Universität Hamburg

Fakultät Wirtschafts- und Sozialwissenschaften

Schwerpunkt Operations & Supply Chain Management

Collaborative Planning in Detailed Scheduling

Der Fakultät Wirtschafts- und Sozialwissenschaften zur Erlangung
des akademischen Grades

Doktor rerum politicarum

vorgelegte

Dissertation

eingereicht im:	Juli 2009
von:	Dipl.-Wi.-Ing. Jan Benedikt Scheckenbach
	geboren am 11. Juni 1980
	in Gräfelfing
Erstgutachter:	Prof. Dr. Hartmut Stadtler
Zweitgutachter:	Prof. Dr. Heinrich Braun
Datum der mündlichen Prüfung:	07.12.2009

Acknowledgements

The author is greatly indebted to many people:

His parents for their unconditional love, constant encouragement and reliability.

Prof. Dr. Hartmut Stadtler and Prof. Dr. Heinrich Braun for their high academic standards, their expertise, their unwavering support, their motivating ideas and uncountable fruitful discussions.

Prof. Dr. Jürgen Branke and Prof. Dr. Kalyanmoy Deb for their education and inspiration.

The whole SAP SCM Optimization team, in particular to Dr. Christopher Sürie and Thomas Engelmann for their professional advice, but also to all others for their loyal support and enrichment of workaday life.

Tatjana Samsonowa, Carolin Püttmann and Martin Albrecht for sharing the experience of a European Union–funded research project. Philipp Schardt, Florian Frey and Heiko Walz, who contributed by their hard work, critiques and commitment.

Contents

Appendices

Mathematical symbols and abbreviations

Repeatedly used mathematical symbols

$\vert . \vert$	cardinality of a set
$\lfloor . \rfloor$	floor function
$\lceil . \rceil$	ceiling function
$x \succ y$	"x is better than y" or "x has higher fitness than y"
a_r	constant capacity of resource r
a_{rt}	time-dependent capacity of resource r
c_{jm}	cost of running activity j in mode m
d_j	duration of activity j as an integral number of periods
d_{jm}	duration of activity j in mode m
dd_j	soft-constrained due date of activity j
dd^*_j	static, non-varying due date of activity j
dl_j	deadline of j
dl^*_j	static, non-varying deadlines of activity j
e	index of a planning domain
ef_j	propagated, earliest finish date of activity j for infinite capacity
E	set of all planning domains
es_j	propagated, earliest start date of activity j for infinite capacity
fd_j	finish date of activity j
hl_m	violation of deadlines of mode m
j, k, l	index for the activities
J	set of all activities
J_S	set of setup activities, $J_S \in J$
J^e	set of activities j within the domain e, $J^e \subset J$
J^e_U	set of upstream-related activities of domain e, $J^e_U \subseteq J^e$
J^e_D	set of downstream-related activities of domain e, $J^e_D \subseteq J^e$
ls_j	propagated, pseudo-hard latest start date of j for infinite capacity
lf_j	propagated, pseudo-hard latest finish date of j for infinite capacity
m	index of mode, $m \in M_j$
m_j	selected mode of activity j
M_j	set of possible modes for activity j
mc_m	Minimum subsequent mode cost of mode m
ml	maximum lateness of a schedule
ml_m	maximum estimated lateness of mode m
ms	makespan of a schedule
ms_m	estimated makespan of mode m

Repeatedly used mathematical symbols

P_j	set of activities which immediately precede j
P_j^e	set of predecessors of activity j within domain e
pdd_j	propagated, soft-constrained due date of j
r	index for renewable resource type, $r \in R$
r_m	primary resource of mode m
R	set of all renewable resources
R^e	set of resources r within the domain e
rd_j	release date of activity j
rd_j^*	static, non-varying release date of activity j
sd_j	start date of activity j
S_j	set of activities which immediately succeed j
S_j^e	set of successors of activity j within the domain e
sl_m	slack consumption of mode m
sc_m	estimated setup cost increase of mode m
st_m	estimated setup time increase of mode m
t	index for periods, $t \in \left[0,\ldots,\overline{T}\right]$
T	planning interval
$\overline{T}$	planning horizon, i.e. the end of the planning interval $\left[0,\ldots,\overline{T}\right]$
thl	total pseudo-hard lateness of the schedule
tl	total lateness of the schedule
tl_m	total estimated lateness increase of mode m
tmc	total mode cost of a schedule
tsc	total setup cost of a schedule
tst	total setup time of a schedule
twc	total weighted cost
u_{jr}	constant per-period demand by activity j for resource r
u_{jmr}	constant per-period demand by activity j in mode j for resource r
wml	weighting of maximum lateness objective
wms	weighting of makespan objective
wtl	weighting of total lateness objective
$wtmc$	weighting of total mode-cost objective
$wtsc$	weighting of total setup-cost objective
$wtst$	weighting of total setup-time objective
λ_{ij}	minimum time lag between activity i and j
$\overline{\lambda}_{ij}$	maximum time lag between activity i and j
$\pi(x)$	permutation of activities ($\forall x, y \in [1 \ldots \lvert J \rvert] : \pi(x) = j \in J$ with $\pi(x) \neq \pi(y)$ for $x \neq y$)
$\pi^{-1}(j)$	inverse permutation ($\forall j \in J : \pi^{-1}\left(\pi\left(j\right)\right) = j$)
ψ_{jk}	setup cost for changeover from activity j to k
$\upsilon_e(p)$	high level ranking of solution p
$\omega_e(p)$	low level ranking of solution p
σ_{jk}	setup duration for changeover from activity j to k

Abbreviations

APS	Advanced Planning System
BOM	Bill-of-Material
CPU	Central Processing Unit
CX	Component crossover
DEAL	Decentralized Evolutionary Algorithm
DS	Detailed Scheduling
DTDG	Deterministic Test Data Generator
EA	Evolutionary Algorithm
FDS	Fix delayed activity-sequence
FS	Fix sequence of all activities
FWD	Forward scheduling
GA	Genetic Algorithm
ICL	Insert components most left
ICR	Insert components randomly
IDD	Insert due dates
LA	Left-alignment
LX	Linear crossover
MLCLSP	Multilevel Capacitated Lot-Sizing Problem
MOEA	Multi-Objective Evolutionary Algorithm
MRP	Material Requirements Planning
PDD	Propagate due dates
PP	Production Planning
RCPSP	Resource-Constrained Project-Scheduling Problem
RA	Right-alignment
RC	Rearrange activities in connected components
RMD	Relax most delayed (upstream-related) release dates
RMP	Relax most promising (upstream-related) release dates
RMS	Relax minimum slack (upstream-related) release dates
RNB	Relax nonbottleneck upstream-related) release dates
rhs	right-hand side
RRD	Relax all (upstream-related) release dates
RTDG	Randomized Test Data Generator
SC	Supply Chain
SCM	Supply Chain Management
SCP	Supply Chain Planning
SGS	Schedule-generation scheme
SPP	Simple Production Planning
SPPM	Simple Production Process Model
SSB	Set start-date bounds
TDG	Test Data Generator

CHAPTER 1

Introduction

This thesis presents a decentralized mechanism for aligning detailed schedules of several companies, being interdependent in form of supplier–manufacturer relationships. In such a relationship, production processes of the manufacturer cannot start until the suppliers have provided the necessary items. Detailed Scheduling is the task of assigning production activities to scarce resources, where resulting plans determine activities' start and finish times down to the second.

In the last decades, Advanced Planning Systems (APS) have been put forward by industry and academia to solve Supply Chain (SC) planning problems. APS employ Operations Research techniques that are able to simultaneously consider constraints related to resource-availability, material flow and time. To cope with the complexity of the planning problems, APS are typically organized in a hierarchical manner. That is, on a mid-term level so-called Master Planning centrally calculates rough cut plans, i.e. quantities of sourced, produced, transported and distributed items across the different production sites and distribution centers of a company. Master Planning relies on aggregated data and generates material quantity targets for subsequent Production Planning and Detailed Scheduling. The latter two planning methods are then typically applied separately for each production site. Production Planning is concerned with translating material quantity targets into concrete production activities, whereas Detailed Scheduling is about computing a good sequence and a feasible resource assignment for the activities. As detailed schedules per plant have to respect superordinated rough cut plans on enterprise level, APS inherently employ a hierarchical form of coordination.

The problem with the above approach is that it is not applicable to companies that are interdependent on delivered items but legally separate organizations. A superordinated decision-making process fails because of the missing central entity and the reluctance of SC members to disclose sensitive data. Typically, a hierarchical coordination requires full visibility of data, such as production and holding costs, Bill-of-Materials, and resource availability. However, keeping this data private is considered vital by most companies. As a remedy, successive planning strategies are employed today, a typical example is upstream planning. In upstream planning, the more powerful downstream entity calculates its production plan first, generating demand for subsequent upstream entities. Most of today's "collaboration" systems, such as vendor-managed inventory, are actually implementations of successive planning strategies. From a mathematical perspective the interorganizational optimization problem is split into separate interdependent subproblems. The subproblems are then solved sequentially, whereas already-solved subproblems define the constraints for yet unsolved subproblems. For example, suppose a manufacturer setting up his production plan according to actual demand and forecasts. By doing this, the manufacturer creates the demand for his suppliers that have to plan their production accordingly. However, successive planning pays no respect to the suppliers' actual production capacities. Not surprisingly, successive planning strategies lead to suboptimal results, which often will result in (either) wasted production capacity, larger safety stocks, decreased service levels and increased production costs.

In recent years, academia (see, for example, Ertogal and Wu, 2000; Fink, 2004; Cachon and Lariviere, 2005; Kutanoglu and Wu, 2006; Sucky, 2006; Stadtler, 2007a; Dudek, 2007; Dahl and Derigs, 2008; Walther et al., 2008) has put forward advanced coordination mechanisms that try to attain the results of hierarchical coordination (or centralized planning) without requiring the parties to exchange sensitive data. Subsuming the different approaches, the term Collaborative Planning has emerged as a new research area. From a practical perspective, Collaborative Planning mechanisms have to cope with three difficulties: First, they should support complex operational planning problems. Second, the exchange of sensitive data must be avoided. Third, the mechanisms are required to be incentive compatible to prevent opportunistically acting partners from supplying systematically biased input that changes overall results to their advantage. It is very difficult to consider all three requirements to the fullest. This, and the still-prevailing paradigm of successive planning have hampered the introduction of Collaborative Planning mechanisms in practice.

This thesis aims to provide a coordination mechanism that focuses on a collaboration of limited scope and thus fulfills the above requirements to a degree acceptable for practical implementations. We restrict our task to coordinating detailed schedules. Being computed only shortly before execution, badly aligned detailed schedules most obviously demand a coordinated solution. Scheduling is about finding the right sequence of production activities, which then determines the activities' start and finish times. However, we will argue that without coordination each partner of the SC would choose a different sequence which often leads to bad overall results. In contrast to prevailing literature, we do not consider side payments transferred from one SC member to another. Though this decision can prevent the mechanism from finding optimal solutions, it gives less incentive to supply biased input. Computational results prove that the performance of the scheme is still of considerable quality. In order to support different kinds of optimization engines, the coordination scheme employs techniques from Evolutionary Algorithms and will be denoted Decentralized Evolutionary Algorithm (DEAL) henceforth.

This research makes use of a proprietary SAP optimization tool—the so-called Detailed Scheduling (DS) Optimizer. In turn, a proof-of-concept is provided, proving that the coordination mechanism can be successfully applied by SC members using this optimizer.

This work is organized as follows. Chapter 2 gives a brief introduction to APS. In Chapter 3, the aim of Collaborative Planning is specified in greater detail and an overview of the current state of the art is given. Chapter 4 presents the general concept of the DEAL framework: the basic working principles independent of the intradomains' optimization problems. A common Detailed Scheduling problem, the so-called Resource-Constrained Project Scheduling Problem (RCPSP), is defined in Chapter 5. In Chapter 6, SAP's proprietary tool for solving RCPSPs—the DS Optimizer—is introduced. Chapter 7 discusses the customized version of the DEAL framework, specifically adapted to the DS Optimizer. In Chapter 8, test instances and the related results are presented. Finally, Chapter 9 summarizes pro and cons of the mechanism and concludes with possibilities for future research.

CHAPTER 2

Overview of Advanced Planning Systems

The planning and controlling of a Supply Chain (SC) are hard tasks. There are various software solutions providing a rich set of tools to support decisions in this complex environment. So-called Enterprise Resource Planning (ERP) systems cover a broad range of functionality, such as Material Requirements Planning, Accounting, Controlling, Human Resources, Research and Development, etc. These systems are transaction based and support all relevant business processes, seamlessly integrated across the SC to enable corporate-wide resource planning based upon a central data warehouse.

However, complex material and resource related constraints in production, procurement and distribution lead to planning problems that are beyond the capabilities of an ERP system. For example, the task of computing the optimal sequence of orders to be produced on different machines with limited capacities while respecting material availability and customer demand defines such a problem. Other examples are the optimal location of distribution and production centers to reach a set of target customers and the optimal route to deliver items from a distribution center to several retailers.

To solve such problems, Advanced Planning Systems (APS) have been intensively researched in recent years and are continuously developed and improved by business software providers such as SAP. APS consider company-wide trade-offs in the objective function and the tight temporal and capacitive constraints encountered in real-world domains, aiming at the integral planning of the Supply Chain using Operations Research methods. This requires proper definitions of alternatives, objectives and constraints for the various planning problems. Note that traditional Material Requirements Planning, which is implemented in nearly all ERP systems, does not have any of the above properties: It is restricted to the production and procurement area, does not optimize and, in most cases, cannot even consider an objective function, and it is a successive planning system (cf. Fleischmann et al., 2007, p. 84).

To cope with the model's complexity, the planning problem is typically split up in hierarchical levels that are solved subsequently by different APS modules. On a long-term planning level, strategic decisions are encountered, such as the planning of locations for production and distribution centers. On a mid-term level, cost-optimal plans are derived in a Master Planning procedure.

Here, the structure of the supply network is viewed as a fixed constraint in an attempt to optimally load production resources and efficiently use available distribution channels taking current orders and forecasts as input. Master plans usually rely on aggregated data. These plans have to be disaggregated to detailed production / scheduling instructions and transportation plans. On the short-term level, the focus is on finding feasible plans that fulfill the aggregated master plans, computing detailed production or transportation schedules.

The length of the planning horizon usually decreases when going from strategic to operational decisions. Commonly, a rolling schedule approach is followed at every level. That is, while maintaining a longer planning horizon, e.g. to capture seasonal fluctuations, planning results are only executed for a much shorter time period, the so-called frozen horizon. After a certain period, plans are re-optimized to include updated demand forecasts, while respecting the results of the previous frozen horizon and the instructions of the next higher decision level.

Figure 2.1 shows the several software modules belonging to an Advanced Planning System in the so-called Supply Chain planning matrix. The modules are arranged according to temporal and functional dimensions.

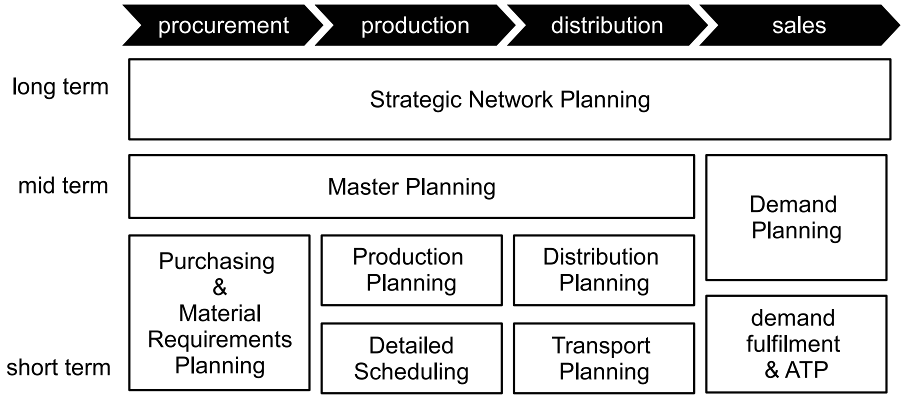


Figure 2.1: Software modules covering the SCP-Matrix, taken from Meyr et al. (2007b, p. 109).

In the following, the functionality of medium- and short-term planning modules, Master Planning, Production Planning and Detailed Scheduling, is discussed briefly.

2.1 Master Planning

Rohde and Wagner (2007, p. 159) give the following definition of **Master Planning**.

> Master Planning supports mid-term decisions on the efficient utilization of production, transportation and supply capacities, seasonal stock, as well as on the balancing of supply and demand. . . . The results of Master Planning are targets / instructions for Production Planning and Detailed Scheduling, Distribution and Transport Planning as well as Purchasing and Material Requirements Planning.

The aim of Master Planning is to synchronize the flow of materials by optimizing an aggregated model. The planning horizon is divided into several periods, so-called time buckets. One of the major characteristics of Master Planning is that the detailed timing of production activities, input requirements and output products is ignored. Instead, cost-optimal quantities of produced, transported, procured and delivered items are sought for each bucket. It is worth stressing that typically only deterministic models serve for Master Planning.

The decision situation for Master Planning differs depending on the problem's characteristic. However, the objective of Master Planning is always to create an optimal plan by balancing the costs of supply–demand matching and inventories while respecting related capacities and material-flow balance constraints. Given the customer demand, the following options are evaluated if bottlenecks on production resources occur (cf. Rohde and Wagner, 2007, p. 160).

- Produce in earlier periods while increasing seasonal stocks.
- Produce at alternative sites with higher production and / or transport costs.
- Produce in alternative production modes with higher production costs.
- Buy products from a vendor with higher costs than your own manufacturing costs.
- Work overtime to fulfill the given demand with increased production costs and possibly additional fixed costs.

For transportation lines, the following alternatives must be taken into consideration.

- Produce and ship earlier while increasing seasonal stock in a distribution center.
- Distribute products using alternative transportation modes with different capacities and costs.
- Deliver to customers from another distribution center

Master Planning problems are typically described as a mixed integer linear program (MILP). As already stressed in the introduction, Master Planning requires the aggregation of sensitive data, such as Bill-of-Materials, production, transport, and holding costs, and capacity profiles. It is thus only suitable for synchronizing production activities between different sites of the same company; it is not suitable for coordinating legally separate enterprises.

2.2 Production Planning and Detailed Scheduling

According to Stadtler (2007b, p. 197), **Production Planning and Detailed Scheduling** aim to generate detailed production schedules for the shop floor over a relatively short period. The primary goal is to create feasible schedules that respect the instructions of the Master Planning level. However, data for Production Planning and Detailed Scheduling is not as aggregated as it is for Master Planning. Instead, detailed start and end dates of activities are of importance. The master plan sets the frame for decentralized Production Planning and Detailed Scheduling decisions at the different locations. According to Stadtler (2007b, p. 198), the Master Planning directives are usually

- the amount of overtime or additional shifts to be used,
- the availability of input items from upstream units in the Supply Chain at different times,
- purchase agreements concerning input materials from suppliers.

Furthermore, directives will be given by the master plan due to its extended view over the Supply Chain and the longer planning interval. As directives, we might have

- the amount of seasonal stock of different items to be built up by the end of the planning horizon (for production units facing a make-to-stock policy),

- given due dates for orders to be delivered to the next downstream unit in the Supply Chain (which might be a subsequent production stage, a shipper or the final customer).

Due to their combinatorial complexity, Production Planning and Detailed Scheduling problems are often tackled by heuristics and metaheuristics.

2.3 SAP SCM APO

Within the SCM business suite, SAP offers an Advanced Planning System called APO ("Advanced Planning and Optimization"). APO is today offered as a planning system on top of SAP's ERP system as well as an independent SCM software suite. APO consists of several modules that can be arranged according to the Supply Chain Planning Matrix (cf. Figure 2.1) using proprietary names. The planning modules Supply Network Planning (SNP), Deployment, and Production Planning and Detailed Scheduling (PP/DS) are most relevant for our discussion.

SNP is used to model the entire supply network including all associated constraints. The module offers support for the creation of feasible master plans for purchasing, manufacturing, inventory and transportation, and for a close match of supply and demand. The planning horizon is divided into discrete time buckets to generate quantity-based plans. The main application fields of the optimizer are *source determination*, *lot sizing* and *inventory control*. On one hand, source determination is concerned with finding the right product mix, deciding which products are to be produced, transported, procured, stored and distributed in which bucket, in which quantities and in which plant. On the other hand, the best mix of resources and the best utilization of production process models are sought, the so-called technology mix. A production process model (or a production data structure) extends the concept of a Bill-of-Materials by defining for each production process a recipe of required input materials, which output materials are produced, which resources can be used, the related resource utilization and so forth. SNP determines the periods and locations of production, procurement, storage and transportation. Lot sizing is concerned with the grouping of orders or items into large lots in order to save setup costs. For inventory control, the planning of safety stocks and target days of supply (the time production can be continued if the flow of input material is interrupted) are supported. Moreover, SNP allows the consideration of maximum stock-level and shelf-life requirements. Using costs in order to evaluate the quality of a plan, the optimizer seeks the least-cost plan in terms of total costs. Total costs refer to production, procurement, storage, and transportation costs, costs for increasing resource capacities (production, storage, transportation, handling) and costs for violating service level requirements (safety-stock-level violation, late delivery, nondelivery). The SNP optimizer uses linear-programming (LP) and MILP techniques to solve the optimization problem. SNP offers support for handling large problems up to the size of several million variables and constraints by providing several problem-decomposition techniques.

According to Dickersbach (2003, p. 151), the idea of SNP is to generate an aggregated, rough-cut, mid-term production plan to get an overview of the overall capacity requirements, the requirements for key components and to trigger the procurement of components with long delivery times. SNP-planned orders are either transferred directly to the ERP system or used for further detailed Production Planning.

After Production Planning decisions have been made, Deployment is used for distribution planning, where it determines which demands can be fulfilled by existing and firmly planned supply. If quantities are insufficient to fulfill the demand or the quantities available exceed the demand, Deployment makes adjustments to the distribution plan created by SNP. However, Deployment does not change the production plan. Similar to the SNP optimizer, the Deployment optimizer bases its decisions on costs defined in the Supply Chain model, such as transportation and storage costs, costs for increasing resource capacities (storage, transportation, handling) and costs for violating service-level requirements (safety-stock-level violation, late delivery, nondelivery). Also Deployment uses linear programming techniques to solve the optimization problem.

The PP/DS module offers functionality for creating orders and for computing production schedules down to the second. Several Production Planning heuristics exist that allow—similar to standard Material Requirements Planning—the calculation of net requirements, procurement quantities, lot sizes and source determination. Production Planning calculates the so-called pegging net, the matching of internal orders. To schedule orders created by Production Planning or SNP, the Detailed Scheduling Optimizer (henceforth abbreviated as DS Optimizer) performs multilevel forward and backward scheduling. A detailed introduction to the DS Optimizer is given in Chapter 6. For more details on other optimizers in SAP SCM APO, the interested reader is referred to Meyr et al. (2007a) and Dickersbach (2003).

CHAPTER 3

Collaborative Planning

The previous chapter introduced Advanced Planning Systems (APS). The modules of an APS can typically be structured in the two dimensions, *planning horizon* and *Supply Chain process*, cf. Figure 2.1. The distinct modules tackle the subproblems by providing holistic approaches that simultaneously consider all relevant constraints, such as capacity and material availability. Coordination is achieved by the inherent hierarchy of the APS modules. That is, company-wide Master Planning defines requirements for Production Planning and Detailed Scheduling usually applied separately for each production plant. This type of coordination is also referred to as **hierarchical coordination** (cf. Stadtler, 2007a; Schneeweis and Zimmer, 2004; Schneeweis, 2003).

However, hierarchical coordination is no longer applicable for companies that are legally independent, but interdependent from a materials-flow perspective. Setting up a superordinate decision-making process fails because the central entity is missing and because the SC members are less willing to exchange **sensitive data**, data that might give other SC members a strategic advantage (e.g., for future negotiations). In particular, data revealing the internal production structure and the product margin, including Bill of Materials (BOMs), resources capacities and production and holding costs, is regarded as sensitive. While there exists certain initiatives, such as Collaborative Planning, Forecasting & Replenishment to exchange less critical demand and supply data, approaches for truly optimizing and aligning interorganizational production plans have only very recently received research attention. Nevertheless there is an evident need for such mechanisms. SCM should tackle the problem of economically supplying the ultimate customer with goods by aligning legally separated, but technically interdependent processes, planning problems and solution techniques of *different* enterprises. Insufficiently aligned production and distribution plans between the companies lead to inefficient SCs. Such inefficiencies will grow with the size of SCs in a world becoming more and more decentralized and specialized. Today's monolithic APS assume a single decision maker who grasps "total visibility" of system details and makes centralized decisions for the entire SC (cf. Nie et al., 2006). Though a central decision entity with global visibility is actually missing, this paradigm is still present in the minds of the decision makers, who focus on maximizing their company's value and not necessarily the value of the whole SC, neglecting that the two goals are interdependent. Today, the most powerful party simply takes the role of the central decision entity and dictates myopically calculated supply and demand patterns to the other members of the SC. This often-observed form of successive planning is also referred to as upstream planning if the most downstream SC member is the dictating entity (cf. Bhatnagar and Chandra, 1993). Upstream planning will typically lead to a suboptimal allocation of production, inventory and transportation resources. Recent research initiatives indicate that employing decentralized—or distributed—Collaborative Planning techniques in order to coordinate local plans can give a strategic advantage over upstream planning decision making. The aim of Collaborative Planning is to compute interorganizational plans with a solution quality close to the ideal solution a central decision entity could compute, but without exchanging sensitive data. Collaborative Planning focuses not on "fighting to get the largest piece of a pie of fixed size," but primarily addresses "how to make the pie bigger" (cf. Fink, 2004, p. 2).

This chapter discusses the state-of-the-art of coordination mechanisms for Collaborative Planning. According to Stadtler (2007a) we define a **coordination scheme** (also known as **coordination mechanism**) as a procedure for aligning plans of two or more decision making units. When referring to the chronological sequence of such a procedure, we will also employ the term **coordination process**. Section 3.1 gives a definition of Collaborative Planning. Section 3.2 provides a summary of related topics and lists relevant literature. Finally, Section 3.3 defines the scope of the thesis.

3.1 Definition of Collaborative Planning

We define a **planning domain** as a given part of an SC and the related planning processes under the control and in the responsibility of one planning organization. **Collaborative Planning** is about coordinating plans between separated planning domains. The terms *collaboration* and *coordination* are frequently used in literature (cf. Stadtler, 2007a; Kilger and Reuter, 2007; Schneeweis, 2003) in an inconsistent manner. For example, Kilger and Reuter (2007) define Collaborative Planning as follows.

> The idea is to directly connect planning processes, that are local to their planning domain in order to exchange the relevant data between the planning domains. The planning domains *collaborate* in order to create a common and mutually agreed upon plan.

According to the authors, a collaboration can be *material related* or *service related*. In both cases, supplier and manufacturer exchange information about demand for and supply of items. However, material-related collaboration is based on information exchange about the item itself, whereas service-related collaboration requires information about how to make the item, to install it, to transport it and so forth. In general, a collaboration process consists of six phases:

1. The *definition* of a collaborative relationship, incorporating mutually agreed goals, the related time horizon, the contribution of each partner and dispute-resolution mechanisms.

2. During *local domain planning*, the planner takes into account current detailed internal information and material or capacity information of adjacent planning domains.

3. By *exchanging plans*, the partners intend to augment the quality of the plans by exchanging nonsensitive data of their ERP or Advanced Planning Systems.

4. *Negotiation and exception handling* deals with aligning and reconciling plans.

5. The adjusted plan leads to replenishment, production and purchasing orders to fulfill the planned goals and is then *executed*.

6. Eventually, several key performance indicators can be computed during *performance measurement*, evaluating the quality of the process during an as-is/to-be analysis (Kilger and Reuter, 2007, cf.).

Schneeweis (2003) distinguishes four different *degrees of coordination* (Schneeweis, 2003, p.5). The lowest level is *data integration* and standardization of information exchange serving as a foundation for *system integration*. *Integration through planning activities* tries to coordination the decision processes of the separate systems. Eventually, *integration through leadership activities* focuses on the strategic orientation of the SC.

In practice, several standards already exist, including EDI and RosettaNet, for exchanging demand and supply data. These standards are often used to enable web-based inventory collaboration platforms (system integration). SAP's Supplier Network Collaboration (formerly known as Inventory Collaboration Hub) is a typical example of such a platform. Supplier and manufacturer agree on upper and lower bounds for stock levels. Exceptions from these levels are reported by automatically generated alerts. Nevertheless, despite recent efforts to standardize information exchange, the amount of sensitive data and the coordination of planning activities has remained limited.

However, today's monolithic APS require all data and are therefore currently applied only locally. One of the main problems of this decentralized application is the consideration of interdependencies between multiple exchanged items within an *automated* exception handling. Today, holistic models are only used to compute local domain plans, but the triggering of alerts is usually still based on simple predefined rules. While today's systems are usually able to detect and report deviations, they are usually unable to automatically compute alternative proposals for complex interorganizational planning problems. Thus, the planner receives only limited information on how to actually resolve a bottleneck. Even worse, the resulting reconciliation efforts might increase with the number of items and the complexity of the planning problems (cf. Kilger and Reuter, 2007). A manual reconciliation of production plans after a machine breakdown has occurred can be a tedious and time-consuming process, since no partner actually knows the other side's planning model.

From a mathematical perspective, the global problem is split up into several, local-domain subproblems. If a planning domain solves its related subproblem, only limited attention is paid to adjacent planning domains. As already indicated previously, the most powerful SC member usually has a *leading* role, while the others are *followers*. The leading entity solves it subproblem first, generating constraints for subsequent planning steps of adjacent followers. Obviously, there is no guarantee that this proceeding leads to optimal results with respect to the whole SC. For example, the leader might dictate supply patterns that lead to huge production cost in adjacent domains by requiring additional overtime to fulfill the proposed demand. It is not surprising that a leader might be tempted to exercise his power to generate additional profit at the expense of his followers. However, what counts for the ultimate customer are quality, availability and cost of the product—not the success of a single enterprise in a multitier SC. In this work, it will be shown that all parties—leader and followers—can realize additional savings over the successive planning strategy of upstream planning. In this light, the definition of Stadtler (2007a) for Collaborative Planning is more appropriate:

> . . . we define Collaborative Planning as a joint decision making process for aligning plans of individual SC members with the aim of achieving coordination in light of information asymmetry.

Information asymmetry is a term stemming from game theory and describes the state where no SC member possesses all the information or preferences of other SC members. According to the author, there exist several definitions for *SC coordination*. The most stringent definition comes from the contract literature (see also the next section).

> A contract coordinates the SC, if (and only if), the set of supply chain optimal actions is a Nash equilibrium, i.e., no firm has a profitable unilateral deviation from this set. Here, coordination requires a solution which represents both a (central) SC optimum as well as a Nash equilibrium (cf. Stadtler, 2007a).

A game-theoretic analysis usually requires restrictive assumptions about the output of the *game*, such as the distribution of SC-wide savings that are realized when applying a coordination mechanism. A softer definition only requires an SC-optimum solution, not a Nash equilibrium. Other authors speak of coordination, if the initial situation could be at least improved. Certainly, this definition is most appropriate from a practical perspective. To summarize the above, current planning systems are set up in a monolithic way, supporting a holistic treatment of planning problems within a single planning domain. The holistic consideration of capacity and material availability constraints is a fundamental progress over Material Requirements Planning. However, from an interorganizational perspective, unaligned local planning activities contradict the holistic idea. Thus, as ultimate goal, Collaborative Planning should enable SC partners to act as one virtual planning organization without requiring them to exchange sensitive data.

3.2 Related topics and state-of-the-art

Coordination mechanisms can be classified by several criteria. Most obviously, the business function can be used for classification, production, inventory, supply or distribution planning (cf. Bhatnagar and Chandra, 1993). A more general taxonomy is provided by Whang (1995). Also Stadtler (2007a) developed a typology including other criteria, such as the number of tiers, the decision models used, the type of information exchange, the objective of coordination and so forth. We recommend readers the above works if they are interested in ways for classifying coordination approaches. Here, we will go rather briefly through the different research areas that provide mechanisms for coordinating decentralized decision units. These areas are outlined below with their strengths and limitations.

3.2.1 Hierarchical coordination

Schneeweis (2003) provides a comprehensive treatise of distributed decision making. A common concept in distributed decision making is hierarchical planning (also called hierarchical coordination). The idea behind this concept has already been discussed in the first two chapters. The author provides an interesting formalization for hierarchical planning approaches, shown in Figure 3.1.

Hierarchical planning can be characterized by a **top level** instructing a **base level**, such as Master Planning providing material quantity targets for subsequent Production Planning and

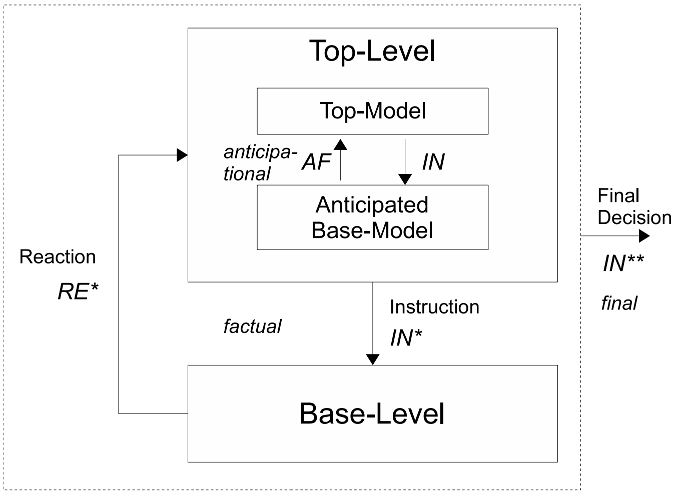


Figure 3.1: Interdependence of hierarchical levels, cf. Schneeweis (2003, p.27) .

Detailed Scheduling. In order to derive purposeful instructions, the top level might anticipate the base level. For example, Master Planning anticipates Production Planning and Detailed Scheduling by considering aggregated capacities and material quantities within the discrete time buckets. Formally, the top level anticipates the base level by hypothetically applying different instructions IN as stimuli to an **anticipated base level**. These instructions give rise to possible (hypothetical) reactions and finally one arrives at an optimal instruction IN^* which is then communicated to the *real* base level. The base-level's reaction RE^* allows the top level to calibrate its anticipating function AF further. This function, respectively the anticipated base model can be part of the top level's decision problem. For example, Master Planning usually anticipates costs for overtime by directly implementing additional constraints and objectives. We will denote the data related to the anticipated base model as **anticipating data** henceforth. There might exist several feedback cycles, after which the hierarchical system arrives at a final decision IN^{**} communicated to the environment.

The above scheme underlies many different approaches, for example Production Planning, Principal Agent Theory, SCM or Managerial Accounting. Characteristically only the top level anticipates the base level but not vice versa. Also successive Production Planning strategies, such as upstream planning, are closely related to hierarchical coordination. In upstream planning, the downstream domain sets the instructions for adjacent upstream domains. One possibility to achieve coordination of production activities over several planning domains is to calibrate the anticipating data of the instructing domain further. Another option is to extend the concept such that the domains iteratively anticipate each other during several cycles. Most approaches for Collaborative Planning presented subsequently employ one of these two ideas.

3.2.2 Multi-echelon systems and contract theory

Several research areas focus on models that allow to derive analytical solutions. A large body of literature is concerned with multi-echelon systems. Such a system includes several stages (e.g., representing production and distribution sites) that are grouped into several echelons (sometimes also called tiers). The body of literature can be divided into two parts. For *deterministic, static systems*, the demand is assumed to be certain and the focus is on finding the optimal trade-off between the holding and fixed-order costs. In contrast, *stochastic systems* assume a stochastic demand distribution, generally with negligible fixed-order costs; the focus is thus on finding the optimal trade-off between holding and stock-out costs.

Static multi-echelon systems rely on restrictive assumptions, such as static demand in a singe-item, single-production-stage scenario. These assumptions allow the analytic derivation of optimal solutions. That is, the computation of *economic order quantities* and *economic production quantities* for fixed costs per ordered lot and inventory-holding costs. Assume a single company choosing its replenishment policy, facing a constant demand rate d for a single product, fixed-order processing cost B and variable inventory-holding costs of h per unit. In such a simplified model, the total cost per time unit amounts to

$$K(x) = B \cdot \frac{d}{x} + \frac{x}{2} \cdot h, \tag{3.1}$$

where x denotes the order quantity. The economic interpretation of the model is that higher-order volumes lead to decreased processing costs per item (since B remains constant), but also to increased holding cost. The optimal quantity x^* can be computed by setting the first-order derivative to zero, resulting in

$$x^* = \sqrt{\frac{2 \cdot B \cdot d}{h}}.$$

Historically, the three main functional areas of the SC (procurement, production and distribution) have been investigated independently, buffered by large inventory before models for multi-stage systems had been developed. The principal difference from a single-stage model is that the analysis seeks not only the optimal trade-offs between order and inventory costs, but also the optimal trade-offs between the preferences of the different stages, whereas each stage might be handled by a different planning domain. For example, large orders might be beneficial to suppliers, because of potential savings in order processing costs, manufacturing setup costs or distribution costs. However, a manufacturer might prefer more frequent, smaller orders due to high inventory cost (cf. Bhatnagar and Chandra, 1993). If the manufacturer is the more powerful party, the supplier's problem is to persuade the manufacturer to change his order policy by making concessions such as price discounts or compensation payments (sometimes also referred to as side payments). The models' restrictive assumptions allow analytical proofs, which makes the field attractive for many researchers. The number of related publications is impressive. Comprehensive reviews can be found in the works of Goyal and Gupta (1989) and Thomas and Salhi (1998). However, from a practical point of view, all studies on static lot-sizing models share a common critique: The assumptions are too strict and the results cannot be carried over to more sophisticated models. In particular, the models give no advice on how to align several APS under information asymmetry. Thus, instead of a comprehensive review only some articles are highlighted here to illustrate this area's main ideas.

Monahan (1984) provides some theoretical considerations for the quantity discount policy of a supplier delivering to one manufacturer. While the buyer is rather interested in relatively small order sizes to keep his inventory costs down, the supplier prefers large order sizes to decrease his order processing, manufacturing and shipping cost. However, being the more powerful party, the manufacturer imposes his order policy which leads to an overall suboptimal solution. The author shows the effects of quantity discounts on the manufacturer's order policy. Suppliers can realize these price discount by sharing a part of their related savings. Banerjee (1986) also proposes an inventory model for an SC consisting of one manufacturer and one supplier. The intradomain problems of both parties can be expressed by Equation 3.1. While the manufacturer wants to minimize the sum of fixed and variable costs (e.g., order and holding costs) of his orders, the supplier aims to minimize fixed and variable production costs (e.g., relating to setup and holding). The author studies the influence of the model's parameters and analytically derives the SC-wide optimal solution, which does not—in most cases—equal the myopic solutions of manufacturer or supplier. The solution leaves all efficiency gains to the supplier, as he shares—as rational player under complete information—only the minimum possible part of a saving.

Banerjee et al. (2007) extend the model to coordinate replenishment decisions for interlinked procurement, production and distribution inventories. The authors assume a scenario of one manufacturer batch producing a single item to be distributed by several retailers, whereas required input material is ordered from several suppliers. The suppliers are allowed to deliver equal-sized subbatches at constant frequency to the manufacturer. At each stage of the SC, the decision has to be made which quantity to order at which frequency. That is, the analysis seeks the optimal trade-off between continuous inventory-holding cost and the fixed-order cost. The demand and production rates are assumed to be constant and transportation and handling times are neglected, as well as production and inventory capacities. Since the manufacturer is assumed to know production rates, demand rates, order costs and inventory costs of all stages, a functional expression of SC-wide costs relating to the order policy can be set up. A difficulty arises because the optimum value cannot be determined analytically due to the integer batch sizes that make the function non-differentiable. A heuristic is implemented that searches for a feasible solution in the neighborhood of the optimal solution of the relaxed problem.

Also, Munson and Rosenblatt (2001) investigate a three-tier SC and related optimal quantity discounts. Abdul-Jalbar et al. (2007) propose a model to retrieve optimal integer-ratio policies for one supplier delivering to two manufacturers. Models for one supplier and multiple retailers are also presented by Chiou et al. (2007), Chen et al. (2001), Lu (1995) and various other authors. The coordination schemes differ in details, such as the type of quantity discount (continuous or discrete). The above investigations assume complete information. Papers that explicitly deal with incomplete information include Sucky (2006), Fransoo et al. (2001), and Corbett and de Groote (2000). Sucky (2006) considers the same setting as Banerjee (1986), where a supplier tries to persuade a (more powerful) manufacturer to change the order policy for an additional compensation payment. Within a bargaining game, the supplier has the chance to announce one take-it-or-leave-it offer. The author shows that under complete information, the supplier acts like a central planner, proposing the joint optimal order quantity derived by Banerjee (1986). In the case of incomplete information, the supplier does not know the cost structure of the buyer. Instead, the buyer's cost structure is assumed to be a random variable that takes two realizations with known probability. Dependent on probabilities and assumed cost structures, Sucky (2006) derives optimal combinations of compensation payments and order policies for rational and risk-neutral players. In the resulting bargaining game, the supplier offers a "menu of contracts," one for each possible realization of the buyer's cost structure. A similar setting is investigated by Corbett and de Groote (2000). Here, the supplier does not know the manufacturer's fixed costs, but only a continuous distribution. Analogous to Sucky (2006), the supplier offers a continuous menu of contracts, consisting of different order-quantity–price combinations.

Stochastic multi-echelon systems are concerned with retrieving the optimal safety stock for a given service level (relating to stock-out costs). For a single stage facing stochastic demand, the optimal safety stock is a quantile of the demand distribution. Within multi-echelon systems, the lead times between the stages might also be stochastic. Research dates back to 1960, with the paper of Clark and Scarf (1960) who derive an exact algorithm for a serial multi-echelon system. Diks et al. (1996) and van Houtum et al. (1996) give an overview about relevant literature since then. Usually, the publications employ a centralized decision making paradigm. As an exception, Fransoo et al. (2001) focus on the noncooperative case. In a two-echelon scenario, they analyze the distribution of inventory stock for cooperative and noncooperative retailers, each requiring a certain service level from the manufacturer.

As a main result, multi-echelon systems provide the insight that lower system-wide costs can be achieved if some planning domains provide transfer payments to create incentives for other planning domains to accept a coordinated solution. However, restrictive assumptions—such as unlimited storage capacity, the consideration of a single item, constant demand and the focus on pure inventory problems (neglecting limited production capacity)—complicate a direct application to practical planning problems of an operational type.

Also contract theory focuses on the analytical investigation of the relationship between a supplier and a buyer of a good. Most of the literature in contract theory is about determining optimal contract parameters given the functional form of a contract. A problem frequently investigated is the so-called newsvendor problem, where a retailer facing stochastic demand has to order the desired quantity from the supplier before the selling season. Both parties are assumed to be risk neutral (i.e., to maximize their expected profit) and to have full information (i.e., each party knows all costs, parameters and rules). The simplest contract is the price-only or wholesale contract that specifies a constant per-unit selling price. Cachon (2003) shows that this type of contract does not coordinate the SC, as the retailer does not order enough inventory to maximize the Supply Chain's total profit. Without any incentives to increase his order quantity, the retailer ignores the impact of his decision on the supplier's profit. This behavior is a manifestation of a phenomenon that has been known for a long time as *double marginalization*, (cf. Spengler, 1950). In a two-tier SC with two monopolies, a markup occurs at every tier, causing the price for a good to be even higher than if the firms where vertically integrated in a single monopoly. As a consequence, the fulfilled demand decreases and the optimal profit cannot be attained.

An investigation of revenue-sharing contracts prevalent in the videocassette rental industry is provided by Cachon and Lariviere (2005). Under this type of contract a percentage of the profit the retailer generates is additionally provided from supplier to retailer. This transfer payment creates an incentive for the retailer to order more videocassettes to better cover the initial peak demand for a new movie. Cachon (2003) lists several types of additional contracts that coordinate an SC by sharing supplier's profit through different forms of transfer payments, namely buyback contracts, quantity-flexibility contracts, sales-rebate contracts and quantity-discount contracts. However, such advanced types of contracts are more costly to negotiate, lead to higher administrative costs and might create additional moral hazard problems. Perakis and Roels (2005) measure the efficiency of price-only contracts for different SC configurations with the "price of anarchy," the difference of profits in the coordinated and uncoordinated solution.

Similar to multi-echelon systems, contract theory provides analytical insights that transfer payments can lead to improved system-wide solutions. However, the prevailing literature constitutes an extreme point by considering very aggregated settings that neglect operational constraints.

3.2.3 Auction theory

Auction theory analyzes price-settling mechanisms under pure market conditions. A central decision entity, the auctioneer, is responsible for determining prices that "clear the market" by finding the compromise with maximal utility for all participants.

There exist two possibilities for applying an auction mechanism to SC planning. First, the partners of choice can be determined by an auction. A typical example in logistics is the choice of third-party logistics providers. However, it is questionable if market conditions really exist *within* an SC: Here, the choice for partners is usually a strategic one, binding a manufacturer to a supplier for a longer time period. According to Stadtler (2007a), pure market conditions only exist at the boundary of an SC. Second, auction mechanisms can be employed for choosing the best interorganizational plan out of a set of alternatives. In this regard, auction mechanisms have been put forward by several authors, in particular for solving scheduling problems. The typical proceeding is that the auctioneer grasps total visibility of resources and sells resource capacity to competing bidding agents, representing, for example, different company divisions. Different mechanisms (in this context also called "auction protocols") exist for finding the optimal schedule based on available bids. A comprehensive study of such mechanisms can be found in Wellman and Walsh (2001). Resource allocation is often interpreted as a combinatorial auction: Each agent bids on a combination of several resource slots (cf. Fan et al., 2003).

Underlying this is the **optimal control problem** of a central authority trying to control several agents in an organization (cf. Kutanoglu and Wu, 2006). The mechanism has to be designed in such a way that the chosen solution best serves the whole organization. In this regard, an important property is **incentive compatibility**. A mechanism is said to be incentive compatible if all the players fare best when they truthfully reveal any private information asked for by the mechanism, i.e. truth-telling is a dominant strategy (cf. Myerson, 1979). These mechanisms are also referred to as **direct revelation mechanisms** as they reveal the true valuation of each agent. General insights were provided independently by Clark (1971) and Groves (1973) in the early seventies; the related mechanisms are thus also referred to as *Clark–Groves* mechanisms. Often, incentive compatibility is achieved by introducing side payments. A common idea is that each player in the auction pays the opportunity cost that his presence introduces to all the other players.

It should be emphasized that auction mechanisms only focus on the problem of selecting the optimal production schedule, assuming the agents' true valuations are considered as private information. The question of how to construct candidate schedules under information asymmetry is not addressed.

3.2.4 Bargaining theory

While auction theory is concerned with pure market environments, bargaining theory focuses on settings with a smaller set of players. Most publications deal with bargaining problems within bilateral monopolies. That is, they provide a game-theoretic analysis for two-player scenarios. For idealized coordination scenarios within an SC, bargaining theory can also give valuable insights. Essentially, a proposal on order quantities or delivery times can be interpreted as a good offered from the supplier to the manufacturer (or vice versa). While each player knows his internal value of the good, he is assumed to have only limited information (in the form of a probability function) of his antagonist's valuation. For such cases of information asymmetry, related publications analyze the efficiency of equilibrium strategies. Chatterjee and Samuelson (1983) provide such an analysis for a bilateral monopoly. Within a nonrecurring sealed-bid double auction, a buyer and a seller of an indivisible good simultaneously reveal their offers. If trading takes place (the buyer's offer is higher than the seller's claim) a settlement is computed according to an *a priori* known mechanism (e.g. by splitting the difference between both offers equally). Though the seller's and buyer's actual values of the good are assumed to be private knowledge, a probability distribution of likely values is supposed to be *common knowledge*; each party knows the probability distribution of the other party, knows that the other party knows the own distribution, knows that the other party knows that its distribution is known and so on. Being rational players, buyer and seller implement "best response strategies," giving a price offer dependent on the own value and the counterpart's probability distribution. This interlinked strategies lead to a Nash equilibrium, a state where no player can increase his expected profit by unilaterally deviating from his chosen strategy. The authors investigate the properties of such equilibrium strategies. A main result is that even if the buyer values the good higher than the seller, a successful sale may be impossible in the equilibrium strategies. Related contributions have been put forward by Harsanyi and Selten (1972), Myerson and Satterthwaite (1983), Samuelson (1984), and Radoport and Fuller (1995).

In addition to the sealed-bid double auction, various price settlement mechanisms exist, such as take-it-or-leave-it offers or the transfer of predefined lump sums. Depending on the mechanism, the bargaining process can be more or less efficient. In bargaining theory, efficiency relates to the percentage of mutual beneficial agreements that can be actually attained by bargaining under the employed mechanism.

Bargaining theory provides interesting analytical insights into strategies leading to Nash equilibria and the related efficiency of the employed price-settlement mechanisms. However, most approaches assume the parties' probability distributions of the value of the good to be public knowledge. It is questionable if this assumption can be justified for practical purposes. A planning domain might already have difficulty estimating its own value distribution for future coordination results based on historical data, since the problem's properties may change frequently and the number of samples is limited. Radoport et al. (1998) and Feltovich (2000) discuss related reinforcement-based learning strategies. As auction theory, bargaining theory is not concerned how the good is created.

Game theory is distinguished from other social sciences by the belief that players act rationally and have well-defined preferences. However, in practice "anomalies" can be frequently observed that contradict this belief (cf. Thaler, 1988). One such anomaly is the ultimatum game, where two players, A and B, have to split up a certain amount of money, say €1. The rules of the game are as follows: First, player A declares the amount he intends to share with B. Second, B can accept or reject the proposal. If accepted, each player receives the share defined by A. If rejected, no player receives anything. The game-theoretic equilibrium strategies for rational players are as follows: A will claim 99 cents for himself as this is the maximum achievable outcome and B will accept the offer, since 1 cent is better than nothing. However, experiments with real humans revealed that low offers from A will be rejected by B. Obviously, B is evaluating A's offer according to its perceived fairness, a criterion which is not included in the rational utility function. Interestingly, the ultimatum game is closely related to coordination mechanisms, where surplus savings have to be shared between the partners. The notion of fairness hence seems to be an important aspect, but it seems questionable if fairness can be captured in a functional form.

3.2.5 Mathematical-decomposition–related techniques

The above approaches allow the derivation of analytical proofs for idealized models. However, in practice, decision makers are confronted with far more complex situations and related models. According to Fink (2004), there is still a lack of investigations that consider deterministic models from Operations Management. In particular, the question of how to align plans computed by APS of different planning domains has come into the focus of research only recently.

Depending on the planning task, intradomain planning problems are often formulated as linear problems (LP) or mixed-integer linear problems (MILP) and the solution is computed by an LP/MILP-solver. From a mathematical perspective, the interorganizational problem is composed by combining the intradomain problems with material-flow-balance equations. Without these constraints, the subproblems would be independent and could be solved separately. There exist several mathematical-decomposition techniques that exploit this fact by splitting up the problem into several slave problems and a master problem to coordinate the slaves. The idea of various authors is to use these mathematical-decomposition techniques to set up a coordination mechanism. This section deals with such decomposition-related techniques. In the case of hard combinatorial or nonlinear problems, the solution is typically computed by a heuristic. Later, Section 3.2.7 discusses coordination mechanisms based on heuristics.

3.2.5.1 Lagrangian relaxation

Lagrangian relaxation was developed to solve large-scale optimization problems more efficiently by exploiting certain problem properties. The principle idea is to split up the original problem into several subproblems that can be treated independently with a master problem to steer the search process. A comprehensive introduction to Lagrangian relaxation can be found in Conejo et al. (2006) and Williams (1999). Here, the approach is only briefly described.

Decomposable problems have a sparse coefficient matrix, where most of the non-zero coefficients can be ordered into a block-angular structure. As an example—taken from Conejo et al. (2006, p. 70)—consider the linear programming problem

$$\min_{x_1,x_2,\dots,x_n} \sum_{j=1}^{n} c_j x_j \tag{3.2}$$

$$\text{s.t.} \sum_{j=1}^{n} d_{ij} x_j \le f_i \qquad \forall i = 1,\dots,q \tag{3.3}$$

$$\sum_{j=1}^{n} a_{ij} x_j = b_i \qquad \forall i = 1,\dots,m \tag{3.4}$$

$$0 \le x_j \qquad \forall j = 1,\dots,n, \tag{3.5}$$

where constraints 3.3 have a decomposable structure in $\overline{e}$ blocks. In the particular case of $\overline{e} = 3$, the problem can be written as

$$\min_{x^{[1]},\, x^{[2]},\, x^{[3]}} \left(\left(C^{[1]}\right)^T \mid \left(C^{[2]}\right)^T \mid \left(C^{[3]}\right)^T \right) \begin{pmatrix} x^{[1]} \\ -- \\ x^{[2]} \\ -- \\ x^{[3]} \end{pmatrix},$$

where the superindices in brackets refer to partitions, subject to

$$\begin{pmatrix} D^{[1]} & \mid & & \mid & \\ -- & - & -- & - & -- \\ & \mid & D^{[2]} & \mid & \\ -- & - & -- & - & -- \\ & \mid & & \mid & D^{[3]} \\ -- & - & -- & - & -- \\ A^{[1]} & \mid & A^{[2]} & \mid & A^{[3]} \end{pmatrix} \begin{pmatrix} x^{[1]} \\ -- \\ x^{[2]} \\ -- \\ x^{[3]} \end{pmatrix} \begin{matrix} \leq \\ \\ \leq \\ \\ \leq \\ \\ = \end{matrix} \begin{pmatrix} f^{[1]} \\ -- \\ f^{[2]} \\ -- \\ f^{[3]} \\ -- \\ b \end{pmatrix}.$$

It is worth mentioning that block-angular structures can be found in many practical problems, see also Williams (1999) and Conejo et al. (2006). In particular, interorganizational planning problems have such a structure because of the interdomain material-flow–balance constraints. This constraint set is usually referred to as the set of "coupling," "linking" or "complicating" constraints.

The problem can be decomposed by "dualizing" the coupling constraints. That is, adding them to the objective function. The problem 3.2–3.5 can be rewritten as

$$\max_{\lambda_1,\lambda_2,\ldots,\lambda_m} \left(\min_{x_1,x_2,\ldots,x_n} \sum_{j=1}^{n} \left(c_j - \sum_{i=1}^{m} \lambda_i a_{ij} \right) x_j \right) \tag{3.6}$$

$$\text{s.t.} \sum_{j=1}^{n} d_{ij} x_j \leq f_i \qquad \forall i = 1, \ldots, q \tag{3.7}$$

$$0 \leq x_j \qquad \forall j = 1, \ldots, n, \tag{3.8}$$

where $\lambda_1, \lambda_2, \ldots, \lambda_m$ denote the Lagrangian multipliers. The idea of Lagrangian relaxation is to iteratively solve the inner minimization problem and—based on the results—update the multipliers until the procedure has converged to a satisfying solution. Since the inner problem has a true block-angular constraint structure, it can be partitioned accordingly and solved separately. Resulting are several independent slave problems and a master problem for updating the multipliers. Obviously, Lagrangian relaxation is attractive if the inner minimization problems are easily evaluated for given multipliers. The difference between the objective function value of the relaxed primal problem for the minimizer and the objective function value of the dual problem of the maximizer is called the duality gap. For an (intermediate) solution of the dual problem the associated solution in the primal problem 3.2—3.5 might be infeasible and therefore feasibility restoring procedures are required. For updating multipliers, subgradient procedures are frequently used (cf. Conejo et al., 2006, p. 195). Here, multipliers are iteratively increased in proportion to their constraint violation in the primal problem. The subgradient procedures are usually easy to implement, but progress slowly to the optimum in an oscillation fashion requiring many iterations—a consequence of the nondifferentiability of the dual objective function, see Zhao et al. (1999) or Van de Panne (1991). Lagrangian relaxation can generally be applied to nonlinear optimization problems. Dantzig–Wolfe decomposition can be regarded as a special application of the Lagrangian relaxation method for linear optimization problems (cf. Conejo et al., 2006, p. 233).

As already discussed, practical interorganizational planning problems usually have a block-angular constraint matrix, which makes Lagrangian relaxation attractive as a building block for coordination mechanisms. The partitioned subproblems can be treated independently and are solely coordinated over multiplier values. Moreover, the master only needs the information of realized material flows to update the multipliers, not the information of domain-internal production constraints. Hence, Lagrangian relaxation inherently supports scenarios with information asymmetry. Additionally, multiplier updating algorithms and related proofs on convergence are already known and can be incorporated. There are several articles in the literature including aspects of Lagrangian relaxation to different degrees. The approaches share a common methodology:

1. A production planning problem is defined as a mixed-integer linear program.

2. Coupling constraints are identified and dualized. The problem decomposes into subproblems that are handled separately by the local planning systems of the distinct planning domains.

3. A coordination mechanism for steering the local planning procedures is introduced. Usually, control over the SC is gained by altering the subproblems' coefficients (multipliers) of decision variables. Either a mediator is used to steer the search process centrally, or planning domains themselves transmit proposals to other planning domains.

4. In some papers, an incentive scheme is defined. Not all planning domains will directly profit from coordination. Though the coordinated solution employs a better resource utilization and realizes cost savings over the initial situation (e.g. the result of upstream planning), the savings might not be distributed fairly and incentives need to be generated for all partners to take part in a coordination process, such as by transferring compensation payments.

In the following, several related literature findings will be discussed briefly. Thomalla (1998) developed an algorithm for solving job-shop problems with Lagrangian relaxation. Several orders, consisting of specific jobs and precedence constraints need to be scheduled on machines with limited capacities. The primal problem is modeled as an integer programming problem. By dualizing the finite-capacity constraints, the problem can be decomposed into independent subproblems, where each subproblem contains the jobs of one order. The Lagrangian multipliers are adapted by a subgradient procedure. In addition, a heuristic is used to construct feasible plans. The algorithm was originally developed to efficiently solve job-shop problems with parallel identical machines, and does not support Collaborative Planning in supplier–manufacturer scenarios. Nonetheless, it is interesting from an algorithmic perspective.

Kutanoglu and Wu (1999) dualize the finite-capacity constraints of an integer programming formulation of a job-shop scheduling problem. As in Thomalla (1998), multipliers are iteratively updated using a subgradient procedure, and a feasibility restoration method is applied to solve resource conflicts. The authors argue that finding the dual optimal solution of the problem is equivalent to finding equilibrium prices in an optimal auction design problem, where job-level subproblems are considered as agents and the subgradient method as auctioneer deciding on prices for resource usage. The auction is of a combinatorial type as each agent bids on a set of timeslots on different machines.

Kutanoglu and Wu (2006) investigate a mechanism that combines auction theory with Lagrangian decomposition. In a first phase, several feasible schedules are computed by using the approach of Kutanoglu and Wu (1999). Then in a second phase, the best solution is chosen in an auction mechanism. Domains bid by reporting their costs on every schedule, whereas an auctioneer sells resource capacity. A direct revelation mechanism is established by linking reported costs to subsequent side payments set by the auctioneer, making truth telling a dominant strategy for each domain.

Other authors use Lagrangian relaxation to decentrally solve master-planning problems. Ertogal and Wu (2000) consider a multilevel capacitated lot-sizing problem (MLCLSP) as an interorganizational problem. In their quantity-based model, setup and holding cost are be minimized, subject to limited capacity and material-flow–balance equations. The planning domains are assumed to share no resources, but to be situated in a supplier–manufacturer relationship. Hence, material-flow–balance equations between the planning domains are relaxed to obtain the decentral submodels. Apparently, the Lagrangian relaxation approach is not without problems, as a nonzero duality gap is likely at convergence (cf. Ertogal and Wu, 2000, p. 936). That is, the mechanism does not terminate in reasonable time with a feasible interorganizational problem. In general, the subgradient method is notorious for oscillations (cf. Zhao et al., 1999).

Regarding the MLCLSP, the oscillations can be explained easily. Supplied or delivered quantities are determined to a large degree by holding costs and Lagrangian multipliers. As a result, the material-flow values tend to take either zero or the maximum value possible, as holding costs are larger or smaller than the Lagrangian multipliers. To counteract the oscillations, Ertogal and Wu (2000) modify the Lagrangian relaxation approach by considering fairness as an additional objective. The fairness objective overcomes the above difficulties of slow and oscillating convergence since it introduces target material flow values into the submodels. At the beginning, demand information is propagated in the network by Bill-of-Material explosions across all planning domains. Then, each planning domain computes its facility-best solution. For each material-flow equation, a "fair" (but infeasible) alignment is computed as the mean of the initial myopic facility-best choices. Deviation from this fair solution is then penalized with a "conflict price" (the value of a Lagrangian multiplier) in the submodels. A mediator is responsible for reducing overall inconsistency by updating conflict prices and target material flows. Analogous to the subgradient procedure, conflict prices are updated proportionally to the deviation of current-to-target material-flow values. In addition, target material flows are updated according to a fairness criterion. The facilities that "suffer" more have a greater influence on new target values. Suffering is measured by the deviation from facility-best solutions. The scheme does not minimize total SC costs, but searches for a consistent and fair solution. The results are compared with the central solution, the solution of the interorganizational MLCLSP solved by a fictitious central decision entity.

Nie et al. (2006) also study the MLCLSP. As above, facility separable subproblems are obtained by dualizing the interdomain material-flow–balance constraints and propagating the demand by a Bill-of-Material explosions. The authors extend the standard Lagrangian relaxation approach in two aspects. First, multipliers are updated by a "surrogate subgradient method," that works if only a subset of planning domains report their solutions. Second, a feasibility restoring method (involving all domains) is applied before multipliers are updated. By doing this, a smoother convergence is achieved, as a comparison with central results indicates.

Walther et al. (2008) explore Lagrangian relaxation to decentrally solve a product recycling problem. In this model, independent recycling companies cooperate in recycling networks steered by a focal company. While the focal company focuses on arranging recycling contracts according to prevailing legislation, the recycling companies are concerned with collecting, recycling and disposing of the material. Since the recycling companies aim to maximize their individual profit, they do not intend to share cost and capacity information. The authors first set up a linear program from the interorganizational problem. The objective is to maximize the network profit, consisting of fees for accepting the material and cost for disposal, transportation and recycling. Next to material-flow–balance and capacity constraints, the recycling network has to obey legal recycling targets. Dualizing the coupling constraints leads to a company-separable problem. For every submodel, multipliers can be directly interpreted as acceptance fees and additional bonus payments. The focal company is responsible for maximizing the network profit by iteratively updating the multipliers based on a subgradient procedure. The recycling companies respond with material quantities they intend to recycle for the given price structures. Global feasibility of the combined solution is preserved by setting upper bounds for material quantities.

It should be noted that generally there might exist several optimal solutions of a subproblem for given multiplier values at convergence. Thus, at termination, the master needs not only to transmit multiplier values but concrete material-flow and resource-usage targets. The organizational interpretation of mathematical decomposition is that a company has a number of divisions that share resources and a management unit that determines prices for resource usage. Van de Panne (1991) argues that the coordination mechanism is not truly decentralized if the management unit also has to tell some divisions their optimal solution.

3.2.5.2 Coordination by exchange of primal information

Lagrangian relaxation requires the exchange of such multiplier values as dual information. Another possibility is to exchange primal information, target right-hand side (rhs) values for decision variables included in the complicating constraints. Jeong and Leon (2002) combine Lagrangian relaxation with agent theory. The primal problem is decomposed into several subproblems by dualizing the coupling constraints. However, instead of having a global multiplier update rule, so-called "coupling agents" control the search process. Each coupling agent connects to a limited number of subproblems. Thus, a net of coupling agents substitutes the central decision entity for multiplier updating. A related "local" multiplier update rule is presented that requires only the current solution of some subproblems. A difficulty arises, as a coupling agent or a local planning domain is not aware of the full set of complicating constraints (the a_{ij} in Equation 3.6). As a remedy, Lagrangian multipliers are locally approximated by Taylor-series expansions. What is ultimately exchanged can be regarded as primal information: target rhs values for decision variables included in the complicating constraints and penalty weights for penalizing any deviation from the target values. In Jeong and Leon (2005), the authors discuss an application of their coordination mechanism to a single-machine scheduling problem. The model decomposes as capacity constraints are relaxed. For each subproblem, a subset of jobs, connected by precedence constraints, needs to be scheduled.

A valuable contribution for coordinating MLCLSP models is from Dudek and Stadtler (2005). The interorganizational model is decomposed by fixing the material flow from a supplier and a manufacturer to certain values. This is achieved by setting the related rhs values for each planning domain. The planning domains are assumed to be able to perform additional shifts at extra expense to fulfill customer demand, such that the problem does not become infeasible, irrespective of the rhs value. Backlogging is not allowed. At the beginning, upstream planning is used to compute the initial "supply pattern." Then the planning domains alternatingly propose supply quantities and associated cost increases. Each planning domain tries to compute reasonable proposals by approximating its antagonist's cost changes within the local objective function. This is realized by penalizing the deviation from the currently confirmed supply quantities. Thus, only effective proposals—that is, supply patterns with the highest trade-off of decrease of internal cost and increase of approximated "external" cost—are computed. The estimation of penalty weights is updated on the basis of historical supply patterns and associated cost increases of the antagonist. Several measures are implemented to escape local optima. For the studied test instances, coordinated results are close to the central solution and are a significant improvement over the initial upstream-planning solution. Planning domains with cost increases are compensated by domains with cost decreases. The remaining savings are split fairly (see also Dudek, 2007).

Jung et al. (2005) present a coordination mechanism for aligning plans of a manufacturer and a third-party logistics provider (3PL). The 3PL has control over a network of distribution hubs, while the manufacturer has multiple production facilities. The 3PL tries to minimize transport, holding and lost-sales cost and requests items from the manufacturer. Due to limited capacity, the manufacturer is not able to fulfill all demand of the 3PL in time, so nonfulfillment is penalized in his objective function. The manufacturer sends back a feasible proposal that serves as input into the 3PL's model. The process is continued iteratively until the 3PLs request can be completed entirely. It can be argued that the manufacturer approximates the 3PL's model by penalizing nonfulfillment of supply requests. In contrast to Dudek and Stadtler (2005), neither are compensation payments transferred, nor are penalty weights updated.

Pibernik and Sucky (2007) investigate a hybrid form between upstream and central planning, where several firms unite to different planning domains. Within a planning domain, all included firms reveal their data and a central model is solved. The different planning domains are coordinated by an upstream planning strategy. Pure upstream planning with ununited firms serves as the initial solution. Cost differences between the coordinated and initial solution are compensated; the remaining savings are distributed in proportion to a company's share of the total cost.

The above coordination mechanisms combine the mathematical decomposition techniques with distributed decision making and mathematical programming. In contrast to contract theory and multi-echelon systems operational constraints are explicitly considered. However, the mechanisms share common weak points: First, the coordination mechanisms are specifically adapted to concrete models and solution techniques. For example, an MLCLSP is used and solutions are computed using an MILP Solver. In practice, the landscape of problems and optimizers is very heterogeneous. Not all practical problems are modeled as an MILP and not all solutions are computed by an MILP Solver. If planning domains are not willing to employ an MILP formulation, major effort would be required to adapt current solution heuristics to deal with Lagrangian multipliers or new soft constraints. In some cases, new constraints might not be added to the model, because the underlying heuristics do not support this step.[1]

Second, the approaches have only been tested for rather small scenarios, in which the local submodels can be solved to optimality. The effects of degraded solutions on the coordination mechanism are not entirely clear. Moreover, the coordination mechanism itself may be aborted prematurely. Mechanisms that do not preserve feasibility are especially likely to block an estimate of the quality of the feasibility-restored solution in such situations.

Third, the mechanisms do not provide incentives to the planning domains to act truthfully. In general, we cannot expect the planning domains to act as a real "team," even though there certainly is a basic intention to collaborate. A crucial point is that a planning domain's adherence to imposed multipliers or penalties cannot be controlled by the other planning domains. Each

[1] This is also true for the SAP DS Optimizer in the focus of our study. More details are provided in Chapter 7.

planning domain can simply boost their local objective value by lowering penalties or multipliers. The transfer of compensation payments is an additional aspect that has to be regarded critically, since reporting higher cost increases immediately leads to receiving a bigger piece of the savings pie. No measure is presented that prevents partners from cheating on their cost increases.

3.2.6 Secure multiparty computation

The above approaches dealt with the exchange of limited information in guiding an interorganizational search process to optimal solutions. The exchanged information provides each planning domain with a local approximation of the interorganizational problem. The domains use these local approximations to iteratively compute solutions that are better from an interorganizational perspective. Nevertheless, it is not guaranteed that exchanging such information does not reveal some sensitive data. In fact, opportunistically acting domains can infer sensitive data by systematically probing other domains. For example, demanding a single unit of an item one period later is unlikely to change computed lot sizes in the counterparty's MLCLSP and the holding costs for that item can easily be deduced from the difference of reported costs. Thus, even if the mechanisms do not directly require the exchange of sensitive data, they are *insecure* in protecting these data.

Security can be defined by means of a trusted mediator (cf. Li and Atallah, 2006). Assume an idealized mechanism, where each planning domain supplies sensitive information to the trusted mediator, the mediator solves the global problem and reports back the individual solutions, such that private data and variables are not disclosed to other parties. A mechanism is said to be **secure** if any adversary interacting in the real mechanism can do no more harm than in the ideal scenario. A central trusted mechanism would be able to compute the optimal solution immediately without providing other domains with probing opportunities. There exist circuit simulation methods to securely evaluate functions. That is, given a certain function to be common knowledge, an algorithm can be constructed that allows the computation of the result of that function without revealing supplied input variables.

This idea can be illustrated by a simple example. Assume three parties A, B, and C and each party holding a stock of items, e.g. A has 3 items, B has 5 items and C has 2 items in stock. Suppose that the parties aim at computing the sum of items without revealing how much items each single party has in stock. A secure sum algorithm can be set up as follows. Party A adds a random number, say 7, to its stock and transmits the sum of 10 to B. B adds its 5 items and transmits the sum of 15 to C. In turn, C adds 2 and transmits the sum of 17 to A. Eventually, A subtracts the initial random number and the correct sum of stock, i.e. 10 items, has been computed. However, no party knows the exact number of items of any other party.

Similar, though more complex, algorithms are available for other algebraic operations and different number of parties. These algorithms can be regarded as "one-way functions," functions that are easy to evaluate but hard to invert (cf. Yao, 1982). In the literature of secure computation, such algorithms are also called **protocols**. Protocols compute only on shares of the data. In simple terms, each protocol takes shares of the input and produces shares of the output (cf. Kerschbaum and Deitos, 2008). Each partner can encrypt his input with a private key, such that it cannot be decrypted by other parties without the private key, whereas the function can still be evaluated despite the encrypted values.

The main disadvantage of secure multiparty computation is that its implementation is very complicated and slow. Moreover, being a sophisticated encryption method, secure multiparty computation itself is not incentive compatible and does not prevent users from supplying systematically biased input. Users are typically assumed to be "honest but curious." Nevertheless, secure protocols can give a guarantee that supplied data is not disclosed. Several authors have investigated secure computation techniques applicable to SCM. Li and Atallah (2006) present a decentralized secure Simplex method for two parties. The global planning problem—represented as matrix containing objective function, objective value and constraints—is additively split between the two parties. Pivoting steps of the simplex algorithm are done in a decentralized fashion, ensuring that no party's partial matrix is revealed to the other party. Instead, only permuted and encrypted data are exchanged. Kerschbaum and Deitos (2008) extend the protocol to multiple parties and study the computational complexity. Atallah et al. (2003) present secure allocation protocols that prevent a disclosure of bidders' sensitive information during auctions.

Leaving aside the complicated computation and the threat of opportunistically acting parties, secure multiparty computation shares another drawback. Assume the idealized setting, where a trusted mediator (responsible for a fair distribution of savings) holds the complete model and SC partners are only providing input. Even under these assumptions we have to face a psychological problem: if results are below a domain's expectations there arises need for explanation. What *exactly* went wrong? How *exactly* do they have to calibrate the data in future? In light of such questions, secure computation is not a solution method easy to comprehend.

3.2.7 Metaheuristic-related techniques

Fink (2004, 2006) investigates a scenario where one supplier delivers to a manufacturer in a just-in-time sequence. The sequence of delivery influences the quality of the related production plans. A mediator repeatedly proposes randomized delivery sequences to the planning domains. The sequence-generating algorithm is assumed to be public knowledge. The planning domains can accept or reject the proposals. The last delivery sequence accepted by both domains is the final outcome of the coordination process. To sustain a fair outcome, both planning domains are required to accept a certain percentage of the proposals. From a myopic perspective, every planning domain would only accept proposals with better objective function value. However, to fulfill the acceptance requirements, the planning domains also need to accept worse proposals. The internal acceptance probabilities are calculated according to a cooling scheme taken from Simulated Annealing. Prior to coordination, the planning domains determine appropriate parameters by a trial run. Summarizing, the coordination mechanism proposed by Fink (2004, 2006) is a special version of the well-known Simulated Annealing metaheuristic, adapted to fulfill Collaborative Planning requirements. From a practical perspective, the mechanism comes with a central difficulty, however. Though proposing purely randomized delivery sequences by a third party can be considered as a very fair approach, it is highly inefficient for real-world production problems, where evaluating each proposal takes a considerable amount of time due to the size of the problems. For the above discussed approaches based on mathematical decompositions, such as Dudek and Stadtler (2005), purposeful proposals were computed by approximating the effects for associated partners. The mediator of Fink (2004, 2006) has no possibility for such a "problem analysis."

Dahl and Derigs (2008) present a mechanism for collaboratively solving a vehicle-routing problem. They assume a scenario where small-sized logistics providers ally in order to gain a competitive advantage. The basic idea is to buy and resell customer orders from and to allies to be able to compute round tours instead of dedicated tours.[2] To compute the tours, each partner employs a heuristic to solve its pickup and delivery problem. The heuristics are extended in such a way that available fleets and already computed tours of the allies are taken into account. Each heuristic may place an order on tours of partners for a compensation payment, determined by a predefined function considering the increase or decrease of the length of tours. The decentrally computed results are very close to the central solution. However, the scheme requires a massive exchange of data, including the location of customers, depots and fleet sizes. It is questionable if companies will agree to such an exchange in a real-world setting.

3.3 Scope of the thesis

Reviewing the above literature, it becomes clear that the design of coordination schemes generally comprises three fundamental difficulties. First, the coordination mechanism must support complex planning problems of operational type, including several variables and constraints. In this regard, it is also important that a (suboptimal) solution should be generated by the mechanism with computational acceptable effort. Second, the coordination mechanism must not lead to a disclosure of sensitive data. Third, the mechanism must be incentive compatible, making truth telling a dominant strategy, must be perceived as fair and must restrain domains from supplying biased input. The difficulty lies in equally considering all the three aspects: For example, multi-echelon systems and contract theory support incentive compatibility and information asymmetry but require very restrictive assumptions and simplified models. The presented coordination schemes relating to mathematical decomposition techniques and secure computation are able to deal with complex models but are not incentive compatible.

In general, side payments are a two-sided sword. For *choosing* the best interorganizational solution from a set of alternatives, side payments are necessary to ensure incentive compatibility and to restrain domains from supplying a biased valuation. In simple terms, linking the reporting of a saving or a cost claim to a subsequent side payment can generate the right incentives for domains to tell the truth. This works in a setting where the set of alternatives is fixed and known to all players. However, the situation is different if the reported values are also used for *constructing* new solutions. Assume a negotiation-based coordination mechanism that requires planning domains to *consecutively* report their costs (or savings) for each solution found by the search process as, for example, suggested by Dudek (2007). At the time a single negotiation takes place, the future outcome of the search process is not determined yet, i.e. the final set of alternatives is not known. Thus, planning domains are tempted to exaggerate their reported costs for receiving a higher side payment in order reap a good piece of the savings pie (as long they have the possibility for doing so). A possible solution to this problem is to decouple side payments from the coordination process, either by computing the payment a priori or a posteriori.

Computing the side payment a posteriori by establishing an auction (cf. Section 3.2.3) requires a set of alternative proposals at the end of the coordination to be known, cf. Kutanoglu and Wu (1999). In this regard, the coordination mechanism should provide *certainty that no party has an incentive to influence the composition of this set to its advantage*. There exist approaches that make it difficult for the parties to influence the search process. For example in Fink (2004, 2006), solutions are generated by an independent mediator as discussed in Section 3.2.7. However, we argued

[2] For an overview about present capabilities of electronic trading platforms for logistic services the interested reader is referred to Bierwirth et al. (2002).

that this search process can not be steered efficiently, as there is no possibility to approximate the effects for associated partners when solutions are generated. Instead, most approaches rely on exchanging cost reports, rankings, dual or primal information to enhance the efficiency of the search process. An exchange of such information might allow a planning domain to influence the composition of the set of alternatives by supplying biased input.

Assume a search process that fosters solutions having a small sum of reported costs. If a planning domain reports exaggerated costs, its own objective gets a larger influence. At the end, those solutions will get explored that have a low weighted sum—solutions that are, ceteris paribus, close to the locally optimal solution of the exaggerating domain. Obviously, if domains supply biased input, the true global optimum with maximum saving might not be found by the search process. This generates an incentive for the domains to tell the truth (by splitting a large pie fairly each player gets a larger piece than by splitting a small pie unfairly). However, if we assume the planning problems to be NP-hard, the solution with maximum global saving is not guaranteed to be found in a reasonable amount of time. In such a case, domains might be tempted to supply biased input. Assuming limited runtime, a search process can only explore a subset of the search space—and it is the composition of this subset that matters at the end.

Moreover, in most approaches, the value of each generated alternative is measured as the difference to the initial solution. However, planning domains are usually not of equal power. A powerful party might be tempted to create bad initial settings for the weaker parties if there is the possibility to gain future compensation payments from the weaker parties' "cost decrease" through coordination. For example, a (powerful) manufacturer could initially order items very early, such that huge overtime costs result in the supplier's domain and additional storage costs in the manufacturer's domain, whereas storage costs are assumed to be less than overtime costs. Now, the coordination mechanism will search for a solution that avoids these costs. In the example, it is likely that such a solution will be found (since the costs have been generated by purpose) and that the coordinated solution will come with a higher cost decrease in the supplier's domain than in the manufacturer's domain. Hence, if savings are split "fairly", the supplier will have to compensate the manufacturer. Thus, the manufacturer used his power to generate an additional margin at the expense of his supplier. Concluding, even if a mechanism was incentive compatible given a set of assumptions, its application in practice might suffer from a violation of some of these assumptions.

Computing a payment a priori is like bargaining over a good (cf. Section 3.2.4), where the good's value is not known yet. In our case, the good refers to the coordinated solution that is finally implemented. In order to be able to choose a side payment that is incentive compatible, at least the distribution of this solution's value must be assumed to be known. From our point of view it is a practical challenge to be aware of such a distribution. Bayesian learning for approximating the distribution could be employed by considering coordinated solutions generated in previous runs. However, this would require that enough samples are available and that these samples are not outdated yet. As the underlying production planning problem changes over time, it might not be appropriate to consider all samples. Still, the above mentioned difficulties remain. In general, cost reports might be regarded as critical as they allow domains to infer sensitive data.

The aim of this thesis is to combine and to extend existing approaches for practical environments that come with additional requirements such as limited runtime. While it seems hardly possible to fulfill the above three requirements to the fullest, the goal is to define a scope in which Collaborative Planning is *applicable* to real-world problems. From a practical point of view, short-term bottlenecks in delivery of intermediate goods urgently demand coordination mechanisms. Hence, the subject of this thesis is the coordination of detailed schedules across company boundaries. Computing detailed schedules requires production activities to be allocated to resources in a multi-item, multi-resource environment where capacity is limited. Because of the combinatorial complexity and size of the problems, detailed schedules are commonly computed by heuristics or metaheuristics. For this study, the SAP Detailed Scheduling (DS) Optimizer serves as underlying base optimization method. An analytical solution as for contract theory, auction theory or inventory systems problems is hardly possible in such a context; we have to rely on computational evaluations. Moreover, the DS Optimizer is specially adapted to the properties of the intradomain problems. Using techniques related to mathematical decomposition in order to solve the interorganizational problem as discussed in Section 3.2.5 is not seen as a suitable approach in this regard for the following reasons: First, metaheuristics generally compute suboptimal solutions which affects the behavior of mathematical decompositions. Second, it is not always possible to adapt the metaheuristics to include required constraints or penalty factors, since the adaptation of the problem would annul fundamental properties that are required by the heuristic to work efficiently. Third, the landscape of solution techniques used in practice is very heterogeneous and forbids the imposition of Collaborative Planning approaches relying on homogeneous models and solvers. However, a purely random generation of delivery sequences as proposed by Fink (2004, 2006) is also regarded as critical, since it provides no possibilities for learning, such as local problem analysis or approximation of the other planning domain's objective function. Though the mechanism will only be verified for a single solver, the SAP DS Optimizer, we will argue that other solvers and models exhibit similar properties and should be supported as well. A generic framework is presented in Chapter 4. Concluding, the scope of this thesis is to

- consider complex interorganizational production planning problems that require the application of Operations Research methods implemented in today's APS,
- develop a specific framework for Collaborative Planning that allows the incorporation of existing intradomain solution techniques without changing their core functionality,
- customize this framework to the SAP DS Optimizer,
- develop a coordination mechanism that does not disclose sensitive data, limits the effects and possibilities of opportunistic behavior, and is perceived as fair by the involved planning domains,
- investigate algorithms for calculating purposeful proposals by analyzing a planning domain's local problem and approximating its effects on other planning domains,
- explore possibilities to speed up the coordination by parallelization, and
- to compare the coordinated with uncoordinated results for practically relevant scenarios.

To create a practically acceptable coordination mechanism, side payments will not be exchanged. The above discussion already revealed that side payments are a two-sided sword, possibly providing a security lack or requiring additional restrictive assumptions. A drawback of not exchanging side payments is that the search space is pruned. Without side payments, interorganizational good solutions might not be contained in the subset of solutions that are acceptable by all domains. However, according to our computational evaluation presented in Chapter 8, this has only a limited effect; the coordination solutions are still of considerable quality. We believe this is due to the combinatorial complexity of scheduling problems. For those problem sizes found in practice, an optimal solution can usually not be generated in reasonable time, even if all data was available to a single solver.

CHAPTER 4

The generic DEAL framework

We propose a **D**ecentralized **E**volutionary **Al**gorithm (DEAL) as a framework allowing planning domains to jointly construct and evaluate solutions of an interorganizational production planning problem. No domain has explicit knowledge of the other domains' constraints, objective functions or solution methods. Using evolutionary principles a solution is sought that satisfies all domains. Being actually independent of underlying optimization models and solvers, the framework is applicable to a wide range of coordination problems. This chapter aims to give a generic outline of the concept. We start by summarizing requirements for a coordination mechanism from a practical point of view in Section 4.1. In Section 4.2 Evolutionary Algorithms are introduced. Section 4.3 gives a brief overview of the key ideas of DEAL. Sections 4.4–4.8 discuss several aspects of the sequential coordination in detail. The sequential coordination is characterized by a single process that sequentially involves the different planning domains. Inactive domains have to wait for the output of active domains which leads to an overall waste of available runtime. Extensions to an asynchronous and parallel coordination are provided in sections 4.9 and 4.10.

4.1 Assumptions and requirements: A practical point of view

In Section 3.2 we already discussed theoretical assumptions and requirements for coordination mechanisms. The practical application of Operations Research optimization techniques adds an additional perspective to theoretical considerations. In production planning, optimization has to cope with large amounts of data and short runtime requirements. Moreover, the data is not always *clean*; it might include some discrepancies stemming from the estimation of parameters or the collection of historical data and forecasts. In general, the objective is not necessarily to find the global optimum, but a satisfying solution in a reasonable amount of time. Also, with regard to a coordination mechanism, real-world requirements prohibit a direct implementation of many theoretical concepts. Extensive interviews with companies and the SAP Solution Management identified several key practical requirements. These requirements had a major impact in the development of the DEAL framework and are presented below.

4.1.1 Nondisclosure of sensitive data

Planning domains are supposed to behave opportunistically and not to be willing to share all available information to reach a higher coordination level. Even though higher savings could be obtained, information is not exchanged if it can be misused by the counterparty, such as for demanding future price decreases. Not surprisingly, most optimization-related costs and constraints are considered sensitive data. For example, production costs directly allow the inference of the product margin. The same is true for holding costs. Considering that one major component of the inventory holding costs is the cost of capital lockup, counterparties may infer data related to the cost and internal value of the product. Also, information about capacity utilization can be misused to negotiate lower prices. Moreover, information about total production and shipping quantity reveals the competitiveness and the strategic positioning of a company. The above list is by no means comprehensive, as the available data depends on the controlling systems and optimization models in use. The fact that a multidomain coordination involves not only a supplier and a buyer, but several (potentially competing) suppliers and buyers, also needs to be taken into account. The danger of internal data being revealed to a competitor adds to the aversion to data exchange. Most firms will not agree to achieving an additional saving at the cost of a strategic disadvantage (visibility of internal cost rates and constraints). In general, we can say that a coordination mechanism that exchanges less information will receive better practical approval.

4.1.2 Availability-based instead of cost-based incentive mechanisms

An SC-wide coordinated solution can entail an unfair sharing of savings between the domains. Some domains might even be worse off than in the initial, uncoordinated setting. Side payments are commonly transferred in state-of-the-art research to generate incentives for domains to join a coordination process, as already mentioned in a sections 3.2 and 3.3. Some general remarks on side payments have already been given there. The interviews revealed that side payments are hardly acceptable in practice. From a practical perspective, there are several additional arguments against side payments:

- Although technically possible, there is usually no direct link between a company's APS and accounting system (cf. Stadtler, 2005b, p. 585). On the one hand, optimization costs often have no equivalent in real-world accountancy costs. For example, the task of defining the penalty for late delivery from an accountancy perspective is very difficult since it requires quantifying the monetary value of a probably damaged customer relationship. On the other hand, accountancy costs are not always represented adequately by the mathematical model. Especially on an operational level, machine-, workforce- and material-related costs are usually considered as sunk costs. The models aim primarily at seeking schedules that make optimal use of the available resource. Most often, fictitious coefficients for the diverse objectives are implemented. Based on historical data, the planner tries to adjust the coefficients until they reflect promising trade-offs of the different objectives. Possibilities for mapping theses coefficients to accountancy costs are regarded as limited for most practical cases.

- Most practical problems are not solved to optimality. Due to their size, exact methods are either stopped prematurely or heuristics are applied. Hence, the resulting costs cannot be considered as deterministic outcome; they must—to a certain degree—be regarded as a random variable with unknown, runtime-dependent probability function.

- Side payments may violate nondisclosure requirements. Domains might not be willing to negotiate a payment, since it gives—at least to some extent—insight into internal cost structures and realized profit. Opportunistically acting domains might abuse the coordination mechanism to gradually probe the cost-structures of other planning domains.

- Already today, the prenegotiation of a frame contract is a very complex and time consuming process. The complexity of the process of setting up a frame contract would increase dramatically if hypothetical side payments needed to be considered too.

- The generation of value might be asymmetrically distributed across the SC. For example, upstream domains might generate only a small share of the ultimate product value and downstream domains add the major part of value. Thus, incurred savings of upstream domains are too little to compensate cost increases of downstream domains. Under such a setting, negotiating on costs would lead to results similar to plain upstream planning. A better approach would be if the downstream domain proposes several, from his perspective equal, alternatives and the upstream domain simply chooses the most suitable one, without any side payments.

At the current stage side payments are viewed very critically from a practical perspective. It might be possible to include side payments at a later stage, when the Collaborative Planning paradigm has experienced broader acceptance. However, such payments should not be the central pillar of the coordination mechanism. As an incentive mechanism in step with actual practice, we briefly outline an alternative concept:

- Each domain is allowed to influence the construction of interorganizational solutions in such a way that its local planning results are acceptable for it.

- Additional savings are not distributed, remaining within the planning domains in which they occur.

- Instead of focusing on cost, we regard availability as the main objective of the coordination mechanism. That is, the parties focus on decreasing idle times on machines, excess stock and delays in the delivery to the ultimate customer rather than on optimizing monetary values. Employed costs are considered as control costs for steering the solvers but not to be relevant for accountancy. This shifts the responsibility from the controlling to the logistics departments. A typical use case is the "restoration" of good production plans after machine breakdowns.

4.1.3 Feasibility preservation and multidomain support

Intermediate proposals should be feasible (in a mathematical sense), such that human planners can constantly monitor the solution quality. If the process has reached a satisfying level, it can be aborted and the best found solution can be put into practice instantly. Theoretical methods based on Lagrangian relaxation that penalize infeasibilities between local domain plans without providing a feasibility-restoring mechanism are regarded critically since they make no estimation on the quality of solutions after feasibility has been restored by a repair mechanism. Today's Supply Chains are complex networks of stakeholders. The coordination mechanism should allow several domains to work together. A common scenario is that an Original Equipment Manufacturer (OEM) is dependent on several suppliers, cf. Chapter 7.

4.1.4 Support of different local optimization engines

Real-world optimization usually requires considerable fine-tuning of parameters and data belonging to a company's specific optimization problem. It is likely that companies would not accept a coordination mechanism that required a major change of the parameters or even abandoned their long-time tested and approved planning system. The risk of exchanging a *working*, fine-tuned, intradomain method producing satisfying results with a coordination mechanism of unknown solution quality is simply too great for many companies. Moreover, companies often have specific planning problems that cannot be tackled by out-of-the-box solutions. In practice, the landscape of optimization tools is hence very heterogeneous and several software companies compete with different versions of different optimization methods with proprietary intrafirm developments. Finding a real-world setting with homogeneous optimization tools is unlikely. Hence, the coordination mechanism should rather treat the underlying solver as a black box. As will be discussed later, most models for production planning share commonalities in their structure. Our intention is to construct a coordination mechanism that dynamically changes the input data relating to common model structures but not the solver itself. Different solvers can then be supported by adapting the interface for changing the data.

4.1.5 Scalability and robustness

As already mentioned, the intrafirm problems are mostly solved by heuristics or exact methods that are aborted at an early stage, such that the solution quality is degraded—i.e., has not reached yet its optimal value. Due to limited runtime, an actually better coordinated solution could be identified as worse and vice versa. The scheme should prove robust to such degraded solutions and support a *graceful degradation*. Let us assume that there is certain probability for generating a (suboptimal) solution in limited time, i.e. the quality of the solution adheres to a certain probability function. Then it makes sense to generate as many "samples" as possible to minimize the effect of low-quality outliers and to generate a good solution. Today, computational grids as a hardware and software infrastructure provide dependable and inexpensive access to high-end computational capabilities. A coordination mechanism providing possibilities for parallelization is regarded as an economic possibility for robustly constructing good solutions in acceptable time.

4.2 Introduction to Evolutionary Algorithms

This work employs principles of nature-inspired optimization techniques, in particular Evolutionary Algorithms (EAs). To ease the understanding of the proposed coordination mechanism, this sections gives a brief introduction to the main principles of evolutionary optimization. These principles have been adopted from Darwin's theory of natural evolution (cf. Darwin, 1859) to computationally solve combinatorial problems. According to Fogel (2000), natural evolution itself can be regarded as an *optimization process*:

> Darwin . . . was struck with organs of extreme perfection that have been evolved, one such example being the image-forming eye . . . . Optimization does not imply perfection, yet evolution can discover highly precise functional solutions to particular problems posed by an organism's environment, and even though the mechanisms that are evolved are often overly elaborate from an engineering perspective, function is

> the sole quality that is exposed to natural selection, and functionality is what is optimized by iterative selection and mutation. It is quite natural, therefore, to seek to describe evolution in terms of an algorithm that can be used to solve difficult engineering optimization problems. The classic techniques of gradient descent, deterministic hill climbing, and purely random search (with no heredity) have been generally unsatisfactory when applied to nonlinear optimization problems, especially those with stochastic, temporal or chaotic components. But these are the problems that nature has seemingly solved so well.

Subsection 4.2.1 gives some basic definitions relating to EAs. In Subsection 4.2.2, analogies to natural evolution are briefly discussed. Subsection 4.2.3 presents the standard template for EAs. Finally, issues relating to multiobjective problems and Goal Programming are discussed in subsections 4.2.4 and 4.2.5.

4.2.1 Basic definitions

This subsection summarizes basic definitions that can usually be found in various books regarding Evolutionary Algorithms (cf. Fogel, 2000; Deb, 2001; Eiben and Smith, 2003; Spears, 2007). For evolutionary optimization methods, there exist two spaces for representing an optimization problem's (suboptimal) solution: the search space and the solution space.

Definition 4.2.1. (solution space) An **instance of an optimization problem** is a pair $(\mathcal{G}, g)$, where the **solution space** $\mathcal{G}$ is the set of feasible solutions and the **objective function** g is a mapping $g : \mathcal{G} \to \mathbb{R}$.

Definition 4.2.2. (search space) The **search space**, $\mathcal{S}$, is the set that is searched by an optimization algorithm. Search space and solution space are connected by a **decoding function** $d : \mathcal{S} \to \mathcal{W} \supseteq \mathcal{G}$. The elements of $\mathcal{S}$ that are mapped to a solution in $\mathcal{G}$ are called **feasible**, all other elements are called **infeasible**. Let $f : \mathcal{S} \to \mathbb{R}$ be the **objective function** defined on $\mathcal{S}$.

An element $x \in \mathcal{S}$ is often called a (feasible or infeasible) solution, although strictly speaking, only the feasible elements are the representations of solutions. Treating the solution and search spaces separately is a fundamental difference from "classical" optimization techniques, such as linear programming. The idea is to reduce the combinatorial problem at hand to its *core*. The solution itself is often decoded by heuristics according to the choice of *core values*. The natural analogy are the genes, which encode different properties of an individual. Having defined search and solution space, we can say what an optimal solution is, distinguishing between global and local optimality.

Definition 4.2.3. (global optimality) A **globally optimal solution** (global optimum) is a solution $x^* \in \mathcal{S}$ such that $f(x^*) \leq f(x)$ (respectively $f(x^*) \geq f(x)$ for maximization problems) for all $x \in \mathcal{S}$.

Definition 4.2.4. (neighborhood) Let $(\mathcal{S}, f)$ be an instance of an optimization problem. A **neighborhood function** is a mapping $\mathcal{N} : \mathcal{S} \to \wp(\mathcal{S})$, which defines for each solution $x \in \mathcal{S}$ a set $\mathcal{N}(x) \subseteq \mathcal{S}$ that are in some sense close to x ($\wp(\mathcal{S})$ denotes the power set of $\mathcal{S}$). The set $\mathcal{N}(x)$ is the **neighborhood** of solution x.

Definition 4.2.5. (local optimality) A solution $x \in \mathcal{S}$ is **locally optimal** with respect to the neighborhood (x) if $f(x) \leq f(y)$ (respectively $f(x) \geq f(y)$ for maximization problems) for all $y \in N(x)$.

4.2.2 Inspiration from nature

EAs are inspired by Darwin's principle of natural evolution. His theory is based on three fundamental observations:

1. Individuals create more offspring than needed for the survival of their race. Nevertheless, the population size stays more or less constant. Limited resources ultimately prohibit an infinite growth of the population. Hence, individuals are struggling for survival. Most individuals die before producing offspring.

2. Two individuals of the same race are never identical in all aspects, differing in minor ways.

3. Skills and properties needed for survival are passed from one generation to the next.

Darwin deduced that only individuals highly adapted to the environment procreate and pass inherent properties to their offspring (survival of the fittest). Today we know that natural evolution is based on three principles:

- **Mutation** is responsible for introducing new gene values.

- **Sexual reproduction** or **crossover** combines existing genes into new ones.

- **Selection:** The fittest, most highly adapted individuals have a larger probability of surviving and a larger probability of producing offspring.

From a gene perspective, individuals can also be interpreted as *vehicles* (cf. Dawkins, 1976). Not only are the individuals struggling for survival. Essentially, the genes themselves are competing against each other by equipping their vehicles with survival strategies and behavioral rules. EAs try to carry principles of natural evolution over to mathematical optimization problems. Therefore, notions of biology are often adopted:

- Solutions are considered as **individuals**.

- The objective value (cost) of a solution relates to an individual's **fitness**.

- An individual's representation in the search space $\mathcal{S}$ is called its **genotype**.

- A solution in the solution space $\mathcal{G}$ is denoted as its **phenotype**.

- EAs work on sets of individuals called **populations**. An iteration of an EA is a **generation**.

- New solutions are denoted as **children** or **offspring** and are formed out of **parents** from the current generation. The offspring are created by using **crossover** and **mutation** operators on the parent generation's genotypes. Crossover combines the genotypes in a reasonable way, whereas mutation performs a move in the neighborhood. As the best offspring replace the worst parents in the population, only the fittest individuals survive.

- Often, a phenotype is back-encoded to its genotype after additionally measures, such as a local optimization, have been applied (to the phenotype). Such functionality is also denoted as **Lamarckism** (in contrast to Darwin, Lamarck believed that also skills acquired during the lifetime of an individual are inherited by the offspring).

As decoding functions not only mathematical functions in the strict sense but various heuristics are often employed. Hence, EAs belong to the class of metaheuristics, allowing to *calibrate* the parameters of an underlying base heuristics to the problem at hand. Tuning the general evolutionary process to a specific problem usually requires an adequate representation, taking problem-specific knowledge into account when designing the mutation or crossover operators. This customization to a specific problem is actually the hard part when setting up an EA.

Besides EAs, other metaheuristics exists, such as Variable Neighborhood Search, Simulated Annealing, Tabu Search and Ant Systems. An overview about the different approaches is provided by Voß (2001). Voß et al. (1999) give a general definition for metaheuristics:

> A metaheuristic is an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high-quality solutions. It may manipulate a complete (or incomplete) single solution or a collection of solutions at each iteration. The subordinate heuristics may be high (or low) level procedures, or a simple local search, or just a construction method. The family of meta-heuristics includes, but is not limited to, adaptive memory procedures, tabu search, ant systems, greedy randomized adaptive search, variable neighborhood search, evolutionary methods, genetic algorithms, scatter search, neural networks, simulated annealing, and their hybrids.

In literature, there often is no clear distinction between EAs and other metaheuristics, cf. Hertz and Kobler (2000). Traditionally, EAs comprise Evolutionary Programming, Evolutionary Strategies and Genetic Algorithms, cf. Spears (2007). Evolutionary Programming, developed by Fogel et al. (1966), commonly uses representations that are tailored to the problem domain, such as real-valued vectors or ordered lists. It arose from the desire to generate machine intelligence and focuses only on mutation operators. Evolutionary Strategies were developed by Rechenberg (1973) and Schwefel (1981), use a real-valued representation and consider mutation as primary operator. On the contrary, Genetic Algorithms, developed by Holland (1975) use a binary representation and have a main focus on crossover. Further details on classification and characteristics of Evolutionary Algorithms can also be found in Braun (1997).

4.2.3 A template for Evolutionary Algorithms

Various templates exists in literature that define the basic working principles of an EA (cf. Eiben and Smith, 2003; Deb, 2001; Spears, 2007; Voß, 2001). These templates differ in the amount of detail and wording, but not in functionality. Algorithm 1 defines our standard template of an EA, close to the one of Deb (2001). In the beginning, a population $P(0)$ of individuals is initialized. Initialization can be at random or follow a start heuristic. In the next step, the population is evaluated and updated: The individuals are compared by their fitness values; fitter individuals will survive to the next generation, while others are sorted out during the update operation such that the population size stays equal. The counter t refers to the current iteration (or generation in EA terms). It should be highlighted that in our template the task of evaluating a population is not concerned with computing an individual's fitness—this is already done during initialization or the construction step described later. Rather, we refer to the task of comparing individuals, e.g. by computing a ranking according to fitness values, as evaluation. Other templates do not distinguish between evaluation and update. However, for this work distinguishing between the two tasks is necessary for describing the coordination process in the following.

Algorithm 1: Standard Template of an Evolutionary Algorithm.

```
$t \leftarrow 0$
initialize population: $P(0)$
initialize offspring: $M'(0) \leftarrow \emptyset$
repeat
    evaluate $P(t) \cup M'(t)$
    update: $P(t+1) \leftarrow u(P(t) \cup M'(t))$
    $t \leftarrow t+1$
    if termination condition then
        final selection
    else
        select mating pool: $M(t) \leftarrow s(P(t))$
        construct offspring: $M'(t) \leftarrow c(M(t))$
    end
until termination condition ;
```

Next, the termination condition is checked. Typically, the algorithm is terminated if a satisfactory fitness level or a time limit has been reached. If terminated, the best or several good solutions are presented to the decision maker. Otherwise, the set of procreating parents, the so-called mating pool $M(t)$, is formed by the selection operator. For doing this, probabilities reflecting the individuals' fitness are computed and define the chance of an individual being selected as a parent. Selection is then usually carried out in a probabilistic manner to allow the population to overcome local optima. Thus, in line with Darwin's principle, better individuals are selected more often for subsequent reproduction. Finally, the new offspring are constructed by combining (crossover) and altering (mutation) the genes from the mating pool's individuals. Changing the genes can be at random or done more purposefully by considering *heuristic knowledge*, such as by implementing rules of thumb that analyze the current solution. During construction, an individual's fitness is also determined. Figure 4.1 illustrates this template graphically and will serve as a cornerstone for further discussion.

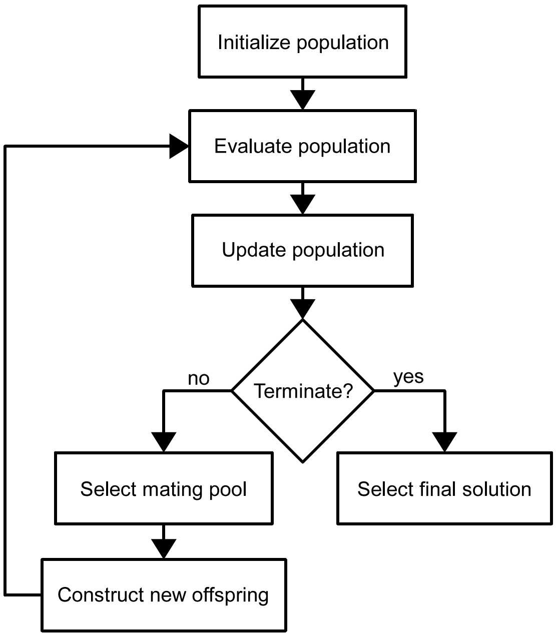


Figure 4.1: Illustration of standard template of an Evolutionary Algorithm.

The operators of an EA usually follow stochastic principles and are steered by parameters. Often, the EAs need to be tuned to a specific problem by a tedious adjustment of these parameters. Under some circumstances, it proves useful to let these parameters themselves be subject to evolution. With a genotype extended by strategy parameters, an EA can **self-adapt** to a specific problem. Note, however, that with an increasing number of strategy parameters, self-adaptation becomes slower and slower.

EAs differ from other metaheuristics, such as tabu search or simulated annealing, by maintaining a population of individuals. For many problems, a population-based approach offers greater flexibility in overcoming local optima. In this regard the so-called *selection pressure*, the intensity with which bad individuals are eliminated is of importance. The selection pressure is defined by the way individuals are compared and selection probabilities are computed, the actual selection mechanism, the way how the population is updated and the population size. A high selection pressure leads to a fast convergence but comes with a higher risk to get stuck early in a local optimum.

4.2.4 Multiobjective Evolutionary Algorithms

Most real-world problems have several conflicting objectives. However, most "classical" optimization methods require single-objective problems. Traditionally, there exist four approaches to deal with conflicting objectives, cf. Domschke and Scholl (2008, pp.56).

- A lexicographic order of objectives is imposed. Then, optimal solutions are generated regarding only the most important objective. Within the remaining set of alternatives, it is searched for optimal solutions regarding only the second important objective. The process is continued to the last important objective or until only one solution remains.
- Extra objectives are converted to constraints—i.e., their value gets bounded from above or below—with the risk that these bounds make the problem infeasible.
- An artificial single objective is constructed as the weighted sum of the multiple objectives.
- A goal is defined a priori and it is tried to minimize the distance to this goal. This technique is denoted as **Goal Programming**, see also Section 4.2.5.

All approaches require user knowledge of the optimal trade-off between the multiple objectives *a priori* to optimization. However, a decision maker is not necessarily aware of this trade-off.

In this regard, Multiobjective Evolutionary Algorithms (MOEAs) propose an entirely different approach. Instead of combining different objectives using a fixed trade-off, they simultaneously develop an efficient set of solutions regarding the multiple objectives. Ideally, this set converges to a discrete approximation of the problem's Pareto-optimal front. If terminated, these metaheuristics return not one, but a set of non-dominated solution candidates. The user can then choose his preferred solution *a posteriori* to optimization.

Figure 4.2 illustrates the relationship between search, solution and objective space for multiobjective problems. In the example, the two-dimensional objective space is spanned by the functions $f_1 : \mathcal{S} \rightarrow \mathbb{R}$ and $f_2 : \mathcal{S} \rightarrow \mathbb{R}$. Fetching ahead a customization to scheduling problems (see Section 5), the solution is illustrated as a Gantt chart. Search, solution and objective space are connected by decoding and fitness functions. A crucial question is how to compare individuals when no

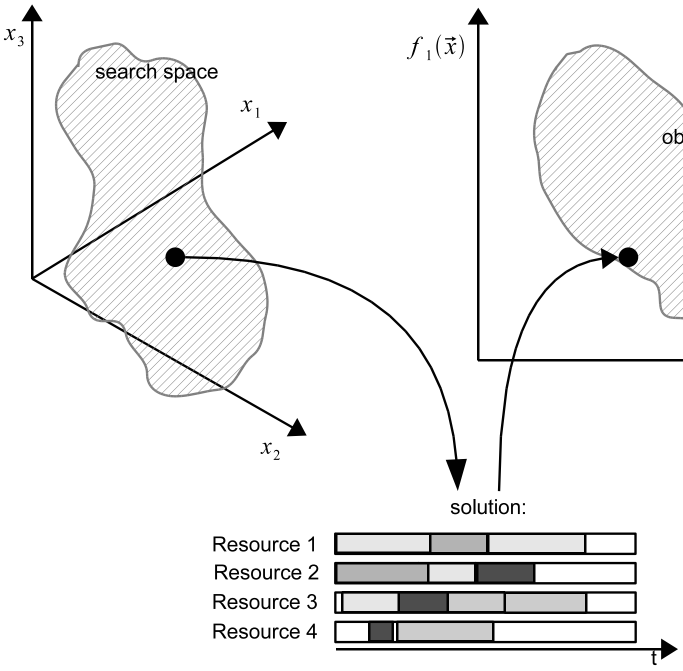
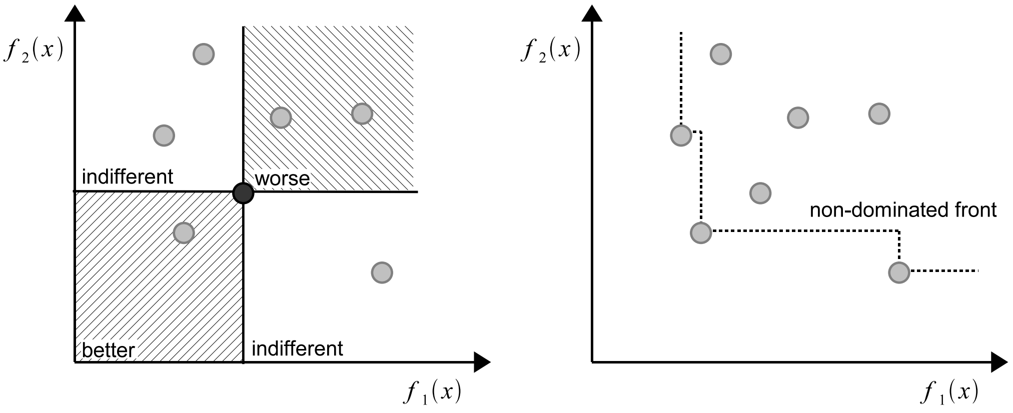


Figure 4.2: Illustration of search, solution and objective space for a scheduling problem.



(i) Dominated and nondominated solutions.

(ii) Nondominated front.

Figure 4.3: Dominated solutions and nondominated front.

trade-off is specified by the user. Most MOEAs use a special sorting technique to evaluate individuals. However, they still fit into the general EA template (Figure 4.1). A multiobjective sorting can be achieved on the basis of the concept of domination. A solution $x^{(k)}$ is said to **dominate** a solution $x^{(l)}$ if $x^{(k)}$ is not worse in any objectives than $x^{(l)}$ but better in at least one objective. Put mathematically, assuming a minimization problem with m objectives,

$$x^{(k)} \succ x^{(l)} \Leftrightarrow f_i(x^{(l)}) \leq f_i(x^{(k)}) \; \forall i \in \{1, \ldots, m\} \text{ and } \exists j \in \{1, \ldots, m\} : f_j(x^{(l)}) < f_j(x^{(k)}),$$

where $\succ$ means “is better than”.

Figure 4.3 (i) illustrates this idea with an example. Within the objective space (both objectives are assumed to be minimized), we can compute several areas for each solution. Taking the individual marked by the black circle as an example, we see that dominated solutions lie in the upper-right quadrant and are worse in at least one objective. Solutions that are better in both objectives dominate the current solution and are situated in the lower-left quadrant. Without any predefined trade-off, we are indifferent to solutions that a better in one but worse in the other objective.

Using the concept of dominance, one can define what an **Pareto-optimal** solution is: a solution not dominated by any other solution in the search space. The entire set of optimal trade-offs is called **Pareto-optimal set**, **Pareto-optimal front** or sometimes **efficient set**. Many MOEAs approximate this set by clustering the population into nondominated fronts. As shown by Figure 4.3 (ii), the nondominated front consists of the solutions that are not dominated by any other solution of the population. The nondominated front can be computed by Algorithm 2 (cf. Deb et al., 2002; Deb, 2001).

Algorithm 2: Nondominated sorting.

```
input  : n individuals $x^{(1)}, \dots, x^{(n)}$
output: nondominated set $P$
$P \leftarrow \{x^{(1)}\}$
for $k = 2$ to $n$ do
    $l \leftarrow 0$
    repeat
        $l \leftarrow l + 1$
        if $x^{(k)} \succ x^{(l)}$ then
            $P \leftarrow P \setminus \{x^{(l)}\}$
        end
    until $x^{(l)} \succ x^{(k)}$ or $l = |P|$ ;
    if $l = |P|$ then
        $P \leftarrow P \cup \left\{x^{(k)}\right\}$
    end
end
```

The second nondominated front can be calculated by removing the elements of the first front and running the algorithm again. If we repeat this procedure until every individual is assigned to a nondominated front, we have roughly sorted the whole population. The lower the front an individual is assigned to, the closer is the distance to the (unknown) Pareto-optimal set. In order to get a complete sorting, we can evaluate the individuals within one front by a second criteria. Usually, some diversity measure is introduced here. The idea is to prefer solutions that have more distant neighbors to solutions that are close to each other. If a diverse population is fostered, the final nondominated front is assumed to cover a broader region of the objective space. For more details, the interested reader is referred to Deb et al. (2002); Deb (2001).

4.2.5 Multiobjective Goal Programming

Instead of minimizing or maximizing objective functions directly, Goal Programming attempts to find one or more solutions that satisfy a number of goals (one for each objective) to the greatest extent possible. If the goals cannot be satisfied, they should be violated as little as possible. In classical optimization, the distance (measured by a metric) to the goals is minimized, cf. Domschke and Scholl (2008, p.58). That is, for m objectives $t_1, \ldots, t_m$, the function

$$\sqrt{\sum_{i=1}^{m} (f_i(x) - t_i)^2}$$

is minimized when using the Euclidian distance. Again, it is the task of the user to specify the distance norm and the goals *a priori*. MOEAs provide an alternative by considering the goal of each objective during the sorting procedure. For example, nondominated sorting can be extended by a conversion operator, $\langle . \rangle$, that returns the value of the operand if positive and zero otherwise. Without loss of generality, assume objective $f_i(x) \leq t_i$ as a goal to be included. Now, nondominated sorting is performed with respect to the converted objective $\langle t_i - f_i(x) \rangle$. In other words, the nondominated sorting only respects yet unsatisfied dimensions. Usually, MOEAs come with the "curse of dimensionality:" the more dimensions are included, the higher the probability that a solution is non-dominated in one dimension. Restricting on only necessary dimensions can give a great speed increase. More details can be found in Deb (1999).

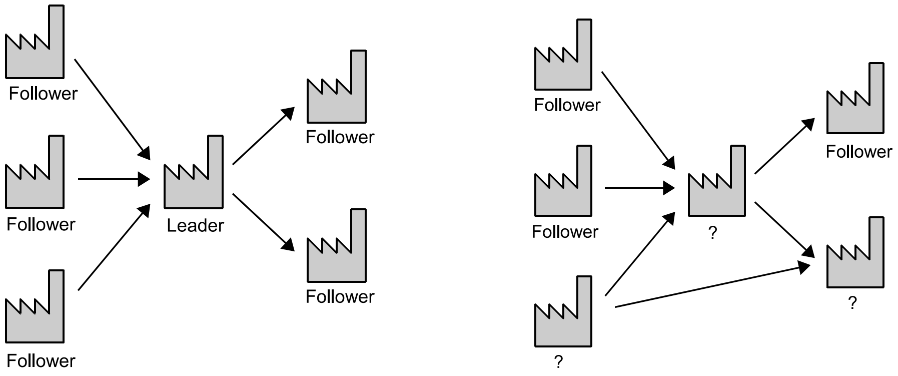
We admit the above introduction to be very brief. However, it explains the main working principles of Evolutionary Algorithms. These principles are used extensively by DEAL, which is introduced in the following sections.

4.3 A brief introduction to DEAL

The basic idea of DEAL is to use existing intradomain planning tools for the coordination of production plans. As discussed previously, the coordination mechanism must not reveal the domains' optimization models, objective functions and private data (capacities, durations, holding costs and so forth), and should generate production plans that serve the interests of all planning domains.

We propose a combination of hierarchical coordination (cf. Section 3.2.1) and Evolutionary Algorithms. The hierarchical part of the mechanism is characterized by different roles and responsibilities assigned to the planning domains. One of the planning domains is the **leader** of the coordination process, the others are called **followers** (cf. Schneeweis, 2003, Section 1.1). For generating an interorganizational solution, the leader sends **proposals** to every follower. These proposals contain instructions relating to the dates or quantities of items to be delivered. Followers implement these instructions by considering them in their optimization model (with certain limitations to be discussed later). The followers are not aware of each other, but the leader is the single point of coordination. It is obvious that such a setting requires the Supply Chain to have at most three tiers with the leader being the only *point of coordination* on the middle tier connecting all other entities, see Figures 4.4 (i) and 4.4 (ii).

For generating and implementing instructions, present intradomain models and solution methods shall be used to the extend possible. Reusing existing functionality allows to keep the cost low when putting the coordination mechanism into practice. Today's Advanced Planning Systems already offer possibilities for each planning domain to take adjacent planning domains into account by employing soft or hard constraints in the optimization model. For example, customer preferences are going into the calculus by setting soft-constrained due dates for orders or material quantities. These due dates *anticipate* the planning problem of the adjacent downstream domains, cf. Section 3.2.1. Setting a due date earlier signals an increased importance of the related order. Cost



(i) Single point of coordination. (ii) Multiple points of coordination

Figure 4.4: Supply Chain with single and multiple points of coordination.

for violating the due dates anticipate the increase of the downstream domains' objective values. Typical examples for hard constraints are release dates or a fixed inflow of required material every period. They define the earliest possible date production activities can start. Hard constraints do not allow to anticipate the models of other domains but directly implement their decisions. DEAL aims at dynamically changing the data of such (hard and soft) constraints by successively sending proposals from the leader to the followers. The parties use their intradomain planning tools to generate new instructions or to implement the instructions by changing their models' data.

Using present intradomain optimization models and solvers leads to several peculiarities, since these models and solvers have not been designed to support an automated coordination. Although the optimization models contain soft constraints to anticipate other domains and hard constraints for implementing instructions, these constraints might not be *distributed* properly among the domains.

On the one hand, a follower might be able to (either) implement the leader's proposal by using soft or hard constraints. Each follower needs to *translate* the leader's instructions. Translation is about mapping the requested delivery dates or quantities to intradomain production activities. Depending on the follower's optimization model, this mapping affects soft or hard constraints. If mapped to soft constraints, a follower is not bound to the leader's instructions and might only partly respect them. Later, we will discuss means how followers report the violation of soft constraints by sending counterproposals. If the leader's instructions are mapped to hard constraints, no feasible solution might exist in the follower's optimization problem (e.g., if the leader supplies items too late). In both cases, an instruction may deteriorate other objective values even if the model remains feasible. A follower might thus not accept the outcome.

On the other hand, the leader might not be able to anticipate a follower. Regarding the interface to the follower, only hard constraints might exist in the leader's model. In such a case, DEAL needs to supplement this functionality. How proposals, counterproposals and interorganizational solutions are actually generated depends on the used solvers and models. Section 4.4 gives an outline on a scheme that reduces infeasibilities in most practical settings.

An interorganizational solution can exhibit a poor quality because of several reasons. It might include infeasibilities, might not be acceptable for all followers or might come with too high values in the objective function of the leader (assuming a minimization problem). All cases result of suboptimally set instructions. Not surprisingly, it is a hard task for the leader to find instructions reflecting the global problem, if only the internal model is known and internal data are visible.

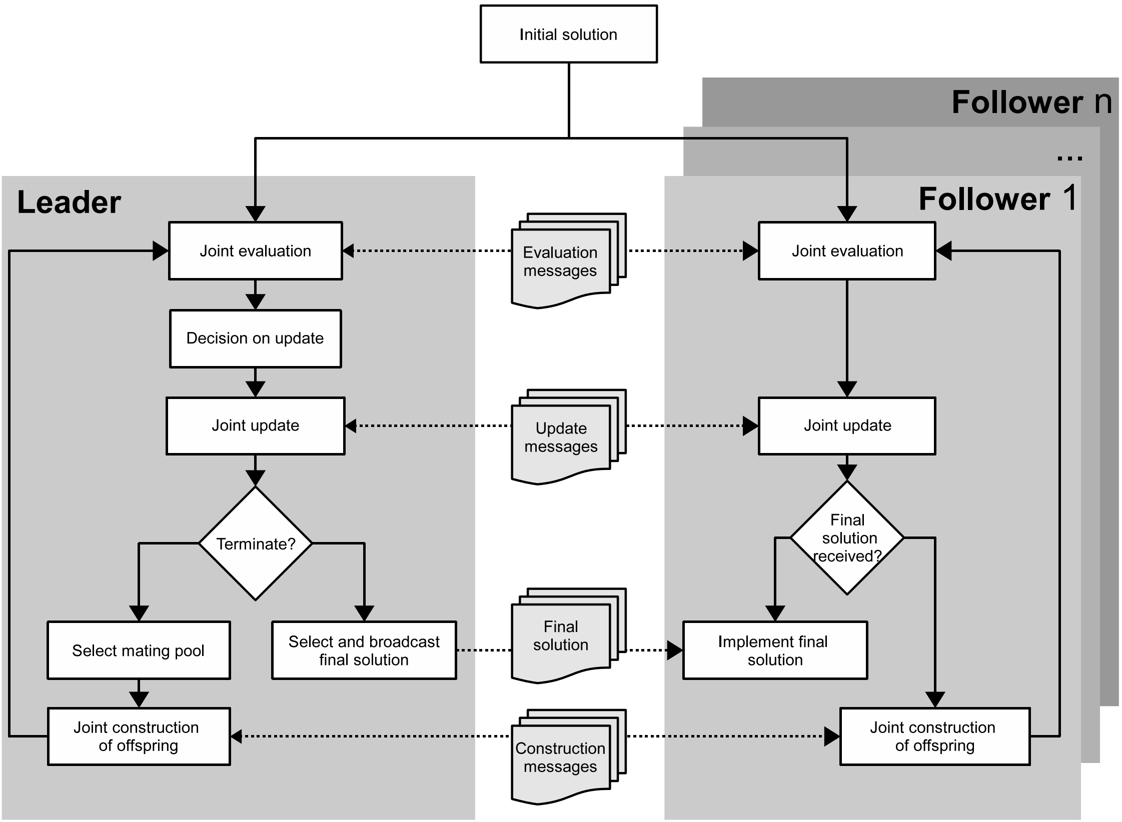


Figure 4.5: Overview of the DEAL framework for the case with one leader and multiple followers.

The task of generating good instructions is left to an Evolutionary Algorithm that is implemented in a decentralized fashion. It has the following main characteristics:

- Each planning domain holds its private subpopulation of individuals.
- The *genotype* of each individual are the instructions sent or received.
- The *phenotype* is the resulting production plan, consisting of private and public data.
- Each solver is part of a global decoding function, mapping instructions to production plans. An interorganizational solution is the combination of individuals of the parties' private subpopulations. However, sensitive data remains confidential as the individuals themselves are not exchanged.
- The followers transmit rankings to the leader that reflect acceptability or the violation of feasibility. The leader combines these rankings to a global fitness value for each interorganizational solution.
- For generating purposeful proposals (containing instructions), the leader analyses previously generated solutions. New proposals can be derived by changing or combining previous proposals, similar to mutation and crossover in Evolutionary Algorithms.

The basic search process for a single-leader setting is depicted in Figure 4.5. Compared with the standard EA template (Figure 4.1), it can be seen that parts of the evolutionary process have been distributed between the domains. Leader and followers jointly perform the evaluation, update and construction by exchanging messages of different types. Constructing an individual involves the tasks of generating proposals and implementing them as outlined above. More details on the construction, evaluation and update of individuals will be presented in sections 4.4 - 4.6. The decision on termination, update and selection is made by the leader. It is worth mentioning that the followers also have an implicit decision right on the final solution. When terminating, the leader may only choose among feasible solutions accepted by *all* followers. If there is no such solution, the SC remains with the initial solution. However, until termination, no follower is aware if any globally accepted solutions exist. Our hope is that this mechanism restrains SC domains from acting too opportunistically. Naturally, a rational leader tries to find a setting that is acceptable to all followers but optimizes his own objective function to the extent possible. In general, the obtained solution will differ from the optimal central solution, the solution a central decision unit—with complete visibility—could compute. However, if side payments are excluded, the *followers define acceptability—leader optimizes* scheme seems to be a reasonable alternative.

4.3.1 Message protocol

Automated communication requires the electronic exchange of messages between the domains. Therefore, the format for the messages and the way of exchanging them must be defined: a **message protocol**. Most researchers do not invest much effort in message protocols (this is true for all the publications listed in Section 5.9.) Data are simply exchanged between the domains. For most academic studies this proceeding is sufficient, since the same type of data is iteratively exchanged several times. Nevertheless, we believe that a few considerations on a message protocol can serve as a guideline to support a future practical implementation. Moreover, since this work proposes a population-based approach capable of parallel computation, the need for a message protocol is more evident than it might be for sequential coordination mechanisms. The principle idea is to regard each planning domain as a state machine. States are defined by the individuals in the current population that have been constructed, evaluated, updated and so forth. Planning domains repeatedly process their local population of individuals. However, in order to get a congruent behavior, the *global state* of the SC needs to be mirrored by every local private population. Following the message protocol, the planning domains synchronize their local populations by a steady message exchange. We regard a message as a structured piece of information with the following properties

- It has one of several *a priori* defined **commands** that are known by all planning domains.
- It has a unique **id**.
- It can be the **reply** to a previous message.
- It has exactly one **sender** and one **receiver**.
- It consists of several **components**, containing the data to transmit.

Commands can be, for example, `send_ranking`, `proposal` or `final_solution`. A message component is a certain type of information relating to a distinct command, for example the *due dates* or a *ranking*. The protocol defines the *sequence of commands* and *which command requires which components*. A more detailed overview of the generic aspects of our proposed message protocol is presented in Section 4.7.

4.3.2 Communication threads

We define a **communication thread** as a message exchange between exactly two planning domains regarding a certain purpose, for example the exchange of a ranking. It will become apparent that constructing individuals or updating the population usually requires more than one message to synchronize the involved domains. The length of a thread is given by the sequence of commands defined by the message protocol. That is, a thread starts with an initial request possibly followed by several consecutive replies. Within a thread, the same two domains constantly reply to each other; no two succeeding messages of the same thread may be issued by the same domain. We assume that upon transmission each message is translated by the receiving domain. Translation means that some components are changed to fit the receiving domain's specific planning system. For example, order ids are mapped from the one APS to the other. Translation has the effect that each domain views the same information from its *perspective*. When referring to a message, each domain always refers to its local copy. A thread is restricted to cover the communication between the leader and exactly one follower. Followers might be competing enterprises, are not supposed to know each other and are not supposed to communicate directly. As connecting entity, the leader needs to build up separate threads with every follower to communicate the information he desires. Moreover, preliminary experiments suggest that a message flow involving more than two domains is hardly traceable. The more entities participate in such a chain of communication, the larger the risk of errors introduced by the weakest entity. A thread is identified by its latest message. If, according to the protocol, no further replies are expected to arrive, we denote a thread as **complete**, otherwise **incomplete**.

4.4 Construction of solutions

After having given a brief overview of the concept, we now step further into the details. This section explains the process of jointly constructing an interorganizational solution from a generic perspective. We first discuss the relationship between communication threads and individuals in Subsection 4.4.1 before giving a generic template for constructing a solution in Subsection 4.4.2.

4.4.1 Individuals connected by threads

An Evolutionary Algorithm involves a steady reevaluation of selection probabilities according to the fitness of individuals. In order to be able to compare the fitness values, each domain needs to hold its local version of the population in memory. These local populations are steadily synchronized by repeated exchange of messages for constructing, deleting or selecting individuals. In essence, an individual defines an intradomain decision problem instance and a related solution. Obviously, not all data defining a problem instance, only relevant constraint settings and the part of solutions that actually changes, need to be stored in an individual. Instructions are exchanged by messages contained in communication threads (cf. Section 4.3.2). The leader has the task of initializing and coordinating the threads between himself and his followers. This way, he can relate his individuals to individuals of his followers without explicitly knowing their content. In general, constructing a new individual affects all followers. However, it is not always necessary that all followers need to be informed. For some followers, a new individual might not mean any change with respect to *their interface*. In such a case, the new individual of the leader is already represented by an individual in the follower's population. Thus, the leader does not need to resend an proposal already sent previously. It is more efficient to connect the individual to an existing communication thread. Vice versa, the follower does not need to recalculate a reply if he can use a previous computation as a substitute. Thus, linking a new communication thread to an existing individual saves precious runtime. For multidomain problems, the state of completion of an individual might depend on the replies of several followers. It will become apparent that we need to record which followers already replied to a request, which are about to reply soon and which are awaiting new input. Especially for the asynchronous and parallel ccordination (see sections 4.9 and 4.10) the need for such a record will become evident. In our implementation, each of the leader's individuals connects to exactly one thread for each follower. In other words, the leader's individuals themselves serve as compound entities between the distinct communication threads. Vice versa, each thread includes the information to which individuals it is connected. If a new message of a follower arrives, the leader updates all individuals affected by it. By doing this, it is ensured that an individual is always connected to the latest message of the related threads. The construction of an individual is done in distinct stages. Representing the completeness of an individual, we can define the following **individual states**.

- **Waiting:** Information from another domain is needed before the next construction stage can be entered.
- **Idle:** New messages need to be issued by the current planning domain.
- **Complete:** All threads are complete and all calculations have been done. Regarding this particular individual, no further messages are expected.
- **Deletable:** The individual was sorted out of the population. Due to issues related to the synchronization of domains, deletable individuals can sometimes not be deleted instantly, but are only marked for deletion. (See also Section 4.10 for further details.)

An example of changing states is illustrated in Figure 4.6, where a leader coordinates two followers (only the interaction with Follower 1 is depicted in detail). In the example, each communication thread consists of two messages: A proposal (containing instructions) issued by the leader and the follower's counterproposal (reporting the violation of instructions). It is supposed that the leader needs this information to finish the construction of his individual. From top to bottom, the negotiation process is illustrated at different points in time. It should be highlighted that the figure does not depict a flow chart but illustrates the states of the system at different points in time.

At $t = 1$, the leader has constructed a new individual, e.g. a tentative production plan that is promising from the leader's perspective but might imply delivery dates that are not feasible from the followers' perspectives. Thus, the leader must verify the feasibility with each of the followers. As it is his turn to start the communication, the individual's state is currently set to idle.

At $t = 2$, the leader sends a proposal to Follower 1. The proposal contains instructions, such as due dates, that affect the calculation of production plans of Follower 1. The data are translated; for example, the leader's order ids are replaced by order ids of the follower's system. Thus, the domains physically hold different messages. Nevertheless, the messages describe the same request and are identified by the same id. Having received the proposal, Follower 1 constructs a new individual, connects it to the proposal and sets its state to idle. Also the leader connects his

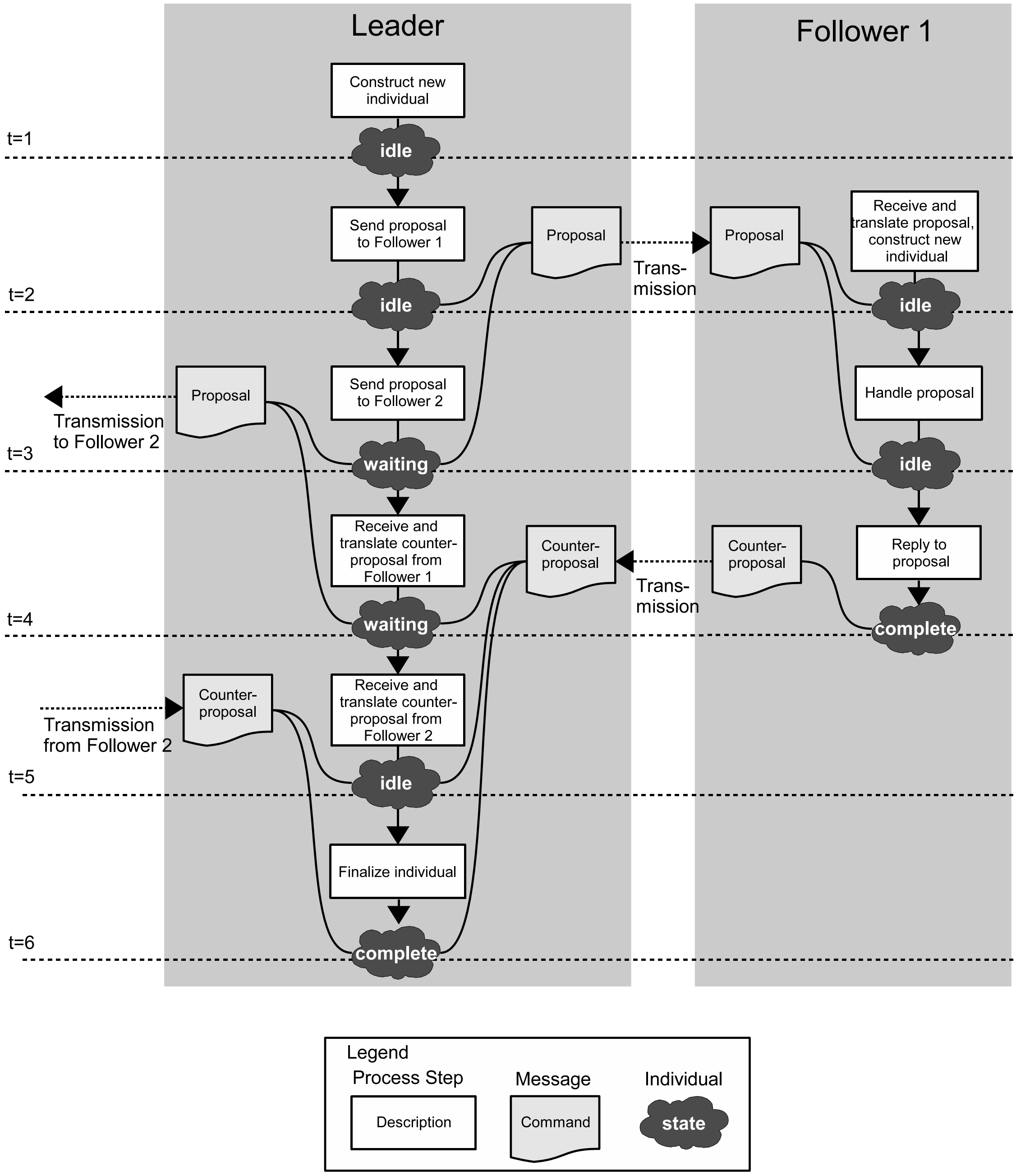


Figure 4.6: Example of an individual connecting to different communication threads.

individual to his local version of the proposals (the leader's and the follower's individuals are said to be connected by the communication thread).

At $t = 3$, the leader sends a similar proposal to Follower 2. As Follower 1 did not reply yet, the individual is still connected to the proposal issued previously (indicated by a second link to the proposal). As no further calculations can be done before the followers have replied, the state of the leader's individual is changed to waiting. Meanwhile, Follower 1 handles the request, e.g. by computing a new production plan. Having done that, he transmits a counterproposal to the leader at $t = 4$. Upon transmission, the data of the reply is again translated. From Follower 1's perspective, the computation is finished and the state of the follower's individual is set to complete. The thread between the leader and Follower 1 is complete, since no subsequent messages are assumed to pertain to this thread in this example. The leader's individual is still classified as waiting since the reply of Follower 2 is missing.

After this message has arrived in $t = 5$, it is the leader's turn to continue the calculation, combining the two counterproposals and recomputing his own production plan. Eventually, the joint calculation of one individual is finished, and its state is set to complete at $t = 6$.

This example illustrates a disparity with the classic understanding of Evolutionary Algorithms: Normally, an individual represents a point in the search, solution and objective spaces. However, within the DEAL framework, an individual also has its own state of completeness. At each iteration, every domain transforms idle individuals into waiting ones or vice versa. Moreover, instead of one central population, several subpopulations are managed by the different domains. Only some combinations of individuals of the different subpopulations yield a feasible interorganizational solution. In the following, we will introduce generic processes transforming waiting to idle or complete individuals. For better legibility, we will omit the illustration of individuals passed between the processes and the translation of messages. Moreover, only the interaction with one follower will be depicted, representing all other followers.

4.4.2 Generic construction template

Constructing an interorganizational solution involves the generation of leader proposals and the computation of followers reactions. Both steps are highly dependent on used optimization models and solvers. However, with regard to present APS it is possible to outline a generic construction template that works with most optimization models and solvers available today. Present models usually have two commonalities. First, due dates (or related target quantities) are usually considered as soft constraints. That is, in order to achieve feasible plans, committed dates of delivery to customers can be violated for additional penalty costs. Especially on a short-term planning level, violating due dates might be the only available vent for achieving feasible plans. Models on a mid-term planning level usually also employ possibilities of capacity increase for additional overtime or storage costs. Second, material availability is usually hard-constrained. That is, on a mid-term planning level, quantities of material inflow (the number of items to be ordered) have an upper bound. On a short term-planning level, release dates define the earliest date a production activity might start.

The above described model structures lead to interdependencies of planning domains that seem paradox at first glance. From a model perspective, upstream domains anticipate downstream domains by employing soft-constrained due dates (or related target quantities). Downstream domains consider upstream decisions (e.g., the release dates) as hard constraints. Hence, in order to achieve interorganizational *feasible* production plans, downstream planning is the preferred choice from a model perspective. However, in practice, upstream planning can be observed most often, i.e. the downstream planning domain is the leading entity, generating instructions for upstream domains. Very often, several suppliers are delivering to a single manufacturer. In such a setting, the manufacturer usually has the best overview about future demand and generates the biggest share of the product value. Hence, the manufacturer most often claims the right for being the leader and for generating proposals. However, his optimization model does usually not anticipate the supplier's production planning model. This fact is historically motivated by a planning paradigm that favors monolithic customer-oriented models but does not foresee the possibility of negotiations on the supply side. If we additionally consider wholesalers to be part of the coordination process, we have a three tier structure as discussed in Section 4.3, where the manufacturer represents the leading middle tier delivering to the wholesalers that are responsible for satisfying the demand of the ultimate customer. Wholesalers are assumed to have a planning problem regarding the distribution of items (otherwise there would be no need to dynamically include their domains in the coordination process). Alternatively, we could assume another tier of manufacturers (having a production planning problem) instead of the wholesalers.

Today, global feasibility is usually preserved by implementing high-level negotiated frame contracts, limiting the amount of items that can be ordered per period or providing a lower bound for the earliest date an item can be ordered (e.g., orders must be communicated four weeks before delivery). However, regarding the global optimization problem, this form of upstream planning usually leads to suboptimal results, as a frame contract does not substitute the global model. Moreover, in case of sudden events, such as machine breakdowns, upstream planning has only limited possibilities to resolve the bottleneck. Since a manufacturer does not anticipate a supplier's planning problem, the manufacturer generates instructions that might not be manageable by the supplier.

As already briefly discussed in Section 4.3, DEAL aims at finding instructions that allow all planning domains to compute improved production plans. However, as calling the solution methods requires a significant amount of runtime, a random mutation of instructions is not suitable. As discussed above, upstream domains presently anticipate downstream domains by employing penalty costs for delayed delivery, but the opposite is usually not true. If we want to reuse present models and solution methods most efficiently, this has two implications. First, demand must be propagated in upstream direction. That is, wholesalers should influence the manufacturer's due dates (or related target quantities) and the manufacturer should influence the suppliers' due dates (or related target quantities). Second, the planning domains should call their solution methods in a downstream oriented sequence to construct a feasible solution. According to the manufacturers proposals, the supplier compute new production plans. Because of their soft-constrained models, there is no guarantee that all instructions are obeyed. Hence, suppliers might send a counterproposal (reporting the deviation from the instructions) and the manufacturer has to replan his production. Eventually, the manufacturer proposes new delivery dates (or related target quantities) to wholesalers. Summarizing, we recommend the following construction sequence for the three tier scenario.

1. The manufacturer takes an existing solution from the mating pool and gathers information on how to improve it from wholesale followers. The wholesalers indicate which changes in the supply pattern would better match their demand and the manufacturer translates the information to new anticipating data, e.g. due dates. We will denote instructions send from wholesalers to manufacturer as **guidance** henceforth.

2. Using the guidance and analyzing his own problem, the manufacturer devises proposals for his suppliers, who can then map these proposals to their anticipating data.

3. The supplier use the information to calculate a new solution for their intradomain problems. They reply with counterproposals representing their decision (the deviation from the instructions) to the manufacturer.

4. Using these new input values, the manufacturer computes his intradomain solution. Based on his results, he in turn sends proposals to the wholesalers.

5. The wholesalers use the latter proposals as input for their own optimization model, reply with a confirmation and the interorganizational solution is constructed.

A proposal might come with changes affecting only a subset of followers. In such cases, newly calculated instructions need not be communicated to all followers. In Chapter 7, we will discuss the different occurrences of redundant proposals. However, irrespective of the exact definition of redundancy, we advocate a double-sided check and elimination of redundant proposals. After having analyzed his local problem, the leader checks for each follower if the new proposal equals existing proposals. Instead of sending a new redundant proposal, the individual is connected to the existing communication thread. Similarly, each follower can check an incoming proposal for redundancy. If a follower identifies a proposal as redundant, he immediately resends the preexisting counterproposal or confirmation. This way, computing capacity can be saved. The double check is necessary since leader and followers can have different models and optimization methods that induce different definitions of redundancy. Figure 4.7 gives an overview about the whole construction process, while figures 4.8 - 4.10 refer to the subprocesses of exchanging guidance, proposals with suppliers and proposals with wholesalers.

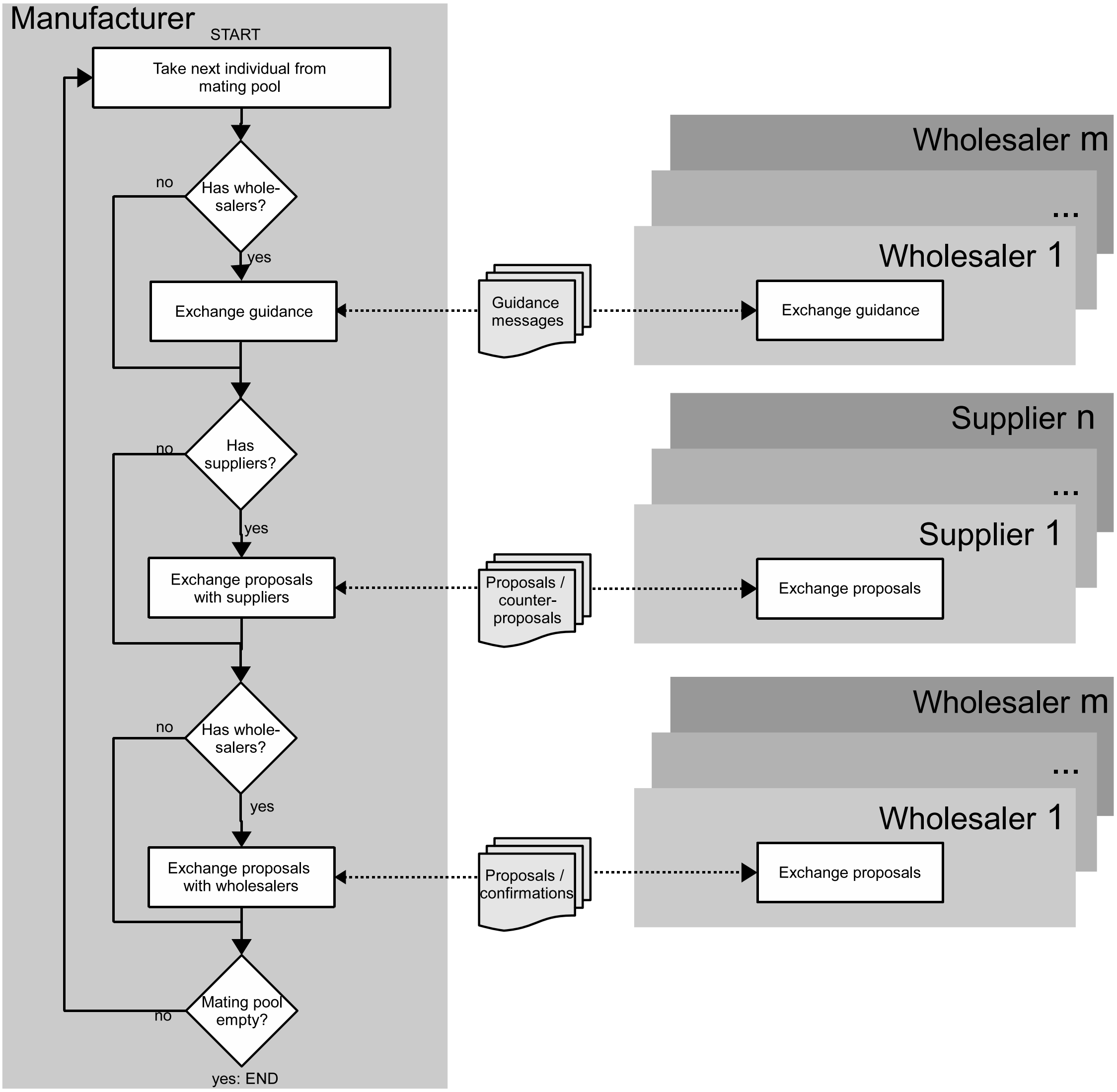


Figure 4.7: Overview of the generic construction of an individual.

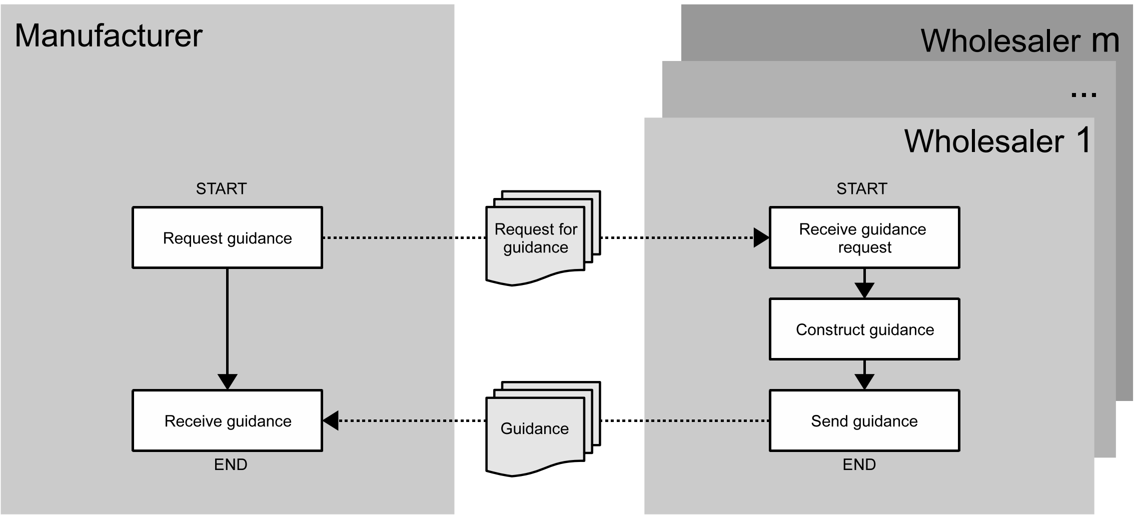


Figure 4.8: First part of the generic construction of an individual.

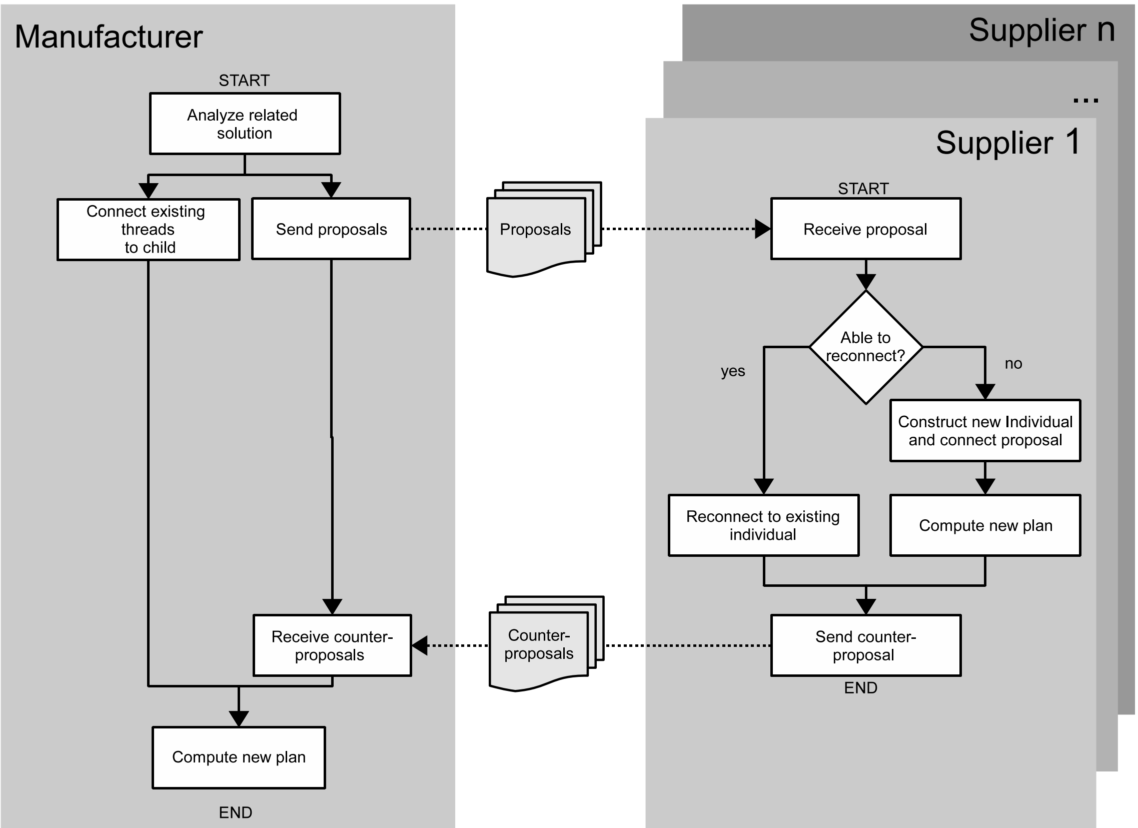


Figure 4.9: Second part of the generic construction of an individual.

Concluding, it should be highlighted that the construction of individuals is a joint process. Although the manufacturer coordinates the process, the suppliers have implicit decision rights as the manufacturer's instructions are mapped to soft constraints in the suppliers' models. Thus, suppliers can deviate from instructions that are *too costly* and reply with a counterproposal. Moreover, all followers are allowed to transmit their preferences to the manufacturer for steering the search process (see next section). The above outlines the generic construction process. How proposals and counterproposals are actually generated depends on the planning problem and used

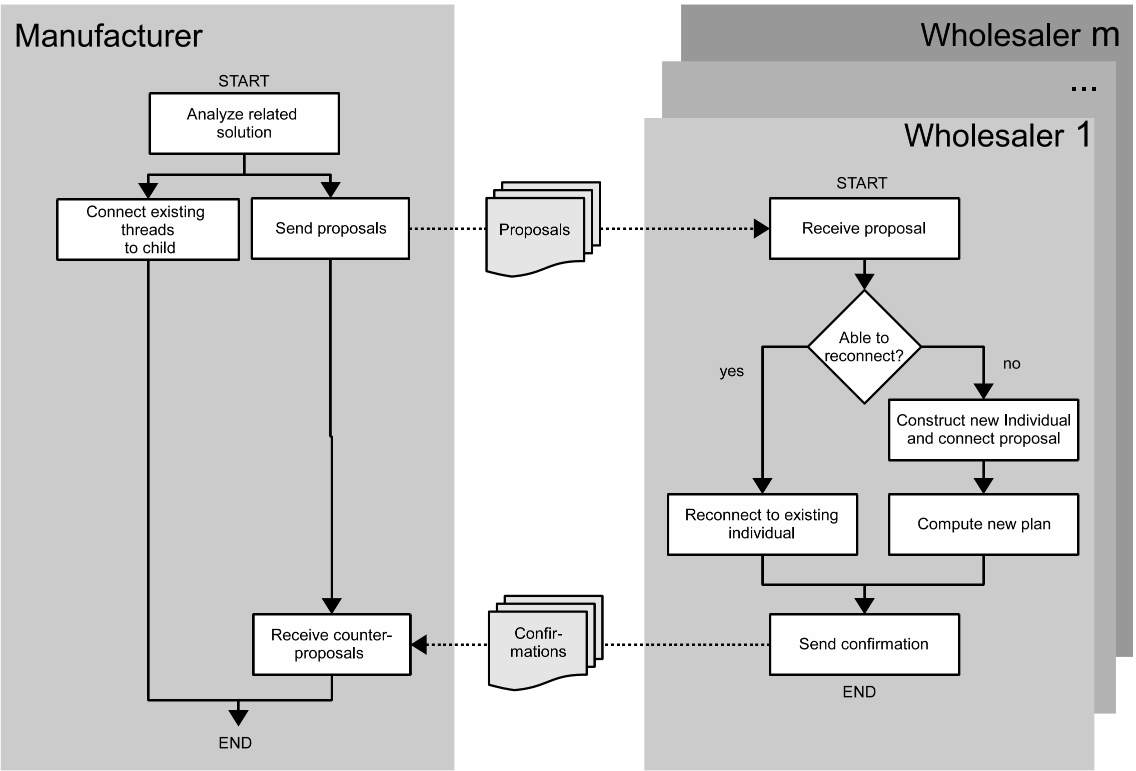


Figure 4.10: Third part of the generic construction of an individual.

solution methods. A detailed discussion regarding scheduling problems solved by metaheuristics, such as the SAP DS Optimizer, will be presented in Chapter 7.

4.5 Evaluation of solutions

In previous sections, we already highlighted the goal of DEAL: to find an interorganizational solution that is acceptable to all followers while optimizing the leader's problem to the extent possible. To inform the leader about their preferences, followers iteratively transmit information on how to rank the population's individuals. The leader combines this information to select the mating pool and update the population. Figure 4.11 shows the generic process for exchanging the ranking information. It should be emphasized that only complete individuals can be ranked. In the current implementation, the leader triggers the evaluation process, since he is aware which of his individuals are complete. A further difficulty arises from the threads serving as a middle layer between the leader's and follower's private populations. A planning domain does not know the other domain's population, instead the ranking must be mapped to existing threads. As can be seen in the illustration, the leader first collects all threads connected to individuals that should be ranked. Now, the leader sends to every follower a list with ids pertaining to communication threads (identified by the latest message) between the two domains. Each follower maps the thread ids to individuals and calculates the related ranking. He then maps the ranking back to thread ids and replies to the leader. Having received replies from all followers, the leader again remaps the rankings to his individuals. On the one hand, the transmitted information should give the leader an unbiased view of the followers' situations. On the other hand, no critical information should be transmitted. For example, forcing the followers to report their (sensitive) objective

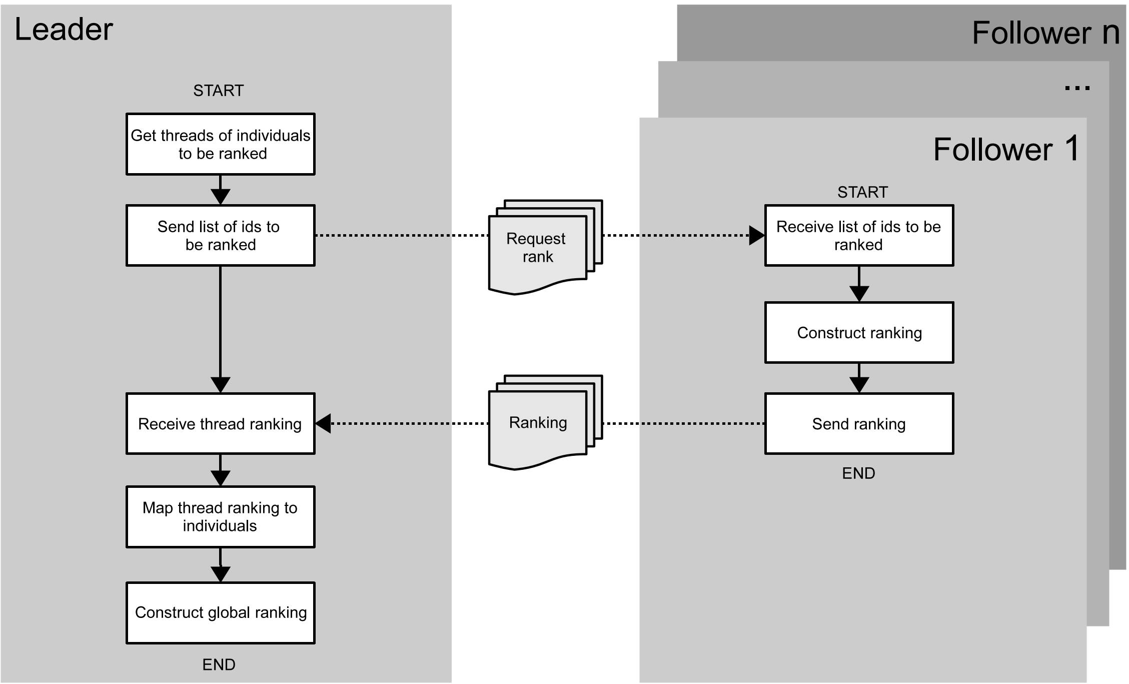


Figure 4.11: Evaluation of the complete population

values to the leader would not be a practically suitable approach. Instead, we recommend the exchange of ordinal rankings. In the following, we will give three examples of concrete ranking schemes that disguise actual objective values.

4.5.1 Restricting to acceptable solutions

A simple case arises if planning domains can configure their intradomain optimization methods to guarantee the generation of acceptable plans. Dependent on their planning models and solvers, followers might be always able to deviate from the leader's instructions (by sending counterproposals) such that the solution is always acceptable from their perspective. Still, a follower can transmit a ranking to guide the search process. For example, a compact production plan might be regarded as preferable since it provides a better starting point for future coordination even though a follower might also accept noncompact production plans. Let the index e denote a planning domain, and E denote the set of all planning domains, where $e = 0$ refers to the leader's planning domain. Each follower $e > 0$ transmits an ordinal rank $\omega_e(p)$ of solution p to the leader with $\omega_e(p) \in \mathbb{N}$. For a population with size n, we assume the best individual to be ranked with 1, the worst with $m \leq n$ and two individuals of equal fitness to share the same rank. Since all followers accept all solutions, we consider the leader's rank $\omega_0(p)$ to be more important. The leader only considers the follower suggestions if he is indifferent between solutions. Put mathematically, the leader applies the following lexicographic comparison between two solutions p and q to sort his local population.

1. **Sort according to the leader's preference:**

$$p \succ q \Leftrightarrow \omega_0(p) < \omega_0(q) \tag{4.1}$$

2. **If indifferent ($\omega_0(\mathbf{p}) = \omega_0(\mathbf{q})$), sort according to the follower's preference:**

$$p \succ q \Leftrightarrow (\omega_0(p) = \omega_0(q)) \wedge \\ (\forall e \in E \setminus \{0\} : \omega_e(p) \leq \omega_e(q)) \wedge (\exists d \in E \setminus \{0\} : \omega_e(p) < \omega_e(q)) \quad (4.2)$$

Relation 4.2 describes the nondominated sorting scheme of Multiobjective Evolutionary Algorithms (cf. Section 4.2.4). Since trade-offs between the follower's rankings are unknown, nondominated sorting is a safe way to combine the different subrankings. If followers are of different importance, the leader might also use a further lexicographic comparison instead of Relation 4.2.

4.5.2 Double ranking

The above ranking scheme can only be applied in special situations where the planning domains can control their intradomain optimization methods to always produce acceptable solutions. In general, the cost of a production plan influenced by the leader's proposal might not be acceptable for a follower. For example, the leader might request items so early that a follower's optimization run delays orders of other customers.

The follower's cost associated with a solution of the interorganizational problem can be decomposed into **coordination-related** and **domain-related** costs. Coordination-related costs are penalty costs resulting from the violation of anticipated target values, such as due dates within the objective function. These costs are mainly for steering the search process. For example, the sum of lateness with respect to the due dates imposed by the leader are not costs the follower can claim in an accountancy sense since they only approximate the leader's costs. Nevertheless, coordination-related costs are in direct relation to domain-related costs. For example, setting a due date for a shipment to the leader earlier might result in a solution with higher setup cost or lateness of other orders of external clients not participating in the coordination process. Concluding, an evaluation of a solution should only be based on domain-related costs (all cost not related to anticipating data), since only these costs are actually relevant. However, for a follower, not all domain-related costs are of equal importance. For example, setup times might be considered within the objective function as a control cost, but what ultimately counts from a short-term perspective is lateness of production activities relating to external customer orders. Hence, in addition to the objective function of the local optimization method, a follower might have another *evaluation function* representing his preferences regarding the leader's proposals within the DEAL framework. A simple, practical evaluation function can be outlined as follows. The function distinguishes between acceptable and unacceptable solutions. Unacceptable solutions result from a setting of instructions that leave the follower's local optimization method too little freedom to produce acceptable results. In order to be able to construct more suitable proposals in the future, the leader needs to be informed about the *degree of acceptability*. We define

$$v_e(p) = \begin{cases} 0, & \text{if the proposal } p \text{ is acceptable,} \\ z \in \mathbb{N}, & \text{if the proposal } p \text{ is not acceptable.} \end{cases}$$

Integer z denotes a ranking of unacceptable solutions. The smaller z, the better the related solution from the follower's perspective. In an ordinal second-level ranking, $\omega_e(p)$, the follower transmits his "optional" preferences on the basis of further domain-related costs not directly linked to acceptability. For example, two proposals q and p having a $\upsilon_e(p) = \upsilon_e(q) = 0$ because of low external customer lateness can be further distinguished according to incurred setup time. Low setup times should be preferred, since they offer better starting points for future proposals. Similarly, the leader has his own evaluation functions, $\upsilon_0(p)$ and $\omega_0(p)$. A global ranking can be constructed using multiobjective sorting approaches within a lexicographic comparison. To sort the leader's population, we apply the following lexicographic comparison between two individuals p and q.

1. **Sort according to nondominated solutions regarding the followers' and leader's acceptance:**
$$p \succ q \Leftrightarrow (\forall e \in E : \upsilon_e(p) \leq \upsilon_e(q)) \wedge (\exists d \in E : \upsilon_e(p) < \upsilon_e(q)) \,. \tag{4.3}$$

2. **If indifferent, sort according to the leader's second-level preference using Equation 4.1.**

3. **If still indifferent, sort according to the follower's second-level preference using Equation 4.2.**

Sorting according to Relation 4.3 has three advantages. First, the transmission of ordinal rankings disguises objective values considered as sensitive by the followers. The leader has only limited possibilities for probing the followers to infer sensitive data. Although he knows that a solution is regarded by a follower as better or worse than another solution, he does not know *how much* better or worse it is. Second, the dimensionality is reduced. The search process is not driven in dimensions were solutions are already acceptable by the domains (having a ranking of zero) but a multiobjective Goal Programming approach, as discussed in Subsection 4.2.5, is implemented instead. Third, and most important, followers have only limited possibilities for cheating. No followers can influence the trade-off between the local objectives by exaggerating his values, as such a trade-off does not exist in non-dominated sorting. Of course, followers can label actually acceptable solutions as unacceptable—but that is something they already can do today by reporting that present delivery dates cannot be maintained. In general, we assume that followers try to comply to delivery dates to the extent possible, as their status of being a reliable partner depends on it. Thus, we are tempted to label DEAL as a *practically incentive compatible* mechanism (however, not in a mathematical sense).

Again, if followers are of different importance or are even considered to be more important than the leader, comparisons 4.3, 4.1, and 4.2 can be rearranged to a more nuanced lexicographic comparison, e.g. prioritizing the rankings of more important followers over less important ones.

4.5.3 Side payments

Theoretically, the exchange of side payments can also be supported by the two-level ranking mechanism. Assume followers and leader to be able to define a mapping of internal costs to side payments, $\upsilon_e(p)$, where

$$\upsilon_e(p) = \begin{cases} \infty, & \text{if the proposal } p \text{ is not acceptable to domain } e, \\ \geq 0, & \text{if the domain wants to be compensated,} \\ < 0, & \text{if the domain intends to share a part of his savings with other domains.} \end{cases}$$

For the initial situation, υ_e is defined to be zero for all domains, for all other solutions it represents the actual value the domain intends to share or wants to receive. In addition to υ_e, the followers transmit a second-level ordinal ranking ω_e, reflecting their additional preferences for solutions having the same side payment (for example, if setup costs do not influence the side payment). Within the above scheme, the leader can compare two individuals p and q as follows.

1. **Prefer high savings:**
$$p \succ q \Leftrightarrow \sum_{e \in E} \upsilon_e(p) < \sum_{e \in E} \upsilon_e(q).$$

2. **If indifferent, sort according to the leader's preference using Equation 4.1.**

3. **If still indifferent, sort according to the follower's preference using Equation 4.2.**

Of course, the ranking mechanism itself does not solve the problems inherent to the use of side payments highlighted above. However, if partners agree on side payments, DEAL can be reconfigured easily to respect this choice.

4.6 Selection of the mating pool and update of population

The evolutionary principle boils down to foster good solutions while eliminating bad ones. This so-called selection pressure can be observed at three different phases: when individuals are ranked, when they are selected according to this ranking and when individuals are deleted from the population. Possibilities for decentrally computing subrankings and merging those subrankings to a global ranking have been discussed in the previous section. Having computed such a ranking, there exist different options to actually select individuals for the mating pool. One possibility is so-called tournament selection. Here, a tournaments is played between two[1] randomly picked solutions, and the better solution is chosen and placed in the mating pool. Two other solutions are picked again and another slot in the mating pool is filled with the better solution. If carried out systematically, each solution can be made to participate in exactly two tournaments (cf. Deb, 2001). Each solution except the worst one has a certain probability of being selected. This property allows EAs to surpass local optima by temporarily selecting individuals that are actually worse but provide a *path* to improved solutions.

After children have been constructed, the population is reduced to its original size. In the sequential coordination, we follow the so called $\mu + \lambda$ approach: from μ parents, λ children are generated. Then, the best μ out of the combined μ parent and λ child individuals form the next parent generation. In other words, the worst individuals are deleted deterministically. Population updating is straightforward, as illustrated in Figure 4.12. The leader uses the global ranking constructed previously to decide which individuals are removed from the current population. Communication threads that are not connected to any undeletable individual are considered to be deletable threads. Followers receive an update request containing lists of deletable threads that affect them. Each follower first disconnects these threads from their individuals. After the threads (respectively, their messages) have been deleted, a confirmation is sent. Subsequently each follower classifies those complete individuals as deletable that have no thread attached. Upon receiving the confirmation, the leader starts a similar process. First the threads are disconnected and deleted, then individuals are marked as deletable. Finally, deletable individuals are deleted if

[1] Theoretically, also larger tournament sizes are possible but are not used here for the sake of simplicity.

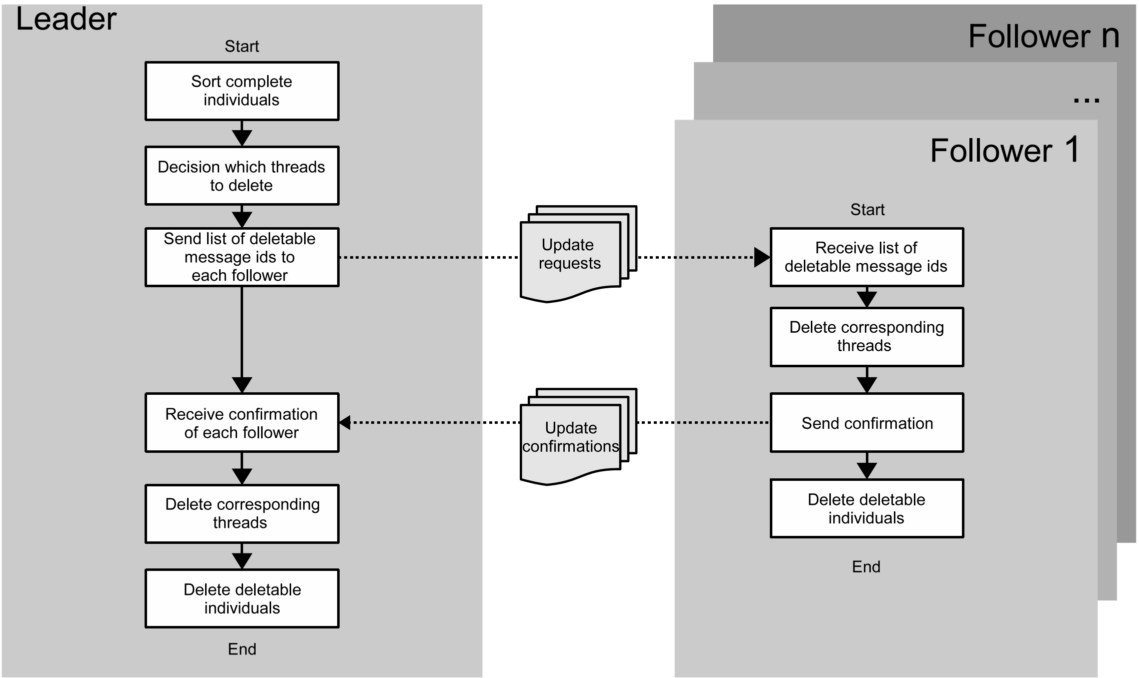

Leader
Start
Sort complete individuals
Decision which threads to delete
Send list of deletable message ids to each follower
Receive confirmation of each follower
Delete corresponding threads
Delete deletable individuals
End
Update requests
Update confirmations
Follower n
...
Follower 1
Start
Receive list of deletable message ids
Delete corresponding threads
Send confirmation
Delete deletable individuals
End

Figure 4.12: Update of a population within the DEAL framework .

Subprocess	Command	Reply	Leader	Follower
Construction & Initialization	request_guidance	guidance	x	
	guidance	proposal		x
	proposal	counterproposal confirm_proposal	x	
	counterproposal confirm_proposal	n.a.		x
Evaluation	request_rank	rank	x	
	rank	n.a.		x
Update	update	confirm_update	x	
	confirm_update	n.a.		x
Implement final solution	final_selection	n.a.	x	
	terminate	n.a.	x	

Table 4.1: Generic commands, replies and issuers of messages

not longer needed. In the asynchronous coordination discussed in Section 4.9, deletable individuals might still be temporarily needed if the construction of related children is not finished yet. However, they are not further considered by evaluation and selection.

4.7 Generic message protocol

In summary, we give an overview of the generic commands used by the domains in the DEAL framework. For better legibility, we classify the commands into *construction*, *evaluation*, *updating* and *final solution implementation* commands. Table 4.1 gives an overview of the commands from a generic perspective, showing the subprocess (cf. Figure 4.5), the initial command, the expected reply and the domain responsible for sending the message. From a design perspective, a message can be regarded as a container of components. During the implementation of the prototype, it turned out that some of the message components can be reused to define the individuals. As an

Component	Purpose	Attached to
predecessor	Stores the information on which message precedes in the current communication thread.	All replies in a thread: `counterproposal`, `confirm_proposal`, `rank`, `confirm_update`, `init_finished` and `solution`.
parent	Stores the information on which message or individual is the parent. The information is needed for a warm reboot of the DS Optimizer solving module	All individuals and messages with command `proposal`.
children	Opposite to parent component. Needed to avoid deletion of individuals whose children are still under construction.	All individuals.
individual link	Links a construction or solution thread to an individual. Necessary to determine the individual's state (idle or waiting).	Messages with commands `proposal`, `counterproposal`, `confirm_proposal`, `request_solution` and `solution`.
rank	Stores the current rank of an individual.	All individuals.
ranking list	Needed for transmitting rankings from follower to leader.	Message with command `rank`.
deletion list	Needed for transmitting update instruction from leader to follower.	Message with command `update`.

Table 4.2: Generic components of the message protocol

individual is constructed, more and more components are added to the individual (or some are modified) until the construction is completed. In other words, the individual can be regarded as component containers, too. Components can be further distinguished as generic and customized components, adapted to a specific optimizer. Table 4.2 gives an overview of generic components. A complete message protocol for the customized framework is presented in Chapter 7.

4.8 Properties of sequential coordination

The above concepts lead to a sequential coordination. That is, following the generic process depicted in Figure 4.5, leader and followers construct generation for generation one individual after the other until a satisfying quality of the interorganizational solution has been reached. The sequential coordination comes with several advantages and disadvantages. An advantage is the generality of the concept. Most requirements mentioned in Section 4.1 are fulfilled, nondisclosure of confidential data, no side payments and feasibility of solutions (regarding the generic construction process highlighted in Section 4.4.2). Moreover, the DEAL concept is—at least to some extent—independent of specific local optimization methods. Knowing the interface of their optimization methods, domains might only need to implement an add-on without changing their model or optimization method.

Some coordination schemes in the literature rely on a mediator. Similarly, the DEAL concept allows for the incorporation of a mediator, though it is not explicitly used in this work. Using a mediator offers three advantages. First, decisions on update and selection can be carried forward to the mediator, who acts in the interest of the whole Supply Chain, trying to equally support the interests of all domains. Second, confidential data, such as true costs, can be reported to the mediator who is committed to nondisclosure. Thus, the mediator could provide a more precise construction and evaluation of individuals. Theoretically, a mediator could substitute the leader in coordinating the message-exchange, evaluating, selecting and updating the individuals but would not have any planning functionality. The former leader would become another follower computing production plans, sending proposals and rankings. Finally, for multitier SCs, the mediator might form the central point of coordination. Nevertheless, from a practical point of view, it is questionable if a mediator really convinces the domains to disclose their confidential data, or only introduces another entity, further complicating the process. Moreover, due to the high responsibility for the coordination result and legal concerns third parties might be difficult to find to take the mediator's role. For these reasons, we did not delve much deeper into the mediator concept, although it will be again briefly mentioned in Chapter 7.

A clear disadvantage is that *decoding* a genotype of instructions to a feasible interorganizational solution requires a lot of time as the different local optimization methods are sequentially applied. Thus, the DEAL framework cannot be regarded as an Evolutionary Algorithm in the pure sense, which usually requires the evaluation of several thousands individuals. Depending on problem sizes and available runtime, we are rather allowed 10 to 500 tries of changing instructions. On the other hand, we probably have to deal with enormous search spaces of instructing data and underlying NP-hard intradomain problems. Thus, a random mutation of instructions does not seem feasible; we must rely on heuristic analyzing operators to find promising proposals. In general, we cannot expect to find the optimal interorganizational solution within a reasonable time. However, we argue that for most practical settings the underlying local optimization methods provide an inherent flexibility to compute "good" production plans even though instructions have only been set in a crude manner. The DEAL concept must rather be seen as a framework that allows an iterative execution and self-adaptation of heuristics to compute proposals. If the population consisted of just one individual and one constructed child, we have a simple hill-climbing procedure. To further increase the speed of computing and evaluating proposals, two approaches are imaginable. One approach would be to use so-called metamodels. Based on historical experience, the leader might be able to estimate a simplified model of his followers' production problems. Then, a lot of genotypes could be identified as "bad" in a pre-evaluation run based on the metamodels. Examples for such approaches can be found in Branke and Schmidt (2005), Ziegler and Banzhaf (2003), Jin (2002), and Ratle (1998). The quality of such approaches is highly dependent on the estimated metamodels. If the metamodels are too simple, a bias results, leading away from actual good regions of the search space. Intuitively, more complex metamodels require more historical data for parameter estimation. Unfortunately, in Production Planning and Detailed Scheduling, most optimization problems have a combinatorial component that complicates the metamodels. In addition, only little historical data (e.g., past coordination proposals) exist that are still relevant to the current situation. Moreover, not only the fitness needs to be estimated, but also functions for calculating counterproposals. Last but not least, an impeded implementation of such models is *intended* by the coordination mechanism—it was purposely constructed to limit possibilities for inferring sensitive data of other domains.

Instead of metamodels, a speed increase was realized by a *brute force* parallel computation. A property of the DEAL-concept is that the construction of proposals and counterproposals is dependent on the related parent solution, but independent of other solutions of the population. Thus, proposals and counterproposals can be calculated in parallel on different CPUs. This gives a speed increase up to the order of the number of parallel CPUs used, not considering the overhead for managing the population. A first step to parallelization is to allow asynchronicity, to allow the planning domains to start the construction of a new solution before the construction of the current solution is finished. In the following sections, we will discuss means of extending the generic concept to the asynchronous and the parallel coordination.

4.9 Asynchronous coordination

Sequential coordination is characterized by periodic idle times. Followers wait for new leader proposals to counter. Similarly, leaders wait for counterproposal before the construction of individuals is continued. Though this process provides a good traceability, it wastes a lot of the available runtime.[2] Asynchronous coordination aims to reduce unnecessary idling by allowing the leader to construct new proposals though followers have not yet replied to the current proposals. Having already invested effort in establishing the message protocol and the concept of communication threads, the implementation of an asynchronous coordination is straightforward. Whenever a message is issued, the subprocesses for construction (figures 4.7 - 4.10), evaluation (Figure 4.11) or update (Figure 4.12) are interrupted, leaving the individual in a waiting state. If the required messages have arrived the related subprocesses continue.

Algorithm 3 shows the generic asynchronous coordination template for the leader and Algorithm 4 shows the same for each follower. First, all incoming messages are processed completely and existing communication threads are extended. If applicable, interrupted subprocesses for update and evaluation are continued. During construction of individuals, the processing of incoming messages sets "waiting" individuals to "idle." In the second step, the construction of the oldest idle individual is continued and its state is either set to "complete" or "waiting" if additional action from the followers is required. Third, conditions for starting the construction of new individuals are tested and new rankings are requested if necessary. The three steps are iteratively continued until the termination condition is met. It should be emphasized that for asynchronous coordination, the selection of the mating pool is different from that of a standard EA. In a standard EA, the whole mating pool is selected every generation and all children are constructed instantly. However, for asynchronous coordination, the term generation is elusive, as the focus lies on the steady flow of messages and the efficient use of computational power. Waiting until all individuals of a generation have been constructed before starting the new generation would be counterproductive to a high utilization of computational power. Hence, new individuals are constructed whenever the sum of idle and waiting individuals drops below a predefined threshold. This ensures that enough work remains in the system and runtime is not wasted. If the threshold is set too low, new individuals are issued too late and domains have nothing to compute during some time periods, as in the sequential coordination scheme. If set too large, however, too many individuals are constructed in advance based on the currently available ranking and population. In other words, the ranking and population information per child is to some extent already outdated when the child is actually constructed. For constructing the children, parent individuals need to be already selected in the mating pool. As the evaluation procedure takes additional time, a new evaluation process is started if the number of individuals in the mating pool drops below a certain threshold. When the evaluation is finished, parents are selected in the mating pool until the threshold is no longer violated. When constructing a new child, the oldest individual of the mating pool is taken. In some sense, the mating pool can be seen as an additional buffer for storing selected parents following a *first-in, first-out* policy.[3] Summarizing, two buffers are needed to

[2] Assuming realistic conditions, where domains do not share a single CPU.

[3] Obviously, also other policies could be implemented. However, from our point of view it is most appropriate to

Algorithm 3: Leader's template for asynchronous coordination

```
while Not terminated do
    if Termination criterion fulfilled then
        Select and broadcast final solution
        Terminate
    end
    forall Incoming messages from follower do
        Attach message to related communication thread
        if Update confirmation then
            if Update messages from all followers have arrived then
                Finish update procedure
            end
        else
            if Ranking message then
                if The subrankings from all followers have arrived then
                    Finish evaluation procedure
                    Select new parents to mating pool until queuing size has been reached
                end
            else
                Retrieve individual connected to thread
                Set individual state to idle (if all required messages have arrived)
            end
        end
    end
    if Population contains idle individuals then
        Continue construction of oldest idle individual by requesting guidance, issuing proposals or
        considering counterproposals
        Set individual state to waiting / complete (if construction is finished)
    end
    if Number of waiting and idle individuals is below queuing size then
        if Mating pool does not contain enough parents to construct desired size of children then
            if Evaluation procedure finished then
                if Update procedure not started yet then
                    Start update procedure
                end
            else
                if Evaluation procedure not started yet then
                    Start evaluation procedure
                end
            end
        end
        if Mating pool not empty then
            Take oldest parent from mating pool and construct idle child
        end
    end
    Sleep
end
```

ensure a steady use of solving units. One consists of a certain number of idle and waiting individuals in the OEM's population, the other of a required number of parents in the mating pool. For simplicity, the numbers are set to be equal and will be denoted as the **queuing size** henceforth. For a queuing size of zero, asynchronous and sequential coordination are very similar. As a future research topic a self-adaptation of the queueing size seems appropriate. For example, the queuing size could be dynamically increased until idle times of the leader have decreased to a predefined

consider "old" information first, before it becomes (even more) deprecated.

Algorithm 4: Follower's template for asynchronous coordination

```
while Not terminated do
    forall Incoming messages from leader do
        Attach message to related communication thread
        if Update request then
            Start update procedure
        else
            if Final solution then
                Implement final solution
                Terminate
            else
                if Evaluation request then
                    Start evaluation procedure
                else
                    if Guidance request then
                        Construct guidance and reply
                    else
                        if Redundant proposal then
                            Retrieve individual connected to thread
                            Reply according to existing counterproposal / confirmation
                        else
                            Construct new idle individual
                            Connect individual to thread
                        end
                    end
                end
            end
        end
    end
    if Population contains idle individuals then
        Construct oldest idle individual and send counterproposal / confirmation
        Set individual state to complete
    end
    Sleep
end
```

level. In our computational evaluation (cf. Chapter 8), a fixed queueing size already generated good results, however.

4.10 Parallel coordination

Parallel coordination is a further extension of asynchronous coordination, motivated by the idea of speeding up proposal and counterproposal generation. Calling the solution methods for computing production plans is actually the most time-consuming task. Some solution methods might already provide built-in support for multiple CPUs, but we cannot assume such a feature in general. However, even if it is not possible to use several CPUs to compute one production plan, we can use the computational power to construct production plans in parallel.

How parallelization can be established depends on technical details. As a concrete example, the SAP Grid Framework is presented later in Section 7.10. On an abstract level, parallelization is achieved by introducing new **solving units**, representing additional computational power for computing an intradomain solution. Leader and followers can delegate the task of calculating a production plan to a solving unit through an additional message exchange, depicted in Figure

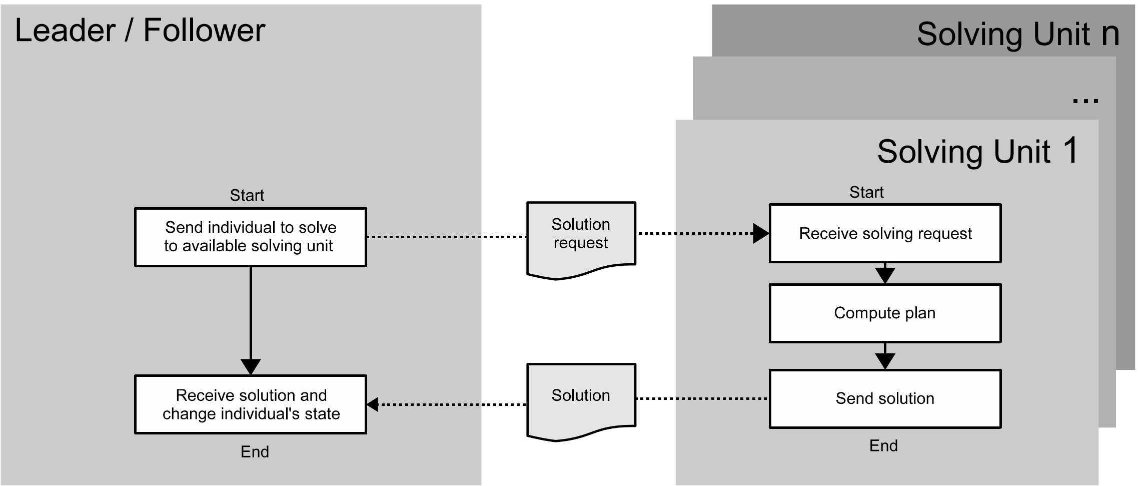


Figure 4.13: Solving an individual within the parallel DEAL framework .

Algorithm 5: Extension of leader's / follower's template to parallel coordination

```
while Not terminated do
    ...
    forall Incoming messages from solving units do
        Attach message to related communication thread
        Retrieve individual connected to thread
        Set individual state to idle
        Mark solving unit as available
    end
    ...
    if Population contains idle individuals then
        if Solution from solving unit received then
            Update individual accordingly
        end
        ...
        if Solution from solving unit needed then
            if Solving unit is available then
                Issue solution request
                Mark solving unit as unavailable
                Set individual state to waiting
            end
        end
    end
    ...
end
```

4.13. Before the solution of the related problem instance has been computed, an individual's state is set to "idle." Available solving units are sought. If a unit can be found, a solving request is

Algorithm 6: Solving unit's template to parallel coordination

```
while Not terminated do
    forall Incoming solution requests do
        Compute plan
        Send solution back
    end
    Sleep
end
```

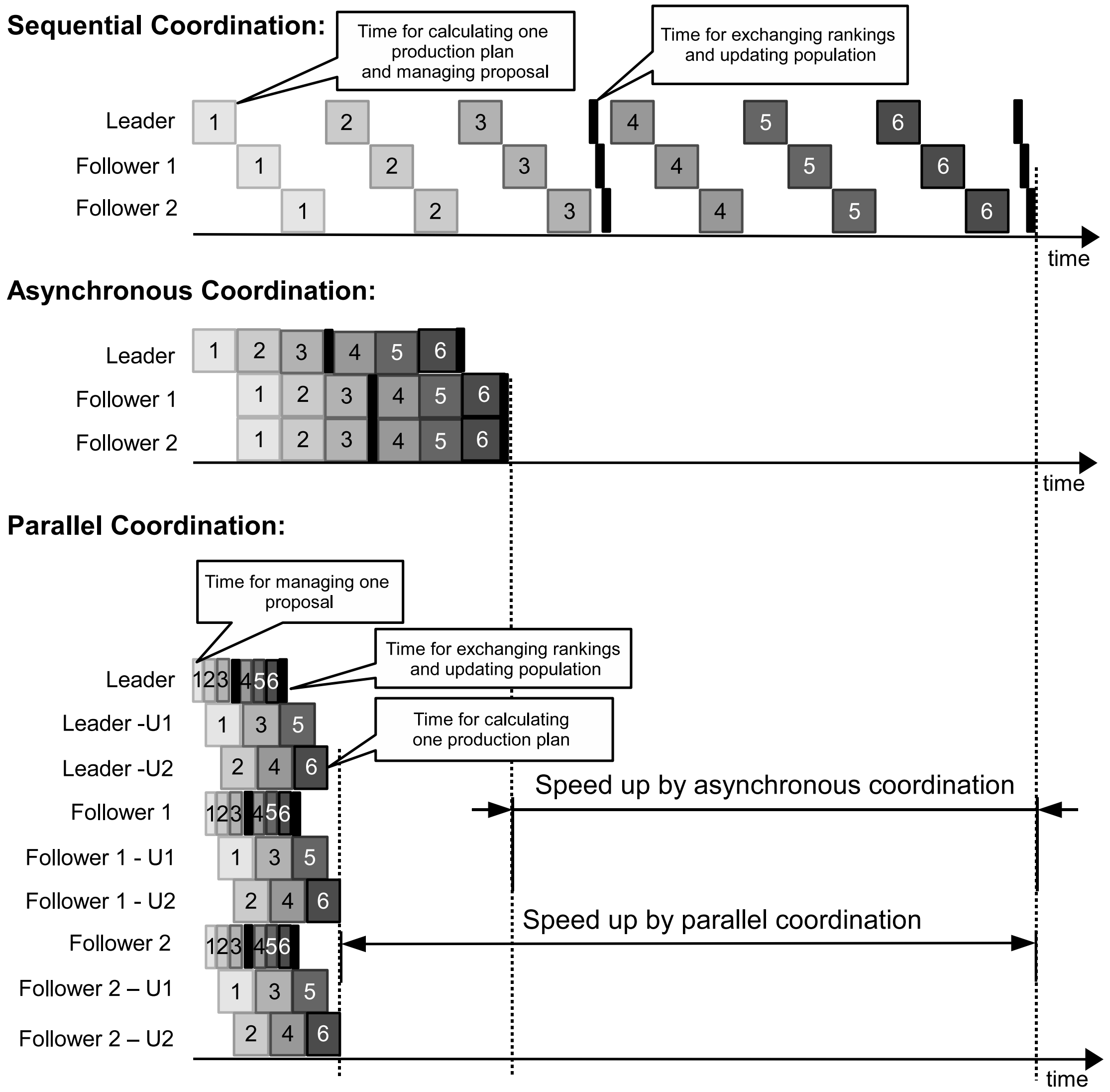

Sequential Coordination:
Time for calculating one production plan and managing proposal
Time for exchanging rankings and updating population
Leader
Follower 1
Follower 2
1 2 3 4 5 6
time
Asynchronous Coordination:
Leader
Follower 1
Follower 2
1 2 3 4 5 6
time
Parallel Coordination:
Time for managing one proposal
Time for exchanging rankings and updating population
Time for calculating one production plan
Leader
Leader -U1
Leader -U2
Follower 1
Follower 1 - U1
Follower 1 - U2
Follower 2
Follower 2 – U1
Follower 2 – U2
123456
1 3 5
2 4 6
Speed up by asynchronous coordination
Speed up by parallel coordination
time

Figure 4.14: Comparison of sequential, asynchronous and parallel coordination .

issued and the individual's state changes to "waiting" until the solving unit replies with the corresponding solution. Algorithm 5 shows how the leaders's and followers's template (algorithms 3 and 4) can be extended. The rest of the templates for leader and follower do not need to be changed (this is denoted by "..." in Algorithm 5). Algorithm 6 shows the template of each solving unit. Concrete procedures for the DS Optimizer will be discussed in Chapter 7. Figure 4.14 schematically illustrates the time requirements of a coordination process for the sequential, asynchronous and parallel coordination. The top part of Figure 4.14 shows sequential coordination for constructing six individuals. The leader begins by constructing his part of Individual 1, followed by Followers 1 and 2. The illustration is idealized, ignoring such details as proposals, counterproposals and so on. After Individual 1 is completed, Individuals 2, 3 are constructed. Then, the population is evaluated and updated. The time required for evaluation and update is depicted by black boxes and the time for constructing an individual by gray boxes. Subsequently, further individuals are constructed. The middle of Figure 4.14 shows asynchronous coordination. A speed up is realized since the followers can run their computations independent of each other and because the OEM can issue new proposals, before the previous ones have been evaluated. The last part illustrates parallel coordination. Let us assume that each domain controls two solving-units, U1 and U2. The task of constructing an individual is schematically split up into two subtasks, one concerned with managing proposals and counterproposals, one concerned with calculating the related production plan, the latter being delegated to a solving unit. With regard to asynchronous coordination, parallel coordination additionally speeds up the coordination process. However, as more solving units are available, more individuals need to be constructed in "advance" based on outdated rankings. Together with increasing the communication overhead, this can have a limiting effect on the performance of the parallel coordination. Also, for asynchronous and parallel computation, the meaning of the term "population" needs to be redefined. Due to time offsets between construction, evaluation and updates, different kinds of individuals are contained in the leader's and followers' populations. In fact, the domains need to manage four populations: idle individuals, waiting individuals, complete individuals and deletable individuals. The terms idle, waiting, complete and deletable refer to an individual's state, cf. Section 4.4.1.

CHAPTER 5

The Resource-Constrained Project-Scheduling Problem

The Resource-Constrained Project-Scheduling Problem (RCPSP) belongs to the class of NP-hard problems. Being a generalization of other well-known problems, such as job-shop or flow-shop problems (cf. Blazewicz et al., 1983; Drexl, 1990; Neumann et al., 2001), the RCPSP has been a focus of research for several decades. In our experimental study, the RCPSP forms the underlying problem of a single planning domain, which is assumed to optimize its production plan using the SAP Detailed Scheduling (DS) Optimizer. We start by formulating the standard version of the RCPSP in Section 5.1. Afterwards, we extend the standard formulation step by step to capture multiple modes, minimum and maximum time lags, varying productivity and sequence-dependent setup times. We conclude this chapter with an overview of present state-of-the-art approaches for solving the RCPSP. The DS Optimizer is presented in Chapter 6. Tables 5.1, 5.2 and 5.3 list the indices, data and variables used in this discussion.

5.1 The basic problem

Very briefly, the RCPSP can be described as follows: A single project consists of several, interdependent activities that need to be scheduled, such that

- the earliest release and latest finish dates of the project and its activities,
- the available resource capacity,
- and the precedence relations between the activities are respected.

To be more precise, we introduce the following definitions:

Definition 5.1.1. (planning interval) The **planning interval** is defined as the discrete time interval $[0, \ldots, \overline{T}] \in \mathbb{N}_0$ consisting of several periods with a given period length (e.g., seconds), in which all activities start and end.

Definition 5.1.2. (activities and resources) An **activity** j refers to a single production activity (e.g., the assembly of parts). An activity has a **start date** $sd_j \in [0, \ldots, \overline{T}]$ and a **duration** $d_j \geq 0$,

j, k, l	index for the activities, $j = 1, \ldots, n$
J	set of all activities
J_S	set of setup activities, $J_S \in J$
m	index of mode, $m \in M_j$
M_j	set of possible modes for activity j
P_j / S_j	set of activities which immediately precede / succeed activity j
$\overline{T}$	planning horizon, i.e. the end of the planning interval $\left[0, \ldots, \overline{T}\right]$
$T := \left[0, \ldots, \overline{T}\right]$	planning interval
t	index for periods, $t \in \left[0, \ldots, \overline{T}\right]$
r	index for renewable resource type, $r \in R$
R	set of all renewable resources

Table 5.1: Indices and index sets for the RCPSP

a_r	constant capacity of resource r
a_{rt}	time-dependent capacity of resource r
c_{jm}	cost of running activity j in mode m
d_j	duration of activity j as an integral number of periods
d_{jm}	duration of activity j in mode m
dd_j	soft-constrained due date of activity j
dl_j	deadline of activity j
ef_j	propagated, earliest finish date of activity j
es_j	propagated, earliest start date of activity j
lf_j	propagated, pseudo-hard latest finish date of activity j
ls_j	propagated, pseudo-hard latest start date of activity j
pdd_j	propagated, soft-constrained due date of activity j
r_m	primary resource of mode m
rd_j	release date of activity j
u_{jr}	constant per-period demand by activity j for resource r
u_{jmr}	constant per-period demand by activity j in mode m for resource r
wtl	weighting of total lateness objective
wml	weighting of maximum lateness objective
$wtst$	weighting of total setup-time objective
$wtsc$	weighting of total setup-cost objective
$wtmc$	weighting of total mode-cost objective
wms	weighting of makespan objective
λ_{ij}	minimum time lag between activity i and j
$\overline{\lambda}_{ij}$	maximum time lag between activity i and j
ψ_{jk}	setup cost for changeover from activity j to k
σ_{jk}	setup duration for changeover from activity j to k

Table 5.2: Data for the RCPSP

fd_j	finish date of activity j
m_j	selected mode of activity j
sd_j	start date of activity j
thl	total pseudo-hard lateness of the schedule
tl	total lateness of the schedule
ml	maximum lateness of the schedule
ms	makespan of the schedule
tsc	total setup cost of the schedule
tst	total setup time of the schedule
tmc	total mode cost of the schedule
twc	total weighted cost

Table 5.3: Variables for the RCPSP

defining the **finish date** $fd_j = sd_j + d_j$. The set of all activities is referred to as J. Moreover, there exists a set R of **renewable resources**.[1] Each resource r has a constant capacity of a_r units that is renewed every period t. Activities compete for scarce resource capacity; that is, an activity can only be executed if there is enough capacity available. When executed, an activity r places a constant **resource usage** u_{jr} on resource r throughout its duration. The time for executing an activity is bounded by its **release date** rd_j and its **deadline** dl_j. We can further divide the set of renewable resources into **unary resources** and **multicapacitated resources**. Unary resources only allow one activity to be scheduled at a specific time, multicapacitated resources can handle several activities at once.

Definition 5.1.3. (precedence relation) The activities have to be executed in a predefined partial order, the so-called activity-precedence relation. For convenience, we introduce the sets $P_j = \{i \in J | i$ is the **immediate predecessor** of $j\}$ and $S_j = \{i \in J | i$ is the **immediate successor** of $j\}$.

Definition 5.1.4. (RCPSP) An instance of the RCPSP is given by

- a planning interval $[0, \ldots, \overline{T}]$,
- a set of activities J,
- a set of renewable resources R,
- a precedence relation given by P_j or S_j for each activity j, respectively,
- the precedence and capacity constraints

$$sd_j \geq rd_j \qquad \forall j \in J \tag{5.1}$$

$$sd_j \geq fd_i \qquad \forall j \in J, i \in P_j \tag{5.2}$$

$$fd_j = sd_j + d_j \qquad \forall j \in J \tag{5.3}$$

$$\sum_{j \in J | \, sd_j \leq t \leq fd_j} u_{jr} \leq a_r \qquad \forall t \in [0, \ldots, \overline{T}], r \in R \tag{5.4}$$

$$fd_j \leq dl_j \qquad \forall j \in J \tag{5.5}$$

- an objective function that shall be minimized or maximized.

Put verbally, an activity must not start before its release date (Equation 5.1) or before its immediate predecessor has been finished (Equation 5.2). After being started, an activity has a constant duration until it finishes (Equation 5.3). During execution, activities may use several resources. In each period, available resource capacity must not be exceeded (Equation 5.4). An activity's finish date must lie before its deadline (Equation 5.5).

An assignment of start and finish dates to all activities respecting the constraints 5.1–5.4 is called a **feasible schedule**, see also Sprecher et al. (1995). Solving the RCPSP aims to find the schedule with the optimal objective function value. A classical objective is makespan minimization, where the makespan is defined as the time from starting the first activity to finishing the last activity. Other objectives are introduced in Section 5.7. Generally speaking, we seek to find the best sequence of activities on the resources for the given objective function while preserving the

[1] Some problem definitions also include nonrenewable resources (e.g., Kolisch, 1995).

precedence relations and obeying capacity constraints. It should be noted that the activities considered here are nonpreemptive, i.e. all activities need to be processed in *one piece*. Furthermore, we assume all data to be deterministic. Note that equations 5.1–5.5 define a conceptual model, because the sets $\{j \in J|\ sd_j \leq t \leq fd_j\}$ in Equation 5.4 are functions of the decision variables sd_j and fd_j. For different integer programming formulations of the RCPSP, see Stadtler (2005a), Neumann et al. (2001) or Klein (2000). Since our focus lies on heuristic solution methods, we omit giving an integer programming formulation.

If hard-constrained deadlines are set too early, the problem becomes infeasible. It should be mentioned that the DS Optimizer introduced in the next chapter treats such deadlines as pseudo-hard constraints. Though the resulting schedule may violate deadlines, it can be at least computed and help the planner to identify capacity problems. Henceforth, we will denote constraints 5.1–5.4 as the standard problem (not including the deadlines as constraints). More details are given in Section 5.7.

Throughout the remainder of this thesis, we assume that enough capacity is available to solve 5.1–5.4. Thus, the computed schedule may violate deadlines but capacity constraints are always respected. We further assume that $\overline{T}$ is sufficiently large to compute feasible schedules.[2] To ease the following presentation, the activities are assumed to be numbered following a topological sorting: $i < j$ if activity i is a predecessor of activity j.

For each activity j, we can calculate lower bounds es_j for start dates and upper bounds lf_j for finish dates by performing a so-called forward or a backward pass (see also Klein, 2000). Starting with $es_1 = rd_1$, the forward pass propagates the earliest start and finish dates through the network by calculating

$$es_j = \max\{rd_j, \max\{es_i + d_i | i \in P_j\}\} \text{ for } j = 1, \ldots, |J|, \tag{5.6}$$

$$ef_j = es_j + d_j,\ \forall j \in J. \tag{5.7}$$

The backward pass propagates the upper bounds from the end of the horizon backward starting with $lf_n = dl_n$:

$$lf_j = \min\{dl_j, \min\{lf_h - d_h | h \in S_j\}\} \text{ for } j = |J|, \ldots, 1, \tag{5.8}$$

$$ls_j = lf_j - d_j,\ \forall j \in J. \tag{5.9}$$

Forward and backward passes allow us to derive tighter bounds for the activities' start and finish dates. That is, we can replace 5.1 with $es_j \leq sd_j \leq ls_j\ \forall j \in J$ and 5.5 with $ef_j \leq fd_j \leq lf_j\ \forall j \in J$. Using the tighter bounds, subsequently applied scheduling heuristics can work more efficiently.

5.2 Active, semi-active and nondelay schedules

Classification of schedules is part of the basic preparation to tackling the RCPSP. Schedules can be distinguished as **active**, **semi-active** and **nondelay schedules**. Traditionally, theses concepts originate from the job-shop literature (cf. Sprecher et al., 1995) and are usually defined rather informally. For the RCPSP, being a generalization of the job-shop problem, the usual informal

[2] That is, $\overline{T} \geq \sum_{j=1}^{|J|} d_j$ in the "worst case" of a single unary resource.

definitions can lead to misunderstandings. Thus, we provide more rigorous definitions, adopted from Sprecher et al. (1995).

Definition 5.2.1. (left shift) A left shift of activity $j \in J$ is an operation on a feasible schedule S that derives a new feasible schedule S^* such that $sd_j^* < sd_j$ and $sd_i^* = sd_i$, for $i \in J, i \neq j$.

Definition 5.2.2. (one-period left shift) A left shift of activity $j \in J$ is called one-period left shift if $sd_j - sd_j^* = 1$.

Definition 5.2.3. (local left shift) A local left shift of activity $j \in J$ is a left shift of j obtainable by one or more successively applied one-period left shifts.

Definition 5.2.4. (global left shift) A global left shift of activity $j \in J$ is a left shift that is not obtainable by a local left shift.

Having defined the above, we can define semi-active and active schedules.

Definition 5.2.5. (semi-active schedule) A semi-active schedule is a feasible schedule where none of the activities can be *locally* left shifted.

Definition 5.2.6. (active schedule) An active schedule is a feasible schedule where none of the activities can be *globally* left shifted.

Informally spoken, a semi-active schedule is a schedule where no activity can be started earlier without changing the sequence of activities on the resources. An active schedule is a schedule where no activity can be started earlier without delaying the start of another activity. Obviously, all active schedules are also semi-active.

A third classification is possible. In the context of job-shop problems a **nondelay schedule** is a schedule where no machine is kept idle at a time when it could begin processing some operation. We can also give a more general definition of a nondelay schedule: According to Sprecher et al. (1995), each RCPSP can be uniquely transformed to a unit-time-duration RCPSP (UTDRCPSP), where each activity $j \in J$ is split into d_j activities, each of which with a duration one. Thus, a feasible schedule S of the RCPSP uniquely corresponds to a feasible schedule $UTDS$ of the UTDRCPSP.

Definition 5.2.7. (nondelay schedule) A feasible schedule S for the RCPSP is called a nondelay schedule if the corresponding $UTDS$ is active.

Theorem 5.2.8. Let S denote the set of schedules, FES the set of feasible schedules (not violating constraints 5.1- 5.4), SAS the set of semi-active schedules, AS the set of active schedules and NDS the set of nondelay schedules. Then the following relations hold (cf. Sprecher et al., 1995).

$$NDS \subseteq AS \subseteq SAS \subseteq FES \subseteq S$$

As will become apparent later, different construction methods exist that produce either nondelay, active or semi-active schedules. Interestingly, the objective function determines whether the optimal schedule can be found within NDS, AS or SAS. We introduce the following definition.

Definition 5.2.9. (regular performance measure) Let $fd_1, \ldots, fd_n$ be the finish dates of activities $1, \ldots, n$, respectively. A performance measure is a mapping, $\Phi : \mathbb{Z}^n_{\geq 0} \to \mathbb{R}_{\geq 0}$, that assigns each n-tuple $fd_1, \ldots, fd_n$ of activities a performance value $\Phi\left(fd_1, \ldots, fd_n\right)$. If Φ increases monotonically with respect to component-wise ordering—mathematically, if $\Phi(fd_1, \ldots, fd_n) > \Phi(fd_1^*, \ldots, fd_n^*)$ implies $\forall j \in J : fd_j \geq fd_j^*$ and $\exists j \in J : fd_j > fd_j^*$—and, in addition, minimization is considered, we denote the performance measure as **regular**.

Makespan minimization is obviously a regular performance measure. It can easily be seen that if Φ is regular and a schedule S^* is obtainable from S by a global *or* local left shift, then S is dominated by S^*, w.r.t. Φ. Hence, for regular performance measures, the optimal schedule is within the set of all active schedules. However, as Sprecher et al. (1995) show by a counterexample, the optimal schedule is not necessarily within the set of all nondelay schedules. Idle times in active schedules allow resources to wait for more critical activities that are due to be available, and this can lead to improved finish dates. A more comprehensive classification of performance measures can be found in Neumann et al. (2001).

The constraints of the standard RCPSP 5.1–5.4 are now gradually extended in the next sections until the problem underlying the DS Optimizer has been defined.

5.3 Minimum and maximum time lags

Until now, we have assumed that each activity is allowed to start directly after the finishing of its predecessor. We can generalize this finish-to-start (FS) relation by introducing **minimum** and **maximum time lags** (also called **links**), λ_{ij}^{FS}, $\overline{\lambda}_{ij}^{FS}$; see Klein (2000), Neumann et al. (2001) and Schwindt (2005) for a detailed introduction to time lags. A minimum time lag constitutes the minimum time needed between the finishing of an activity i and the starting of its successor j, the maximum time lag the maximum allowed time. Moreover, there exist further relation types such as start-to-finish ($\lambda_{ij}^{SF}, \overline{\lambda}_{ij}^{SF}$), finish-to-finish ($\lambda_{ij}^{FF}, \overline{\lambda}_{ij}^{FF}$) and start-to-start ($\lambda_{ij}^{SS}, \overline{\lambda}_{ij}^{SS}$) relations. To incorporate these new constraints in the RCPSP formulation, Equation 5.2 needs to be replaced by:

$$sd_j \geq fd_i + \lambda_{ij}^{FS} \qquad \forall j \in J, i \in P_j \tag{5.10}$$
$$sd_j \geq sd_i + \lambda_{ij}^{SS} \qquad \forall j \in J, i \in P_j \tag{5.11}$$
$$fd_j \geq fd_i + \lambda_{ij}^{FF} \qquad \forall j \in J, i \in P_j \tag{5.12}$$
$$fd_j \geq sd_i + \lambda_{ij}^{SF} \qquad \forall j \in J, i \in P_j \tag{5.13}$$
$$sd_j \leq fd_i + \overline{\lambda}_{ij}^{FS} \qquad \forall j \in J, i \in P_j \tag{5.14}$$
$$sd_j \leq sd_i + \overline{\lambda}_{ij}^{SS} \qquad \forall j \in J, i \in P_j \tag{5.15}$$
$$fd_j \leq fd_i + \overline{\lambda}_{ij}^{FF} \qquad \forall j \in J, i \in P_j \tag{5.16}$$
$$fd_j \leq sd_i + \overline{\lambda}_{ij}^{SF} \qquad \forall j \in J, i \in P_j. \tag{5.17}$$

By considering the min-links, the earliest start dates can be calculated by a modified forward pass (cf. Klein, 2000, p.44) as

$$es_j = \max\left\{rd_j, \max_{i \in P_j}\left(es_i + \lambda_{ij}^{SS}\right), \max_{i \in P_j}\left(es_i + d_i + \lambda_{ij}^{FS}\right), \max_{i \in P_j}\left(es_i + \lambda_{ij}^{SF} - d_j\right),\right.$$
$$\left.\max_{i \in P_j}\left(es_i + d_i + \lambda_{ij}^{FF} - d_j\right)\right\} \quad \forall j \in \{1, \ldots, |J|\}. \tag{5.18}$$

Analogously, the latest finish dates can be calculated by

$$lf_j = \min\Big\{dl_j, \min_{i\in S_j}\left(lf_i - \lambda_{ij}^{SS} + d_j\right), \min_{i\in S_j}\left(lf_i - \lambda_{ij}^{FS} - d_j\right),$$
$$\min_{i\in S_j}\left(lf_i + d_i - \lambda_{ij}^{SF}\right), \min_{i\in S_j}\left(lf_i - \lambda_{ij}^{FF}\right)\Big\} \quad \forall j \in \{|J|,\ldots,1\}. \tag{5.19}$$

Maximum links do not affect earliest start dates and latest finish dates. Instead, they determine the *latest* start date and the *earliest* finish date of an activity. Backward and forward passes considering max-links can be implemented analogously to 5.18 and 5.19. If max-links *and* min-links form part of the problem such simple forward and backward passes are no longer possible because max-links can outdate previously computed upper and lower bounds, calculated on the basis of min-links, and vice-versa. Instead, a dynamic programming approach (cf. Schwindt, 2005, p.13) is applied that processes a set of outdated (or yet unconsidered) activities until this set becomes empty. However, for the sake of convenience, we will still speak of forward and backward passes henceforth, although both terms relate to the same dynamic programming approach in the case of min- and max-links. We do not go further into the details here, simply assuming that the deadlines are set in such a way that there exist feasible intervals for earliest and latest start and finish dates for all activities.

5.4 Multiple modes

A **mode** is a resource and processing time assignment to an activity. For example a product can be produced on one of several alternative machines. Put mathematically, each activity j can be scheduled in a specific mode m_j. Each mode m defines the capacity utilization on the **primary resource** r_m and optionally other **secondary resources**. Primary resources are the resources of main interest and have the additional property of varying setup states, which will be explained later in more detail. We refer to the set of an activity's possible modes as M_j. To capture multiple modes, the standard RCPSP 5.1 to 5.4 needs to be extended to

5.1–5.4

$$m_j \in M_j \qquad \forall j \in J \tag{5.20}$$

$$d_j = d_{jm_j} \qquad \forall j \in J \tag{5.21}$$

$$\sum_{j\in J|\ sd_j \le t \le fd_j} u_{jm_jr} \le a_r \qquad \forall t \in [0,\ldots,\overline{T}], r \in R. \tag{5.22}$$

Moreover, there are often **mode compatibility constraints**. For example, if the first activity of a production process was scheduled on a specific line, all subsequent activities have to be scheduled on the same line. In these cases, M_j depends on the activity's predecessors and needs to be dynamically adapted during scheduling, as will be discussed in Chapter 6.

The backward and forward passes are extended in such a way that the computed upper and lower bounds relate to the "best combination" of modes (shortest and longest durations). During backward and forward passes, some modes might also be identified as infeasible in a practical setting. Their duration might either be too long to meet the required deadlines or their resource utilization might exceed present resource capacity profiles. Such modes are thus marked as inaccessible (removed from the set M_j) when the forward and backward pass is applied.

5.5 Varying capacity and calendars

The complexity of many practical problems has its origin in the time dependence of the resources' properties. First, capacity may vary over time, and we must ascertain that, during the duration of an activity, enough capacity is available as expressed by Equation 5.24. Second, productivity may vary. Imagine a late-night shift, where only half of the workforce is available. This might influence activities scheduled overnight to take twice as much time as during the day. During a weekend, productivity might also drop to zero. That is, an activity of 10 hours can be started before the weekend, where 3 hours of work are finished on Friday and the remaining seven hours on Monday. Put mathematically, an activity's **gross-duration** is a function of its start date, the net-duration, and the involved resources, as shown in Equation 5.23. The standard RCPSP formulation 5.1–5.3 can be modified to

$$\begin{aligned}
&5.1\text{–}5.3, 5.20, 5.21 && \\
&fd_j = sd_j + BFD(sd_j, d_{jm_j}, r_{m_j}) && \forall j \in J && (5.23)\\
&\sum_{j \in J|\ sd_j \leq j \leq fd_j} u_{jm_jr} \leq a_{rt} && \forall t \in [0, \ldots, \overline{T}], r \in R \backslash B. && (5.24)
\end{aligned}$$

When performing backward or forward passes (5.6, 5.7, 5.8, 5.9,5.18,5.19), the gross-duration rather than the net-duration d_j of an activity is taken, if calendars are included. The duration of an activity is extended by breaks and lower productivity in the time period the activity currently occupies. Calendars can be different for different resources. For simplicity, only the productivity and break calendars of one resource (usually the primary resource) are considered for each mode. Thus, the function BFD calculates the gross finish date for a given start date sd_j, a given net-duration of the selected mode, and the related primary resource. For backward passes, a function BSD exists analogously calculating the gross start date, given the current finish date, the net duration and the primary resource. It should be emphasized that calendars do not affect length of time lags, however.

5.6 Sequence-dependent setup times

In the standard RCPSP formulation 5.1–5.4, it was implicitly assumed that the duration d_j of an activity j reflects its setup and processing time. This assumption makes sense if the setup times are small compared to the processing times. However, if setup times are considerable large and if some activities require the same setup, the makespan could be reduced by batching these activities. Moreover, the duration and cost of a setup could be sequence dependent, its length might depend on the previous activity scheduled on the same machine. For example, painting a product white after red can require considerably more cleaning of the equipment than vice versa (cf. Engelmann, 1998; Kolisch, 1995).

In our RCPSP formulation, **sequence-dependent setup times** are modeled by **setup activities**. Setup activities are like standard activities except for the fact that their net-duration varies, depending on the required setup-state transition. Setup-activities do not stand alone but "prepare" a resource for an appurtenant standard activity by changing the resource's setup state to the state required by the standard activity. Intuitively, the two appurtenant activities need to be scheduled on the same resource, which can be modeled by a mode-compatibility constraint. We can extend our formulation by

5.1–5.3, 5.20, 5.23, 5.24

$$d_j = d_{jm_j} \qquad \forall j \in J \backslash J_S \tag{5.25}$$

$$d_j = \sigma_{ij} \qquad \forall j \in J_S, i \in J \backslash J_S \text{ if } r_{m_j} = r_{m_i} \text{ and there is no activity } k \text{ with } r_{m_j} = r_{m_i} = r_{m_k} \text{ and } s_j \leq s_k \leq s_i. \tag{5.26}$$

Equation 5.26 defines that the length of a setup activity depends on its machine predecessor on the same *primary* resource. Moreover, we assume the related primary resources to be unary resources. In contrast to multicapacity resources, unary resources allow only one activity to be scheduled at any time:

$$a_{rt} \in \{0, 1\} \qquad \forall r \text{ that are primary resources, } t \in T \tag{5.27}$$

$$u_{jmr} = 1 \qquad \forall j \in J \text{ with } r \text{ as primary setup resource.} \tag{5.28}$$

We need to make this restriction to have well-defined setup-state transitions, see also Engelmann (1998). If required, multicapacity setup resources can be included by splitting them up into multiple unary resources accessible over the modes of the related activities.

5.7 Objective function

Besides minimizing the makespan, other objectives are important. This section lists all objectives that can be considered in the DS Optimizer. To consider multiple objectives within one objective function, a weighted sum approach is usually used. That is, for each of the (possibly conflicting) objectives, a weight is specified *a priori* and the weighted sum of objectives is then minimized—see also Section 4.2 for more details on this approach. An exception is the minimization of pseudo-hard lateness[3] (relating to the violation of deadlines), which dominates the weighted sum of other objectives.

5.7.1 Minimize total pseudo-hard lateness

According to 5.1–5.5, activities' finish-dates are constrained to be below the pseudo-hard bounds dl_j. However, for the sake of calculating a finite plan, the DS Optimizer is allowed to violate deadlines. Hence, the most important objective is to reduce this violation. Put mathematically, we want to

$$\text{Minimize } thl = \sum_{j \in J} \max \{fd_j - lf_j, sd_j - ls_j, 0\}. \tag{5.29}$$

Obviously, lateness minimization is a regular performance measure. The minimization of pseudo-hard lateness dominates as top-priority single objective the weighted sum of the following objectives.

[3] In literature also the term tardiness is used frequently. In this thesis we will stick to the term lateness as it is used as standard in the SAP Software.

5.7.2 Minimize makespan

The makespan-minimization objective can be formulated as

$$\text{Minimize } ms = \max_{j \in J} fd_j - \min_{j \in J} sd_j. \tag{5.30}$$

5.7.3 Minimize total lateness

In addition to deadlines, **due dates** are modeled as soft constraints (dd_j), influencing the weighted sum of second-priority objectives. Usually, soft-constrained due dates are used to model customer preferences, whereas deadlines refer to technological constraints. In practice, the lateness between promised soft-constrained due date and the realized finishing date of production is important since it measures the service level provided to the customer. The total lateness can be taken into account by including 5.31 in the objective function.

$$\text{Minimize } tl = \sum_{j \in J} \max \{fd_j - dd_j, 0\} \tag{5.31}$$

It is worth mentioning that backward passes can be performed not only for deadlines, but also for the soft-constrained due dates. The proceeding is analogue to deadlines (cf. equations 5.6 , 5.7 and 5.19). For the standard problem, **propagated due dates** pdd_j can be computed as

$$pdd_j = \min \{dd_j, dl_j, \min \{pdd_h - nd_h | h \in S_j\}\}; \text{ for } j = n, \ldots, 1. \tag{5.32}$$

Propagated due dates are not used as objectives, but rather influence the construction of initial solutions as will be discussed in Chapter 6.

5.7.4 Minimize maximum lateness

In addition to the total lateness also the maximum lateness is considered to be important. Including the minimization of the maximum lateness in the objective function attempts to avoid situations where a few customers experience extraordinarily huge delay. We can consider maximum lateness by minimizing

$$ml = \max \left\{ \max_{j \in J} (fd_j - dd_j), 0 \right\}. \tag{5.33}$$

Obviously, the maximum-lateness objective belongs to the class of regular performance measures, too.

5.7.5 Minimize total mode cost

In the multi-mode case 5.1–5.4 and 5.20–5.22, we can also consider the cost of running an activity in a specific mode in the objective function. Let c_{js} be the cost associated with running an activity in a specific mode. The total mode cost can be optimized by

$$\text{Minimize } tmc = \sum_{j \in J} c_{m_j}. \tag{5.34}$$

Mode costs facilitate the introduction of preferences about the loading of machines.

5.7.6 Minimize total setup time

Sequence-dependent setup times have a direct impact on the makespan of a project. To some degree, minimizing the makespan requires sequences of activities with small setup times. Past experience with the DS Optimizer indicates, however, that including setup times as a separate, additional objective can help to steer the search process in the right direction. Minimizing the total sum of setup times can be formulated as

$$\text{Minimize } tst = \sum_{j \in J_S} \sum_{i \in J \setminus J_S} \sigma_{ij} \cdot \mathbb{1}(i,j), \text{ where} \tag{5.35}$$

$$\mathbb{1}(i,j) = \begin{cases} 1 & \text{if } r_{m_j} = r_{m_i} \text{ and there is no activity } k \text{ with} \\ & \quad r_{m_j} = r_{m_i} = r_{m_k} \text{ and } s_j \leq s_k \leq s_i \\ 0 & \text{otherwise.} \end{cases} \tag{5.36}$$

Setup times do not belong to the class of regular performance measures. An example can easily be imagined where a global left shift of an activity leads to unfavorable setup-state transitions with increased setup times. If setup times are considered within the objective function, the optimal schedule may not lie within the set of all active schedules, but rather in the set of all semi-active schedules.

5.7.7 Minimize total setup cost

In addition, cost can be assigned for changing a resource's setup state:

$$\text{Minimize } tsc = \sum_{j \in J_S} \sum_{i \in J \setminus J_S} \psi_{ij} \cdot \mathbb{1}(i,j). \tag{5.37}$$

As with setup times, setup costs are not a regular performance measure.

5.7.8 Total objective function

Concluding this section, the overall goal is to minimize a weighted sum of different objectives without violating the bounds for pseudo-hard lateness. Mathematically, the weighted sum of low-priority objectives can be stated as follows.

$$twc = wms \cdot ms + wtl \cdot tl + wml \cdot ml + wtst \cdot tst + wtsc \cdot tsc + wtmc \cdot tmc \tag{5.38}$$

To compare two schedules, the following lexicographic sorting is used.

$$p \succ q \Leftrightarrow \left(thl^{(p)} < thl^{(q)}\right) \vee \left(thl^{(p)} = thl^{(q)} \wedge twc^{(p)} < twc^{(q)}\right) \tag{5.39}$$

As we will see, the DS Optimizer uses this relation extensively to rank different schedules that are consecutively constructed.

5.8 Further constraints and objectives

The DS Optimizer supports other constraints that are not explicitly treated here. They are just mentioned for the sake of completeness. In practice, activities may exist without any mode or resource assignment. Moreover, several activities may be fixed at certain positions. Also, another type of calendar, the so-called shift calendar, is supported where activities must start and end within the same shift. Furthermore, soft-constrained due dates can have different objective weights assigned: For example, one due date can be considered to be "twice as important" than another. With regard to the proposed coordination mechanism, we assume all due dates to be of equal importance in order to stay as generic as possible w.r.t. the underlying optimizer. Our hope is that the coordination mechanism is applicable to a larger set of optimizers as only a few requirements have been stated. The DS Optimizer also supports due dates and deadlines for the start date of an activity, which will be omitted too.

From a business perspective, the related **order** rather than the activity is relevant. Essentially, an order comprises several activities belonging to the same production (sub)process. For example, a setup activity cannot stand alone; it must come in connection with another nonsetup activity. Orders are closely linked to certain master-data structures, such as production process models, and are, for example, of importance for calculating lot sizes in previously executed Production Planning. Except for the alignment heuristics to be presented in Section 6.2, orders play a negligible role for the DS Optimizer, however.

5.9 State-of-the-art heuristics to solve the RCPSP

This aim of this section is to give an overview of the state-of-the-art heuristics that will be used to solve the RCPSP. Being an extension of the job-shop problem, the RCPSP belongs to the class of NP-hard problems (cf. Blazewicz et al., 1983). Searching for an optimal allocation of scarce resources usually involves two steps (cf. Herroelen et al., 1998): sequencing and scheduling. Sequencing concerns the ordering in which the activities have to be performed. Scheduling deals with the actual placement of the activities by defining which activities are to be performed at a particular time. For practical problem sizes and runtime requirements, heuristic approaches are indispensable and form the focus of this thesis (cf. Neumann et al., 2001, p. 110). For an overview of exact approaches, such as branch-and-bound schemes, the interested reader is referred to Manne (1960), Blazewicz et al. (1996), Herroelen et al. (1998), Brucker et al. (1999), Klein (2000), Brucker and Knust (2000), Neumann et al. (2001), and Damay et al. (2007). Comprehensive overviews of heuristic approaches are provided by Kolisch (1995), Kolisch and Hartmann (1999), Hartmann and Kolisch (2000), Klein (2000), Ponnambalam et al. (2001), Alcaraz and Maroto (2001), and Kolisch and Hartmann (2006). The following classification has been adopted from Hartmann and Kolisch (2000), Alcaraz and Maroto (2001), and Kolisch and Hartmann (2006).

5.9.1 Schedule-generation schemes

The basic building block of most heuristics is a so-called schedule-generation scheme (SGS). An SGS starts from scratch and builds a feasible schedule by iteratively extending a partial schedule. According to Hartmann and Kolisch (2000) and Kolisch and Hartmann (1999), the two main SGS types can be distinguished w.r.t. activity and time incrementation. **Serial SGS** perform activity incrementation while **parallel SGS** perform time incrementation.

5.9.1.1 Serial schedule-generation scheme

For the standard RCPSP 5.1–5.4, the serial SGS consists of $|J|$ stages in each of which an activity is selected from the **eligible set** and scheduled at the earliest precedence- and resource-feasible time. The eligible set comprises all unscheduled activities whose predecessors have already been completed. Usually, the eligible set contains several candidates and one activity needs to be chosen by some additional rule. In general, the serial SGS creates active schedules (Hartmann and Kolisch, 2000). Another variant is to presort all unscheduled activities once at the beginning (Hartmann, 1998; Engelmann, 1998; Alcaraz and Maroto, 2001; Jozefowska et al., 2001; Hartmann, 2002; Bouleimen and Lecocq, 2003; Debels and Vanhoucke, 2005), such that the sorted sequence of activities already respects the partial order implied by the precedence constraints. This encoding is also referred to as an **activity list**. Using an activity list supersedes the computation of the eligible set.

5.9.1.2 Parallel schedule-generation scheme

The parallel SGS does time incrementation. For each iteration there is a schedule time and a set of eligible activities. For parallel scheduling an activity is eligible if it can be precedence- and resource-feasibly started at the current schedule time. Activities are chosen from the eligible set and started at schedule time until there are no more eligible activities left. Afterwards, the schedule time is incremented. It has been shown by Kolisch (1996) that the parallel SGS produces non-delay schedules. Thus, for regular objective functions the optimal solution is not necessarily obtainable by parallel SGS.

5.9.2 X-Pass methods

X-Pass methods, also known as **priority rules**, employ one or both forms of SGS in order to construct one or more schedules. We distinguish between **single-pass methods**, which only construct one schedule, and **multipass methods** (cf. Hartmann, 1998). It should be emphasized that multipass methods usually do not consider knowledge of previously generated solutions when constructing a schedule.

A priority rule is employed to select one of the eligible candidates at each stage of the SGS. A function, $v(j)$, assigns a priority value to each eligible activity. Among all candidates, the one with the lowest value is chosen (as there is no convention, some authors prefer highest values). In the case of ties, one or several tie-breaking rules are employed. Typical priority rules are, for example, minimum processing time (the activity with shortest duration is chosen), maximum processing time, minimum latest finish date (the activity with earliest propagated due dates is chosen) or minimum slack time (the activity with smallest interval between earliest start date and propagated due date is prioritized).

Multipass methods combine single-pass methods in multiple ways. **Multipriority rules** derive several schedules by using different rules for the same SGS. Eventually the best schedule is finally chosen. **Sampling methods** generally make use of one SGS and one priority rule. Different schedules are constructed by biasing the selection of the priority rule through a random device.

For more details the interested reader is referred to Hartmann (1998), Kolisch (1996) or Kolisch (1995).

5.9.3 Metaheuristic approaches

Though multipass methods allow the rapid computation of a variety of schedules, they do not support the learning of effective priority values for the problem at hand. This is the point where metaheuristics come into play. Due to the combinatorial complexity of the RCPSP, a direct encoding, a vector of start dates, cannot be recommended, since the resulting schedules would be highly infeasible. Instead, metaheuristics are used as learning strategies to calibrate the parameters of the underlying heuristics. With regard to the RCPSP, a very common approach is to encode the sequence of activities in form of permutations or probabilities and to construct the related schedule by a serial or parallel SGS. Initialized by standard priority rules, the aim of the metaheuristic is then to find the best suited permutation or probability distribution for the problem at hand. It should be noted that serial or parallel SGS is generally not a bijective function[4], i.e. several encodings are usually mapped to the same schedule. The following encodings dominate the present literature.

5.9.3.1 Permutation encoding

Mattfeld (1995, p. 74) gives an overview of common encodings in the job-shop literature. For his own computational analysis, he encodes the priorities of tasks in a permutation, whereas a job comprises a set of tasks. To ensure feasibility w.r.t. precedence constraints between the tasks, not the task-number but the job number is used repetitively. For each permutation entry the first unscheduled task of the indicated job is chosen to be scheduled within a serial SGS. If the set of eligible candidates comprises several tasks, the task belonging to the most prioritized job is chosen. More details on this "permutation with repetition" encoding can also be found in Bierwith (1995). In Mattfeld and Bierwith (2004), two variants of a permutation-based Genetic Algorithm (GA) are compared, where the permutations are encoded by either a serial or a parallel SGS to active or nondelay schedules, respectively. While the set of active schedules always contains an optimal schedule for regular performance measures, the set of nondelay schedules is usually smaller and can be explored more quickly. Regarding the RCPSP, Kim et al. (2003) use a permutation encoding and the serial SGS to construct RCPSP schedules in their GA.

5.9.3.2 Activity-list encoding

As already discussed in Section 5.9.1.1, an alternative to permutation encoding is a presorted list of activities. Presorted lists have the advantage that activities can be directly scheduled one after the other, without the need for computing the eligible set. Using activity lists, Bouleimen and Lecocq (2003) tackle the RCPSP with a simulated annealing method. They define the neighborhood of an activity list as the set of perturbations reachable by insert operations. An insert operation selects an activity from the list and inserts it at another position between the related preceding and succeeding activities. This guarantees that the altered activity list does not violate

[4] In mathematics, a bijective function is a function f that maps a set X to a set Y with the property that, for every $y \in Y$, there is *exactly one* $x \in X$ such that $f(x) = y$. A function f is bijective if and only if its inverse relation f^{-1} is a function, too.

any precedence constraints. For the multi-mode version of the RCPSP, a second encoding is introduced, storing the selected mode for each activity. The neighborhood is then defined by an insert operation on the activity list and a random change of a selected mode. Once the modes have been selected, a serial SGS is used as decoding function. A similar representation is used by Jozefowska et al. (2001). A GA to tackle the multi-mode RCPSP is also presented by Hartmann (2001).

For the standard RCPSP, Alcaraz and Maroto (2001) present a GA that decodes an activity list by serial scheduling. The focus of their paper is on suitable crossover operators. Also Hindi et al. (2002) describe a GA relying on activity-list encoding and a serial SGS. Hartmann (2002) not only encodes the sequence of activities, but also the decision whether to use parallel or serial SGS as the decoding procedure. While a parallel SGS usually leads to good solutions at the beginning of an optimization run, it has a weak performance towards the end of the run since optimal or close-to-optimal schedules may not be contained in the set of nondelay schedules. However, it is usually impossible to predict which SGS will perform better for an arbitrary instance of the RCPSP. According to the author, the self-adapting GA outperforms other state-of-the-art metaheuristics.

Debels and Vanhoucke (2005) apply a forward and backward serial SGS to obtain left- and right-aligned schedules, cf. Section 5.9.4. So-called backward serial scheduling constructs a right-aligned schedule by placing the activities *as late as possible*, starting with the rightmost activity of the list. Using two populations—one being decoded by forward serial SGS to left-aligned schedules, the other being decoded by backward serial SGS to right-aligned schedules and crossover operators with the ability to merge parents of the two different populations to new children—the authors search for minimum-makespan schedules.

The above approaches restrict neighborhood moves and mutations in such a way, that the precedence relation between entries of an activity list is not violated. Thus, within the current list, an activity is never shifted before its predecessors or behind its successors. This restricts the neighborhood in a nondeterministic and unpleasant way. If, for example, an activity is initially positioned towards the right end of the list it prunes the possibility of left moves of all its direct and indirect successors. To be able to shift the successors to the left, the obstructing activity has to be removed first. However, shifting only the obstructing activity and not its successors might worsen the fitness, such that this direction is not explored further or with only low probability. Thus, local optima are artificially introduced by the above neighborhood definition. To overcome the above difficulties, Fleszar and Hindi (2004) employ another neighborhood definition in a hill-climbing algorithm using a serial SGS. Neighborhood moves are obtained by shifting one list entry, j, to another position. The activity list is subsequently repaired by shifting all predecessors and successors of j until precedence constraints are obeyed.

Being an GA, also the DS Optimizer relies on an activity-list representation, see also Engelmann (1998). The algorithm impresses through its variety of genetic operators and its applicability to a wide range of practical problems, including max-links, multi-modes, setup times and calendars and combines several of the concepts mentioned in this section. A detailed discussion is presented in Chapter 6.

5.9.3.3 Random-key representation

Other representations encode the decision to select an activity of the eligible set directly. Such random-key representations assign a number to each activity of the project. Again, both the serial and parallel SGS can be used to construct the related schedule, whereas the activity with highest priority is taken from the eligible set. In contrast to the activity-list representation, every random-key assignment leads to a feasible schedule. Leon and Ramamoorthy (1995) use a parallel SGS to construct schedules within a local search algorithm and test an EA as well. Naphade et al. (1997) use a so-called problem-space representation. Basically, they use the min-slack priority rule to decide which activity to select from the eligible set. The slack is computed as the difference between latest (static) finish date and the earliest start date (according to the partial schedule): $lf_j - \max_{k \in P_j} fd_k$ (not considering min-links). Instead of priority values, the authors encode perturbations of the latest finish dates and apply a local search heuristic. Parallel or serial scheduling is randomly applied as a decoding function. Valls et al. (2003) employ tabu-search heuristics based on a topological-order representation. Here, the numbers assigned to activities respect the topological order implied by the precedence constraints, which is an effective strategy to reduce the size of the search space. Simulated Annealing based on a random-key representation and a serial SGS is presented by Cho and Kim (1997). To solve the familiar multiproject RCPSP, Goncalves et al. (2007) use a parametrized SGS: A maximum allowed delay time assigned to each activity determines the degree between nondelay and active scheduling. To allow for self-adaptation, the maximum allowed delay times are also encoded with the priorities.

5.9.3.4 Priority-rule representation

In the priority-rule representation, a solution is represented by a vector with as many positions as activities in the project. Each position is occupied by a rule from a given set of priority rules. The i-th rule of the vector is used to decide which activity to schedule at stage i of a serial or parallel SGS. Hartmann (2002) uses a serial SGS in combination with a GA based on a priority-rule representation. More examples are listed by Alcaraz and Maroto (2001).

5.9.3.5 Other representations

Other metaheuristics also rely on serial or parallel SGS. For example, Merkle et al. (2002) developed two ant-colony optimization algorithms, based on the serial and parallel SGS, respectively. At each stage of the SGS, the activity to be scheduled is randomly selected from the eligible set on the basis of probabilities stored in a so-called pheromone matrix. The pheromone values are iteratively updated based on the fitness of previous completed schedules.

5.9.4 Improvement by right-left-alignment

An interesting technique to improve feasible schedules is a right–left-alignment. We define a **right-aligned** schedule as a schedule where no activity can be finished later without advancing some other activity, or violating the constraints, or increasing the objective function. **Right-alignment** considers activities from right to left and shifts each activity to the latest feasible date (global right shift) that does not lead to an increase of the objective function. Vice versa, a **left-aligned** schedule is a schedule where no activity can be started earlier without delaying another activity, or violating the constraints, or increasing the objective function. **Left-alignment** considers activities from left to right and shifts each activity to its earliest feasible date (global left shift). Section 6.2 discusses in detail the related heuristics of the SAP DS Optimizer. We speak of **right-**

left-alignment if first a right-alignment is applied to a schedule, followed by a left-alignment.[5] By definition, a right-left-alignment cannot deteriorate the objective value. However, applying right-left-alignments iteratively can lead to more compact schedules and hence better objective values for many performance measures. Li and Willis (1992) explain the improvements on the later schedules by an analogy:

> Suppose we have a box of wooden blocks of various lengths and sizes. If we turn the box over a few times, the movement will shift some blocks into alignment with each other and consequently, the overall volume occupied by all blocks will become smaller. (p. 275)

The authors developed a heuristic that iteratively reschedules an initial schedule forward and backward until no further improvement is possible. Instead of a right-alignment based on a series of global shifts, the authors apply a backward serial SGS, where priorities are determined by the sequence of the activities of the initial schedule. The resulting schedule may be infeasible as it reaches into the past. A subsequent left-aligned schedule is derived by a serial forward SGS with priorities based on the backward schedule and so on.

Valls et al. (2005) considered the right-left-alignment technique as a local-search strategy for different metaheuristics and were able to show improvements over the standard decoding procedure of applying a serial SGS. Tormos and Lova (2001) combine a sampling method and a serial SGS with a subsequent right-left-alignment. In Tormos and Lova (2003), the authors enhance this approach. Right-left-alignment is only applied if the schedule constructed by sampling is better than the average of solutions generated by sampling so far. Applying a so-called local-constraint based analysis, Özdamar and Gündüz (1996) use the dynamic time windows given by a forward SGS to select activities of the eligible set in the subsequent backward SGS and vice versa, which is slightly different from the right-left-alignment defined by Valls et al. (2005). Also Klein (1998) combines forward and backward scheduling to increase the performance of standard priority rules.

We do not claim the above selection of papers to be complete. Our primary intention was to show that many researchers consider metaheuristics based on serial or parallel SGS, as well as alignment techniques, to be efficient methods to tackle the RCPSP. For a more detailed review and numerical comparisons, the interested reader is referred to the works of Hartmann and Kolisch (2000) and Kolisch and Hartmann (2006).

[5] This technique is also called forward-backward improvement, cf. Kolisch and Hartmann (2006) or double justification, cf. Valls et al. (2005). However, in this thesis we will stick to the term right-left-alignment to be able to better differentiate from related, but different techniques presented later.

CHAPTER 6

The SAP Detailed Scheduling Optimizer

The SAP Detailed Scheduling (DS) Optimizer is a tool for computing detailed schedules on an operational level. Assuming an integrated distribution and production planning scenario (cf. Dickersbach, 2003, Chapter 6), a rough-cut quantity-based plan is calculated first on a Master Planning level using the SAP Supply Network Planning (SNP) Optimizer, cf. Chapter 2. Essentially, the SNP Optimizer calculates resource allocation, product mix and distribution for several production plants. Based on this first allocation, production plans for each plant are calculated during a subsequent SAP Production Planning (PP) run. SAP Production Planning extends the common Material–Requirements–Planning (MRP). Briefly summarized, it provides the following functionality. First, several heuristics are available to calculate lot sizes, ranging from simple lot-for-lot strategies to heuristics supposed to find a good trade-off between fixed and variable costs for lots. Producing a lot triggers a production activity. Second, PP creates the so-called pegging-net, where each activity is related to activities producing needed input products. Third, the activities can be roughly scheduled by some further heuristics. Obviously, it is a very hard task to compute the pegging-net and the lot sizes and to schedule the resulting activities simultaneously. A strategy to reduce complexity is to freeze the pegging-net after the PP run. The task is then to find a satisfying sequence of the activities on the resources without changing activity-precedence relations or violating available capacity—this combinatorial problem belongs to the class of RCPSPs.

The DS Optimizer has been developed to solve large-scale RCPSP instances and relies on a generic model formulation supporting several RCPSP-related extensions, as described in Chapter 5. The DS Optimizer itself comprises several modules, as depicted in Figure 6.1. The connecting entity is the so-called **context**, an accumulation of C++ objects that represents an RCPSP instance and its solution (assigned modes, start and finish dates) in the memory of the computer. The context is created by a model generator, which either uses data from the application database, or—in case of a stand-alone usage of the DS Optimizer—uses text files or hard-coded C++ classes to generate a context. The model generator not only supports the "downloading" of a problem, but also allows "uploading" a solved context into the database, or writing the results in text files.

Once a context has been generated it can be passed to several modules. The solving module is responsible for deriving a solution and modifying the relevant objects of the context accordingly,

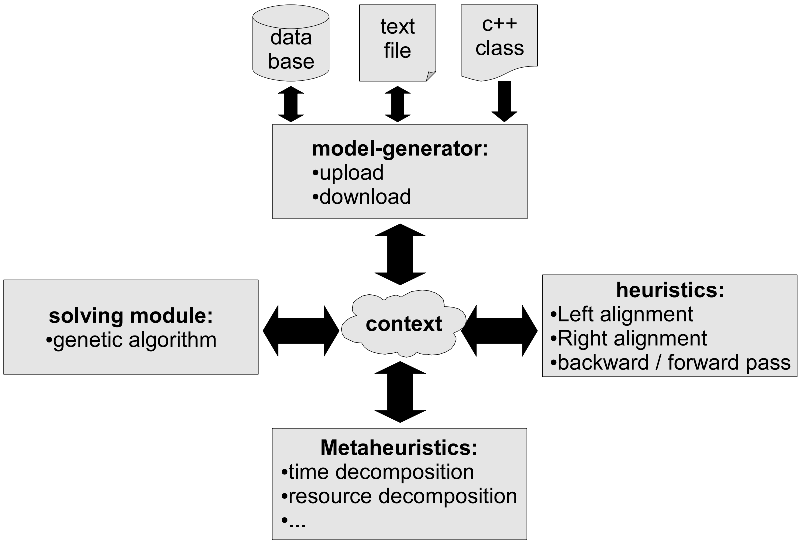


Figure 6.1: Main modules of the DS Optimizer.

such as setting the start and release dates or selecting the modes. A Genetic Algorithm (GA) using a serial SGS as decoding function proved most successful and is nowadays used as the standard. The algorithm is described in detail in the following sections. Moreover, several heuristics exist to modify a context, for example by calculating forward or backward passes. Other important heuristics allow the shifting of activities left or right, similar to the right- and left-alignment procedures mentioned in Section 5.9.

Last but not least, several top-level metaheuristics complete the portfolio of tools. These metaheuristics allow the extraction, modification and backinsertion of subcontexts out of and into the original context. For example, there exists a time decomposition that sequentially improves a large context (representing an already feasible solution) by extracting subcontexts of activities currently located in a certain time window. Each subcontext is then separately passed to the solving module which tries to improve the isolated problem without violating precedence and capacity constraints at the boundary to the remaining (global) context. The overall solution is improved by letting the time window iteratively glide over the set of all activities. As the problem decomposes this way, the solution quality usually degrades, but the time for finding a solution can be quasi-linearized. By adjusting the size of the time window, we can calibrate the trade-off between solution quality and convergence speed. Such metaheuristics proved to be an efficient means to efficiently tackle large-scale scheduling problems. However, since they are not used for our computational tests, these top-level metaheuristics won't be explained in detail.

6.1 The Detailed Scheduling solving module

When it comes to the task of scheduling many activities, methods relying on a serial SGS are commonly used to construct good schedules in a limited amount of time, as already discussed in Section 5.9. With respect to capacity constraints, the generated schedules are always feasible (for reasonable capacity profiles). Due dates might be violated, however. In a practical environment such plans containing delayed activities are preferred over plans that violate capacity constraints. The GA underlying the DS Solving module relies on an activity-list representation, cf. Section 5.9. The activity list is then decoded by a serial SGS. However, max-links and setup times require a more sophisticated SGS than those described in Section 5.9. This decoding procedure as well as mutation and crossover operators are the topic of the following subsections. It is worth mentioning that the solving module is applied after forward and backward passes computed earliest and latest start and finish dates es_j, ls_j, ef_j and lf_j.

6.1.1 Encoding and decoding

In the current implementation, a solution is represented as follows. First, an activity list π determines the sequence of activities to be serially scheduled. Second, a string of integer numbers suggests which mode to select for each activity. However, the mode suggestion is not definitive. In fact, an activity might not be "scheduleable" in a suggested mode, if the mode violates latest finish dates or capacity restrictions. In such cases, the best possible mode is selected by a greedy heuristic described later. Moreover, it is possible that the encoding suggests no mode at all, which immediately triggers the greedy heuristic.

Activity lists always represent a feasible order w.r.t. to precedence constraints. Put mathematically,

$$\forall\, k \in J : \pi^{-1}(j) < \pi^{-1}(k) \Leftrightarrow j \in P_k, \tag{6.1}$$

where π^{-1} denotes the inverse list, storing the position of each activity within π. Working with partially ordered lists has two advantages. On the one hand, we can immediately decode activity after activity without needing to check if all activity predecessors have already been scheduled. On the other hand, the actual sequence of activities in the schedule is likely to be closer to the sequence defined by the list, as it would be to the one defined by a permutation, not respecting the precedence constraints. Hence, activity lists enable the mutation and crossover operators to act more "purposefully." As will become apparent later, an activity list is arbitrarily sorted after initialization or mutation. There are two ways implemented for "repairing" a list.

- Establish the activity precedence by pulling "too late" activities "earlier," where the notions "early" and "late" in this regard refer to the position within the list π. The list is re-sorted by Function `establishPrecedenceByPullFwd` on Page 90. The function uses a set U of eligible activities whose successors are already in correct order. Initially, the set U consists of activities that don't have any successors (Line 1). Additionally a set S of already considered activities is used (Line 2). Iteratively, the repaired permutation $\pi_{repaired}$ is filled up from right to left (Line 3). At every iteration, the activity $j^* \in U$ that is positioned most right in the current permutation π is taken (Line 4) and inserted into the repaired permutation $\pi_{repaired}$ (Line 5). The sets S and U are updated accordingly (lines 6 and 7). Afterwards, those predecessors of j^* whose successors are already included in the repaired permutation are added to U (lines 8-12). Hence, activities that are not contained in U are pulled forward in the repaired permutation.

- Establish the activity precedence by pushing "too early" activities "later." Here the list is re-sorted as described by Function `establishPrecedenceByPushBwd`. This function works analogously to the above function, but in the other way round.

Many mutation operations shift activities only in one direction, which requires exactly one of the repair procedures to be applied. Otherwise the mutation might be reversed. If there is

```
Function establishPrecedenceByPullFwd($\pi$)
  input   : Permutation $\pi$ to be repaired
  output  : Returns repaired permutation $\pi_{repaired}$
  $U := \{j \in J | S_j = \emptyset\}$
  $S := \emptyset$
  for $x := |\pi|$ to $1$ do
      Compute $j^*$ with $\pi^{-1}(j^*) > \pi^{-1}(j)\ \forall j \in U$
      $\pi_{repaired}(x) := j^*$
      $S := S \cup \{j^*\}$
      $U := U \backslash \{j^*\}$
      forall $i \in P_{j^*}$ do
          if $S_i \subseteq S$ then
              $U := U \cup \{i\}$
          end
      end
  end
  return $\pi_{repaired}$
```

```
Function establishPrecedenceByPushBwd($\pi$)
  input   : Permutation $\pi$ to be repaired
  output  : Returns repaired permutation $\pi_{repaired}$
  $U := \{j \in J | P_j = \emptyset\}$
  $S := \emptyset$
  for $x := 1$ to $|\pi|$ do
      Compute $j^*$ with $\pi^{-1}(j^*) < \pi^{-1}(j)\ \forall j \in U$
      $\pi_{repaired}(x) := j^*$
      $S := S \cup \{j^*\}$
      $U := U \backslash \{j^*\}$
      forall $i \in S_{j^*}$ do
          if $P_i \subseteq S$ then
              $U := U \cup \{i\}$
          end
      end
  end
  return $\pi_{repaired}$
```

no clear direction of mutation, one of the two repair procedures is chosen randomly.[1] As already mentioned in Section 5.9, allowing arbitrary mutations followed by a subsequent re-sorting allows larger moves within the neighborhood.

6.1.1.1 Scheduling a single activity for a given mode

When building an infinite plan, i.e. without considering capacity constraints, the scheduling of a single activity is straightforward. In case of minimum finish–start links, the start date of an activity j can be calculated by $sd_j = \max_{k \in P_j} fd_k + \lambda_{kj}^{FS}$. As capacity constraints do not hold, the

[1] For generating random variables, a random number generator needs to be implemented. The computer as a deterministic machine can not generate truly random values, but uses a certain function to generate *pseudo random numbers*. For the DS solving module and prototypical implementation of the DEAL framework, the so-called Park-Miller random number generator has been used. Here, a random number x_t is computed as $x_t = x_{t-1} \cdot g \mod n$, with $n = 2^{31} - 1$ and $g = 16807$. The number x_{t-1} is called the random seed and x_0 is the initial random seed. Drawing a random number $r_t \in [0.0, 1.0)$ is simulated by computing $r_t = \frac{x_t}{n}$. It should be emphasized that for the same initial random seed, the same "probabilistic" choices are computed.

calculation equals the forward pass mentioned in Section 5.1. For other types of min-links, we can proceed analogously as discussed in Section 5.3. Hence, the time complexity of scheduling such a simple problem is $\mathcal{O}(n)$, for n activities to be scheduled (and if we assume the numbers of predecessors to be independent of n).

If a problem with infinite capacity contains max-links, the time complexity is a polynomial function of the number of activities, since one violated max-link can require a recalculation of all activity predecessors (cf. Neumann et al., 2001, p. 214).

If capacity is limited, not only does the earliest feasible start date according to precedence constraints need to be computed, but also a feasible slot with sufficient capacity on primary and secondary resources needs to be searched. For building active schedules *not* containing max-links or sequence-dependent setup activities the slot-finding procedure for activity j works as follows: Starting at the earliest time, $t_{start} = es_j$, the next date with sufficient capacity on the primary resource is searched. Having found such a date, we check if the slot length is sufficient, both on primary and secondary resources. To calculate the required slot length, several factors are of importance—the activity's net-duration, the productivity and the break calendar—which eventually lead to the activity's gross-duration. For a feasible slot, the available capacity must not drop below the required resource usage during the gross-duration. Otherwise, t_{start} is set to the end of the current, infeasible slot and the next slot is searched. It is clear that the time complexity of finding a slot depends on the initial capacity profiles, the number of secondary resources, and the number of gaps at the beginning of the schedule that are of insufficient length.

For determining the net-duration of a setup activity, the current setup state of the primary resource is of importance, cf. Equation 5.26. In turn, the current setup state is defined by the activity scheduled before the current slot. Henceforth, we will denote activities before and after the current slot as **machine predecessors** and **machine successors**. It is important to highlight the difference to the sets of immediate predecessors P_j and successors S_j of an activity j. While the latter sets are part of the problem formulation (cf. Equation 5.2), do not relate to resource assignments and are independent of the generated schedule, machine successors and predecessors relate to a particular activity (or a slot of interest) with regard to a particular resource and a (partially) generated schedule.

The above method yields active schedules. If setup times or costs are considered as non-regular performance measures, the optimal schedule is not necessarily contained within the set of all active schedules, as discussed in Section 5.7. Under such circumstances, an SGS yielding semi-active schedules might be preferred. To calculate semi-active schedules it is sufficient to initially set t_{start} to the largest finish time of already scheduled activities. As setup times or costs only occur on primary resources, only those activities of the selected mode's primary resource need to be considered. Furthermore, the calculation of t_{start} needs to be modified only for setup activities.

6.1.1.2 Greedy heuristic for mode selection

As already mentioned, the encoding does not necessarily suggest a mode for every activity. And even if a mode was suggested it might not be "scheduleable," if, for example, no slot was found before the latest finish date. In both cases, the most promising mode is then selected by a greedy heuristic that compares the results of scheduling the activity in each mode, respectively. During the heuristic run, the slot finding is not restricted to slots before the latest finish date, instead, the violation is penalized. From a practical perspective, finding a slot behind this date is preferred to

Abbreviation	Priority	Weight	Description
hl_m	1	n. a.	violation of deadlines
ms_m	2	wms	estimated makespan
tl_m	2	wtl	total estimated lateness increase
ml_m	2	wml	maximum estimated lateness
st_m	2	$wtst$	estimated setup time increase
sc_m	2	$wtsc$	estimated setup cost increase
mc_m	2	$wtmc$	minimum subsequent mode cost
sl_m	2	0.01	slack consumption

Table 6.1: Figure, priorities and weights for comparing possible modes.

not finding a slot at all. Eventually the schedules are compared by a relation and the best mode is selected. The relation differs from the standard relation 5.39 and takes into account the current and subsequent consequences of the mode selection. Because of mode-compatibility constraints, a selected mode can determine settings for succeeding activities and possibly have a major impact on the overall objective function. The figures relevant for the comparison and the related priorities and weights are listed in Table 6.1.

Let j denote the job to be scheduled and S the set of already scheduled activities. The deadline-violation calculation is straightforward: $hl_m = \max\{fd_j - lf_j, sd_j - ls_j, 0\}$. The estimated increases of setup time and cost, st_m and sc_m, are calculated as the difference when comparing the current partial schedule before and after the activity of interest was scheduled in mode m. The figures are an estimate in the sense that—when building active schedules—subsequent planning steps can schedule activities before j and change the setup state of the resource such that the net-duration of j changes (if $j \in J_S$), see also Subsection 6.1.1.4. The increase of total estimated lateness is calculated as the difference between finish date and propagated due date, $tl_m = \max\{fd_j - pdd_j, 0\}$. The maximum estimated lateness is set to

$$ml_m = \max_{k \in S \cup \{j\}} \left(\max\{fd_k - pdd_k, 0\}\right).$$

The estimated makespan is calculated as

$$ms_m = \max_{k \in S \cup \{j\}} \left(fd_k + ttf_{km}\right) - \max_{k \in S \cup \{j\}} \left(sd_k\right),$$

where ttf_{km} stands for "time to finish" and denotes the minimum sum of processing times of the direct and indirect successors of k, under infinite capacity consideration, but respecting mode-compatibility constraints for mode m. When analyzing mode-compatibility constraints, the minimum sum of mode costs of j and its direct and indirect successors is computed for the chosen mode, m. "Slack consumption" is another decision criteria. The number is computed as

$$sl_j = \begin{cases} 1, \text{ if } fd_j \geq pdd_j \\ 0, \text{ if } pdd_j - es_j = 0 \text{ or } tss_j + ttf_j + fd_j - sd_j = 0 \\ \frac{ttf_{jm} j}{tss_{jm} + fd_j - sd_j + ttf_{jm}} - \frac{pdd_j - fd_j}{pdd_j - es_j}, \text{ otherwise,} \end{cases}$$

where tss_j denotes "time since start" of activity j. Analogously to "time to finish," "time since start" is a static number expressing the sum of minimum durations of the direct and indirect predecessors of j under infinite capacity considerations but respecting mode-compatibility constraints. The term

$$\frac{ttf_{jm}}{tss_{jm} + fd_j - sd_j + ttf_{jm}}$$

approximates the relative remaining work, whereas

$$\frac{pdd_j - fd_j}{pdd_j - es_j}$$

expresses the relative free slack if j is scheduled in the mode currently investigated. The "slack consumption" can simply be regarded as a weighted sum of two criteria: A mode leading to short remaining work should be preferred, as should a mode with a large slack w.r.t. the propagated due date.

6.1.1.3 Aggregating activities to blocks

Decoding an activity list to a schedule by a serial SGS for problems containing max-links leads to several difficulties. Already the task of testing whether there is a feasible schedule for a given RCPSP instance is NP-complete (cf. Neumann et al., 2001). When applying a serial or parallel SGS, we cannot guarantee that a max-link is preserved until both the preceding and succeeding activities have been scheduled. If a violation occurs, already scheduled activities need to be rescheduled until all max-links are obeyed. Rescheduling consists of deallocating some activities and scheduling them in a different way. However, deallocating the preceding activity of a max-link and scheduling it later can lead to further max-link violations between the activity and its predecessors. In the worst case, a chain reaction requires all activities of the already existing schedule to be planned again. In general, we have to decide whether to deallocate an activity itself or its predecessor to fulfill a max-link. Deallocating a predecessor can lead to incorrect schedules since no feasible later slot with sufficient capacity might be found to fulfill the max-link. Also the setup time of the predecessor's machine successor (cf. Subsection 6.1.1.1 for the definition of the term machine successor) might increase, such that the remaining schedule might not be feasible, even without considering the preceding activity. The only activity that can be deallocated safely is the latest scheduled activity since we know that the previous schedule is already correct. Hence, in case of a violated max-link, we need to deallocate all activities in reverse order, back to the violating predecessor, before applying a repair algorithm.

If max-links occur only rarely, it makes sense to build blocks of the related activities and schedule them "together" to keep the time expenditure for rescheduling small instead of allowing arbitrary activity sequences. The above ideas can be summarized to a "block concept" as follows:

- **Block building:** Single activities connected by max-links are aggregated to blocks. The precedence constraints of activities outside and inside of a block remain unchanged. We define a block's predecessors (successors) as all activities (possibly contained by other blocks) outside the block that precede (succeed) activities within the block. Now, by summarizing activities to blocks, it might happen that an activity outside the block becomes both, predecessor and successor of the same block. This ambiguity cannot be handled by serial scheduling. A solution can be developed by illustrating the problem as a directed graph, where each node refers to an activity or block, respectively, and each min-link is mapped to a directed edge from predecessor to successor. Now, for each max-link, an additional directed edge from *successor to predecessor* is introduced. For more details on such "activity-on-node networks" Neumann et al. (2001) is recommended.

If the graph contains cycles, a topological sorting is not possible. That is, serial scheduling will fail for the current block structure. This fact can be used to build feasible blocks. Each cycle of a directed graph is contained by a maximal connecting component. Between the distinct maximal connecting components only acyclic relations exist. Hence, enlarging blocks to maximal connecting components leads to a block structure that can be handled by serial scheduling. Blocks built by this method also have minimal size since splitting a block up would immediately lead to a cycle (cf. Engelmann, 1998). The building of blocks can be done once during a preprocessing step.

The aggregation of blocks applies only to the activity-list part of the encoding. The remaining part of the encoding, i.e. the string of integer numbers suggesting a mode for each activity, is not aggregated, since the activities of a block can still be scheduled in different modes.

- **Block scheduling:** Having calculated blocks, there is no need for list entries to further refer to single activities; instead, they refer only to the blocks themselves, decreasing the size of the list. The serial SGS now subsequently schedules blocks instead of single activities. Within a block, the activities are (arbitrarily) topologically sorted according to activity precedence and are scheduled by a dynamic programming procedure:

    1. Schedule the next activity of the block's remaining unscheduled activities by finding a feasible slot for the suggested mode. If no mode was suggested or no slot was found on the primary and secondary resources of the suggested mode, use the greedy heuristic to determine the most promising slot. Terminate if all activities of the block have been scheduled.
    2. Check if a max-link to any predecessor (located inside the block) is violated. If there was no violation go to step one and continue with the next activity.
    3. Deallocate all activities that violate any max-link in reverse order until the earliest predecessor belonging to a violated max-link.
    4. Increase the earliest feasible start date of this predecessor, such that the max-link is not violated.
    5. All affected activities need to be scheduled again, continue with step 1.

A rigorous algorithmic formulation of the above procedure can be found in Neumann et al. (2001, Section 2.6.4). For the sake of completeness, it should be mentioned that, in the current implementation, steps 3 and 4 of the above procedure are enriched by additional forward passes to detect violations of precedence constraints as early as possible.

The block-building concept has its limitations. If a problem instance contains many max-links, many and/or large blocks are built. In these cases, the dynamic programming method leads to backtracking with exponential time complexity. With respect to optimality, it should also be mentioned that the building of blocks can obscure the global optimum. For each block, a certain activity sequence is fixed during the preprocessing. However, to reach the optimum, another sequence might actually be necessary. Hence, for each possible sequence of every block, the dynamic programming procedure would actually need to be called, further increasing the time complexity.

For practical problems not containing too many max-links, the block concept proved to be a suitable approach. It guarantees that all generated schedules are feasible with respect to the max-links. As already mentioned, searching for maximally connected components also leads to minimally sized blocks. In most practical cases containing only a limited number of max-links, the deterioration of the objective value will be small as only a few small blocks are generated.

6.1.1.4 Scheduling of setup activities

Serially scheduling sequence-dependent setup activities comes with several difficulties. First, on multicapacitated resources, we would need to determine a setup activity's machine predecessor to calculate the required setup state transition. This feature is not supported because a single setup state for such a resource would be very restrictive. Alternatively, the planner can split up multicapacity resources into several unary resources, and introduce alternative modes for the related activities.

Second, between a setup activity and its successor no other activity must be scheduled as stated by Equation 5.26. However, applying a serial SGS in a straightforward manner can lead to gaps between setup activities and their successors due to other precedence constraints. As the gaps cannot be used by the list's subsequent activities, the idle time would be totally wasted. Thus, we need to assure that the two activities are scheduled as close together as possible in order to avoid a waste of resource capacity. As for max-links, we can aggregate each setup activity and its successor to a single block. If idle times occur between the two activities or setup constraints of type 5.26 are violated, we reschedule the setup activity, increase the start date and reschedule the block.[2]

Third, if an active scheduling policy is pursued, activities may be placed before already scheduled setup activities. Thus, a different setup-state transition might be required for the *already scheduled* setup activity following on the same resource. That is, its duration needs to be recomputed. The slot-finding procedure needs to be extended in this regard. If a promising slot has been found, the duration of the machine successor needs to be recomputed, which in turn alters the slot's length. If the slot is of insufficient capacity, this correction needs to be reversed before the next slot is sought.

Fourth, in the case of setup activities, the optimal schedule is not necessarily contained in the set of all active activities. As described above, a semi-active planning is possible, but at the expense of an increased solution space. As a rule of thumb, a semi-active scheduling policy is applied if the sum of weights for setup times and setup costs exceeds the sum of the remaining (regular) performance measures.

6.1.1.5 Serial scheduling example

For simplicity we refer to a list containing blocks and activities simply as the block list henceforth. A block might contain one or several activities to be scheduled. To explicate the serial scheduling procedure, we present a concrete example consisting of ten activities to be scheduled. The calendar is shown in Figure 6.2 and the activities to schedule in Figure 6.3. The activities are numbered $j = 1, 2 \ldots, 10$. For better legibility, additional letters indicate the type of an activity, whereas "J"

[2] Actually, the rescheduling of activities on primary resources is a simplified description of the current implementation. To avoid idle times and find suitable slots, several computations are performed to restrict the interval of earliest start and latest finish dates and to estimate the required slot length. These techniques contribute by saving runtime when building a schedule. However, a detailed description is beyond the scope of this thesis.

denotes a standard activity, "MJ" denotes a standard activity with multiple modes (i.e., relating to alternative machines) and "S" refers to setup activities. The activities are to be placed on two resources, R1 and R2. For simplicity, a mode utilizes only one resource. Black boxes in the calendar denote breaks. Dashed lines between two activities represent an activity precedence constraint, for example J4 is predecessor of J6. As mentioned previously, activities and their setup activities are aggregated to blocks to facilitate scheduling. Max-links do not occur in the example, giving seven blocks, B1 to B7. As can be seen B1, B4 and B6 consist of a setup activity and a standard activity, whereas the other blocks each consist of a single standard or multimode activity, respectively.

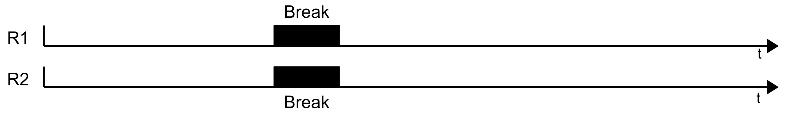


Figure 6.2: Serial scheduling example—calendar.

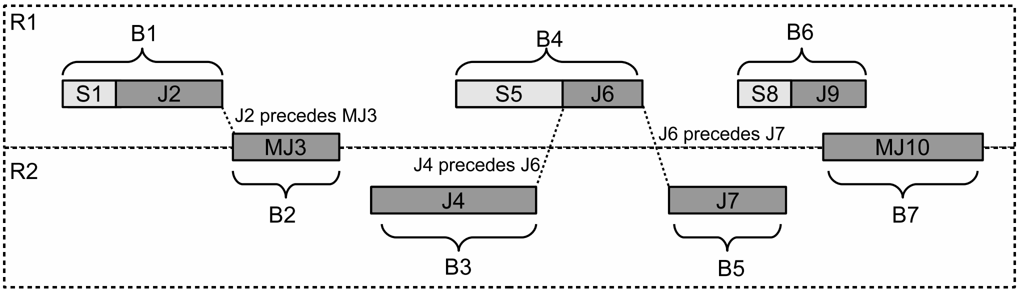


Figure 6.3: Serial scheduling example—activities and precedence constraints.

The length of a setup activity depends on the setup state the resource is currently running on and on the required state of the related standard activity following the setup activity. In the example, we assume that each activity requires a different state. The underlying setup matrix is illustrated in Figure 6.4.

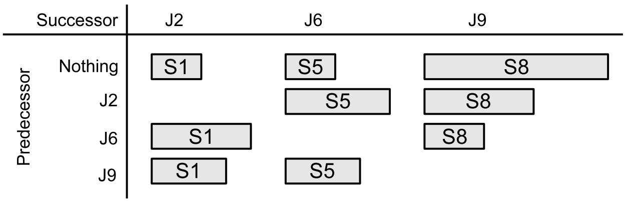


Figure 6.4: Serial scheduling example—underlying setup matrix.

In the following, we describe step by step the scheduling of a concrete block list that implies the following order: B3 → B4 →B1 →B2 →B6 →B5 →B7. Note that this order does not violate any activity-precedence constraints. That is, the repair procedures of Page 90 have already been applied. We further assume that the mode of MJ3 is set to resource R2 and that for MJ10 no mode has been selected yet. All other modes are set to the resources indicated by Figure 6.3. Figure 6.5 shows the result of scheduling the list's first block, B3. The block consists of just one activity (J4) to be scheduled on R2. Since no other activity has been planned yet, J4 is scheduled as early as possible (leftmost in the picture).

Next, the block B4 is scheduled as shown in Figure 6.6. Since B4 consists of two activities, S5 and J6, the dynamic programming approach is applied. The setup activity, S5, is scheduled first

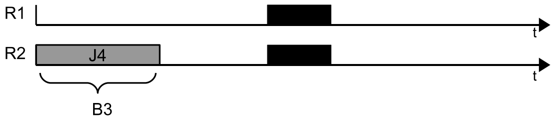
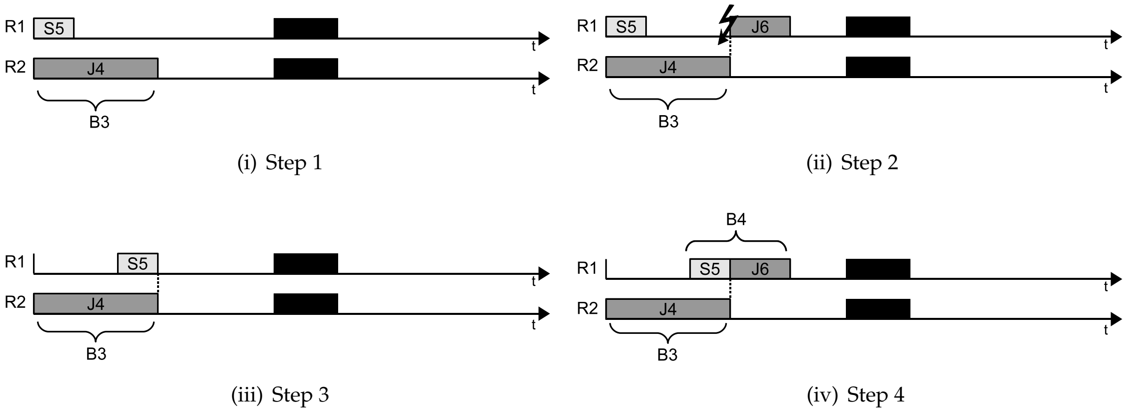


Figure 6.5: Serial scheduling example—scheduling of block B3.



(i) Step 1

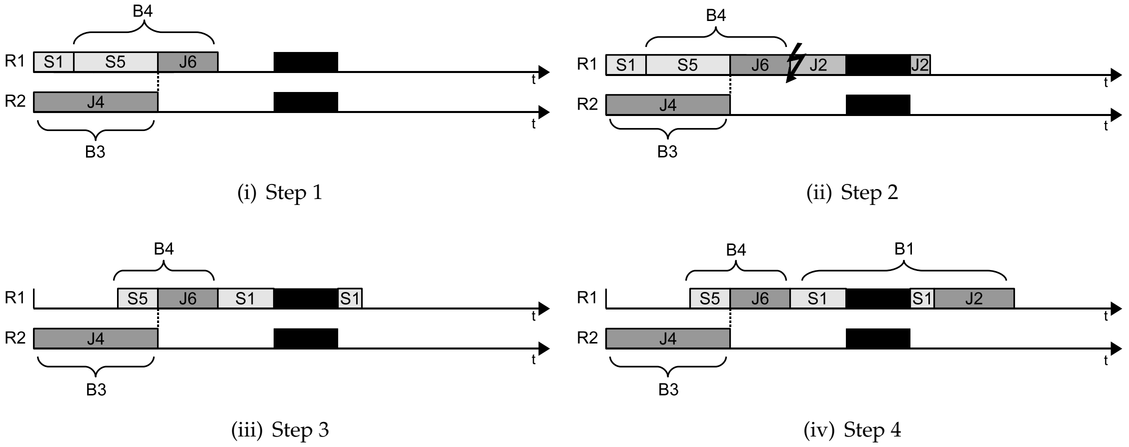
(ii) Step 2

(iii) Step 3

(iv) Step 4

Figure 6.6: Serial scheduling example—scheduling of block B4.

to the most left (Step 1). In the second step, J6 is scheduled as early as possible but must not start before its predecessor J4. Now, in order to avoid the idle time between J6 and S5, J6 is deallocated[3] and S5 is shifted to the right (Step 3). After a new scheduling of J6, the dynamic programming procedure ends and the block is successfully scheduled (Step 4).



(i) Step 1

(ii) Step 2

(iii) Step 3

(iv) Step 4

Figure 6.7: Serial scheduling example—scheduling of block B1.

The scheduling of the next block B1 is depicted in Figure 6.7. First the block's setup activity, S1, is scheduled. A possible slot for S1 is found right at the beginning of the schedule prior to S5 (Step 1). Note that the length of S5 increased (cf. Figure 6.6 (iv)), as S1 before S5 implies a different setup-state transition. However, after scheduling J2 in the second step, a violation of setup constraint 5.26 occurs, since J2 is not the machine successor of S1. Both activities are rescheduled, and the earliest start date of S1 is increased, such that S1 is located after J6. The result of a new slot finding for S1 is shown in Step 3. Note that the activity is stretched by the duration of the break

[3] In the current algorithm, a faster method is implemented that does not require the explicit deallocation of J6, instead calculating feasible slots of S5 and J6 before the block is scheduled. However, for didactic purposes, we illustrated the slower, simplified version.

(illustrated by the black box). Moreover, the length correction of S5 is reverted. Searching a slot for J2 in the next step leads to a feasible schedule (Step 4) and the dynamic programming procedure terminates. The above sequence of steps is a result of the active scheduling policy. Using the semi-active policy, S1 would immediately be scheduled after J6 (which in this case leads to the same result).

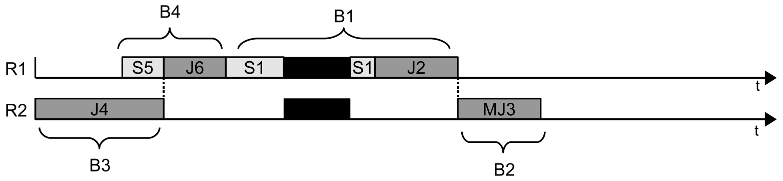
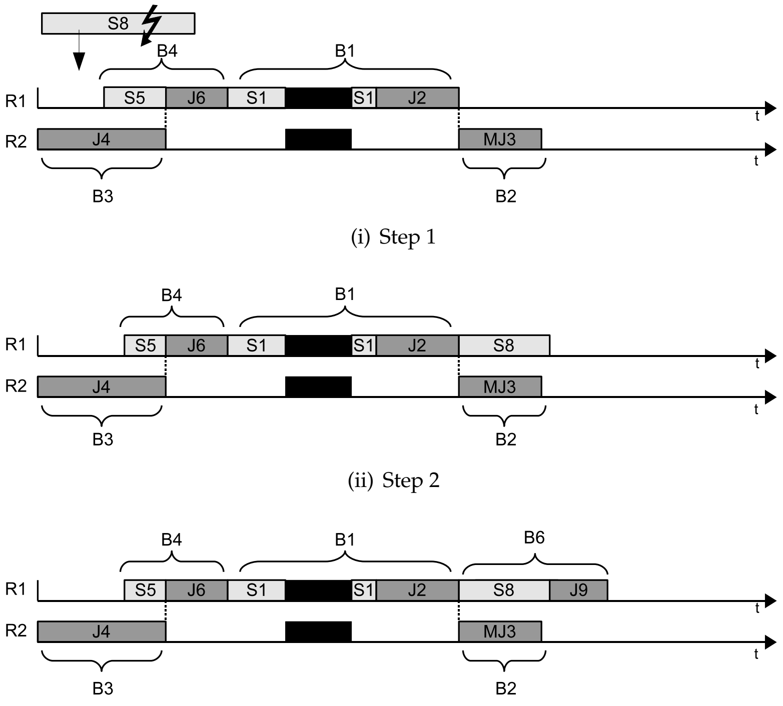


Figure 6.8: Serial scheduling example—scheduling of block B2.

After B1, block B2 is scheduled according to the encoding. The block consists of just one activity, MJ3, which is the successor of J2. As per the assumption, the suggested mode referred to R2. The corresponding schedule is depicted in Figure 6.8.



(i) Step 1

(ii) Step 2

(iii) Step 3

Figure 6.9: Serial scheduling example—scheduling of block B6.

The next block, B6, contains again two activities, the setup activity S8 and its successor, J9. The single steps of the dynamic programming procedure are illustrated in Figure 6.9. First, S8 is scheduled. The earliest date with sufficient capacity on R1 is the most left. However, after S5 has been modified and the net-duration of S8 has been calculated it becomes apparent that S8 is "too long" to be scheduled before S5 (see the first Gantt chart). Setup activity S5 is reset to its original duration and the slot-finding procedure then searches the next feasible start date, which is just after J2 (second Gantt chart). Scheduling J9 immediately after S8, the dynamic programming procedure terminates.

The scheduling of the next block, B5, is again straightforward. The block contains only a single activity, J7 to be scheduled on R2. J7 must not start before its activity predecessor, J6, terminates. Assuming an active scheduling policy, J7 is positioned before MJ3 as shown in Figure 6.10. Note that for semi-active scheduling, J7 would be placed after MJ3.

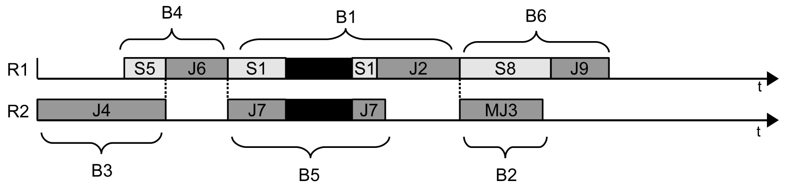
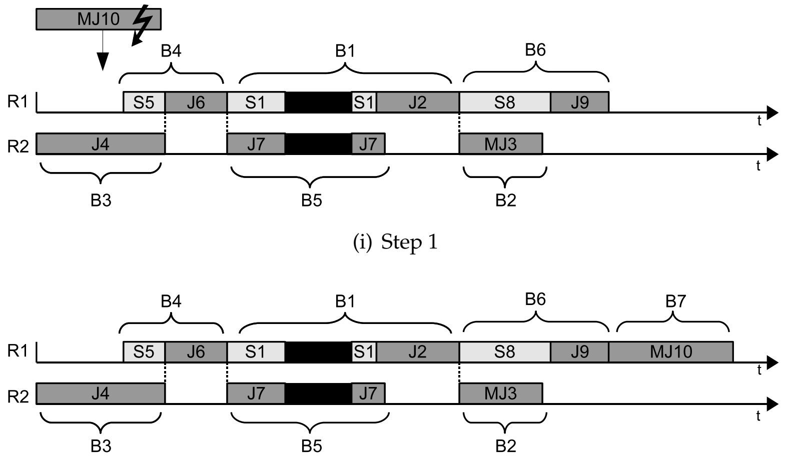


Figure 6.10: Serial scheduling example—scheduling of block B5.



(i) Step 1

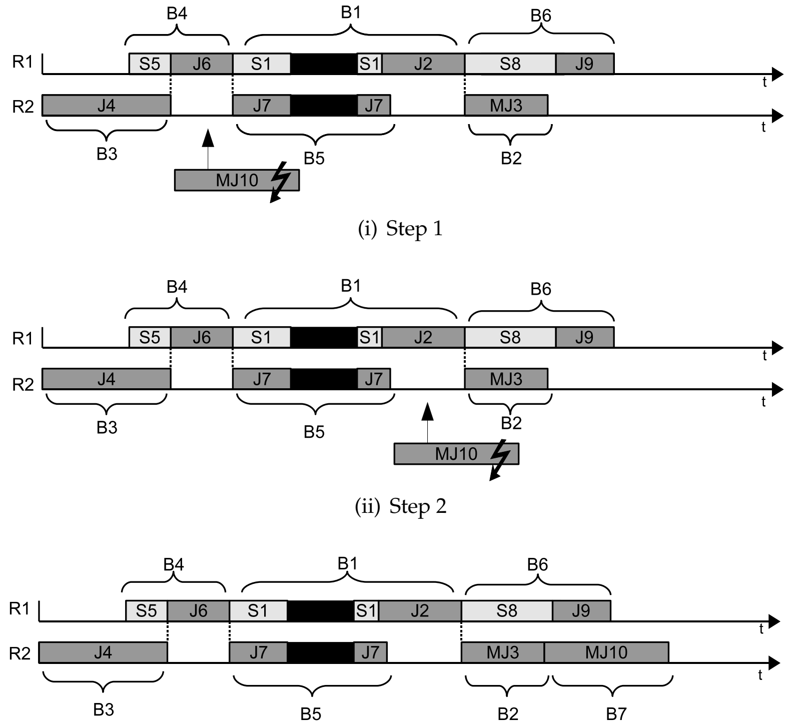
(ii) Step 2

Figure 6.11: Serial scheduling example—scheduling of block B7 in mode 1.

Finally, the last block B7 is scheduled. B7 consists of only one activity, MJ10. For MJ10, no mode selection has been suggested by the encoding. Therefore, the best mode has to be selected by the decoding procedure itself. The greedy heuristic builds for each mode a separate schedule, eventually choosing the one with the best outcome. The steps of the slot finding procedure for the first mode are illustrated in 6.11. The scheduling procedure for the second mode is shown in Figure 6.12. With respect to the makespan and lateness of activity MJ10, the latter schedule proves best. Since the total setup time stays equal and no deadline is assumed to be violated, the greedy heuristic eventually chooses the second mode.

6.1.2 Initialization of population

The GA underlying the DS solving module offers different methods to initialize a population of block lists and mode suggestions. Furthermore, in order to consider results of other heuristics a backencoding of already existing schedules is supported. The rest of the population is initialized in the following way. First, block lists are sorted according to different sorting criteria. Then, in a second step, the diversity is increased by further disturbing the sorted populations. Eventually, activity lists are repaired before the above SGS is applied. The modes are not initialized; rather, they are selected automatically using the greedy heuristic. The initialization methods are described in detail in the following subsections.



(i) Step 1

(ii) Step 2

(iii) Step 3

Figure 6.12: Serial scheduling example—scheduling of block B7 in mode 2.

6.1.2.1 Backencoding of existing schedules

Before the DS solving module is called, other heuristics may have calculated an initial schedule or the planner may have set up a solution manually. In many cases, this solution could greatly improve the search process if it was backencoded to a block list and injected into the initial population. The backencoding procedure considers semi-active planning. That means that applying the semi-active version of the serial SGS to the backencoded list should result in the original schedule of interest. If no setup times or costs are included, an active scheduling policy will only further increase the solution quality. In other words, the schedule of the backencoded list will be at least as good as the original schedule regarding the objectives makespan, lateness and mode costs. If the problem contains setup times or costs, the solving module automatically decides if active or semi-active schedules are to be constructed, as previously covered.

The backencoding of mode selections is obviously trivial. For each activity, we can set the suggested mode of the encoding to the selected mode of the schedule. Backencoding the current sequence of activities into a block list can be trickier. First of all, it should be recalled that we aim at a list that respects activity-precedence constraints by representing a feasible topological sorting of the related activity-on-node graph. For the standard RCPSP (5.1–5.4), the backencoding procedure is simple, as the activity list can be sorted according to the start dates of activities in the schedule. If, however, due to max-links or setup times, activities are aggregated to blocks, a backencoding cannot be guaranteed for all schedules. A heuristic that tries to backencode a schedule “as close as possible” can be outlined as follows. (For more details, the interested reader is referred to Engelmann (1998).) First, a directed “block-on-node” graph is constructed. If an activity of a block, A, is an activity *or* machine predecessor of an activity contained by another block, B, the graph contains a directed edge from A to B. Edges can be categorized in 4 priorities:

1. An edge due to an activity-precedence constraint has the highest priority.
2. If two activities are scheduled subsequently on the same unary resource, and the machine predecessor is delaying the start of the machine successor (cf. Subsection 6.1.1.1 for the definition of the terms machine predecessor and successor), the corresponding edge has high priority.
3. An edge for two subsequent activities on the same unary resource, where the machine predecessor is not delaying the start of the successor, has medium priority.
4. An edge for two subsequent activities on multicapacity resources has low priority.

The list entries are now sorted according to the directed graph as follows. The block list is filled up from left to right. Blocks with an indegree of zero are set at the right end of the initialized list as it stands at the time, and the related node and edges are deleted in the graph. If no such block exists, edges with lowest priority are deleted from blocks with minimal indegree. The procedure terminates if a position has been selected for every block.

6.1.2.2 Initial sorting of block lists

Initializing an activity list is commonly achieved by applying standard priority rules, cf. the works mentioned in Section 5.9. Also the DS Solving module supports the initial sorting of a list according to one of the following three criteria. First, activities can be sorted according to their propagated due dates:

$$\pi^{-1}(j) < \pi^{-1}(k) \Leftrightarrow pdd_j < pdd_k.$$

This initialization is based on the intuitive idea that activities with an earlier due date should also be scheduled earlier. If activities are aggregated to blocks, the lowest propagated due date of activities within the block is taken.

In a similar way, the earliest start dates can be taken into account:

$$\pi^{-1}(j) < \pi^{-1}(k) \Leftrightarrow es_j < es_k.$$

In the case of blocks, the highest start is used as a substitute.

Third, combining the above two criteria activities can be sorted according to their slack:

$$\pi^{-1}(j) < \pi^{-1}(k) \Leftrightarrow (pdd_j - es_j) < (pdd_k - es_k)$$

Here, activities with smaller intervals are scheduled first, whereas more “flexible” activities are scheduled later. In the case of blocks, the same substitutes as above are taken. Finally block lists can be initialized purely randomly. For the computational test, a population of 60 individuals is taken, whereas 15 individuals are constructed by the same sorting method.

6.1.2.3 Disturbing methods

The initialized block lists are now disturbed by one of the following two mechanisms. If the problem contains setup times or costs, the program tries to build campaigns of activities requiring the same setup state. Scheduling activities requiring the same setup state in a sequence saves setup times and costs. Intuitively, the size of a campaign is of critical importance. If too small, there will only be a negligible decrease of setup times or costs. If too large, more setup-related costs will be saved, however, at the expense of an undesirable increase in other objective criteria. The optimal trade-off is highly problem dependent and not obvious. The approach here is to simply use different search window sizes on the same initial block lists. The construction of campaigns is achieved by the following heuristic.

1. Set the initial window size to $ws = 1$.
2. Select the next block list with the same initial sorting. Cluster similar setup activities by the following subprocedure:
   (a) Select the first block of the list.
   (b) Search within the next ws list entries for blocks containing setup activities with the same setup key on the same primary resource. If such a block has been found, insert it before the window's leftmost block with a different setup state.
   (c) Select the leftmost block of the current window with a different setup state as a new starting block. Continue with (b) until all blocks of the block list have been considered.
3. Increase the window size by a constant factor. (For the computational tests, 1.8 was used.)

The heuristic is called once for each set of block lists initialized by the same sorting. If the problem does not contain any setup times or costs, the initialized block lists are disturbed randomly. To do this, activity/block positions are arbitrary swapped within a gliding window. For the computational tests, the following values have been used. Starting leftmost, the next 16 list entries are shuffled. Then, the windows is moved by 8 list entries. After a list has been initialized and disturbed by a combination of the above methods, it is repaired: sorted according to either Function `establishPrecedenceByPullFwd` or Function `establishPrecedenceByPushBwd` (cf. Page 90) with equal probability.

6.1.3 Mutation and crossover operators

The DS solving module relies on several mutation operators that can be categorized into four basic principles. First, a list entry can be inserted at another position. Second, an entry can be swapped with another entry. Third, the block sequence remains unchanged, but modes are altered. Fourth, not only a single list entry but whole clusters are shifted to different positions. This makes sense in the case of setup times and costs to keep the disruption of setup campaigns low. The single mutation operators are explained in the following subsections.

6.1.3.1 Insertion of blocks

We speak of an insert operation if the block sequence remains constant except for the single block to be inserted at another position. Thus, an insert operation consists of three steps. First a block to be shifted needs to be identified, either randomly or based on heuristic principles. Second, a target position is calculated. Third, entries located between the current and the target position are locally shifted by one to the left or to the right (depending if the target is situated right or left of the current block) and the block is placed at the target position. The following insert operators are implemented:

- **Random insert on same primary resource:** "Wrong" sequences of activities on the resources are the origin of considerable lateness and large setup costs and times. However, since it is not clear which sequence leads to the best result, it makes sense to randomly select a block and insert it before or after other blocks on the same resource. To be more specific, the term "on the same resource" needs to be clarified. We say two blocks are situated on the same resource if at least one activity of block 1 has the same primary resource in the suggested mode as any activity of block 2. If no mode was suggested, the primary resources must be equal for at least one of all possible modes.

  If two such blocks can be identified, we either insert the later one before the earlier one, or insert the earlier one after the later one. In the first case, the block list needs to be repaired subsequently by pulling "too late" blocks forward, by sorting according to Function `establishPrecedenceByPullFwd`. In the latter case, we need to push "too early" blocks backward by applying Function `establishPrecedenceByPushBwd`. Before the new genotype is decoded, the mode suggestions of activities of the inserted block are deleted so that the greedy heuristic will be applied. We illustrate the result of the operator on the basis of an example: Schedule 6.12 (iii) will be mutated. The original block list of the example was B3 → B4 →B1 →B2 →B6 →B5 →B7. Assume that B7 is now inserted before B5 so the new order becomes B3 → B4 →B1 →B2 →B6 →B7 →B5. The resulting schedule (for which the greedy heuristic selected the second mode for B7) is shown in Figure 6.13.

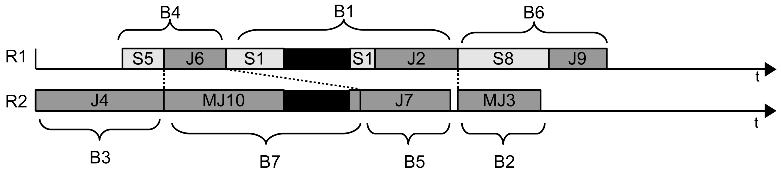


Figure 6.13: Serial scheduling example—result of shifting a block on the same resource.

- **Random insert across resources:** Even if not situated on the same primary resource, shifting a block before another block can lead to a different schedule if the activities involved run on similar secondary resources or if there is an indirect influence over other machine successors or predecessors. This operator randomly selects a block and shifts it to another randomly chosen target position. The mode suggestions of activities contained by the shifted block are deleted. Afterwards, the positions of activity predecessors and successors are corrected by sorting the block list by either Function `establishPrecedenceByPullFwd` or Function `establishPrecedenceByPushBwd`, depending on the direction of the shift. As an example, assume that the original block list (see Figure 6.12 (iii)) is altered to B1 → B3 →B4 →B2 →B6 →B7 →B5, as B1 is shifted before B3. The resulting schedule is shown in Figure 6.14.

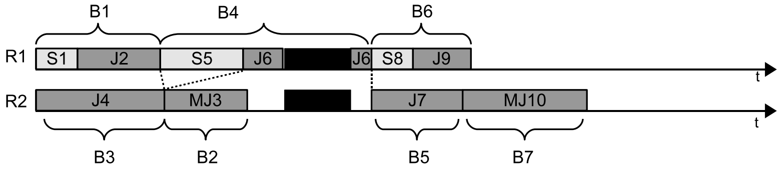


Figure 6.14: Serial scheduling example—result of shifting a block on a different resource.

- **Changing critical sequences:** The makespan of a schedule equals the length of its critical path. The length of the critical path in turn contains a fixed and a variable component. The fixed component is the sum of the durations and minimal distances implied by the activity-precedence constraints. The variable component, however, is determined by the sequence of activities along the critical paths without an activity-precedence relation but succeeding each other on the same resources.

This operator selects such a sequence, disturbs it and hence tries to reduce the length of the critical path. In our example (Figure 6.12 (iii)), the critical path is given by the activities J4, J6, S1, J2, MJ3 and MJ10. On R1, the critical subsequence of machine successors is J6, S1, J2 or blocks B4 and B1, respectively. On R2, blocks B2 and B7 are critical. Now, by choosing and disturbing the latter sequence—i.e., inserting B7 before B2—we try to reduce the length of the critical path. The new block list becomes B3 → B4 →B1 →B7 →B2 →B6 →B5, which leads to the already discussed schedule shown in Figure 6.14. We use this observation to emphasize the effects of using a SGS as decoding function. By applying an active or semi-active scheduling policy, two different block lists can be decoded to the same schedule.

In general, the operator can be described as follows. First, the critical graph is computed (cf. Engelmann, 1998). Second, succeeding activities on similar resources are identified and the corresponding blocks are aggregated to critical sets. One of these sets is then chosen randomly. Finally, with a probability of 0.5, the last block of the chosen set is randomly inserted at another position of the set and the block list is repaired by pulling "too late" blocks forward (i.e., sorted according to Function `establishPrecedenceByPullFwd`). Otherwise, the first block of the set is inserted at a later position and the block list is repaired by pushing "too early" blocks later by re-sorting according to Function `establishPrecedenceByPushBwd`.

- **Shifting delayed block:** This operator tries to decrease the maximum and total lateness of a schedule. A quarter of all blocks is randomly chosen and compared by their lateness. Eventually, the block with the most delayed activity is selected and inserted at an arbitrary position before its current position. Afterwards, the position of its activity predecessors is corrected by sorting the block list according to Function `establishPrecedenceByPullFwd`: "too late" predecessors are shifted as well.

6.1.3.2 Swapping of blocks

In contrast to an insert operation, a swap alters the block list by exchanging exactly two list entries. However, a subsequent repair can cause further changes. The following mutation operators rely on swap operations.

- **Multiple swaps:** For our computational tests, this operator randomly selects one block after the other and swaps each block with another block at a random position. The selection of blocks continues until a total limit of 100 activities (contained in the selected blocks) has been reached. Afterwards, the block list is repaired by either pulling "too late" blocks forward or pushing "too early" blocks later with probability 0.5.

- **Disturb:** The disturb mutation radically changes a part of the block list. Within a certain randomly positioned window, the sequence of blocks is randomized and the related mode suggestions are deleted. For the computational test, a window size of 20 was chosen.

- **Reduction of first-priority lateness:** Minimizing the violation of deadlines is of first priority, as defined by the lexicographic sorting 5.39. This operator tries to correct those parts of a block list that lead to such a lateness by swapping several blocks. With a probability of 99%, a block containing such a lateness is swapped with its preceding list entry, if this does not violate activity-precedence relations. Otherwise the block is inserted at an arbitrary point before its current position and the list is repaired according to Function `establishPrecedenceByPullFwd`. We apply this correction to all blocks leading to a first-priority lateness. Using this operator sequentially shifts blocks causing such a lateness to the left, with the ultimate goal to reduce the total first-priority lateness to zero. If deadlines are violated, this operator is always applied before the list is subsequently altered by another (randomly chosen) operator. The full sequence of operators is discussed later in Subsection 6.1.5.

6.1.3.3 Mode alternation

Another category of mutation operators does not alter the block sequence, but changes the modes instead. Two mode-alternation operators are implemented.

- **Reset modes:** All mode suggestions of the activities contained by the blocks within a window of subsequent blocks are deleted. The position of the window is calculated randomly. For the computational test, a window size of 50 blocks was used.

- **Change mode:** This operator randomly selects one multi-mode activity that already has a mode suggestion. A new mode suggestion is randomly drawn from all possible modes. If the new mode suggestion equals the old one, the suggestion is deleted and the greedy heuristic is applied.

6.1.3.4 Campaigns

If the problem contains setup times or costs, it makes sense to aggregate activities with similar setup keys into campaigns. The following campaign mutation operators are implemented.

- **Merge campaigns:** This operator builds a larger campaign out of two smaller campaigns. First, a setup activity is randomly selected. Now, the related campaign, A, is calculated as the set of all machine successors and predecessors (cf. Subsection 6.1.1.1 for the definition of the terms machine predecessor and successor) requiring the same setup state on the same primary resource. Afterwards, another campaign, B, scheduled on the same primary resource requiring the same setup state is searched. Based on a random drawing B is either searched to the left or to the right of A. If found, all list entries belonging to the first campaign, A, are now inserted next to list entries of campaign B. Depending on the search direction, the list is either repaired by pulling "too late" blocks forward or by pushing "too early" blocks later. If the new list is decoded to a schedule, both campaigns will be scheduled next to each other and will thus merge to a single campaign.

- **Insert campaign:** First, a setup activity is chosen randomly and the related campaign, A, is computed. Then, all other campaigns (including those with different setup keys) within a certain window on the same primary resource are calculated. For our computational tests, the window size was uniformly distributed to include between 5 to 20 of the next left and right campaigns, respectively. For all campaigns, the cost differences are calculated that would result if we insert campaign A before or after each candidate. The candidates are sorted according to increasing cost difference, and we select the final insert target according to a geometric distribution. The candidate with the highest saving (or least cost increase) is chosen with a probability of 80%, the second one with $20\% \cdot 80\%$, the third one with $20\% \cdot 20\% \cdot 80\%$ and so on. Eventually, after having chosen the target, campaign A is inserted in the desired position.

6.1.3.5 Linear-order–based crossover

Linear-order–based crossover tries to merge the sequences of two parent encodings. First, two different crossover points are chosen randomly. The part of the block list before the lower point and after the higher point are directly copied from the first parent to the child. The same is true for the suggested modes of the activities contained by the copied blocks. The remaining part of the child block list is filled up in the order defined by the second parent. The related mode suggestions are also copied from parent 2. An example with the crossover points 3 and 7 is given below.

	Sequence 1	Sequence 2	Sequence 1
Parent 1	B3 → B4 → B1	→ B2 → B6 → B5	→ B7
Parent 2	B6 → B1 → B2	→ B7 → B3 → B4	→ B5
Child	B3 → B4 → B1	→ B6 → B2 → B5	→ B7

Linear-order–based crossover preserves the precedence constraints, so a subsequent repairing of the child is not necessary. In "classical" Evolutionary Algorithms, crossover and mutation are applied in sequence to the same parent. This is not the case here. The above-presented mutation operators require problem-specific knowledge which is derived from the phenotype, the actual schedule. After crossover, the phenotype must be constructed again by the serial SGS, however. Thus, for each parent individual, either crossover *or* mutation is applied.

6.1.4 Mutation and crossover probabilities

Mutation and crossover probabilities depend on the problems' properties. For example, if makespan minimization is less important, applying the "changing critical sequences" operator will not necessarily lead to good solutions, since minimal-makespan schedules do not necessarily imply a decrease in other objectives. For our computational tests, the operators are initialized with base probabilities as shown in Table 6.2. We speak of a "dominant setup" if the weight of setup-related objectives exceeds the weight of the remaining objectives: $wtst + wtmt \geq wtl + wml + wms$. Minimizing the makespan becomes dominant if wms is the largest weight among all objective function weights. The actual mutation and crossover probabilities are normalized values of the base probabilities.

6.1.5 The evolutionary cycle

This section reviews the phases of the GA (cf. Figure 4.1, Section 4.2) underlying the DS Solving module.

1. **Preprocessing:** Before starting the core GA, a preprocessing takes place. Preprocessing involves a forward and a backward pass to compute feasible intervals of start and finish dates

Operator	Base Probability
Random insert on same primary resource	0.2
Random insert on across resources	0.2
Changing critical sequences	0.2 if makespan is dominant 0 otherwise
Shifting delayed block	0.1
Multiple swaps	0.2
Disturb	0.2
Reset modes	0.05 if alternative modes are present and backencoded after the greedy heuristic 0 otherwise
Change mode	0.05 if alternative modes are present 0 otherwise
Merge campaigns	0.2 if setup is dominant 0 otherwise
Insert campaign	0.2 if setup is dominant 0 otherwise
Linear-order–based crossover	1.0

Table 6.2: Mutation base probabilities used for computational tests.

for each activity. According to these intervals, some modes might actually be infeasible. For example, a mode's duration might be either too long or too short such that the finish and start date could not be both in their feasible interval or the mode's capacity requirements might exceed the resource availability. Such modes are marked as inaccessible and are not considered further.

2. **Initialization:** The population is initialized using the methods described in Section 6.1.2. The current schedule is backencoded to the first individual. The remaining part is sorted in equal portions according to the due dates, the earliest start dates or the slack or randomly. If setup times are included, the setup-clustering method is applied afterwards. Otherwise the initial block lists are disturbed randomly.

3. **Evaluation:**

   (a) **Decoding by serial scheduling:** First, all individuals of the population that have not been decoded so far are decoded to a schedule using the aforementioned serial SGS.

   (b) **Backencoding of solutions:** If the "reset modes" operator was assigned a positive probability, the modes selected by the greedy heuristic are backencoded to the individuals. Also, the actual schedule could be backencoded to the block list to further support Lamarckism (cf. Subsection 4.2.2). However, as backencoding of blocks is a rather time-intensive procedure for more complicated problems, this option was not activated for our computational tests.

   (c) **Computation of rankings:** Eventually, the individuals are ordinally ranked according to Criterion 5.39.

4. **Update:** Applying a so-called $\mu + \lambda$ selection, the worst individuals out of the combined parents and offspring are removed, and the remaining form the set of new parents. In addition, duplicates are filtered out already before the $\mu + \lambda$ selection takes place. A child is considered to be a duplicate if it has the same objective values as any individual of the

parent population. Different schedules can have the same objective values, such that the duplicate elimination method actually decreases the diversity of the population. An advantage is that it is a fast method. Checking for true duplicates requires a comparison of the actual schedules, which is a rather time-consuming task.

5. **Check for termination:** The GA terminates when it reaches a predefined total runtime.

6. **Selection:** Parent selection is based on repeated tournaments. In each tournament, two candidates from the parent population are randomly chosen and compared according to their fitness values (cf. Equation 5.39). The fitter parent is then selected into the mating pool.

7. **Construction of offspring:** Each selected parent is modified and saved as new child as follows.

   (a) **Reduction of first-priority lateness:** Before the actual mutation, the operator "Reduction of first-priority lateness" is applied if such a lateness occurs.

   (b) **Mutation or crossover:** To every child, one mutation or crossover operator is applied, selected according to the probabilities mentioned in Section 6.1.4. The second parent required for crossover is chosen randomly from the set of all individuals of the mating pool having a different fitness than the first parent.

   (c) **Sorting out failed mutations:** In some cases an operator might not be successful. For example, a campaign with the same setup key might not be found to merge. Block lists of such failed mutations are sorted out immediately.

8. The process continues with Step 3 until termination.

6.2 Alignment heuristics

Besides the solving module, different heuristics exist to modify an RCPSP context. For example, one heuristic performs backward and forward passes, propagating due dates and start dates through the network of activities as already discussed in Chapter 5. In this section, we introduce two other heuristics. Starting with a feasible solution (i.e., a suboptimal solution of the related RCPSP), these heuristics shift activities left and right without disturbing the sequence of activities "too much." These heuristics are proprietary versions of the right- and left-alignment techniques mentioned in Section 5.9 and are explained in the following subsections.

6.2.1 Right-alignment heuristic

Earliness as a non-regular performance measure precludes the use of the serial SGS; the optimal schedule no longer lies within the set of all active schedules (or semi-active schedules for setup times), but within the set of all *feasible* schedules. Applying a serial or parallel forward SGS is obviously not the right approach to construct good schedules with respect to the earliness objective. Moreover, the set of feasible schedules is usually much larger than the set of all active or semi-active schedules. Thus, only the lateness plays a role within the objective function of the DS Optimizer. The schedules constructed by the serial SGS may contain activities with a huge earliness, which is seen as an undesirable "side effect" of many practical problems.

The strategy is then to modify the schedule in a subsequent step *after* optimization in order to reduce the earliness without causing additional lateness. The **right-alignment heuristic** iteratively applies a series of global right shifts (cf. Definition 5.2.4) as follows.

1. As for the Genetic Algorithm presented in Section 6.1.1, activities are aggregated to blocks. However, for the alignment heuristics, a block comprises at least all activities of an order. An order is a combination of activities from a business level perspective (cf. Section 5.8). If max-links to activities from other blocks exist, the related blocks are merged to a single block. The choice of the order as the minimal building block is motivated by the preferences of the users and also by the fact that setup constraints, max-links and mode-compatibility constraints are usually only used within an order.

2. Within this set of yet unshifted blocks, sort all blocks according to the minimum finish date of their activities.

3. Try to find the *latest* slot for the unshifted block with the largest finish date, such that the lateness of the current schedule does not increase. The slot-finding procedure follows a dynamic programming approach similar to the one discussed in Section 6.1.1, with the difference that it starts with the last activity of the block and that late start dates are preferred. The following additional restrictions can be applied optionally. First, the set of feasible modes can be restricted to the currently selected mode. Otherwise the best mode, leading to the largest start date is selected for every activity. Second, an upper bound, b, for setup time and cost increase can be specified. A slot is only feasible if $wtst \cdot \Delta tst + wtsc \cdot \Delta tsc \leq b$, where Δ denotes the difference between the activity's new and old position for setup times and cost, respectively. If no later slot can be found, the activities remain at their current position.

4. Remove the block from the set of unshifted blocks. If there are still blocks to be shifted continue with 3, otherwise stop.

Applying the above heuristics several times in a row can further improve the earliness, since gaps created by shifted blocks of the first run can be filled during subsequent runs. The right-alignment heuristic will be used later in order to create new proposals. For the computational tests, the following configuration was used.

1. Shift all activities right without changing modes or increasing setup times or costs.

2. Shift all activities right allowing mode changes and arbitrary setup-time or -cost increases.

3. If any block was shifted previously, shift again all activities right, allowing setup-time and -cost increases but not mode changes.

4. If any block was shifted previously, go to step 3, otherwise stop.

6.2.2 Left-alignment heuristic

Similar to shifting activities to the right, also a left-alignment functionality is provided. It is obvious, however, that the result of serial scheduling using an active-scheduling policy cannot be improved. If a slot of sufficient capacity had been available, it would have already been chosen by the DS solving module. Nevertheless, for construction of new proposals the **left-alignment heuristic** proved useful as will become apparent later. A changing of modes is not supported and a temporary degradation of the objective value is not supported by the current implementation. Thus, the left-alignment is only applied once.

6.3 Further extensions

For the sake of completeness it should be mentioned that the DS Optimizer as well as the alignment heuristics additionally support a deallocation of orders. If activated, whole orders are deallocated until no due dates are violated. Deallocated orders are penalized in the objective function, such that the underlying Genetic Algorithm tries to find undelayed schedules with as few deallocated orders as possible. The resulting schedule provides the planner with additional information on capacity shortages. However, since the goal of the coordination mechanism is to calculate an interorganizationally feasible schedule, the coordination mechanism does not rely on the deallocation functionality. For ranking individuals, several further methods are supported, such as nondominated crowding-distance sorting (cf. Deb, 2001). Moreover, the campaign definitions can be broadened to activities requiring similar setup states or to setup activities with similarly small setup times, for example.

CHAPTER 7

The customized DEAL framework

This chapter discusses the customization of the DEAL framework to the SAP Detailed Scheduling (DS) Optimizer and further specifies the coordination mechanisms for planning domains that intend to coordinate their intradomain Resource-Constrained Project Scheduling Problems (RCPSPs). Section 7.1 lists underlying assumptions and defines the interorganizational RCPSP. In Section 7.2, we describe the standard two-tier scenario, where several suppliers are delivering to one Original Equipment Manufacturer (OEM). The remaining sections describe in detail the customized steps of DEAL—evaluation, construction and initialization—with regard to the standard two-tier scenario.

7.1 The interorganizational problem

We assume that each planning domain has its unique set of resources, disjoint from the resource sets of other domains. The collaboration process remains entirely on an operational planning level. That is, neither are aggregated quantities of delivered items communicated nor is the pegging net recomputed. An interorganizational solution is constructed by following a successive planning approach, where each partner applies his DS Optimizer to his intradomain RCPSP. Activities between the different planning domains are connected by precedence constraints. That is, activities of the OEM can only be started after the suppliers' preceding activities have been finished. To be more specific, we make the following assumptions.

- There is no cyclic dependency between the domains. Several suppliers are delivering to one OEM and to other, external customers (not participating in the coordination process), but not to another supplier. Also, the OEM is only delivering to ultimate customers but not to any supplier.

- For computing schedules, OEM and suppliers have available the tools presented in the previous chapters, the solving module of the DS Optimizer, the heuristic for computing backward and forward passes, and the alignment heuristics.

- Suppliers and OEM are assumed to have unlimited inventory capacity to store intermediate goods at no cost.

- The activity network is generated during a preprocessing step. That is, the OEM first calculates his lot sizes, the related number of activities and the pegging net (i.e., the precedence relations between activities).[1] Moreover, initial release and due dates are computed. Orders are generated and communicated to the suppliers, who in turn proceed with similar calculations. This way, an interorganizational activity network is computed, where each domain only knows its part, however.

- There is no intermediate recalculation of the lot sizes (influencing activity durations) or the pegging net. Thus, the number of activities and the related net-durations remain constant.

- Activity-precedence constraints within a planning domain remain constant during the entire collaboration process. Also, activity precedence *between* the partners is not altered.

- Resources are not shared by the domains and the available resource capacity remains constant as well. Moreover, mode-resource assignments, activity durations, calendars and so forth do not change either.

- During the coordination process there is no change of due dates belonging to customers not taking part in the coordination process. The same is true for release dates related to technical properties or to suppliers excluded from the collaboration process.

- We assume that no maximum time lag exists between two activities of different planning domains.

The RCPSP formulation of Chapter 5 is extended by defining additional data, index sets and superscripts summarized in Table 7.1. For example, the set of all jobs is distributed between different planning domains e, i.e.

dd_j^*	static, non-varying due date of activity j
dl_j^*	static, non-varying deadlines of activity j
e	index of a planning domain
E	set of all planning domains
J^e	set of activities j within the domain e, $J^e \subset J$
J_U^e	set of upstream-related activities of domain e, $J_U^e \subseteq J^e$
J_D^e	set of downstream-related activities of domain e, $J_D^e \subseteq J^e$
R^e	set of resources r within the domain e
rd_j^*	static, non-varying release date of activity j
P_j^e	set of predecessors of activity j within domain e
S_j^e	set of successors of activity j within the domain e

Table 7.1: Additional data, index sets and superscripts for the interorganizational RCPSP

$$\forall e \in E : J^e \subset J,$$

$$\bigcup_{e \in E} J^e = J,$$

$$\forall e, f \in E, e \neq f : J^e \cap J^f = \emptyset.$$

[1] Algorithms for calculating lot sizes and the pegging net are not subject of this chapter. For an overview of different approaches, the interested reader is referred to Voß and Woodruff (2006). Moreover, in Chapter 8, a test data generator simulating the SAP Production Planning functionality is presented.

The same is true for resources, modes, and so forth.

For downstream domains, release dates are determined by corresponding finish dates of upstream domains and static, nonvarying, release dates rd_j^*, influenced by technical circumstances or partners not participating in the coordination process. This dependency of upstream domains is expressed by including the following equations in a downstream domain's RCPSP formulation.

$$rd_j = \max \left\{ \max_{k \in P_j^g, g \in E \setminus \{e\}} (fd_k), rd_j^* \right\} \quad \forall j \in J_U^e \tag{7.1}$$

The set of **upstream-related activities** J_U^e is defined as the set of all activities with predecessors in other planning domains: $J_U^e = \left\{ j \in J^e | \exists i \in J^g \wedge i \in P_j^g \text{ with } g \in E \setminus \{e\} \right\}$. A forward pass propagates the earliest start and finish dates through the network of activities.

For downstream planning, the process is straightforward: the upstream domain schedules its activities first. The derived finish dates form the release dates for activities of adjacent downstream domains. After the downstream domains have scheduled their production, a feasible solution of the interorganizational problem has been constructed. For upstream planning, we need a different process. Since violating due dates is the "vent" for computing feasible schedules, upstream domains might not fulfill all proposed dates of the downstream domains. Due dates of upstream domains belong to anticipating data, since they provide an approximation of the downstream domains' scheduling problems. Thus, it should be the task of the downstream domains to propose promising due dates. However, in general, the downstream domains are not aware of the upstream domains' model formulations. In order to separate the analysis of the local problem from subsequent translation issues, each downstream domain calculates promising settings for their upstream-related *release dates*. These settings will be denoted as **proposed dates** pd henceforth. The upstream domains map these dates to due dates by computing

$$dd_j = \min \left\{ \min_{k \in S_j^g,\, g \in E \setminus \{e\}} (pd_k), dd_j^* \right\} \quad \forall j \in J_D^e, \tag{7.2}$$

where $J_D^e = \left\{ j \in J^e | \exists i \in J^g \wedge i \in S_j^g \text{ with } g \in E \setminus \{e\} \right\}$ denotes the set of **downstream-related activities**. Static, non-varying due dates dd_j^* pertain to external customers not taking part in the coordination process. Upstream domains may experience two kinds of lateness for each activity j, **external-related lateness** ($\max\{fd_j - dd_j^*, 0\}$) and **downstream-related lateness** ($\max\{\max\{fd_j - dd_j, 0\} - \max\{fd_j - dd_j^*, 0\}, 0\}$). Naturally, both kinds of lateness should be considered in the upstream domain's objective function. Henceforth, we suppose that an upstream domain considers external-related lateness to be more important than downstream-related lateness. Coordinating a plan with downstream domains should not (further) delay delivery to external customers. This can be achieved during a preprocessing before the DS solving module is called. First, external-related due dates are converted to pseudo-hard deadlines $dl_j := \min\{dl_j^*, dd_j^*\}$, whereas static, non-varying deadlines dl_j^* may refer to further technological constraints. Second, we set $dd_j^* := \infty$ for all $j \in J^e$. Finally, the due dates dd_j are computed using Equation 7.2. This procedure only applies to upstream domains. Thus, while external-related deadlines are prioritized, the violation of downstream-related due dates is treated as the standard lateness in the objective function, cf. Relation 5.39.[2]

[2] This form of prioritizing can be regarded as a crude approach. However, it facilitated the development of the prototype. For a real-world application, a finer tradeoff between external- and downstream-related lateness might be more appropriate.

For two-domain upstream planning, we have the following process. The downstream domain calculates proposed dates for the upstream domain. The upstream domain maps these proposed dates to due dates as shown by Equation 7.2 (external-related due dates have been set to deadlines previously). Depending on the weighting of the objective function, the upstream domain "tries" to fulfill the proposal to the extent possible. However, due to finite capacities and prioritized deadlines, the upstream domain might not be able to entirely fulfill the proposal. Some activities might be scheduled earlier and others later. The upstream domain then in turn sends the resulting finish dates as a counterproposal to the downstream domain. The downstream domain maps the finish dates to release dates, cf. Equation 7.1. After applying a forward pass, the downstream domain eventually calculates its own schedule based on the new earliest start and finish dates. In some sense, upstream planning can be regarded to always involve downstream planning in order to get feasible schedules for all involved partners. For *translating* proposals, a planning domain requires knowledge of which activities of other domains are directly affected by its own activities. This information is not considered as critical.

Preliminary experiments suggested that converging BOM structures at the boundary between upstream and downstream domain can lead to performance losses. Situations are imaginable where several activities of an upstream domain need to be completed before a single activity of the downstream domain can be started, whereas each of the upstream activities has its own due date. As the DS Optimizer tries to minimize the total lateness, however, the finish dates of the upstream domain's activities are not necessarily aligned and the latest activity determines the start of the downstream domain's activity! For the considered set of activities the maximum rather than the total lateness should be prioritized. This value is not necessarily included in the objective function, since the overall maximum-lateness objective can pertain to another (even more delayed) activity. In some sense, the upstream domain does not have the correct *view* of the downstream domain's situation. As a remedy, we introduce artificial activities on the supplier's side during a preprocessing procedure. The artificial activities use no resources, have a duration of zero and directly precede the downstream domain's activities while succeeding the critical supplier's activities. The DS solving module itself is not changed, however.

Figure 7.1 gives an overview of the different dates relevant to a planning domain's activity j.[3] It should be emphasized that parts of the DS Optimizer's data are treated as variables by DEAL.

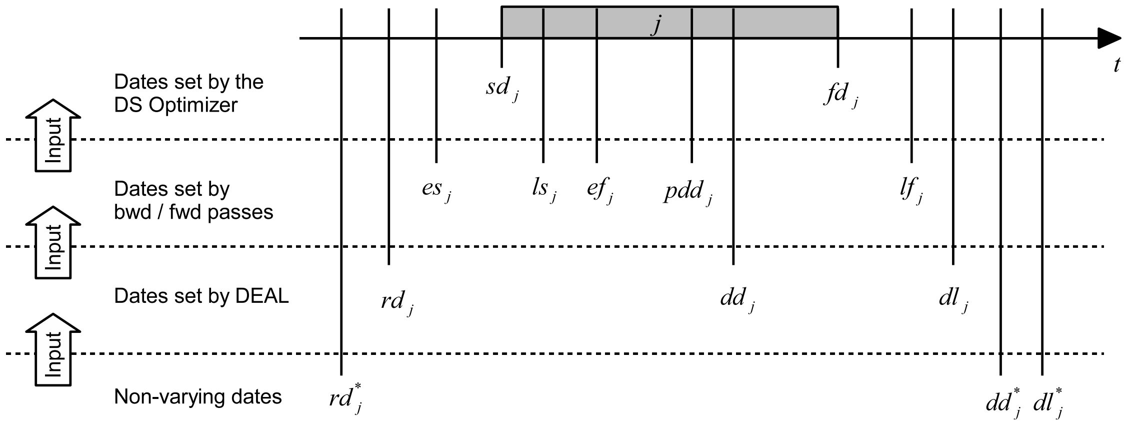


Figure 7.1: Overview of the dates set by DEAL and DS Optimizer for a single activity j

[3] This figure does not include proposed dates pd_j as these dates do not directly relate to an activity but influence the due dates dd_j, cf. Equation 7.2.

Dates related to technological constraints or external partners not participating in the coordination (i.e., rd_j^*, dd_j^* and dl_j^*) form the nonvarying basis of an intradomain RCPSP problem instance.[4] The DEAL framework dynamically changes the problem definition by aligning release dates, rd_j and due dates, dd_j. Propagated due dates, pdd_j, and latest start and finish dates, ls_j and lf_j, are computed by a backward pass. A forward pass sets the earliest start and finish dates, es_j and ef_j, according to the dynamically changing release dates, rd_j. The above settings define the problem instance for the underlying DS Optimizer that computes a solution w.r.t. the updated constraints and objective function. Eventually the DS Optimizer sets start and finish dates, sd_j and fd_j, for each activity. Due to the serial schedule-generation scheme in use, an activity is never scheduled before its earliest start date. The latest finish date or the (propagated) due date may be violated, however.

7.2 The standard two-tier business case

In the following, we focus on two-tier business cases, where the OEM takes the leading role. We suppose that all domains have external customers not taking part in the coordination process and that the OEM's and suppliers' production processes are tightly coupled in the sense of a just-in-time delivery. Though inventory buffers are assumed to be available at no cost and are not included in the RCPSP model formulation, early due dates of external customer orders are assumed to require the coordination of delivery sequences between the planning domains. According to discussions with SAP Solution Management, this setting is of major importance. Within the DEAL framework, the OEM iteratively sends proposed dates to his suppliers who map the proposals to due dates pertaining to their planning problems, cf. Equation 7.2. The suppliers also have external customers and will fulfill the OEM's proposals only as long as their external customers are not harmed. As already discussed, external customer orders are prioritized by converting the related due dates to deadlines. Following the convention of Chapter 4, we denote the OEM's domain with $e = 0$ and refer to the suppliers' domains with $e > 0$. For a supplier e, an individual is encoded by downstream-related due dates dd_j pertaining to activities that precede activities of the OEM's domain; $j \in J_D^e$.

Downstream-related due dates belong to the class of anticipating data, since they provide the supplier with a crude approximation of the OEM's planning problem. Together with other unvarying constraints and objectives, these due dates define an RCPSP problem instance. The supplier's DS Optimizer *maps* this instance to a solution. The solution part relevant to the OEM consists of the finish dates, fd_j, of activities that precede activities of the OEM's domain; $j \in J_D^e$. All suppliers send their finish dates to the OEM. The OEM maps these finish dates to upstream-related release dates by applying Equation 7.1. Together with his static constraints, these release dates define his problem instance, after a forward pass was applied. Applying the DS Optimizer yields the corresponding solution. Also from the interorganizational perspective, a feasible solution was created since the connecting constraints (i.e., the OEM's release dates) were not violated.

[4] As discussed above, static, non-varying due dates of upstream domains are converted to deadlines in our prototype. However, this must not be true for the general case.

7.3 Initialization

As standard Evolutionary Algorithms do, DEAL requires an initial population. The initial population consists of only one individual (i.e., a feasible interorganizational solution) and is subsequently filled up with children until the desired population size has been reached. For the computational tests, we used two methods to construct the initial individual. Both methods presume that activities, precedence constraints, and initial release and due dates have already been generated (as has been discussed when presenting the assumptions in Section 7.1). For the computational tests, methods for simulating this step will be presented in Chapter 8.

7.3.1 Upstream planning

In the standard initialization, the OEM proposes dates that myopically optimize his intradomain problem. To do this, all release dates rd_j are first set to their static, non-varying values rd^*_j. Based on this relaxed problem, the OEM computes his schedule first. To do this, first a backward and forward pass is applied to set the bounds for each activity j—the earliest (latest) start dates es_j (ls_j), the earliest (latest) finish dates ef_j (lf_j), and the propagated due dates pdd_j. Then, the DS solving module calculates a solution (setting start dates sd_j and finish dates fd_j) by letting the underlying GA run for a predefined amount of time. The proposed dates are then derived from the related right-aligned schedule. The suppliers translate the proposed dates into downstream-related due dates, solve their own scheduling problem and report the realized finish dates to the OEM. As previously discussed, the OEM in turn converts these finish dates into upstream-related release dates and recalculates his schedule.

In the literature, upstream planning often leads to high costs in the suppliers' domains. For example, Dudek and Stadtler (2005) assume a Master Planning problem where backlogging is not allowed but suppliers have to use costly overtime capacity to fulfill a manufacturer's initial request. It is worth highlighting that the situation is entirely different for our two-tier business case with parties relying on the DS Optimizer. As mentioned previously, the DS Optimizer calculates schedules that respect capacity and activity precedence constraints at the cost of lateness. Additional overtime capacity is usually not explicitly modeled by the RCPSP.[5] Thus, the suppliers are not able to guarantee to fulfill the OEM's initial request. The trivial strategy of proposing very early dates will usually not lead to satisfying results for the OEM. More important than absolute values are the implied relative priorities of the proposed dates. By transmitting proposed dates, the OEM indicates to his suppliers which activities are more and which are less important. However, if the decision on the delivery sequence is myopically made by the OEM, the suppliers' counterproposals cause lateness in the OEM's own schedules. Simply put, the suppliers convert an—on first perception—advantageous sequence of delivery into a sequence that ultimately leads to lateness for the OEM's customers. The goal of the customized DEAL framework can be stated as *finding a delivery sequence that causes minimal lateness in the OEM's domain and can be executed efficiently by the suppliers*.

[5] It should be noted, however, that there are two ways to implicitly model overtime capacity. First, additional modes with short duration and higher mode costs can be introduced. However, using modes to model overtime is not regarded as the standard business case. Second, deallocation of activities allows one to compute schedules without lateness at the cost of deallocated activities. The planner has then to consider additional means, such as overtime, to fulfill the deallocated customer orders. As mentioned previously, deallocation capabilities of the DS Optimizer have not been used as the coordination mechanism was defined to aim primary at the construction of interorganizational feasible schedules.

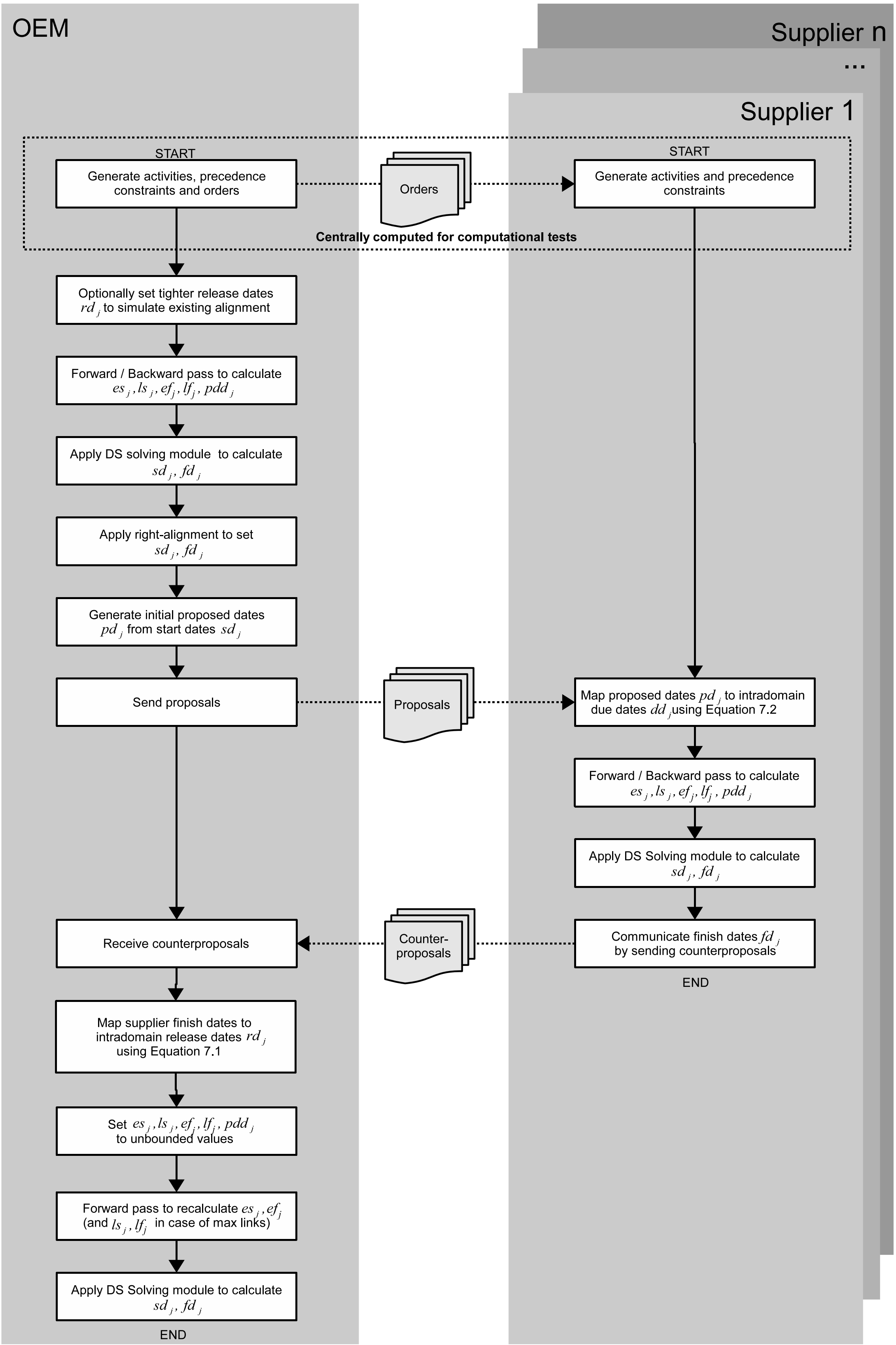

OEM
Supplier n
...
Supplier 1
START
Generate activities, precedence constraints and orders
Orders
START
Generate activities and precedence constraints
Centrally computed for computational tests
Optionally set tighter release dates rd_j to simulate existing alignment
Forward / Backward pass to calculate $es_j, ls_j, ef_j, lf_j, pdd_j$
Apply DS solving module to calculate sd_j, fd_j
Apply right-alignment to set sd_j, fd_j
Generate initial proposed dates pd_j from start dates sd_j
Send proposals
Proposals
Map proposed dates pd_j to intradomain due dates dd_j using Equation 7.2
Forward / Backward pass to calculate $es_j, ls_j, ef_j, lf_j, pdd_j$
Apply DS Solving module to calculate sd_j, fd_j
Receive counterproposals
Counter-proposals
Communicate finish dates fd_j by sending counterproposals
END
Map supplier finish dates to intradomain release dates rd_j using Equation 7.1
Set $es_j, ls_j, ef_j, lf_j, pdd_j$ to unbounded values
Forward pass to recalculate es_j, ef_j (and ls_j, lf_j in case of max links)
Apply DS Solving module to calculate sd_j, fd_j
END

Figure 7.2: Overview of initialization phase of DEAL's customized version.

Figure 7.2 illustrates the above initialization procedure. It substitutes the generic "initial solution" process of Figure 4.5 in Section 4.3. Two more issues should additionally be highlighted. As can be seen, forward and backward passes are applied multiple times. For the sake of completeness, it should be mentioned that before reapplying such a method, the dates es_j, ef_j, ls_j, lf_j and pdd_j need to be reset to their unbounded values for all $j \in J^e$. Otherwise these bounds would only get tighter and tighter but not reflect the current (counter-) proposal. In order to simulate an existing alignment, initial release dates can be precomputed in addition. This issue is discussed in the following subsection.

7.3.2 Simulation of an existing alignment

In practice, a previously good alignment between OEM and suppliers (e.g., as part of a frame contract) can become outdated by sudden events, such as machine breakdowns. A question worth investigating is to what extent the DEAL mechanism supports partners reacting to such scenarios. We simulate a *good alignment* by centrally applying the DS Optimizer to the interorganizational problem, formulated as a single RCPSP instance. The resulting start dates of activities at the boundary between OEM and suppliers are set as release dates in the OEM's initial subproblem. These dates help the initialization procedures of the DS Solving module (cf. Section 6.1.2) to construct good activity lists at the beginning.

However, capacity shortages or changing external-related due dates are introduced as well into the separated problem instances. Hence, the precomputed release dates (simulating a good alignment) don't fit any more. It is continued with the above described planning procedure—both methods are in fact upstream planning methods. However, when calculating his upstream proposal, the OEM has now to respect the centrally computed, outdated, release dates.

7.4 Evaluation

This section is about customizing the generic evaluation schemes presented in Section 4.5 (For a general overview, how evaluation is embedded in the DEAL framework, see also Section 4.9). We assume that most individuals are acceptable for the suppliers. On the considered short-term level, suppliers are supposed to support the OEM's proposals even though these proposals cause high setup times or costs, mode costs or makespan within their local planning problems. However, as already discussed, delivery dates to external customers are considered as deadlines in the current implementation. Recall that deadlines are considered top-level objectives. For reasonable values of these bounds, the supplier will thus compute acceptable schedules irrespective of the OEM's proposals and a single-ranking scheme as described in Section 4.5.1 would be sufficient. Unfortunately, due to limited runtime or if deadlines were are set to early, a supplier's local optimization method might not be able to compute a schedule without any deadline violation. To counteract such cases, we implemented the double-ranking scheme of Section 4.5.2. The top-level ranking, $v_e(p)$, reflects a supplier's e weighted sum of total and maximum violation of deadlines (the same weights as for the standard DS Optimizer total and maximum lateness objective are used). The second-level ranking, $\omega_e(p)$, reflects the weighted sum of a suppliers's makespan, setup times, setup costs and mode costs. To calculate the sum, the weights of the supplier's objective function (cf. Equation 5.39) are used. Standard lateness (i.e., due date violation) is not included, since it does not relate to external customers, referring only to due dates imposed by the OEM. As discussed in Section 4.5, these costs are not accountable by the supplier, as a late delivery actually harms the OEM. The OEM's top-level ranking, $\upsilon_e(o)$, reflects the weighted sum of total and maximum lateness. As for the suppliers, the second-level ranking $\omega_e(p)$ reflects the OEM's weighted sum of makespan, setup times, setup costs and mode cost.

Minor changes are applied to the generic scheme of Section 4.5 for eliminating duplicates. Preliminary experiments suggested that the elimination of duplicates is necessary to ensure a diverse population and to reduce the risk of premature convergence to a local optimum. Two different kinds of duplicates might exist from the OEM's perspective: Individuals that have similar upstream-related release dates but might be differently evaluated[6] and individuals that are similarly evaluated but have different upstream-related release dates. According to its duplicate type, an individual suffers a different degradation of its ranking. If two individuals are equally evaluated, the older one is preferred, which allows the computation of more promising proposals (see also Section 7.8). Summarizing, we get the following ranking scheme to sort a population of complete individuals.

1. **Sort according to nondominated solutions regarding suppliers' and OEM's acceptance.**

$$p \succ q \Leftrightarrow (\forall e \in E : \upsilon_e(p) \leq \upsilon_e(q)) \wedge (\exists d \in E : \upsilon_e(p) < \upsilon_e(q))$$

2. **If indifferent, sort according to the OEM's second-level preferences.**

$$p \succ q \Leftrightarrow \omega_0(p) < \omega_0(q)$$

3. **If indifferent, sort according to the suppliers' second-level preferences.**

$$p \succ q \Leftrightarrow (\forall e \in E \backslash \{0\} : \omega_e(p) \leq \omega_e(q)) \wedge (\exists d \in E \backslash \{0\} : \omega_e(p) < \omega_e(q))$$

4. **If indifferent, sort according to the time when an individual was completed.**

$$p \succ q \Leftrightarrow \text{The construction of } p \text{ was completed earlier}$$

After the population was sorted, a second duplicate check is performed. The OEM traverses the sorted list of individuals from the best to the worst. For each individual, lower-ranked individuals are sought with similar upstream-related release dates. If such cases exist, then the lower-ranked individuals are shifted to the end of the list. After having performed the duplicate check, the position in the final list expresses the final, joint ranking used by subsequent selection. To actually choose the mating pool, a tournament selection is carried out, as already explained in Section 4.5. For parallel and asynchronous coordination, the mating pool is steadily refilled and new children are constructed to maintain a constant queuing size, as discussed in Section 4.9.

As a possible extension, upper acceptance bounds for different objectives are imaginable. For example, suppliers might accept only a limited increase of setup time. If above a certain level, setup time could be considered an unacceptable consumption of resources. Such excess values

[6] As discussed in Chapter 6, the DS Optimizer's underlying GA is based on probabilistic decisions. Thus, different intradomain schedules are generated if the DS solving module is applied multiple times to the same RCPSP instance. In general, a schedule can only be reproduced if the same random seed was taken.

could additionally be considered when calculating the top-level ranking. In such a case, also the underlying local optimization methods should be adapted accordingly. Regarding the objective function of the DS Optimizer, such an adaptation was realized quickly. However, new objectives actually require changes to various mutation operators of the DS Optimizer that are currently designed to minimize the original objective function. Moreover, criteria of the greedy heuristic for automated mode selection are affected. Changing the related implementation was considered a major effort beyond the scope of this thesis. The computational results thus build on the aforementioned assumption that suppliers focus on the reduction of lateness of external customer orders, but are indifferent to increases in other costs.

7.5 Construction of solutions

The starting point for constructing new proposals is the selected individuals in the mating pool (the parents). For each parent, a new child is constructed. Construction might require setting up and solving an intermediate problem. To further distinguish among the problem instances of parent, intermediate and child, we introduce the additional superscripts (p), (i) and (c). If the setting is clear, the superscripts are omitted. We first describe the general process of constructing a solution before going into the details in the subsequent sections. The basic principle underlying the construction of a child is upstream planning, where the OEM proposes dates close to the release dates of the parent. To be more specific, the construction of a child requires the following steps.

1. The OEM computes new proposed dates $pd_j^{(c)}$ for all activities $j \in J^0$. To do this, several operators are available to analyze existing solutions. To compute proposed dates, these operators might require the setup and solution of an intermediate problem (i). The different operators are presented in Section 7.7.

2. The OEM proposes the dates to all suppliers $e \in E\backslash\{0\}$, who calculate the corresponding due dates $dd_j^{(c)}, \forall j \in J^e$ according to Equation 7.2. It should be emphasized that every supplier gets a separate proposal (starting a separate communication thread) containing only those dates that are relevant for his planning domain. The suppliers update the propagated due dates $pdd_j^{(c)}$ by applying a backward pass. As already discussed in Section 4.4.2, for some suppliers proposed dates might not mean any change to their interface. In this case, the OEM's child is reconnected to an existing communication thread. Analogously, the suppliers perform an additional check for redundancy. A detailed discussion of this topic follows in Section 7.9.

3. Each supplier calculates a new schedule by applying his DS Optimizer and derives the finish dates $fd_j^{(c)}$ for activities $j_e \in J_D^e$ that precede the OEM activities. Each supplier counterproposes his finish dates to the OEM.

4. The OEM maps all finish dates to release dates $rd_j^{(c)}$ using Equation 7.1 and applies a forward pass to calculate earliest start and finish dates es_j, ef_j for all $j \in J^0$.

5. The OEM calculates the child's schedule by applying the DS Optimizer to the updated RCPSP instance.

Constructing a child can vary in the analysis of the problem instance p and the derivation of the new proposed dates $pd_j^{(c)}$. The proposal generating operators are discussed in Section 7.7. However, steps 2–5 of the above sequence remain the same for all operators. Figure 7.3 illustrates the above procedure. It is the customized version of the generic construction template shown in Figure 4.9 of Section 4.4.2.

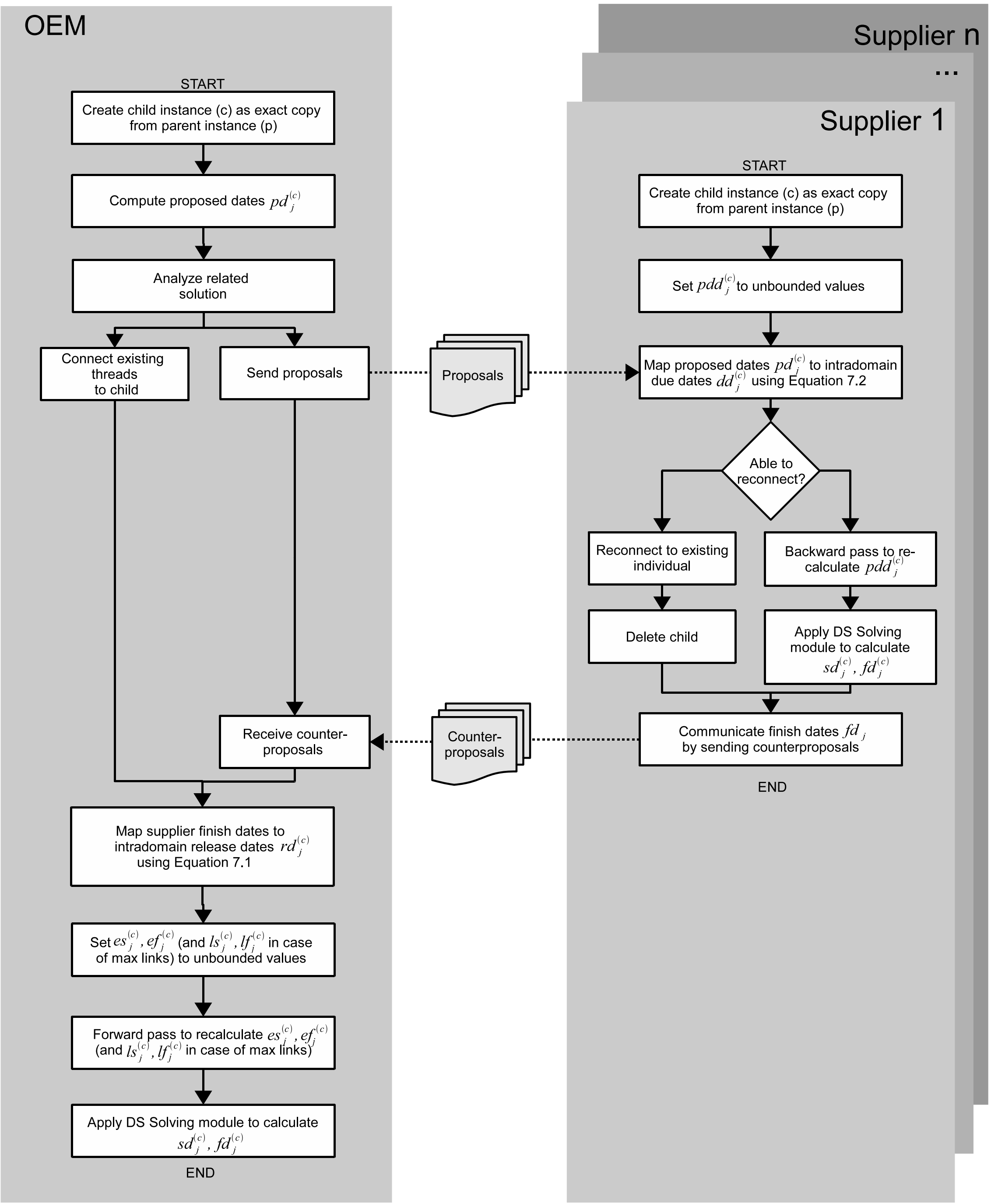

OEM
START
Create child instance (c) as exact copy from parent instance (p)
Compute proposed dates $pd_j^{(c)}$
Analyze related solution
Connect existing threads to child
Send proposals
Proposals
Receive counter-proposals
Counter-proposals
Map supplier finish dates to intradomain release dates $rd_j^{(c)}$ using Equation 7.1
Set $es_j^{(c)}$, $ef_j^{(c)}$ (and $ls_j^{(c)}$, $lf_j^{(c)}$ in case of max links) to unbounded values
Forward pass to recalculate $es_j^{(c)}$, $ef_j^{(c)}$ (and $ls_j^{(c)}$, $lf_j^{(c)}$ in case of max links)
Apply DS Solving module to calculate $sd_j^{(c)}$, $fd_j^{(c)}$
END
Supplier n
...
Supplier 1
START
Create child instance (c) as exact copy from parent instance (p)
Set $pdd_j^{(c)}$ to unbounded values
Map proposed dates $pd_j^{(c)}$ to intradomain due dates $dd_j^{(c)}$ using Equation 7.2
Able to reconnect?
Reconnect to existing individual
Backward pass to re-calculate $pdd_j^{(c)}$
Delete child
Apply DS Solving module to calculate $sd_j^{(c)}$, $fd_j^{(c)}$
Communicate finish dates fd_j by sending counterproposals
END

Figure 7.3: Overview of construction phase OF DEAL's customized version.

7.6 Coordination example

This sections aims at complementing the above discussion by providing a concrete example.[7] Assume an OEM who wants to produce two lawnmowers (Mow). A lawnmower consists of five components: A motor (M), a tank (T), a set of wheels (W), a handle (H) and a chassis (C). The components are delivered by two suppliers: Supplier Plastic, producing the sets of wheels and the tanks, and Supplier Steel, producing chassis, motors and handles. When changing from wheel to tank production, Supplier Plastic needs a sequence-dependent setup (S). In a first production step, the OEM assembles the chassis and the wheels (CW) and the motor and the tank (MT). Then, all components are assembled to a lawnmower. In addition to the OEM, Supplier Steel delivers to another, external, customer by producing an external good (Ext). Figure 7.4 illustrates the ideal situation.

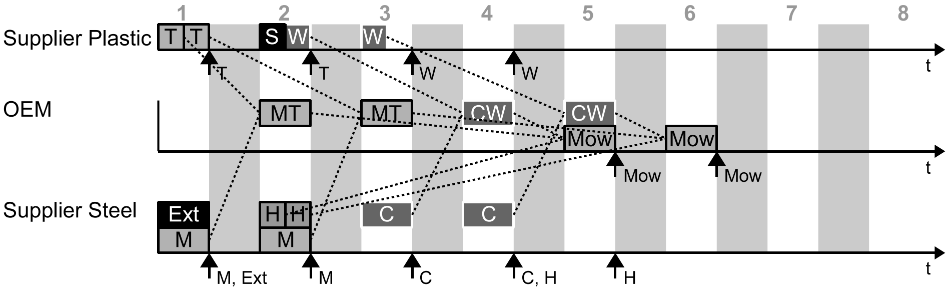


Figure 7.4: Coordination example—ideal situation.

It can be seen that OEM and Supplier Steel own two resources each, while Supplier Plastic only handles a single resource. Multiple modes are not present, and each activity requires a single resource, as illustrated in the figure. For better legibility, the example consists of 8 periods, separated by breaks during which production is not possible. Activities either require a full or a half period length. Dotted lines illustrate the pegging net and vertical arrows the due dates. For example, the lawnmowers (Mow) shall be finish at the end of periods 5 and 6, respectively. Therefore, Supplier Plastic needs to finish the production of the tanks (T) at the end of periods 1 and 2 and the production of the wheel-sets (W) at the end of periods 3 and 4, respectively. The schedules of Figure 7.4 have been constructed by upstream planning. That is, the OEM first applies his DS solving module, and then proposes delivery dates implied by the related right-aligned schedule, as discussed previously (cf. Figure 7.2). Since the suppliers can fulfill all proposed dates (respectively the related due dates), there is no need for coordination in this idealized setting.

A need for coordination results if the proposed dates can not be fulfilled by the suppliers. Figure 7.5 depicts a situation where Supplier Steel cannot produce the motor as promised due to a machine breakdown during the first two periods. As a consequence, the lawnmowers are delivered to the ultimate customer too late. For the remainder of this section we assume this to be the initial situation, generated by the initialization procedure, cf. Figure 7.2.

The machine breakdown can be counteracted by changing the sequence of delivery and production. Producing chassis (C) and wheel-sets (W) before before handles (H) and tanks (T) makes a nondelayed delivery of lawnmowers possible. Figure 7.6 shows the related schedules. However, even in this small example, it is a hard task to retrieve these sequences manually as each party has only partial information. On the one hand, the suppliers have no indication to change the

[7] For didactic purposes, this example bases on an ideal, trivial scheduling problem and was not used for the computational evaluation.

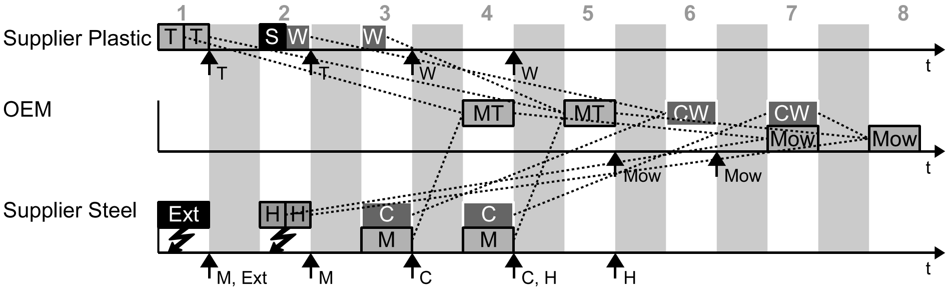


Figure 7.5: Coordination example—initial situation.

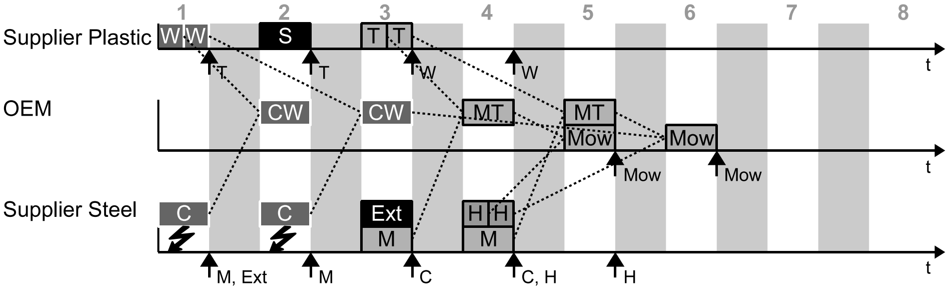


Figure 7.6: Coordination example—a possibility to counteract the machine breakdown.

sequence as most due dates are fulfilled in the initial situation (cf. Figure 7.5). Only the delivery of the motor is delayed, which is unavoidable from the perspective of Supplier Steel. On the other hand, the OEM is not aware of the suppliers' resource situation. Moreover, the schedules of Figure 7.6 imply that Supplier Steel needs to delay the delivery of the good relating to the external customer (Ext). As assumed in Section 7.1, suppliers are not willing to do this. Additionally, Supplier Plastic needs to run a longer sequence depending setup activity (S) when switching from tank to wheel production.

As discussed in Section 7.5, DEAL implements different operators that analyze the OEM's current situation in order to generate new proposals. These operators represent different rules-of-thumb. One of these rules (cf. Subsection 7.7.2.5) works in the following idealized way.

First, all release dates confirmed by the suppliers that match the initially proposed dates are relaxed in the OEM's planning problem, i.e. set to the beginning of the planning horizon. Then, the OEM computes the related left-aligned schedule by applying the left-alignment heuristic presented in Section 6.2 (also a rescheduling by calling the DS solving module is possible). The generated schedule is shown in Figure 7.7. Dotted lines of the pegging net are only drawn for nonrelaxed release dates. It can be seen that the "CW" activities have been placed very early and that the OEM's lateness decreases to zero.

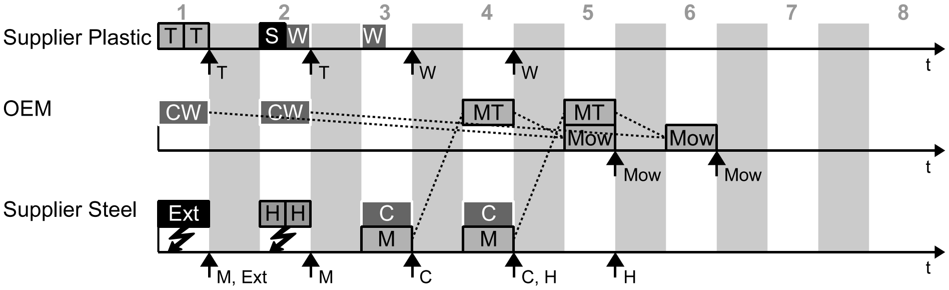


Figure 7.7: Coordination example—left-aligned version of the OEM's relaxed problem.

To reduce the earliness, a right-alignment is applied subsequently. As discussed in Section 6.2, the right-alignment heuristics is not allowed to generate additional lateness when shifting activities to the right. Figure 7.8 shows the related schedule. The OEM uses this schedule for deriving new proposed dates which are transmitted to the suppliers. The figure also shows the implied suppliers' due dates.

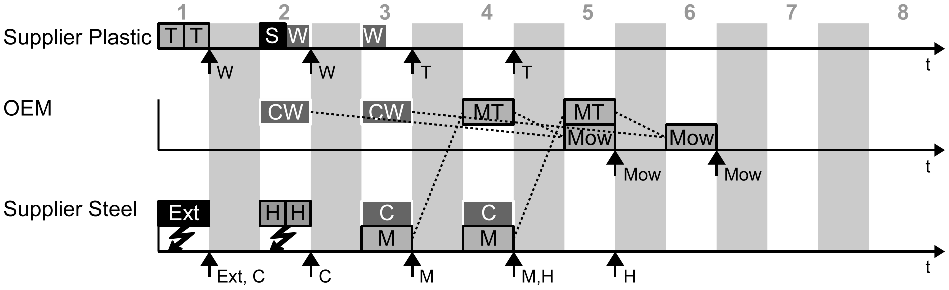


Figure 7.8: Coordination example—left-right-aligned version of the OEM's relaxed problem.

For computing counterproposals, the suppliers reschedule their production according to the new due dates by calling their DS solving module. In the example, Supplier Plastic can fulfill all due dates, although this implies are larger sequence depending setup time[8] as shown in Figure 7.9.

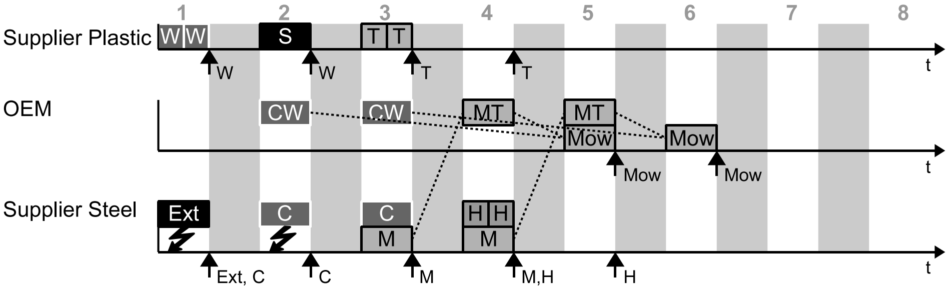


Figure 7.9: Coordination example—rescheduling of the suppliers.

When rescheduling his production, Supplier Steel prioritizes his external customer due dates as discussed in Section 7.1. Therefore, not all new due dates imposed by the OEM can be fulfilled. Thus, the OEM needs to reschedule his production according to the counterproposal of Supplier Steel (the counterproposals of the two suppliers are aggregated by using Equation 7.1). As can be seen in Figure 7.10, the delay of lawnmowers can only be partially decreased as a consequence. However, an improvement that is acceptable by all parties was generated. Comparing the final schedules (Figure 7.10) with the initial ones (Figure 7.5) we can make the following observations:

1. Regarding their top-level ranking (reflecting the violation of deadlines) the suppliers are indifferent between the two solutions.
2. The OEM prefers the solution of Figure 7.10, as due dates to external customers are less violated.
3. Regarding the suppliers' second-level preferences, the solution relating to Figure 7.5 with less setup time should be preferred.

[8] We assume that violating due dates is penalized more than larger setup times in the objective function of Supplier Plastic.

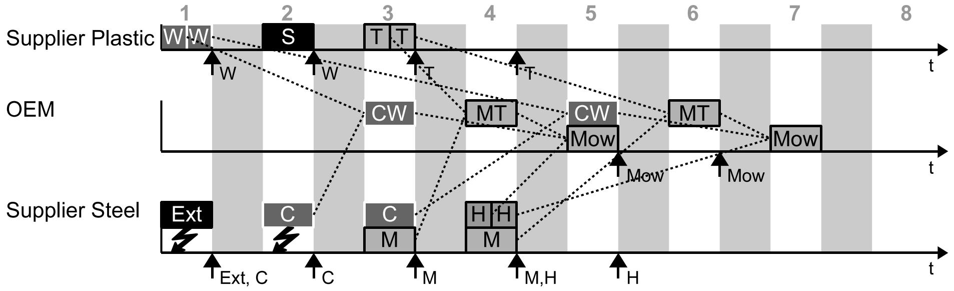


Figure 7.10: Coordination example—improved feasible schedule.

According to the ranking scheme of Section 7.4, the parties thus prefer the schedules of Figure 7.10. If the OEM's lateness was not reduced, the initial situation (Figure 7.5) would have been preferred as it implies less setup time for Supplier Plastic. If Supplier Steel would have violated the deadline relating to his external customer (Ext), either because local runtime was short or the scheduling heuristic does not support a prioritization of deadlines, the initial situation would have been preferred according as well. Thus, the ranking scheme serves as a safety mechanism for the suppliers, as it steers the search process in the right direction. Moreover, the OEM may finally only choose among solutions accepted by all suppliers, as discussed in Chapter 4.

In general, we can not expect that a single operator totally resolves all drawbacks of the initial due-date-setting. However, by applying several operators iteratively, the initial situation can be improved gradually, as indicated by our computational tests presented in Chapter 8.

7.7 Proposal generating operators

In the following, we will present operators that enable the OEM to construct new proposals by analyzing his complete individuals (not being under construction) and the related schedules. The operators try to derive a promising setting of proposed dates for reducing the lateness incurred by the OEM's customers. As discussed above, the suppliers will try to comply with these dates to the extent possible, but are allowed to deviate from the proposal if necessary. Each operator follows one of four basic principles:

1. **Propagation of lateness:** Proposed dates are calculated by propagating the OEM's lateness w.r.t. external customers to upstream-related activities by means of backward passes. A detailed discussion follows in Subsection 7.7.1.

2. **Left-right-alignment of a partially relaxed problem instance:** An intermediate problem instance is generated by partially relaxing release dates of the parent instance. Lateness to ultimate customers is then reduced by applying a left–right-alignment (cf. Section 6.2) to the intermediate problems. The child's proposed dates are derived from the start dates of the subsequently right-aligned intermediate problem. These operators are discussed in Subsection 7.7.2.

3. **Rescheduling:** An intermediate problem instance is generated as a partially relaxed copy of the parent instance. By applying the DS Optimizer to the intermediate instance, we search for better sequences of activities. The child's proposed dates are derived from the start dates of the intermediate instance. Subsection 7.7.3 deals with this kind of operators.

4. **Crossover:** Multiple existing individuals are used to derive new proposed dates, more details are presented in Subsection 7.7.4.

In general, the construction of a child is subject to the following trade-off. If, on the one hand, the OEM is granted too much freedom to readjust the proposed dates, a solution similar to initial upstream planning will result. If, on the other hand, possibilities for changing proposed dates are too limited, the schedules will not change and the evolutionary process will get stuck. Thus, the fundamental idea is to restrict the OEM to those changes that have a large relative impact on his objective value with regard to the change of proposed dates. A similar concept also underlies the work of Dudek and Stadtler (2005). Here, deviating from the current solution is explicitly penalized in the objective function of the proposal-issuing planning domain. However, such an explicit formulation is not possible for the DS Optimizer on the basis of the current schedule generation scheme (more details on this topic will be presented in Subsection 7.7.5). As a remedy, several operators have been implemented, representing different "rules of thumb" for calculating a proposal. These rules of thumb have been developed on the basis of intuition and results of preliminary experiments. Regarding their performance, there is no guarantee in form of a mathematical proof or formula.

An operator might not always be able to calculate proposed dates that substantially differ from previous release dates. In such cases another operator is applied to avoid producing redundant proposals and to keep runtime requirements as low as possible. In addition, a self-adaptation of operators is supported. More details are given in Section 7.8.

CX	Component crossover
FDS	Fix delayed activity-sequence
FS	Fix sequence of all activities
FWD	Forward scheduling
ICL	Insert components most left
ICR	Insert components randomly
IDD	Insert due dates
LA	Left-alignment
LX	Linear crossover
PDD	Propagate due dates
RA	Right-alignment
RC	Rearrange activities in connected components
RMD	Relax most delayed (upstream-related) release dates
RMP	Relax most promising (upstream-related) release dates
RMS	Relax minimum slack (upstream-related) release dates
RNB	Relax nonbottleneck upstream-related) release dates
RRD	Relax all (upstream-related) release dates
SSB	Set start-date bounds

Table 7.2: Abbreviations of operators' subprocedures.

For the sake of clarity we will abbreviate the different subprocedures that define an operator. Table 7.2 presents an overview of used abbreviations. An operator is defined by a sequence of subprocedures, e.g. FS | PDD means that the sequence of activities is fixed before a propagation of due dates takes place. The single operators and subprocedures are presented in the following subsections. For easy reference, an operator's subprocedure sequence is contained in the heading. Table 7.3 gives an overview of every subprocedure sequence, the related subsection and the page number.

Sequence of subprocedures	Subsection	Page number
CX	7.7.4.3	140
CX I FWD I LA I RA	7.7.4.3	140
FS I ICL I FWD I RA	7.7.3.2	135
FS I ICR I FWD I RA	7.7.3.2	135
FS I PDD	7.7.1.2	128
FS I RMD I LA I RA	7.7.2.3	131
ICL I FWD I RA	7.7.3.2	135
ICR I FWD I RA	7.7.3.2	135
IDD I FWD I RA	7.7.3.3	136
LX	7.7.4.1	138
LX I FWD I LA I RA	7.7.4.1	138
PDD	7.7.1.1	127
PLX	7.7.4.2	139
PLX I FWD I LA I RA	7.7.4.2	139
RA I FDS I PDD	7.7.1.3	129
RC I FWD I RA	7.7.3.1	134
RMD I LA I RA	7.7.2.2	131
RMS I LA I RA	7.7.2.4	131
RNB I LA I RA	7.7.2.5	132
RMP I LA I RA	7.7.2.6	132
RRD I LA I RA	7.7.2.1	130
SSB I FWD I RA	7.7.3.4	137

Table 7.3: Overview of operator subprocedures, related subsections and page numbers.

7.7.1 Propagation-based mutation operators

A simple approach to calculate new promising release dates is to propagate the lateness through the activity network by means of backward passes. We implemented three different operators that rely on the propagation principle.

7.7.1.1 Propagate due dates (PDD)

Propagation of due dates is the most simple operator to achieve a crude alignment between the OEM's and the supplier's schedules. By applying a backward pass (see equations 5.8, 5.9 and 5.32), the OEM calculates propagated due dates pdd_j and latest finish dates lf_j. For the standard problem 5.1–5.5, we can calculate the required **infinite start date** isd_j of an activity as

$$isd_j = \min\left\{lf_j^{(p)}, pdd_j^{(p)}\right\} - d_j. \tag{7.3}$$

The infinite start date denotes the latest possible time an activity has to start to obtain a nonlate schedule (i.e., a schedule with a total lateness of zero) under infinite capacity. The proposed dates are then set to these infinite start dates. If calendars are included, the gross-duration rather than the net-duration needs to be subtracted in Equation 7.3. Recall that the propagation of due dates works on an infinite basis, it considers neither the current schedule nor capacity constraints.[9] The results are proposed due dates that give the suppliers a very rough view of the OEM's situation. If

[9] As propagated due dates are solely dependent on calendars and precedence constraints, they need actually only be computed once. However, as will become apparent, other operators might change the precedence constraints. To assure that the propagated due dates match the current precedence constraints, a recomputation is performed whenever necessary. Since the numerical test cases contain no max-links, the computational overhead of this strategy is negligible. For problem instances containing max-links, a more advanced implementation supporting a backup of already computed propagated due dates is recommended however.

the OEM's own due dates have been altered due to changing customer preferences, this operator realigns the network of activities to the new situation, however, without considering capacity. For most coordination problems, this strategy is too simple, since bottlenecks arise *because of* limited capacity. An advantage is that backward passes can usually be computed very quickly.

A further difficulty arises if release dates of OEM activities are not determined by a supplier's counterproposal, but by other technological constraints. Considering lateness related to technological constraints during the backward passes is not useful. Even if the supplier delivered earlier, lateness caused by technological constraints would not change. To reduce the propagation to lateness actually caused by a supplier, we apply a forward pass that only considers technology-related release dates rd_j^* that are independent of a supplier's delivery. The forward pass thus calculates the earliest start dates that are invariant to the results of the coordination process. Adding the activity's gross-duration to this earliest start date yields the related earliest finish date. If a due date was set prior to this earliest finish date, a lateness will result, irrespective of a supplier's delivery. Such due dates (or deadlines) are then temporarily set to the related earliest finish date, if applicable. The aforementioned backward pass is then applied after this preprocessing.

7.7.1.2 Fix sequence and propagate due dates (FS | PDD)

One way to roughly consider capacity constraints is to fix the sequence (FS) of activities before applying the backward pass. Fixing the sequence means introducing artificial finish–start links with a minimum time lag of zero between activities succeeding each other on similar resources. For each resource, a list of activities currently scheduled on the resource is computed. Then, for every activity j of a list, the earliest succeeding activity k contained in the same list is searched, with $fd_j \leq sd_k$. If such an activity can be found, an artificial link between j and k is introduced.[10] Furthermore, unused modes are temporarily removed from the model. Otherwise, forward and backward passes, which always consider the "best combination" of modes neglecting capacitative constraints do not work as intended and can lead to inconsistencies.

After the sequence is fixed, a backward pass is applied to calculate the proposed dates. Again, technological constraints can be considered by a forward pass and a subsequent correction of due dates. This measure is applied before the sequence is fixed to limit the left shifting of the due dates. After the proposed dates have been computed, the artificial links are removed.

In general, fixing the sequence comes with two major disadvantages: First, it only provides a crude approximation of available capacity. Especially for variable-capacity profiles, the proposed dates are likely to be either too late or too early and do not guarantee a subsequent calculation of a feasible schedule respecting capacity without lateness. Second, as will become apparent later, many coordination problems result because of a delivery sequence that does not match the constraints of the suppliers and OEM. Obviously, the operator has only limited ability to change the sequence. Nevertheless, the proposed dates may provide more information to the suppliers than the infinite propagation of the previous section.

[10] Obviously, for multi-capacity resources, this strategy does not fix the whole sequence. To do this, minimum time lags need actually be introduced between an activity and every successor on the same machine, which—for large instances—generates an significant amount of additional precedence constraints and might slow down forward and backward passes. Instead, our strategy only introduces new constraints between immediate successors. Hence, on multicapacitated resources not the whole sequence, but only subsequences are fixed. Preliminary experiments suggested that this fixation is most promising for approximating which release dates should actually be altered to decrease lateness to ultimate customers.

7.7.1.3 Fix delayed sequence and propagate due dates (RA | FDS | PDD)

There exists also a middle way between propagation without fixing any sequence and a propagation with fixing the complete sequence. The operator also employs a single right-alignment and works as follows.

1. An intermediate problem instance (i) is constructed as the right-aligned version of the parent (p), using the right-alignment heuristic presented in Section 6.2.

2. Lateness relating to technological constraints is tackled by a forward pass and subsequent due date correction, as discussed above.

3. Late activities are sorted in order of monotonely decreasing lateness. Only those activities that contribute to the first α percent of total lateness are regarded as critical. For the remaining activities, the due dates are relaxed in the intermediate problem instance. That is, $dd_j^{(i)} := fd_j{}^{(p)}$.

4. A backward pass is applied to propagate the partially relaxed due dates.

5. A set of fixation candidates is computed, consisting of all activities with violated propagated due dates, $pdd_j < fd_j$.

6. The sequence is fixed (as with the previous operator), while only activities in the candidate set are considered.

7. A backward pass is applied to recalculate propagated due dates and infinite start dates.

8. The proposed dates are derived as $pd_j^{(c)} = \min\left\{sd_j^{(i)}, isd_j^{(i)}\right\}$, where $isd^{(i)}$ is the infinite start date according to the partially fixed schedule (cf. Equation 7.3).

In other words, the operator does the following. It keeps the sequence of nonlate activities and the sequence of "late" activities that violate the propagated due dates ("fix delayed sequence", FDS). However, proposed dates of late activities are set earlier. The parameter α allows further tuning of the operator.

7.7.2 Alignment-based mutation operators

The above presented propagating operators support the domains in finding a crude alignment between the schedules. Nevertheless, the infinite treatment of actually finite capacity and the limited possibilities for changing sequences of delivery dates have a limiting effect. The alignment heuristics presented in Section 6.2 can be used as a means to overcome the shortcomings of pure propagation. Different proposals can be computed by aligning only a subset of activities (henceforth called the set of **activities to align**). The remaining activities are fixed at their current position. To be more precise, a **left-right-alignment** consists of the following steps.

1. Construct an intermediate problem instance as an exact copy of the parent.

2. Identify the critical set of *activities to align* by some rules of thumb, to be presented in the following subsections 7.7.2.1–7.7.2.6.

3. Compute the set of **activities to fix**. An activity to be fixed is neither within the set of *activities to align* nor a direct or indirect successor of such an activity.

4. Fix all *activities to fix* at their current position. The fixing is achieved by temporarily setting the release date to the currently selected start date $rd_j^{(i)} = sd_j^{(p)}$ and the deadline to the currently selected finish date $dl_j^{(i)} = fd_j^{(p)}$.[11]

5. For all *activities to align*, relax release dates $rd_j^{(i)}$. Relaxing the release dates means setting them to rd_j^*, which refers to static technological constraints.

6. Propagate the release date relaxation and the fixing through the network by applying forward and backward passes.

7. Left-align (LA) the schedule w.r.t. to the relaxed release dates. During the left-alignment lateness usually decreases.

8. Right-align (RA) the schedule in such a way that no new lateness results (cf. Section 6.2).

9. Remove the fixing by setting $rd_j^{(i)} = rd_j^{(p)}$ and $dl_j^{(i)} = dl_j^{(p)}, \forall j \in J^0$. Update the earliest start dates by a backward pass.

10. Again right-align the schedule to close remaining gaps, cf. Section 6.2.

11. Set the proposed dates to the currently selected start dates: $pd^{(c)} = sd^{(i)}, \forall j \in J_U^0$.

It turned out that the decision of which release dates to relax is crucial. We adopted the following rules of thumb to work on intermediate problem instances.

7.7.2.1 Relax all upstream-related release dates (RRD | LA | RA)

The most basic approach is to relax all release dates (RRD) related to upstream activities. If upstream-related release dates are relaxed early enough, a subsequent left-right-alignment will decrease the lateness caused by suppliers to zero (if static release dates do not lead to problems). In contrast to fixing the sequence and propagating the due dates (cf. Section 7.7.1.2), the subsequent left-right-alignment truly respects the capacity. Moreover, a left-right-alignment can have a positive effect on the compactness of the schedule, as already mentioned in Section 5.9.4. However, the sequence of activities is likely to remain constant if many release dates are relaxed. By keeping the sequence, the operator generates proposed dates that pertain to a low-lateness feasible schedule for the OEM. Suppliers "correct" these dates using their DS Optimizer in the subsequent steps of constructing a solution. The hope is that iteratively relaxing the release dates followed by left-right-alignment and correction by suppliers can lead to aligned schedules.

Relaxing all upstream-related release dates with a subsequent left-right-alignment is likely to change the position of most activities, but unlikely to change the sequence of activities on the resources. For interorganizational problems that require a certain sequence of delivery, this operator might have limited potential as it keeps most parts of the sequence unchanged. Instead of focusing on those release dates that actually cause the biggest problems, proposed dates for all activities are derived. It might be advantageous to focus only on a **critical set** of release dates. The following operators implement different definitions of criticality.

[11] Fixing an activity is not related to fixing the sequence between two activities.

7.7.2.2 Relax release dates of the most delayed upstream-related activities (RMD | LA | RA)

An activity is either delayed because of its predecessor's late finish time or because of its resource's limited capacity. Lateness with regard to the computed *propagated due date* indicates the amount of lateness for which the current activity or its predecessors are responsible, even though the related products are not directly delivered to the customer. If the finish date fd_j of an activity j violates its propagated due date pdd_j and the activity itself is bounded by an upstream-related release date, it makes sense to request an earlier delivery from the responsible suppliers. This way, earlier slots on the resources may be found for scheduling the activity and its successors earlier. Thus, the idea is to compute the critical set of activities as the set of those activities that have an upstream-related release date and violate their propagated due date. It might be advantageous to further restrict the critical set to those activities with the highest violation of propagated due dates ("relax most delayed", RMD). Summarizing, we apply the following procedure.

1. Propagate due dates and deadlines through the network by performing a backward pass.
2. Sort the set of upstream related activities J_{UP}^0 in descending order of propagated due date violation.
3. Choose the first α percent of the sorted activities as the critical set and continue with the left-right-alignment procedure as described above.

The parameter α can be seen as a further tuning parameter, defining the trade-off between a proposal's locality and innovation potential. It should be noted that the relaxation of most delayed activity has the tendency to shift delayed activities to the beginning of the planning horizon. Instead of shifting all activities "a bit" as in the previous procedure, it is likely that only a few activities are shifted by a large amount.

7.7.2.3 Fix sequence and relax release dates of the most delayed upstream-related activities (FS | RMD | LA | RA)

The above procedure propagates lateness caused by predecessor constraints to current upstream-related release dates. However, if the OEM's production includes many stages, the potential of the procedure may be limited. A finite capacity of intermediate production stages can lead to delayed final products, although the propagated due dates of early production stages are respected. In order to still be able to relate lateness to upstream-related release dates, we fix the sequence (FS) by introducing artificial finish–start links as already described in Section 7.7.1.2. Then, the process continues as for the previous operator. A difference with the operator described in Section 7.7.2.1 is that—dependent on the parameter α—not all activities are relaxed. Instead, activities that are scheduled on resources with relatively small idle times have a higher chance of being realigned.

7.7.2.4 Relax minimum slack of upstream-related release dates (RMS | LA | RA)

This operator does not consider any lateness, but focuses on start dates of activities. Due to the OEM's limited resource capacity, not all activities are usually scheduled immediately at their upstream-related release dates. Instead, some activities are scheduled closer to their upstream-related release date while others are scheduled not so close. The difference, or slack, between the start date, sd_j, and release date, rd_j, can be regarded as an indicator of the importance of that activity. Relaxing activities with minimum slack (RMS) is supposed to lead to a larger delay reduction than relaxing activities with large slack. It should be emphasized that due to the combinatorial nature of the RCPSP, no general relationship of such type exists, however. Concluding, the operator works as follows.

1. Sort the set J^0_{UP} in order of increasing slack, $sd^{(i)}_j - rd^{(i)}_j$.
2. Choose the first α percent of the sorted activities as the critical set and continue with the left-right-alignment procedure as described above.

Again, α can be used a further tuning parameter.

7.7.2.5 Relax nonbottleneck upstream-related release dates (RNB | LA | RA)

The above operators enable the OEM to derive promising release dates by analyzing a previous local problem instance and the related local solution. Previous counterproposals, defining the *history of coordination*, are not taken into account however. The process can get stuck at a local optimum, as the OEM proposes similar release dates over and over and suppliers reply always with similar counterproposals. The effectiveness of proposals can be improved by analyzing previous counterproposals, with the goal to identify release dates that determine the bottlenecks at a supplier's side. One approach is to measure the difference between proposed dates and upstream-related release dates of the parent solution. We define a job's **amount of correction** in the parent solution, p, as

$$\Delta^{(p)}_j = rd^{(p)}_j - pd^{(p)}_j,$$

whereas $rd^{(p)}_j$ denotes the release date that results from the suppliers counterproposal to the proposed date $pd^{(p)}_j$ when constructing the parent individual. Note that $\Delta^{(p)}_j$ can also become negative if a supplier exceeds the OEM's requirements. Intuitively, a relaxation of release dates of activities with a large amount of correction is not promising, as the supplier is apparently not able to fulfill the resulting proposed dates. Instead, it is a promising strategy to relax only the nonbottleneck (RNB) release dates, i.e. to restrict the critical set to those activities with a low $\Delta^{(p)}_j$.[12] After having sorted all activities with upstream-related release dates in ascending order of $\Delta^{(p)}_j$, we restrict the critical set to the first α percent of the activities before continuing with the left-right-alignment procedure as described above.

7.7.2.6 Relax most promising upstream-related release dates (RMP | LA | RA)

Another possibility to estimate trends for setting release dates is to compare previous start dates. A start date can be regarded as the ultimate result of the OEM's proposal, suppliers' optimization runs and counterproposals, and OEM's optimization. Let the superscript (pp) denote the parent's parent. The difference $\Lambda_j = sd^{(pp)}_j - sd^{(p)}_j$ measures the effect of the previous mutation. If $\Lambda_j > 0$, the activity was started earlier; if $\Lambda_j < 0$, it was started later. If the previous mutation was successful—solution (p) is better ranked than (pp)—it might be advantageous to shift activities with $\Lambda_j > 0$ even earlier and activities with $\Lambda_j < 0$ even later in order to follow the historical trend. Hence, the idea is to relax the most promising release dates (RMP). We adopt the following proceeding.

[12] Of course, no general relationship of such type exists, but the operator represents a rule of thumb that might work in some cases.

1. Sort all activities of J^0_{UP} according to descending Λ_j.
2. If solution (p) is better ranked than solution (pp), relax the first α percent of the sorted set of upstream-related activities. If not, relax the last α percent in order to steer the search process in the opposite direction. Continue with a left-right-alignment as discussed above.

Again, we can regard α as a further tuning parameter.

7.7.3 Rescheduling-based mutation operators

The above realignmment-based operators have three central advantages: They respect finite capacity, are fast and keep most parts of the current activity sequence, increasing the effectiveness of a warm reboot at both the supplier's and the OEM's side. However, not altering the sequence can also become a disadvantage. As will become apparent later, many coordination problems in scheduling occur because of bad delivery sequence imposed by the OEM on his suppliers or vice versa. While the above shifting and propagation operators are able to roughly and quickly correct the mismatching between the planning domains, they are likely to fail in changing the position of activities in a fine-granular manner.

To enable the OEM to introduce small but important changes in the schedule, a third set of so-called rescheduling operators has been implemented. Rescheduling operators rely on the idea of partially relaxing an intermediate problem instance, rescheduling it in forward direction using the DS Optimizer (FWD) followed by a right-alignment (RA) in order to find new promising delivery dates. Also for the above alignment-based operators one could argue that, instead of a left-alignment, a reoptimization of the intermediate relaxed problem instance could be advantageous. Without objecting to this idea in general, the following arguments should be considered. If the release dates were relaxed in the same manner as described above, set to their static bounds, a reoptimization would have the effect of scheduling all relaxed activities and their successors very early. A subsequent right-alignment would be mandatory in order to compute meaningful proposed dates. Moreover, as the sequence of relaxed and reoptimized activities might not fit the calendar at the original position, the right-aligned schedule could come with major idle times. If only a few release dates are relaxed to achieve a limited deviation from the parent solution, a left-alignment is supposed to bring results similar to those of a reoptimization. Moreover, a left-right-alignment has the advantage of shorter runtime requirements. Especially when taking into account that the suppliers alter the OEM's proposal anyway, a left-right-alignment seems to be sufficient to find a crude alignment of schedules. We could verify this proceeding during our computational evaluation, cf. Chapter 8. Only for small test instances including sequence-dependent setup times, the left-alignment failed. This was due to the current implementation of the left-alignment heuristic that does not support a temporary increase of setup times, cf. Section 6.2. For such test instances, we additionally tested operators, where the left-alignment (LA) step was replaced by a forward rescheduling (FWD).

In general, when rescheduling, release dates should not be relaxed completely to their static bounds rd^*_j. A critical point is the estimation of suitable values for the relaxed release dates. If set too early, a left-right-alignment is supposed to bring similar results in shorter time as argued above. If set not early enough, the schedule will only change slightly or not at all, wasting runtime. The following operators try to estimate suitable values for relaxed dates based on the schedule of the parent individual.

7.7.3.1 Rearrange activities of connected components (RC | FWD | RA)

Imagine an undirected activity-on-node graph, where activities define the nodes and precedence relations the edges. Usually, this graph can be decomposed into several disjunct (weakly) connected components. With regard to production lots, a connected component mirrors the Bill of Materials on the lot level.[13] As mentioned previously, badly aligned activity sequences between OEM and suppliers lead to idle or setup times and eventually result in late delivery to the ultimate customer. The OEM should be provided with the functionality to recalculate the sequence. However, totally unbounded rescheduling would lead to results similar to upstream planning and even abandon previous rough alignments found by the propagating and shifting operators. Allowing the OEM an intermediate reoptimization within the bounds of an already roughly aligned superordinate sequence might be an advantageous strategy. One possibility to derive a superordinate sequence uses the information of connected components and works as follows.

1. Generate an intermediate problem instance (i) as an exact copy of the parent (p).

2. Compute the connected components. Reduce the components to those activities that are dependent on upstream-related release dates, the set J^0_{UP}. This step only needs to be done once at the beginning of the coordination procedure, as precedence constraints are assumed to remain unchanged.

3. For each component $C \subseteq J^0_{UP}$, two numbers are computed. The number

   $$lis_C = \min_{j \in C} (isd_j) \tag{7.4}$$

   indicates the latest infinite start date of the component: that is, the date the first activity of the component has to be started to avoid lateness in the infinite capacity case, cf. Equation 7.3. Again, this number remains unchanged by the coordination procedure and has to be calculated only once. The number

   $$ess^{(p)}_C = \min_{j \in C}(sd^{(p)}_j)$$

   denotes the earliest selected start date of the component.

4. For each component C, the upstream-related release dates of the intermediate instance are then set to

   $$rd^{(i)}_j = \max\left\{ess^{(p)}_C + isd_j - lis_C, rd^*_j\right\} \ \forall j \in C. \tag{7.5}$$

   The difference $isd_j - lis_C$ is the infinite offset of an activity to the first activity of the component. From an infinite capacity perspective, placing an activity before its infinite offset will not lead to reduced lateness.[14]

5. The OEM reschedules his production by applying his DS Optimizer to the relaxed problem.

[13] If lots of early production stages serve as input for several subsequent lots, the connected components will rather mirror BOM set unions.

[14] Regarding the current implementation, the following caveat should be mentioned. Recall that isd_j is computed on the basis of the infinite capacity, and does not refer to the start date an activity is currently scheduled on. Thus, the infinite values reflect a calendar structure that is not necessarily valid for the current slot the activity occupies. This can lead to situations, where Equation 7.5 shifts release dates to the right. To avoid such outcomes, the calculation of isd_j and lis_C must actually be done on the basis of an artificial calendar without breaks and full productivity. However, as the experimental results incorporate calendars only in a noncritical way, this workaround has not been implemented. it is mentioned only for the sake of completeness. The release dates of all remaining activities are updated by forward passes.

6. A right-alignment is applied.

7. The proposed dates are derived by the temporarily computed start dates, $pd_j^{(c)} = sd_j^{(i)}$.

Simply put, the operator respects a superordinate sequence given by the components. However, within each component the optimizer is allowed to reschedule activity sequences (we abbreviate this as "RC"). This can help to solve conflicts caused by release dates that are not in line with a component's precedence constraints. For problems containing only one component, no release date will be relaxed. If none of the intermediate release dates, $rd_j^{(i)}$, is smaller than in the parent solution, $rd_j^{(p)}$, the procedure is aborted before rescheduling.

7.7.3.2 Insert connected components (ICL | FWD | RA, FS | ICL | FWD | RA), ICR | FWD | RA and FS | ICR | FWD | RA)

Generally, the OEM can create new proposals by selecting one or several components, setting the release dates of contained activities earlier, rescheduling his production and deriving the proposed dates. The idea of this operator is to allow the OEM to insert a whole component at another position. Intuitively, relaxing release dates of activities belonging to the same component makes more sense than for activities belonging to different components. The OEM's freedom of proposal generation can be restricted by limiting the number of components and by defining lower bounds for the relaxed due dates. If too much freedom is granted to the OEM, the generated proposals will deviate heavily from previous alignments and no improvement will be realized. If too few components are relaxed, the DS Solving module is not able to compute an improved schedule. A decision rule on which components to select analyzes the differences between finish dates and propagated due dates. Lower bounds for release date relaxations are derived based on the current schedule. The whole procedure works as follows.

1. Compute the connected components and generate an intermediate problem-instance (i) as an exact copy of the parent (p).

2. Consider lateness caused by technological constraints by a forward pass and subsequent due date correction, as discussed in Subsection 7.7.1.1.

3. Optionally fix the sequence (FS) of activities in (i) as discussed in Subsection 7.7.1.2.

4. Apply backward passes in order to calculate propagated due dates.

5. For each component $C \subseteq J$, the maximum difference between finish date and propagated due date is computed as
$$md_C = \max_{j \in C} \left(fd_j^{(p)} - pdd_j^{(p)} \right).$$

6. Sort the components in descending order of md_C.

7. Reduce the set of components by removing all activities not directly dependent on upstream-related release dates, J_U^0.

8. Relax the release dates of the activities contained in the first $\lfloor \alpha \cdot n_C \rfloor$ components, where n_C is the total number of components and α a further tuning parameter (the symbol $\lfloor . \rfloor$ denotes the floor function). The release dates of a component C are always relaxed on the

basis of another reference component D. Approaches for selecting the reference component are presented later. First the minimum start date ess_C of C is computed as

$$ess_C = \min_{j \in C} sd_j^{(p)}.$$

The related activity is denoted by $k = \arg\min_{j \in C} sd_j^{(p)}$. A difference Υ_{CD} between C and D is computed as

$$\Upsilon_{CD} = \max_{\substack{j \in D \mid \exists m \in M_j, o \in M_k, r \in R \\ \text{with } u_{rmj} > 0 \wedge u_{rok} > 0}} \left(\max \left\{ ess_C - sd_j^{(p)}, 0 \right\} \right).$$

This number measures the maximal difference between of the earliest activity of C and any activity of D that can be scheduled on at least one similar resource. Obviously, $\Upsilon_{CD} \geq 0$ in all cases. Similar to the rearrangement of connected components, the latest infinite start date, lis_C, is computed, cf. Equation 7.4. The release dates of the activities of C are shifted by the amount Υ_{CD} by computing

$$rd_j^{(i)} = \max \left\{ ess_C + isd_j^{(p)} - lis_C - \Upsilon_{CD}, rd_j^* \right\}, \forall j \in C.$$

The release dates of the jobs contained in C are set earlier, whereas the earliest job of C is set as early as the earliest job of the reference component D that could be scheduled on a similar resource. This way we try to bound the release dates of the intermediate problem from below, such that the DS Optimizer cannot reduce the lateness by simply shifting all activities earlier but is required to change the sequence. We implemented two ways of retrieving a reference component.

(a) Out of all components D that could be scheduled on a similar resource and have a $\Upsilon_D > 0$, one is selected randomly as reference (ICR).

(b) The component D with largest Υ_{CD} is selected as a reference (ICL) .

If no reference component could be found, Υ_{CD} is set to zero. That is, the operator RC | FWD | RA of Section 7.7.3.1 is applied.

9. The OEM reschedules his production by applying his DS Optimizer to the relaxed problem.

10. A right-alignment is applied.

11. The proposed dates are derived by the temporarily computed start dates, $pd_j^{(c)} = sd_j^{(i)}$.

Obviously, if a problem consists only of a few components, the possibilities for this operator are limited.

7.7.3.3 Reschedule with penalty (IDD | FWD | RA)

This operator is an extension of operator IC | FWD | RA of Subsection 7.7.3.2. To decrease the lateness of his production schedule, the OEM should be allowed to reschedule the most critical activities. In the previous approach, an activity was regarded as critical if it belongs to a component with large lateness. However, this criticality definition is actually an arbitrary decision, based on intuition. An advantageous approach would be to introduce soft constraints in the model formulation that penalize deviations from the parent solution. This favors the computation of small changes with a huge intradomain cost decrease. In other words, the decision on which activities are critical is met by the solution method itself according to the actual problem at hand. As discussed previously, other academic approaches are based on this idea as well, for example Dudek and Stadtler (2005).

Unfortunately, the DS solving module's serial scheduling scheme relies on hard-constrained release dates and a backward scheduling functionality is not supported so far (more details are given in Section 7.7.5). Regarding our problem, the model formulation cannot be changed easily; it would require major efforts in changing the DS Optimizer's underlying Genetic Algorithm. A workaround that does not require changes to the current base optimization method, but does implement a deviation penalty, works as follows:

1. Create an intermediate problem instance (i) as an exact copy of the parent (p).
2. For each activity $j \in J_U^0$, calculate a relaxed release date. To do this, the connected components are used. The release dates of *every* component C are shifted by the largest Υ_{CD} to the left, similar to operator IC | FWD | RA. However, also nonlate components are considered now.
3. Left-align the problem instance.
4. Perform a backward pass to calculate the propagated due dates.
5. For each activity $j \in J_U^0$, an artificial due date is introduced with $dd_j^{(i)} = \min\left\{pdd_j^{(i)}, fd_j^{(i)}\right\}$.
6. The intermediate problem instance is rescheduled by applying the DS Optimizer.
7. The artificial due dates are removed again, i.e. $dd_j^{(i)} = dd_j^{(p)},\ \forall j \in J_U^0$.
8. A right-alignment is applied.
9. Proposed dates are derived as $pd_j^{(c)} = sd_j^{(i)},\ \forall j \in J_U^0$.

On first sight, it seems counterproductive to introduce artificial due dates. However, due dates are the only possibility to penalize activity shifts in the current DS Optimizer. After the left-alignment (Step 3), the schedule usually still contains delayed activities or idle times, due to wrong delivery sequences. The lateness can be reduced if the sequence of activities is altered. The violation of artificial due dates provides a rough measure of the degree of sequence alteration.

Concluding, our hope is that the operator focuses on changing the most critical sequences, and that each supplier receives a proposal containing only the most necessary sequence alternations.

7.7.3.4 Reschedule within hard bounds on start dates (SSB | FWD | RA)

Another way to generate new proposals is to allow the OEM to reschedule every activity within certain bounds.

1. Create an intermediate problem instance as an exact copy of the parent.
2. Set the release dates to $rd_j^{(i)} = sd_j^{(p)} - \alpha \cdot ml^{(p)},\ \forall j \in J_U^0$, where $ml^{(p)}$ denotes the maximum lateness of the parent schedule.
3. Introduce artificial latest start dates, $ls_j^{(i)} = sd_j^{(p)} + \alpha \cdot ml^{(p)},\ \forall j \in J_U^0$.

4. Apply a forward and a backward pass to propagate the new bounds through the network.
5. Apply the DS Optimizer and find a solution for the intermediate problem.
6. Right-align the intermediate instance.
7. Derive the proposed dates as $pd_j^{(c)} = sd_j^{(i)}, \ \forall j \in J_U^0$.

Again, α can be regarded as a further tuning parameter. If $\alpha = 1$, the DS solving module has a lot of freedom for optimization, but there is actually no need for changing the sequence of activities. If $\alpha \to 0$, there is a major need for rescheduling but not enough freedom.

7.7.4 Crossover operators

Inspired by the role of sexual reproduction in the evolution of life, an Evolutionary Algorithm attempts to combine elements of existing solutions in order to create new solutions. The elements of existing solutions are combined in a crossover operation, imitating natural crossover of DNA strands that occurs in reproduction of biological organisms.

Crossover should combine information of parent individuals in a purposeful manner, preserving basic building blocks. Regarding the studied interorganizational scheduling problem, the sequence of delivery from supplier to OEM determines the solution quality. Combining delivery sequences from parent individuals is achieved by either taking mean values of parents' release dates or randomly choosing the dates from one of two parents.

The calculated values are then either directly communicated to the suppliers as proposed dates or an intermediate rescheduling and left-right-alignment (FWD | LA | RA) takes place. For the latter option, the OEM uses his DS Optimizer to calculate an intermediate schedule based on the combined release dates. The intermediate schedule can come with larger total and maximum lateness than both parent schedules because of averaged or randomly picked release dates. Finally, the related compact schedule is derived by a subsequent left-right-alignment. The proposed dates are then set to the start dates of the left-right-aligned schedule.

In classic approaches, crossover combines the genotypes of several individuals of the mating pool in order to generate multiple children. In our approach, we adopted a slightly different variant because the number of individuals in the mating pool is not constant, fluctuating due to the asynchronous exchange of messages. There are situations possible in which the mating pool does not contain the required number of individuals to perform the crossover operation. Moreover, the mating pool contains multiple copies of the same promising parent and a crossover combining such copies would not introduce new information in the population.

Instead, a slightly different crossover variant was implemented. To cross an individual p of the mating pool, a set of candidates is computed consisting of all complete and evaluated individuals with different release dates. If no candidates could be found (e.g., after initialization), the operator aborts. Otherwise, two individuals are randomly picked out of the set of candidates and the better one is finally chosen as the crossover counterpart, q.

7.7.4.1 Linear crossover (LX and LX | FWD | LA | RA)

In the linear crossover procedure, the release dates of the intermediate instance, i, are computed as

$$rd_j^{(i)} = \frac{rd_j^{(p)} + rd_j^{(q)}}{2}, \ \forall j \in J_U^0. \tag{7.6}$$

As discussed above, either the averaged release dates are directly proposed to the suppliers or a rescheduling with subsequent left-right-alignment takes place.

7.7.4.2 Linear crossover of evaluated population (PLX and PLX" | FWD" | LA" | RA)

Instead of taking only two parents, the whole population can be used as basis to compute average values. Assume a population of n complete individuals that have already been evaluated and not sorted out by the update operation. The release dates of the intermediate instance are computed as

$$rd_j^{(i)} = \frac{1}{n} \sum_{p=1}^{n} pd_j^{(p)}. \tag{7.7}$$

By averaging values of previously proposed dates we might be able to identify trends. Our reasoning is as follows. Suppose an idealized situation, where proposed dates are either very early or very late. Now, if the OEM proposed a similar date for a certain activity for all individuals, we can argue that there is a high certainty of this date. For example, the activity might be proposed always very early because it lies on a critical path in the OEM's activity network or it might be proposed very late because it is considered as uncritical by the GA. By taking the average, this promising setting is proposed again.

On the contrary, if an activity exhibits sometimes an early and sometimes a late proposed dates, the OEM has a rather high flexibility. Proposing averaged dates over such activities grants the suppliers more freedom in their decision. Naturally, the suppliers cannot commit to all the averaged dates. When computing the counterproposal they have to decide which activity to delay and which not.

The proposed dates determine a delivery sequence that seems promising from OEM perspective. Suppliers have a large incentive to stick to the proposed sequence if there is a large relative distance between proposed dates. If the distance is rather small, other objectives, such as a setup time minimization, might get a larger influence and it is more likely that the suppliers will change the sequence. Under this light, averaging of proposed dates does the following. If the OEM is "certain" about demanding a delivery very early or very late he will communicate so to the supplier. If there is a large distance between the due dates in a supplier's domain the OEM's proposal implies a high incentive for sticking to the sequence. Otherwise, the proposed dates will lie "somewhere in the middle" and the implied incentive for sticking to the sequence might be rather weak, giving the suppliers more freedom for building better batches or allocating the resources more efficiently.

Of course, in general, proposed dates are neither proposed very early nor very late but are distributed along the whole planning horizon. Then, the above described effect is enhanced by a clustering of proposed dates. Figure 7.11 exemplarily illustrates previously proposed dates for

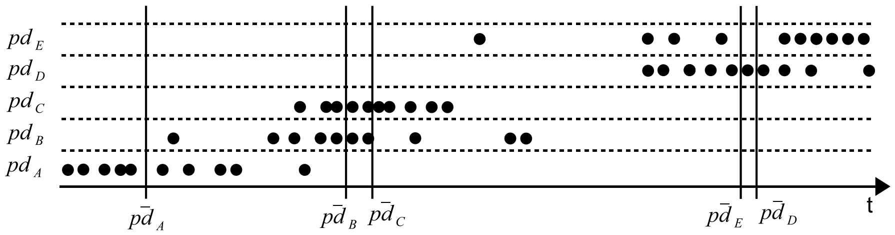


Figure 7.11: Example for crossover of proposed dates

five activities A, B, C, D, E. Vertical lines show the mean values that the OEM will propose to his supplier. Regarding the delivery sequence, the OEM transmits the following preferences to the supplier: "Most important for me is that A is delivered before E and D. Second, send me B and C before E and D. If possible, deliver A before B and C. However, I don't care much if B is sent before or after C and if E is sent before or after D." Our hope is that such priorities on preferences become clearer when taking the average of the population.

Again, instead of directly proposing the averaged values to the suppliers a rescheduling with subsequent left-right-alignment can take place.

7.7.4.3 Component crossover (CX and CX | FWD | LA | RA)

Instead of calculating the mean of previous dates, we can iterate through the upstream-related activities and randomly pick between the release dates of two different parents. In order to not disrupt previous alignments, we restrict the release date of activities belonging to the same component to be selected from the same parent. The whole procedure works as follows.

1. Compute the connected components and reduce them to the relevant, upstream-related activities (as discussed in Subsection 7.7.3.1).
2. Iterate through the components. For each component, the start dates are set to those of either the first or the second parent with probability of 0.5.

Again, either the mixed release dates can be directly proposed to the supplier or a rescheduling with subsequent right-alignment can be applied first.

7.7.5 An ideal operator

As discussed previously, it seems a promising idea to construct new proposals by changing a few release dates that have a relative large impact on the OEM's objective function. For this purpose, the use of a backward schedule generation scheme seems to be the most promising approach. Backward scheduling first schedules activities with no successor as late as possible, given the due dates. Predecessors are then added until the schedule is completed. Instead of lateness, the violation of release dates is a soft constraint and penalized in the objective function. In some sense, backward scheduling can be imagined as forward scheduling on an inverted timeline. Suppose a metaheuristic, similar to the DS Optimizer, for backward scheduling would exist. The OEM could then generate proposals that respect his due dates and violate the upstream-related release dates of the parent individual as little as possible. For further tuning, some of the due dates could be relaxed, i.e. set to the computed finish date of the parent individual. Unfortunately, the DS Optimizer does not support backward planning. An implementation with the same functionality as forward planning is considered as major effort, taking up to several years. Simply inverting the timeline and using forward planning is not possible, since setup-activities would not be interpreted correctly. Instead, we were interested in opportunities to achieve a coordination of schedules with simpler means. The operators in use are based on functionality that already exists (one of the requirements discussed in Section 4.1.4). As computational results are of considerable quality, we did not pursue the implementation of a backward scheduling DS Optimizer.

In general, it should be emphasized that all operators tend to propose early dates to the suppliers. The operators have been explicitly constructed for the case of all domains using the SAP DS Optimizer. In most cases, the DS optimizer allows suppliers to construct feasible, acceptable schedules despite the early setting of soft-constrained due dates. Suppliers employing other models or solution methods are not guaranteed to find acceptable solutions, so operators may need to be adapted.

Until now we assumed that a supplier is counterproposing the finish dates of the best GA-individual found for a specific OEM proposal. In general, suppliers could also counterpropose several alternatives, derived from other GA-individuals. However, each alternative requires an additional call of the OEM's DS solving module. Preliminary experiments suggested that it is more advantageous to generate many proposals than to generate only a few in a very detailed manner. Hence, we did not investigate this approach further. However, if the OEM possesses more computational resources than the suppliers, this strategy might suit better.

Another assumption was made for recombining suppliers' counterproposals. Only counterproposals belonging to the same proposal were combined before calculating the OEM's schedule. In other words, when constructing an individual, the OEM generates proposals and the suppliers react by sending counterproposals and rankings. In general, it might also be promising to combine counterproposals that do not relate to the same original proposal—still feasible interorganizational solutions would be computed. Again, this comes with the cost of additional computational power and might prove useful if the OEM's computational resources exceed those of the suppliers. Both aspects might be topics for future research.

7.8 Self-adaptation of operators

The above discussion suggests that the effect of operators is problem and time dependent. At the beginning of the coordination process, a crude alignment might be achieved by simple propagation operators, but later more sophisticated alignment- or rescheduling-based operators might yield better results. In general, the user will not be able to specify the correct sequence of operators *a priori*. The prototype thus supports a self-adaptation of operators. Based on the history, the most promising operator is chosen for each selected parent. Two methods were implemented for choosing the operators, one deterministic and one stochastic.

7.8.1 Deterministic self-adaptation

Deterministic self-adaptation was designed for cases with a limited amount of runtime. We assume that the user already has an approximate idea of a good sequence of operators and can build up a list of operator specifications. An operator specification consists of the operator name and possible further settings such as the parameter α. A list of yet untried operator specifications is attached to every individual. If an operator specification was already applied to a parent—because the parent was selected into the mating pool multiple times—we postpone a reapplication. The reasoning here is that applying the same operator to the same parent will probably lead to similar results. Each time an individual is selected as a parent, the related child is constructed using the list's first entry. After construction, the entry is removed from the list. The constructed child is initialized with a complete list specified by the user. If a parent is selected several times, such that its list contains no further entries, it is reset to the complete list. Maintaining an operator list is only useful together with an explicit consideration of duplicates, as mentioned in Section 7.4. If parent and child are of equal fitness because the chosen operator did not bring any improvement, the sorting method favors the selection of the parent (with the reduced operator list) over the child.

As already indicated, the success of operator specifications may be problem and time dependent. The basic idea is to sort the list according to the history of coordination. An operator specification is considered as successful if child c was better ranked than its parent p, i.e. $c \succ p$. Means for aggregating the rankings of the different parties have already been discussed in Section 7.4. We define the **success of an operator** o as

$$\eta_o = \begin{cases} 1, \text{ if } c \succ p \\ 0, \text{ otherwise,} \end{cases}$$

if o was used to construct the child. Moreover, the OEM can measure the **magnitude of success** κ_o by comparing the difference between the OEM's relevant cost of child and parent, the weighted sum of maximum and total lateness, cf. Section 7.4.

$$\kappa_o = \left(wtl \cdot tl^{(p)} + wml \cdot ml^{(p)}\right) - \left(wtl \cdot tl^{(c)} + wml \cdot ml^{(c)}\right)$$

After applying an operator (and receiving the suppliers' rankings of the child), the OEM updates the *estimated* values of future operator success $\overline{\eta}(o)$ and magnitude $\overline{\kappa}(o)$ by exponential smoothing:

$$\overline{\eta}_o := \xi \cdot \overline{\eta}(o) + (1 - \xi) \cdot \eta(o),$$
$$\overline{\kappa}_o := \xi \cdot \overline{\kappa}(o) + (1 - \xi) \cdot \kappa(o),$$

where η_o and κ_o refer to the operator of the current child. Initially, we set $\overline{\eta}_o = 0.5$ and $\overline{\kappa}_o = 0.0$. Before constructing a new child, the parent's operator list is sorted according to the current estimation, giving preference to operators with larger estimated success. If the estimated success is equal, the operator with larger estimated magnitude is preferred. If still no decision could be made, the preferences of the user, expressed in the order of the initial list are taken into account. Thus, while at the beginning the user-specified list plays a crucial role, self-adaptation gets stronger influence as more operators are evaluated.

Obviously, the performance of the deterministic adaptation depends on the initial list. If specified correctly, some quick wins at the beginning of the coordination process are likely. Deterministic adaptation is thus suitable for situations where only a short runtime is available and where the user already has *a priori* knowledge about the effectiveness of the operators. For the numerical tests, ξ was set to 0.8.

7.8.2 Stochastic self-adaptation

Stochastic self-adaptation uses the history of coordination to calculate *probabilities* used later to choose the operator. Principles from ant-colony-optimization (ACOs) algorithms have been employed to calculate operator selection probabilities. In biology, ant colonies are known for finding the shortest path (the column of ants) between source (e.g., the ant colony) and sink (e.g., some food), despite an ant's limited vision. This surprising ability is achieved by a swarm intelligence: each ant leaves a trail of pheromones on its individual path. As time passes, shorter paths obtain larger quantities of pheromone since ants travel more frequently from source to sink (and back) than on longer paths. In addition, pheromone evaporates steadily. Each ant decides about the future direction it takes based on the amount existing pheromones on the way. Eventually the behavior of the whole colony converges to taking the shortest path. In ACOs, artificial ants build a solution step by step by going through several probabilistic decisions required to construct a solution (cf. Merkle et al., 2002). In general, ants that find a good solution mark their paths by putting some amount of pheromone on the edges of the path. The following ants of the next generations are attracted by the pheromone, so that they search in the solution space near previous good solutions.

We regard the decision of selecting an operator as a dynamically changing optimization problem. For each operator o, let τ_o denote the pheromone value. The probability for choosing operator o out of the set of all operators O is computed as $\frac{\tau_o}{\sum\limits_{o \in O} \tau_o}$.

After an operator has been applied, all pheromone values are decreased by a damping factor δ. If an operator has been applied successfully (the child is better globally ranked than the parent, cf. Section 7.4), its pheromone value is increased by the relative improvement of v_0 (the weighted sum of the OEM's maximum and total lateness):

$$\tau_o := \tau_o + \frac{v_0(p) - v_0(c)}{v_0(p)}.$$

For the stochastic self-adaptation, the list of available operators per individual is not decreased dynamically. For the numerical tests, a damping factor of $\delta = 0.95$ and an initial value of $\tau_o = 0.5$ have been chosen. Thus, $\tau_o > 0, \forall o \in O$ throughout the whole coordination process. The damping is applied to the whole population once an individual is completed, i.e. $\tau_o := \tau_o \cdot \delta, \forall o \in O$.

Same principles could be applied to the parameter α of each operator. Due to the limited number of samples a self-adaption is here regarded critically, however. In the current implementation α is simply randomly drawn from the interval $(0, 1)$, if needed.

7.9 Reusing previous solutions

When constructing an interorganizational solution, the largest part of the time is used for calculating the intradomain schedules. Most parts of the RCPSP problem instance (e.g., number of activities, precedence constraints, resource utilization and capacity, calendars and so forth) remain unaltered. We tried to keep memory and runtime requirements low by exploiting this property in two ways.

1. Nonvarying parts of the model persist in memory. Instead of iteratively starting the DS Optimizer, loading the data, executing the solver and writing the solution back, the coordination mechanism is implemented as a separate DS Optimizer module (cf. Figure 6.1) in our prototype. This avoids time-consuming and unnecessary downloads and uploads of data from and to the databases. Once the DS Optimizer has been started, the coordination module takes control of the context, solver and other heuristics until the final solution has been computed.

2. The time for calculating a solution is considerably reduced by using preexisting solutions related to previous proposals.

In this section, we shed some light on the latter point: reusing previous solutions. Intuitively, resolving a bottleneck in production, such as one caused by an unforeseen machine breakdown, requires the cooperation of several partners. Regarding our standard two-tier business case, several suppliers need to be coordinated to establish an aligned flow of intermediate goods to the

OEM. Nevertheless, not all suppliers are necessarily affected by an OEM's proposal. To some suppliers, a new proposal might not mean any change with respect to their orders. Dynamically restricting the proposal exchange to those suppliers that are actually affected has the potential to speed up the whole coordination process. Moreover, in a practical environment, suppliers could become displeased if forced to recalculate proposals that do not mean any change to already known solutions.

However, calculating a target set of suppliers before a proposal is constructed is hardly possible due to the complex relations of activities within an RCPSP instance. From our point of view a superior approach is to analyze the problem as a whole in order to calculate proposed dates before checking which supplier is actually affected. The danger of redundancy lurks at several points during the joint construction of new solutions.

- Applying an operator several times to the same parent individual leads to the same or similar children. We already discussed several means for an effective operator selection in the previous section.

- An intermediate problem might equal a previous intermediate problem. In the current implementation, intermediate problem instances are not checked. Our reasoning here is as follows: For alignment-based operators, an intermediate check is actually not necessary, since the alignment heuristics are quite fast. Instead of keeping track of previous intermediate proposals, we decided to keep memory requirements low by avoiding the additional check. For rescheduling operators, the outcome of a local optimization is not determined because of the underlying Genetic Algorithm.

- The proposed dates might have no **innovation potential**. We define the innovation potential of a child (c) under construction on the basis of the set of current evaluated, complete and deletable individuals. If an individual (p) can be found in this set that

  - is accepted by all planning domains, i.e. $\forall e \in E : v_e(p) = 0$,
  - whose release dates are all smaller or equal than the child's proposed dates, i.e. $\forall j \in J_U^0 : pd_j^{(c)} \geq rd_j^{(p)}$, and
  - in case the child's partial construction involved an intermediate schedule (i) and this intermediate solution additionally shows no improvement regarding the lateness, i.e. $\exists j \in J^0$ with $dd_j < fd_j^{(p)} \vee dd_j < fd_j^{(i)} : fd_j^{(i)} \geq fd_j^{(p)}$,

  than the child is said to have *no* innovation potential and the construction is aborted. If an individual acceptable to all suppliers exists, it is a waste of time to send proposed dates that are all equal or larger than the upstream-related release dates of the existing individual. The suppliers can fulfill the proposal without even scheduling their production. Several mutation operators of the underlying solving module are lateness triggered and will not work effectively if the suppliers' downstream-related due dates are not set early enough. If the other two conditions are met, chances are limited from the OEM's perspective, that the proposed dates will lead to improved solutions since individuals with equal or smaller upstream-related release dates already exist.

  After a child has passed the above global check, it is checked for each supplier in a similar way. If the proposed dates have no innovation potential with regard to each supplier's interface, the individual is connected to a preexisting communication thread and the proposal is not resent, cf. Section 4.4.2).

- After having received the proposed dates, the suppliers calculate their downstream-related due dates according to Equation 7.2. Finally, a third redundancy check is performed on the supplier side. This is necessary as different proposed dates from the OEM might lead to the same downstream-related due date for the supplier (if the due date is calculated as a minimum value of several release dates). If a supplier identifies redundant proposals, he immediately replies with the finish dates of the related existing solution.

It is worth mentioning that due to memory and runtime considerations the complete coordination history is not checked for redundancy. Only the current complete and deletable population are checked. Thus, for very small population sizes, already proposed dates may be proposed again if the related individuals lie too far in the past and have already been deleted.

Moreover, previous solutions can also be reused in a further way. Assume a situation wherein the OEM has to generate a new solution according to a counterproposal that is—regarding upstream related release dates—only slightly different from a known solution. Under such circumstances, runtime would be wasted if the existing results were not exploited in constructing the new schedule. Recall that one GA-individual[15] is initialized according to an existing schedule (cf. Section 6.1.2.1). This back encoding of existing schedules can be exploited for a "warm reboot" each time the solving module is started. Thus, when initializing the GA-population of the DEAL-child, the parent schedule is back encoded.

We illustrate this idea by providing a small example. Assume an OEM, who needs to schedule five single-mode activities (A1,A2,B1,B2,C) on two resources, R1 and R2. The activity A2 is direct successor of A1 and B2 of B1, respectively. Figure 7.12 shows the parent schedule. It can be seen that the upstream-related release date rd_{A1} has been set earlier than the dates rd_{B1} and rd_C. In the parent schedule, the activities A2, B2, and C finish behind their due dates (indicated by the vertical arrow). Suppose that the OEM proposes to shift rd_{B1} and rd_C earlier to reduce the lateness and that the supplier confirms this proposal.

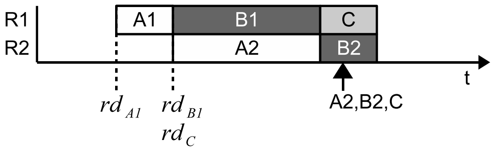
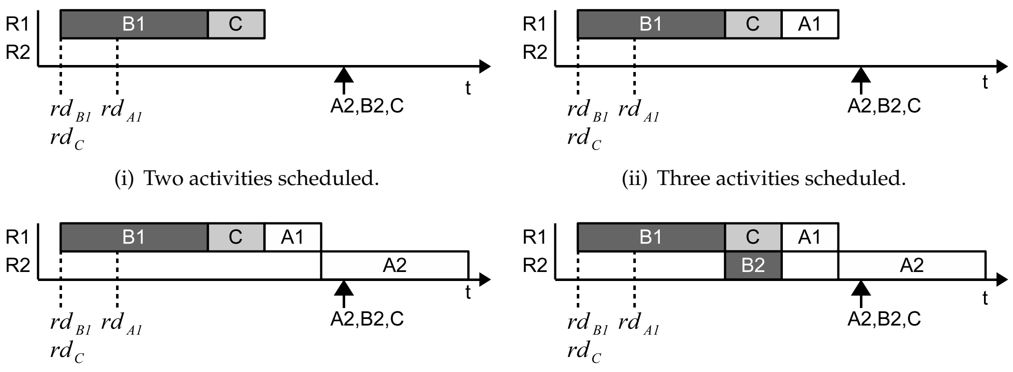


Figure 7.12: Example for back-encoding previous solutions—parent schedule.

Having received the counterproposed release dates (see Figure 7.13), the OEM reschedules his production. Assume that, for initializing his GA, the activities are sorted according to the new release dates, cf. Subsection 6.1.2.2. For this sorting, the initial permutation implies the order B1 →C→A1→A2→B2 in our example. Figure 7.13 shows the steps of the OEM's SGS (generating active schedules), cf. Section 6.1.1. Although the release dates have been set earlier, the lateness increases. Obviously, this initialization strategy does not fit.[16]

[15] To indicate the difference between individuals used by DEAL and individuals used by the GA of the DS Optimizer's solving module, we use the terms DEAL- and GA-individual henceforth. A DEAL-individual is the local representation of an interorganizational solution. A GA-individual encodes an activity list and a mode selection as discussed in Chapter 6 (strictly speaking, we are focusing on the genotype).

[16] For the sake of simplicity, we do not consider the other initialization strategies of Section 6.1.2.2 in this example.



(i) Two activities scheduled.

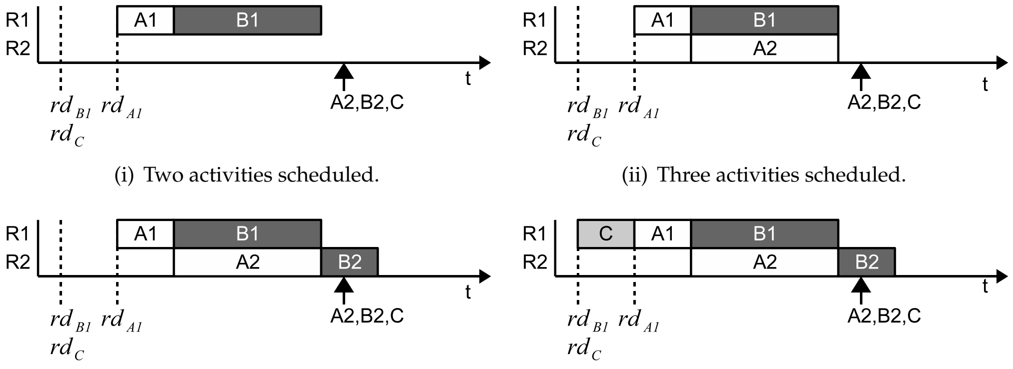
(ii) Three activities scheduled.

(iii) Four activities scheduled.

(iv) All activities scheduled.

Figure 7.13: Example for back-encoding previous solutions—initializing the child schedule according to release dates.

Back-encoding the parent schedule can provide a better initialization strategy. In the absence of max-links and sequence-dependent setup times, back-encoding means to sort the initial population according to the activities' start dates *in the parent schedule*, cf. Subsection 6.1.2.1. In our example, this might imply the order A1 →B1→A2→B2→C. The steps of the OEM's SGS for this order are shown in Figure 7.14. It can be seen, that the lateness decreases.



(i) Two activities scheduled.

(ii) Three activities scheduled.

(iii) Four activities scheduled.

(iv) All activities scheduled.

Figure 7.14: Example for back-encoding previous solutions—initializing the child schedule according to the parent schedule.

Hence, back-encoding the parent schedule allows to carry over knowledge from previous DEAL-individuals and can contribute in saving runtime, especially if the problem instances consist of several hundreds activities with multiple modes (which are back-encoded as well, cf. Section 6.1.2.1). If the parent's schedule does not suite the child's problem instance, other initialization operators of the underlying GA will still be able to set up an initial GA-population.

If the construction requires solving an intermediate problem, the parent's schedule is used in solving the intermediate problem, and the schedule of the intermediate solution is used in solving the child's problem. From a computational perspective, a schedule is completely defined by the activities' start and finish dates and selected modes, limiting memory requirements for storing a schedule.

The same procedure is applied for OEM and supplier. For sequential coordination and a population consisting only of one individual, this measure proved to be an efficient means to avoid a recomputation from scratch and thus to limit local runtime. For parallel coordination or larger population sizes, we implemented additional principles that require a slight change in the DS Solving module's implementation. As preliminary experiments suggested, it is a nontrivial task to decide which DEAL-individual provides the best starting point (if several such individuals exist) *before* computing the solution of the child. Instead of focusing only on the parent individual, we use the information of all existing individuals. Recall that the underlying GA itself is a population-based approach. The idea was thus to inject all available information into the initial GA-population. The standard GA initialization methods of Section 6.1.2 have not been altered, however. Instead, additional GA-individuals were added, leading to an oversized initial GA-population. After the GA's first generation, the population shrinks to its standard size, as the worst GA-individuals are sorted out. The following concept was implemented for storing the additional information. Each DEAL individual has a list of promising GA individuals. After the GA has completed the construction of an intermediate solution or a child, the related final GA-individual is taken and added to the list of each idle, waiting and complete DEAL-individual.[17] If a new DEAL-individual is constructed it inherits the parent's list. Whenever a DEAL-individual's schedule is computed, its stored GA-individuals are added to the initial GA-population when calling the solving module. Adding an individual to the initial population means that the activity lists and the mode selection of a new individual is initialized according to a stored activity list and mode suggestions resulting from a previous call of the DS Solving module.[18] This way, even DEAL-individuals that have already been deleted get a chance to contribute to the construction of a child. Intuitively, the warm reboot has a larger potential if new DEAL-individuals deviate only moderately from already existing solutions. A potential drawback is that local optima might have greater influence since they are *inherited* between the DEAL-individuals.

In conclusion, reusing previous solutions can be regarded as a way to combine several local optimization runs into one global optimization run. From this perspective, the coordination mechanism can also be regarded as a single optimization run interrupted from time to time to exchange information to alter the properties of the underlying model.

7.10 Implementation details

Having discussed the properties of the RCPSP problem, the DS Optimizer, the generic DEAL framework and its customized version, this section supplies various implementation details. The prototype was implemented in C++ and extends the present DS Optimizer, version 5.1, part of the SCM Business Suite 2007. As mentioned in Chapter 6, the DS Optimizer can be run as a standalone executable[19] with flat text files, not connected to the SAP GUI or the database (LiveCache). The prototype builds on this standalone mode. The focus is thus on the functional part of the

[17] For our computational evaluation, the list consists of 60 entries. If adding a new GA individual violates this bound, the oldest list entry is removed. In principle, not only the best one but several solutions could be carried over from each GA run, which opens an interesting topic for future research.

[18] Carrying over mode suggestions is not always possible. As discussed in Subsection 6.1.5, a preprocessing makes infeasible modes inaccessible. As release dates change and artificial deadlines or precedence constraints are added by some operators, the number of accessible modes might differ from DEAL-individual to DEAL-individual. Hence, stored mode suggestions might not be accessible for some RCPSP instances. In such cases, the related mode suggestions are not initialized, leaving the choice to the greedy heuristic presented in Subsection 6.1.1.

[19] In computing, an executable causes a computer to perform indicated tasks according to encoded instructions as opposed to a file that only contains data.

coordination mechanism and not on issues regarding visualization or database design. During coordination, there is no upload or download from the database. The source code of the DS Optimizer was extended by several classes representing planning domains, interdomain messages and populations of interorganizational solutions. The object oriented design allowed the implementation of the coordination process as a single executable. For the sequential coordination, only one instance of this executable (a so-called process), suffices to simulate the coordination mechanism: planning domains are encapsulated as autonomous objects and communication takes place by exchanging message objects between the domain objects. Special attention was paid to ensure that sensitive data was kept private and not accidentally exchanged between the domain objects. In the parallel coordination, several instances of the executable run as separate processes on the same or on different machines. An interprocess communication was realized on the basis of SAP Remote Function Calls (RFC).[20] Here, message objects need to be serialized, sent via RFC and deserialized on the other end.[21] The different functionalities were encapsulated in such a way that only the coding of the "physical" message exchange is different for sequential and parallel coordination. The planning functionality, methods for managing the population and the creation and evaluation of proposals are exactly similar for both coordination modes.

The implementation can be briefly explained by considering the (high-level) architecture of the DS Optimizer (see Figure 6.1, Chapter 6). Each intradomain planning problem is stored as a separate context. (Recall that a context defines a single RCPSP instance and its related solution, see Chapter 6.) These contexts can be passed to the solving module or to heuristics and metaheuristics. The main working principle of the DEAL framework is to change some of the data of a context (cf. Figure 7.1) and thus to create new RCPSP instances. Different input dates (e.g., release dates, due dates) will in general result in different solutions (start dates, finish dates and mode selections). The rest of the context (activities, precedence constraints, capacity profiles and so on) does not change however. Hence, in addition to a **working context**, representing the nonvarying part of the RCPSP instance, an individual encodes only the varying part of an interorganizational solution for a planning domain. As highlighted in Chapter 4, each planning domain can be seen as a state machine that works on a population of individuals. Several classes have been implemented to manage the population for leader and follower. This functionality will henceforth be subsumed as a **domain-control procedure**. In general, a domain control procedure is the inner part of the while loops of Algorithm 3 or Algorithm 4 presented in Section 4.9.

Briefly summarized, a domain control procedure processes incoming messages, manages individuals and produces output messages. When evaluating or constructing an individual, the control procedure initializes the working context with the content of an individual before passing it to the solving module or the heuristics. Messages for synchronizing the local populations are represented as separate objects in every planning domain. Each partner has a **message inbox** and a **message outbox**. Each message is identified by its id and stores the ids of sender and designated receiver, cf. Section 4.3.1. A domain control procedure puts messages to send into its outbox and looks for incoming messages in the inbox. A **distributor** object without any planning functionality handles the exchange of messages. The distributor's **distribution-control procedure** first copies a planning domain's outbox messages into a temporary storage and then distributes them

[20] Remote Function Calls are SAP's proprietary standard of protocols and interfaces for calling functions in other systems. In general this technology is also known as remote procedure calls.

[21] In this regard serialization refers to the task of converting an object to a text message transferred from one process to the other. Deserialization refers to the opposite, the instantiation and initialization of an object according to the received textual specification.

into the designated inboxes. In addition, a **controller** object initializes and monitors the system. To be able to do this, the controller stores the global context (i.e. the interorganizational RCPSP instance), but the controller does not intervene in the coordination process and has no planning functionality at all. Instead, the controller reassembles the domains' partial context information that belongs to the same interorganizational solution and checks if this global context is feasible, such as if all precedence constraints are fulfilled. The feasibility check is performed for every solution constructed. The checks are performed by a standard module provided by SAP, the so-called SolutionChecker. The checks slow down the overall process a bit but confirm that the interplay of C++ objects really works.

A detailed description of the OEM's control procedure in pseudocode for the customized DEAL framework is given in Appendix B. The customized control procedure for the supplier can be found in Appendix C and the controller's control procedure in Appendix D. The descriptions of control procedures are valid for the asynchronous and parallel coordination and include additional functionality for debugging by sending local solutions to the controller. Further explanations are provided in the following subsections.

7.10.1 Sequential and asynchronous coordination

Figure 7.15 illustrates the interplay of modules for the sequential and asynchronous coordination. On a high level, the architecture of the DS Optimizer (Figure 6.1) is extended by a domain-control module, a message inbox and outbox, a population of DEAL-individuals and a dictionary of interorganizational precedence constraints, the mapping of activities between the planning domains. Each domain-control procedure successively changes the population, processes inbox messages, reinitializes the local context, triggers the solving module and produces output messages. Every domain-control procedure is a sequence of instructions, performing only a limited number of predefined steps. Hence, a simple means to achieve a sequential coordination is to iteratively switch between the instructions of the distribution-control and the domain-controls within a single process, calling the domain-control procedure of domain A, the distribution-control procedure of controller, the domain-control procedure of domain B, the distribution-control procedure of controller and so on. A call executes a control procedure once. This approach has the disadvantage of wasting computational power as the control procedures may be called with nothing to do. A critical point are deadlocks, cases where no planning domain can produce any output since it is waiting for input from other planning domains. To avoid deadlocks, the procedures have been designed to be applied to any state of incoming messages and individuals. It is guaranteed that at least one control procedure can continue computation. A central advantage is that the functionality of distributing messages is kept separated from the functionality of calculating proposals. There is only a single focal distributor taking care of the message exchange. The focal distributor is also required for technical reasons. An existing grid framework was used to parallelize the process. This framework does not allow bilateral communication between individual processes. Instead, slave processes must communicate through a master process. To manage this job, the focal distributor needs to be placed on the master process (more details can be found in Subsection 7.10.3). For the sequential coordination, there is no difference between several domains with separated distribution functionality or a single distributor iteratively managing the distribution tasks of all the domains.

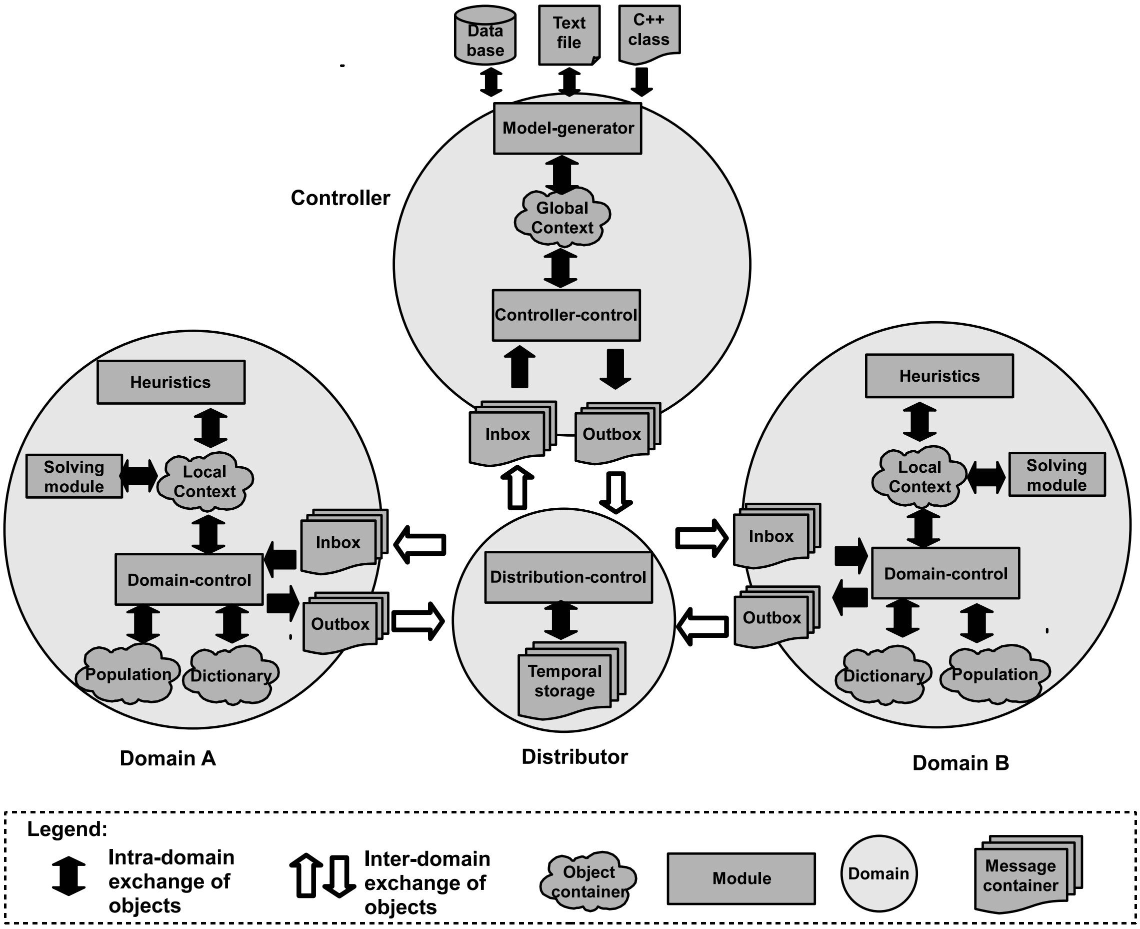


Figure 7.15: Modules of the asynchronous DEAL prototype implementation

7.10.2 Parallel coordination

One of the main advantages of DEAL is the support for parallel computation. As long as the domains reply to each other's requests, the actual sequence of replies is irrelevant for the functioning of the mechanism. Simply put, it does not matter how the planning domains compute their proposals, as long as they do it some way.

There exist numerous possibilities for parallelization. One idea would be to divide the problems into several parts (e.g., according to the currently selected start dates of activities) and to coordinate these parts in parallel. Obviously, this would require additional measures to preserve feasibility. Moreover, a good solution might require shifting activities across the whole planning horizon and the parallelization would prune important regions of the search space. For these reasons this idea was not investigated further.

Another possibility is to parallelize the local optimization method, the solving module of the DS Optimizer. For example, the population of permutations could be split into several islands, each situated on another process. From time to time, individuals are exchanged between the islands to synchronize the convergence of the whole system (cf. Engelmann, 1998). However, we cannot expect to successfully parallelize other local optimization methods in a similar way.

Therefore, we implemented a third variant that is supposed to work with most optimization methods. Recall, that the DEAL process iteratively changes data of a local RCPSP problem. In fact, several RCPSP instances are generated, each of which can be solved separately. By far the

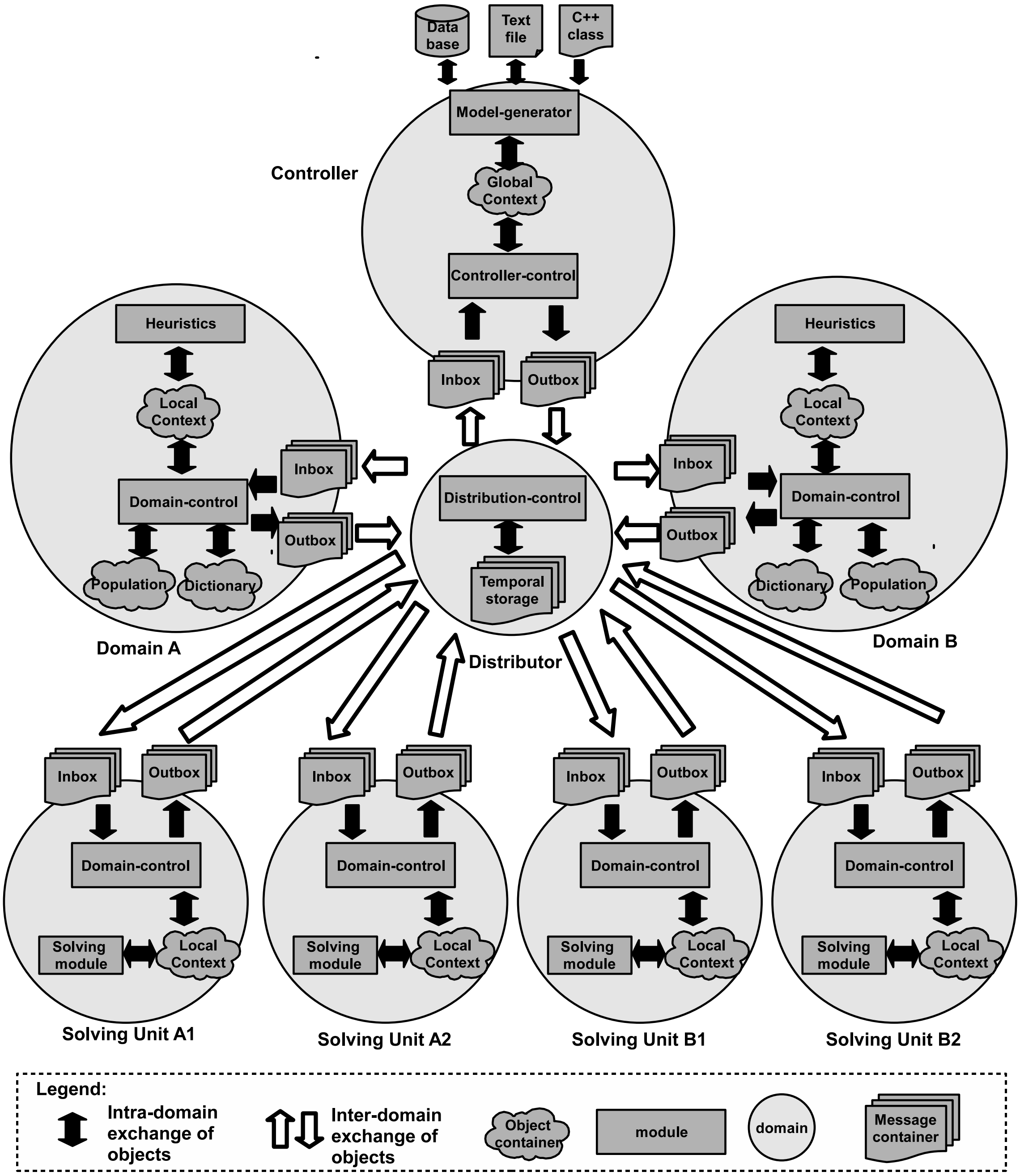


Figure 7.16: Modules of the parallel DEAL prototype implementation

most time-consuming task is computing a schedule by calling the optimizer's solving module. As indicated in Section 4.10, the idea is hence to send ready-made RCPSP instances to solving units that only call the solving module and send the results back when finished. The interplay of modules for the parallel coordination is illustrated in Figure 7.16. As for the sequential and asynchronous coordination, all messages are routed over a focal distributor. If, for example, Domain A wants to send an instruction to Solving Unit A1, it cannot do so directly. Instead, a message for the receiver A1 needs to be constructed and placed in the outbox. The distributor will then gather this message and place it in the designated inbox of A1. As already mentioned, this design was motivated by properties of the SAP SCM grid framework, explained in the next subsection.

7.10.3 The SAP SCM grid framework

The SAP SCM Business Suite contains a grid framework to support parallel computation. A communication between different processes (run on different machines) is achieved by RFCs. Within this framework a master–slave mechanism was established providing encapsulated functionality for the optimizers. Using this predefined mechanism, communication between processes can be programmed conveniently via C++ streams and one need not struggle with the underlying remote function calls. However, communication can only take place between slaves and the master, not between the slaves directly. For standard intradomain parallelization tasks, this architecture is sufficient, as the master takes care of administering the overall process and the slaves carry out predefined jobs.

However, to simulate a parallel coordination, several planning domains need to communicate with each other *and* with the solving units. Extending the master–slave mechanism itself to such a communication was considered a major effort beyond the scope of this thesis. A naive approach would be to situate all planning-domain objects on the master process and each solving-unit object on a several slave process. Here, the communication to solving units would be possible over RFCs, and the communication between planning domains by exchanging objects within the master process. However, this would not parallelize the computation of the planning domains. On the master process, the domain-control procedures need still to be called sequentially as discussed in Section 7.10.1! Thus, a different approach was used as illustrated in Figure 7.17. All planning domains and solving units are situated on the slaves, only the controller is on the master. Additionally, the master contains a global distributor object. The controller's control procedure and the global distributor's control procedure are called iteratively in the master process,[22] as in the

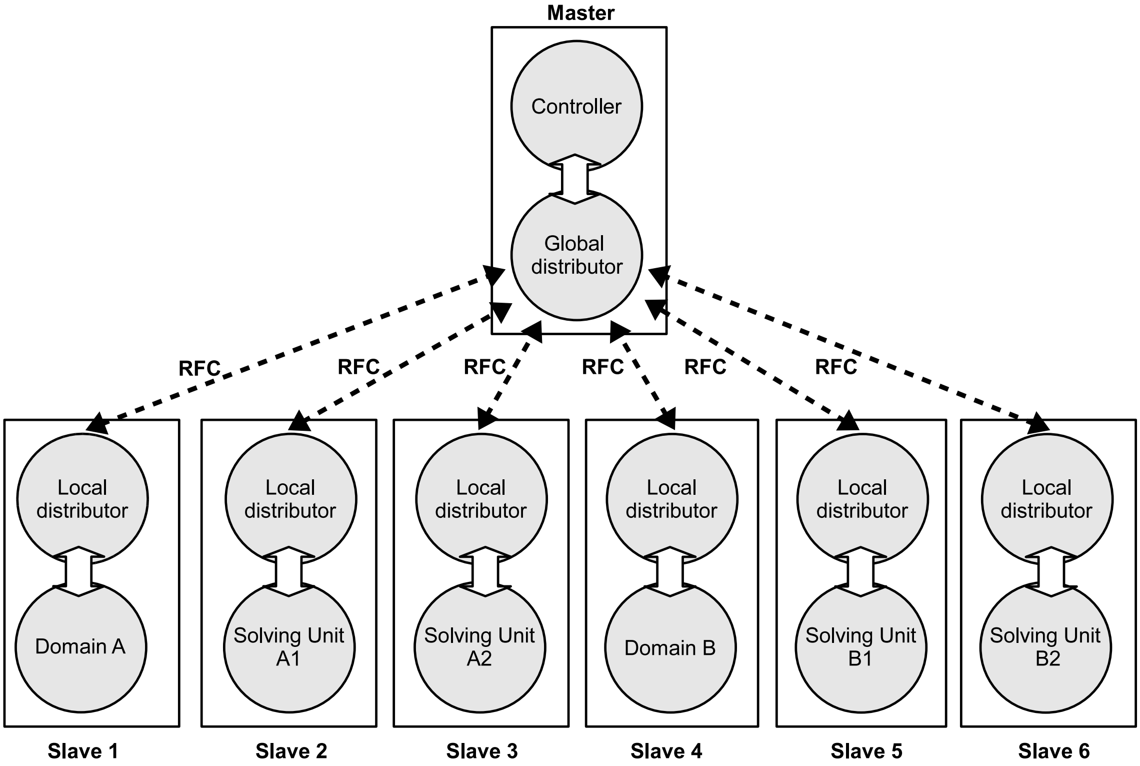


Figure 7.17: Distribution of modules in the grid-framework

[22] Alternatively, the controller could be located on a separate process. However, since runtime requirements are small,

sequential case. The global distributor has the additional information on which domain or solving unit is on which slave. Next to planning domains or solving units, every slave contains a local distributor. A slave's domain or solving unit control and local distributor control procedures are also called iteratively. Sending an instruction from a planning domain to a solving unit involves the following. The planning domain places a message into its outbox. The local distributor copies the message into its temporary storage and tries to find the receiver on the same slave. If a local receiver could not be found, the message is forwarded to the global distributor via RFC. The global distributor again forwards the message to the slave that contains the designated solving unit via RFC (or to the locally attached controller if he is the designated receiver of the message). The message is deserialized and temporarily stored by the local distributor on the receiving slave. Finally, the local distributor puts the message into the inbox of the designated solving unit. Theoretically, several planning domains or solving units could be placed on the same slave. However, to enable a true parallelization, the tests where run with only one domain or solving unit per slave. Additionally, enough CPU power was ensured, one CPU for each slave.

It is worth highlighting the three layers of communication in the parallel coordination. The basic physical layer consists of remote function calls in the SAP SCM grid framework, providing a connection between different processes. The second, intermediate, layer is given by the master–slave mechanism that provides common communication functionality for C++ executables. The top layer is defined by the messages exchanged between planning domains and solving units. As already mentioned, the same top layer is used for sequential, asynchronous and parallel coordination. The sequential coordination, however, does not require the other two layers. In the parallel coordination, the functionality of each layer is encapsulated, the control procedures do not "know" whether they are called in a single process or distributed over several processes. For parallel coordination, the master–slave mechanism must first be started, before the coordination can take place. A comprehensive overview is given later in Subsection 7.10.5.

With regard to the parallel coordination, several further issues need to be mentioned. As already mentioned, not all control procedures will have work to do every time they are called. In order to limit CPU use, the following strategy has been implemented. At every iteration, the master or slave process measures the time difference between calling the first control procedure under its responsibility and finishing the last. If there are no messages to process or idle individuals to construct, the control procedures terminate fairly quickly and the measured time difference is much smaller than one second. If applicable, the subprocess pauses additional milliseconds such that the time between two iterations of control procedure calls takes at least 1 second. Due to these pauses, transmitting a message from sender to receiver can take several seconds. However, if there is a constant flow of messages in the system, this per-message delay is negligible.

Another issue is related to a technical detail concerning the interprocess transmission of messages via RFC. If one process intends to send a message to another process, the latter needs to be ready. If, for example, the global distributor sends a message to a slave and the related domain-control is currently calculating a proposal, the master process is held until the local distributor on the slave has been called to receive the message. During the waiting time, the master cannot receive further messages. This can lead to a cascade, where a single process forces the whole system to pause. Obviously, such a system behavior hampers the simulation of a true parallel computation. As a remedy, the global distributor stores messages temporarily until the slave is ready, until all control procedures of the current iteration are finished. To indicate its readiness, a slave

the controller was put on the master process to avoid additional RFC calls and to limit the processes count.

sends a short command to the master, polling for new messages. The master then forwards all messages to the slave gathered after the previous poll. If no message arrived, the master replies with a `no_message` command and the slave continues its iteration of control procedure calls. In some sense, the global distributor acts as a thread for each planning domain, buffering incoming messages until the domain is ready to receive them. To limit communication overhead, the slaves have to maintain a minimum time between to consecutive polls (set to 1 second for the numerical test).

Another point to be mentioned relates to the domain-control procedure itself. During a single iteration of a domain-control, several messages can be computed and placed into the outbox. As described above, the messages would first be gathered by the (local) distributor and sent to the master if the domain-control procedure was finished and the distribution-control procedure could be started. However, calculating a proposal might take several seconds. Hence, it makes sense to distribute a message as soon as it was placed in the outbox before calculating the next proposal. This behavior was realized by terminating the domain-control and calling the local distributor each time a message was issued. Finally, it should be emphasized that the parallel coordination does not fulfill the definition of a grid in the close sense. For example, Foster and Kesselmann (1998) define a grid as follows.

> A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive and inexpensive access to high-end computational capabilities.

In the current implementation, an assignment of solving units is fixed and not changeable during the coordination process. In addition, the local optimization time was set to be similar for all solving units during the numerical tests. As the OEM has to solve intermediate problem instances, he gets more solving units assigned than the suppliers. However, the distribution of solving units was done on the basis of a crude estimation of computational efforts. Moreover, not all local problems might be of similar complexity. For example, a supplier's problem might have less activities than the OEM. A dynamic adaptation of local runtime to actual requirements is a subject of future research.

Concluding, it can be said that setting up the parallel coordination was a technically challenging task. Especially, debugging in a parallel environment turned out to be difficult since the origin of failures in a parallel working system is not intuitively clear. In this regard, using the sequential mode proved useful for finding and eliminating programming bugs. Another critical point is that the physical message exchange is not truly decentralized but routed over the master process, as required by technical conditions. This comes with the danger of the master being a bottleneck, not being able to redistribute messages at a sufficient rate. However, such an behavior could not be observed in any of the conducted empirical tests.

7.10.4 Customized message protocol

Sections 4.3 and 4.7 gave an overview of the message protocol from a generic perspective. The generic protocol was summarized in Table 4.1. Aside from not using the guidance messages, this protocol is still valid for the scenario of one OEM and several suppliers employing the DS Optimizer. Additional commands are used to control the initialization and the termination, for feasibility checking and message exchange in the parallel case. These commands are summarized

Command	Reply	Planning domain	Controller
`start`	n.a.		x
`terminated`	n.a.	x	
`check_feasibility`	n.a.	x	

Table 7.4: Additional commands and issuers for initialization, termination, feasibility checking and parallel message distribution

Command	Reply	Planning domain	Solving unit
`request_solution`	`solution`	x	
`solution`	n.a.		x

Table 7.5: Additional commands and issuers for message exchange with solving-units

in Table 7.4. Other commands for communicating to solving units are listed in Table 7.5. Both tables relate to the top layer of messages exchanged between domains and solving units. The controller sends a message with a `start` command to the OEM if all processes have been set up. The domain-controls send a `terminated` back if they have finished all their calculations and the system of processes can power down. Intermediate feasibility checks are triggered by messages headed by the `check_feasibility` command. Planning domains send requests for computations, messages with `request_solution`, to the solving units, which reply with `solution` messages once the calculation is finished. More details regarding the interplay of domain- and solving-unit–control procedures can be found in Appendices B and C.

In addition, Table 7.6 lists commands of the intermediate layer, pertaining to the master–slave mechanism. These commands summarize the above discussion for handling the "physical" message exchange between processes. More details are provided in the next subsection. Table 7.7 gives an overview of customized components, their purpose and the individuals and messages that use this components.

7.10.5 The process of parallel coordination

Concluding the description of implementation details, this subsection reviews the communication between master and slave processes in the grid framework. Figure 7.18 illustrates the main sequence of master and slave communication. Initially, all processes start and are initialized per parameters from a text file. After activating the slaves (command `init`), the master runs the model generator module to build the global context. After initialization, the global context is then split up into the separate subcontexts, where each subcontext represents a planning domain's RCPSP problem instance. In some cases, the initial situation is generated by centrally applying

Command	Reply	Master	Slave
`init`	n.a.	x	
`receive_entity`	n.a.	x	
`start_coordination`	n.a.	x	
`stop_coordination`	`coordination_stopped`	x	
`coordination_stopped`	n.a.		x
`receive_messages`	n.a.	x	x
`ready`	n.a.		x
`no_message`	n.a.	x	
`exit`	n.a.	x	

Table 7.6: Additional commands and issuers for instruction between master and slave processes

Component	Purpose	Attached to
start dates	Start dates of upstream-related activities.	Individuals of OEM, messages with command `solution` sent by solving units to OEM.
proposed dates	Proposed dates from OEM to supplier. Translated to supplier's due dates upon transmission.	Message with command `proposal`—sent by OEM.
due dates	Proposed due dates from OEM to supplier.	Messages with command `proposal`—received by supplier. Messages with command `request_solution` sent from supplier to solving unit.
finish dates	Realized finish dates of supplier's downstream-related activities. Translated to release dates upon transmission to OEM.	Messages with command `counterproposal` sent by supplier. Message with command `solution` sent from solving units to supplier.
release dates	Upstream-related release dates received by OEM from supplier.	Messages with command `counterproposal` received by OEM, individuals by OEM, `request_solution` messages sent to solving units.
start dates of parent	Stores the start dates of the parent individual. Needed by operator RMP | LA | RA	Individuals of OEM.
finish date to external	Realized finish dates of downstream-related activities to external customers. Needed to quantify costs.	Individuals of supplier and OEM. Messages with command `solution`.
internal cost	Internal cost associated with a solution.	Individuals of supplier and OEM, messages with command `solution`.
latest feasible start dates	Upper bounds implied by operator ISB | FWD | RA. Needed for communicating the bounds to the solving units.	Messages with command `request_solution`.
reboot information	Stores all start and finish dates and selected modes of a solution, needed for warm reboot.	Messages with commands `request_solution`, `solution` and `check feasibility`, all individuals.
operator list	Stores a list of available operator specifications for further construction of children.	All individuals.

Table 7.7: Customized components of the message protocol

the solving module to the global context prior to splitup. Next, objects representing the planning domains and solving units are instantiated and initialized with the respective subcontexts and assigned to the slaves (command `receive_entity`). In addition, the controller object is instantiated on the master process. At the end of the initialization phase, the domain and solving-unit objects are distributed to the slaves and the global and local distributors are instantiated.

During coordination, the control procedures exchange messages through the master. The slaves iteratively poll for new messages by sending the command `ready` to the master. If the master process' global distributor has new messages for a slave, the transmission is started with the command `receive_messages`, otherwise the master returns `no_message`. For sending messages

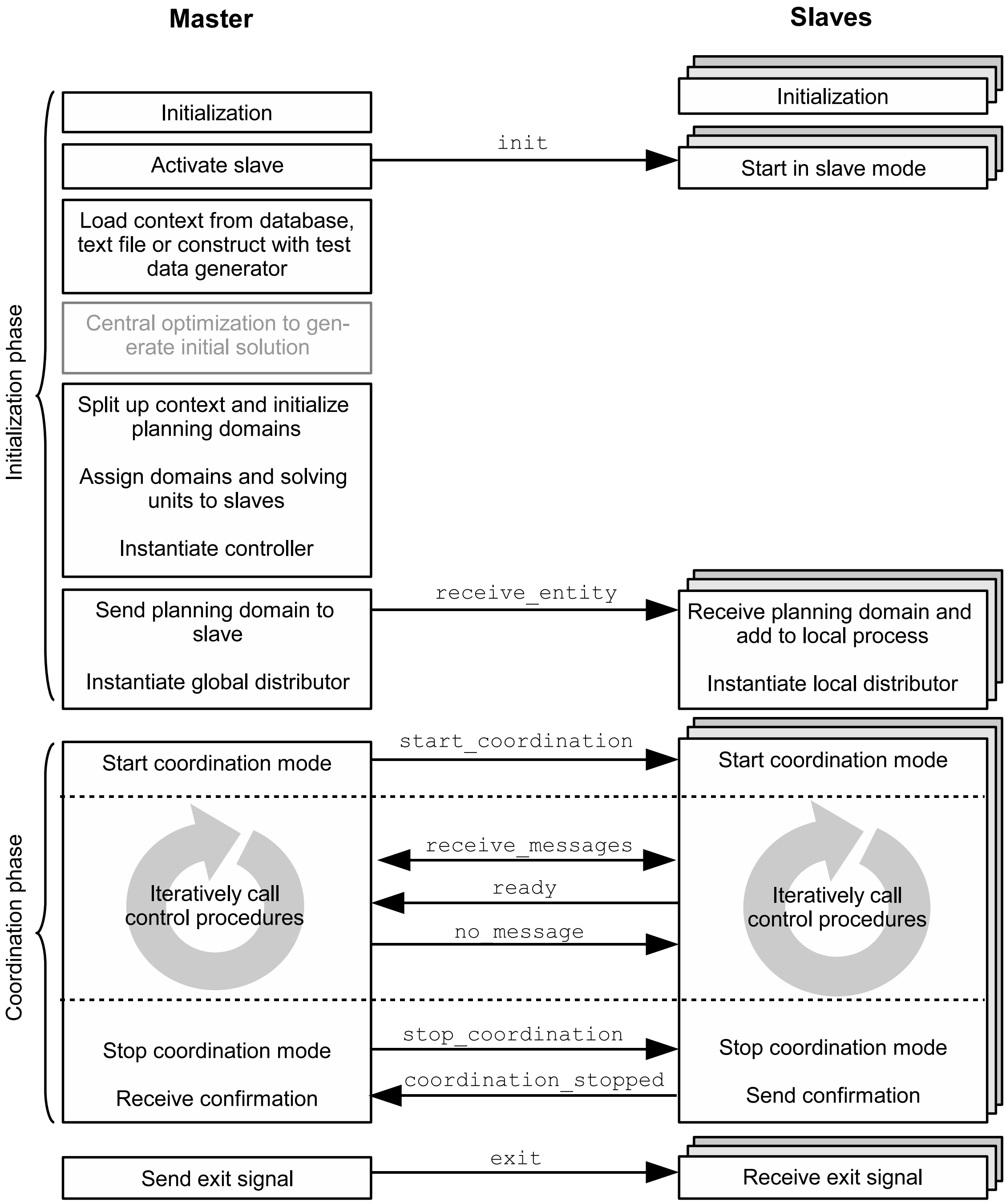

Master
Slaves
Initialization phase
Coordination phase
Initialization
Activate slave
Load context from database, text file or construct with test data generator
Central optimization to generate initial solution
Split up context and initialize planning domains
Assign domains and solving units to slaves
Instantiate controller
Send planning domain to slave
Instantiate global distributor
Start coordination mode
Iteratively call control procedures
Stop coordination mode
Receive confirmation
Send exit signal
init
receive_entity
start_coordination
receive_messages
ready
no message
stop_coordination
coordination_stopped
exit
Initialization
Start in slave mode
Receive planning domain and add to local process
Instantiate local distributor
Start coordination mode
Iteratively call control procedures
Stop coordination mode
Send confirmation
Receive exit signal

Figure 7.18: Processes of the parallel coordination

to the master, also the slave use the command `receive_message`.

The coordination phase ends if all planning domains have terminated. Further commands (`stop_coordination`, `coordination_stopped` and `exit`) are exchanged to end the coordination phase and end the processes in a controlled way.

7.11 The three-tier scenario

So far, this chapter has discussed methods for constructing and evaluating solutions in the two-tier scenario. In such a setting, the construction of solutions essentially follows a two-phase up- and downstream planning procedure. In the first phase, the OEM sends proposed delivery dates to his suppliers. In the second phase, the suppliers reply with feasible delivery dates. The question of interest is if this coordination scheme can be extended to more than two tiers. Consider a three-tier SC, consisting of suppliers, OEM and wholesalers with planning functionality. An extension of the two-tier approach is possible if the OEM represents the single point of coordination, cf. Figure 4.4 (i). A generic construction template has already been given in Section 4.4.2. The high-level idea is that the leader asks wholesalers for guidance (i.e., data to construct purposeful proposals). For the customization of the DS Optimizer, the following proceeding can be outlined.

1. The OEM sends a guidance request to each wholesaler.
2. The wholesalers derive proposed dates to increase their service level using the operators presented in Section 7.7.
3. The OEM converts the proposed dates to internal due dates by applying Equation 7.2 discussed in Section 7.1.
4. Based on the updated due dates, the OEM constructs proposed dates using the operators presented in Section 7.7.
5. As in the two-tier scenario, the suppliers convert the proposed dates to due dates, call the optimizer to construct a schedule and reply with (feasible) finish dates.
6. The OEM converts the suppliers' finish dates to release dates. Based on the new release dates and previously received due dates, he schedules his production by calling the DS Optimizer. Realized finish dates are in turn submitted to wholesalers.
7. The wholesalers convert the OEM's finish dates to release dates and schedule their own production.
8. The OEM collects rankings from all suppliers and wholesalers. A global ranking for further selecting and updating the population is constructed by the method discussed in Section 7.4.

It should be noted that the wholesalers do not necessarily need to be questioned for guidance every time the OEM constructs proposals for his suppliers. Probably a less frequent updated guidance (every 2nd or 3rd round) might suffice. Moreover, for solutions that have already been accepted by upstream partners additional guidance information might not be necessary at all.

An extension to more than three tiers or several points of coordination is not without problems. Under the current coordination scheme, several leaders would be required. A major difficulty would be that the leaders need not only to coordinate their followers, but also to synchronize their activities with each other. This would require major parts of the SC structure to be public knowledge. However, revealing suppliers and customer contracts (to possibly competing companies) might be regarded as critical disclosure of sensitive data. If revelation of suppliers and customers is regarded as noncritical, a superordinate mediator might be advantageous for large SCs with complicated structures. A population-based mechanism is imaginable in which each planning domain sends a delivery date proposal (consisting of proposed start and finish dates representing a local improvement of the initial schedule) to the mediator. Knowing the structure of the material flow, the mediator trades off conflicting proposed start and finish dates by, for example, taking the average. As a fairness criterion those planning domains evaluating the current solution as bad with regard to the remaining population could get higher influence. The mediator reports back tentative delivery dates to the planning domains. Now, a global two-phase upstream and downstream planning takes place. During upstream planning, no item may be requested prior its tentative delivery date. During downstream planning each planning domain tries to fulfill the upstream requests to the extent possible. It should be noted, however, that taking averages of proposed dates possibly contradicts the intention of some of the operators presented in Section 7.7.

If revelation of suppliers and customers is regarded as critical, another mechanism can be imagined. Following Dudek (2007), an active subset of the SC, containing only a single point of coordination (taking the leading role) can be extracted. All inactive planning domains not included are handled as "external" suppliers or customers, i.e., the related release dates are not relaxed during proposal generation and the related due dates are treated as deadlines, cf. Section 7.1. In other words, the planning domains of the active subset aim to eliminate inefficiencies by using the DEAL framework without invoking passive domains. The idea is then that this active subset cycles through the whole SC, iteratively excluding active and including inactive planning domains while maintaining the *single point of coordination* requirement. The above ideas might be the starting point for future research.

7.12 Appropriate real-world business cases

Concluding the description of the DEAL framework, it is interesting to examine, which practical requirements of Section 4.1 are fulfilled. Certainly, exchanging ranks and delivery dates does not disclose sensitive data. Moreover, since side payments are not exchanged, the focus is on availability and not on costs which eliminates various problems associated with side payments. An obvious drawback is that some areas of the search space are pruned; they cannot be reached if side payments are not exchanged. However, the central question in this regard is if the planning domains actually intend to reach the global optimum. Section 4.1 lists several facts that contradict this hypothesis. The other requirements are also fulfilled to a large extent. The mechanism preserves feasibility, allows the coordination of a limited number of domains, supports the black-box treatment of underlying local optimization engines and, as a population-based approach, provides possibilities for parallelization.

Though an analytical proof is hardly possible, the customized DEAL framework is intuitively incentive compatible because no player has anything to lose when participating in the coordination. Suppose that contingents have been agreed in a frame contract but are outdated due to unforeseen events, such as a machine breakdown or increased customer demand. Not using the coordination mechanism, the OEM can optionally claim a contract penalty but cannot fulfill his service level targets. He is thus interested in employing the coordination mechanism to realize a potential decrease of late delivery. Also the supplier is interested in participating in the coordination since he can increase the customer satisfaction of the OEM and possibly avert a contractual penalty. However, the OEM has no possibility to overexercise his power by forcing the supplier to obey the proposed delivery dates. Suppliers are always allowed to deviate from the OEM's proposal in order to fulfill service level agreements with other customers. Being a loyal partner, the supplier intends to fulfill the OEM's proposals to the extent possible, however. As underlying optimization methods and related data remain unaltered, the costs for implementing the coordination mechanism seem to be manageable.

Setting up a today's planning system with a high degree of automation already requires a huge investment. Licences and IT infrastructure have to be bought, planning processes need to be adapted, planners need to be trained and data need to be prepared. However, if conflicts occur across company boundaries, today's planning systems are still unable to automatically generate alternative solutions. Regarding the effort companies have already put in their planning systems it seems rational to further consider a relative small investment in connecting the systems of different companies. This thesis aims at giving a blueprint how such an endeavor could be realized.

Having set up the customized DEAL framework, the planners could let it run overnight and discuss the results in the next morning. The prioritization of deadlines pertaining to external customers (or other local objectives) and the implementation of acceptance bounds for evaluating solutions automatically allow the planners to define the behavior of the search process a priori. As proposals, counterproposals and rankings are generated fully automated, there is no necessity to evaluate each solution manually.[23] Since side payments are not computed, DEAL automatically generates only a set of alternatives, and the best alternative found (or several good alternatives) are presented to the planners for future consideration. Even if prenegotiated contingents are sufficient, the coordination mechanism could help to search for more compact plans, releasing additional production capacity.

Nevertheless, fulfilling practical requirements does not mean that all business cases would benefit to the same degree from implementing this mechanism. The following list summarizes several additional aspects for business cases to benefit by implementing the DEAL framework as well as some pitfalls.

- Supply chains with an equally distributed value creation seem to have a larger incentive for introducing a coordination mechanism. Consider the case where the OEM creates 99% of the value (and the associated costs) of the SC. Obviously, in such a setting the OEM will have greater power than if he only created 50% of the value. From this perspective, the OEM might be tempted to obstruct the introduction of a coordination scheme. Later numerical tests suggest, however, that in the end, this paradigm harms the OEM. For example, due to a machine breakdown, suppliers might simply not be able to fulfill the OEM's request. In such cases, the OEM's production plan gets distorted and it might be difficult to manually find alternatives.

- For a successful application of the DEAL framework, the OEM's production policy needs to allow a certain degree of freedom to produce alternative plans. There exist certain forms of make-to-order production that simply not "allow" deviations from the optimal sequence. A typical example is the automotive industry, where suppliers have to fulfill high standards to attain a just-in-time delivery of assembly parts without any errors.

- The focus should be on coordinating few but "important" items, items that have only a limited safety stock due to the expenses for storage or capital lockup.

[23] In analogy, even today no intermediate solution of a local GA is examined manually.

- The planning problems need to have a certain complexity to generate a considerable coordination potential. For example, cases in which many supplier exists, but each supplier only delivers one kind of item only have a low coordination potential.

- The benefit of a coordination scheme comes with costs for implementation. For some industries, cheaper "coordination" possibilities might exists. For example, in consumer electronics, other components are usually installed to counteract supply shortages.

CHAPTER 8

Computational evaluation

8.1 Building blocks defining the complexity of coordination

Solving an intradomain RCPSP is about finding the right sequence of activities, which then determines the related start and finish times. Also, from an interorganizational perspective, the sequence of delivered items is important since it determines the earliest start dates for the OEM and proposed due dates for the suppliers. There exist several building blocks for interorganizational conflicts. In general, such conflicts can result if a domain's *local planning properties* lead to a delivery sequence that generates disadvantageous RCPSP instances for other domains. Different setup costs and times can cause such conflicts. While the schedule of one planning domain may include only a few minor setup activities, fulfilling the resulting delivery sequence might require major setup activities in the schedules of other domains. Hence, the setup activities consume resource capacities or generate penalties in the objective function (in the form of setup costs) such that delivery to the ultimate customers is delayed. Also, different activity durations may require a certain product mix in one planning domain to avoid idle times. However, the resulting delivery sequence might lead to idle times in other planning domains. Thus, accumulated idle times can again cause a delayed delivery to the ultimate customer. Last but not least, different mode costs and durations can also lead to conflicts. In general, the effect of the conflicts can be reduced by finding a good compromise between the different preferences on the delivery sequence. For example, the total lateness might be smaller if all domains experience small setup times than if one domain experiences very little and the others very large setup times. Next to these local problem properties, *interface properties*, such as transports between supplier and manufacturer, can further deteriorate the outcome. Finally, *unforeseen events*, such as machine breakdowns, demand action.

Subsections 8.1.1–8.1.3 present a few examples for the above building blocks. The aim is to present examples that are condensed to those ingredients that make the standard real-world problems difficult to coordinate. Later discussed test instances are compositions of these building blocks. If the coordination mechanism works on these test instances, it has a good chance to be successfully applied to a wide range of real-world problems.

For the computational tests, we assume that the parties generally aim at decreasing the lateness of goods delivered to the OEM's ultimate customer. As discussed in Chapter 7, suppliers are thus supposed to support the OEM's proposals even though these proposals cause high setup times or costs, mode costs or makespan within their local planning problems. However, suppliers are not willing to accept an increased delay of orders to their own (external) customers. On a short-term level, the idea of fulfilling customer demand in time is in line with the prevailing practice. In general, however, the ranking scheme can also be adapted to support other objectives as has been discussed in Section 7.4.

8.1.1 Local properties

In general, conflicts can result if suppliers' objectives contradict the OEM's objective. For example, while the OEM is interested in finding schedules with minimum lateness, his suppliers might focus on schedules with minimum setup times. Under such circumstances, coordination will likely turn out to be useless; no matter what proposal the OEM is sending, the suppliers will continue to prioritize their setup time objectives and the resulting counterproposals will not change substantially over time. As discussed, we suppose that the suppliers are willing to fulfill the OEM's proposals to the extend possible. Hence, even though the suppliers' objective functions do include setup time and makespan objectives to steer the search process, total and maximum lateness objectives are assumed to play a major role. Conflicts between the suppliers and the OEM arise, if the objectives lead to a bad initial schedules generated by upstream or downstream planning. The following subsections present several examples for such cases.

8.1.1.1 Conflicting setup times

One source of conflicts is sequence-dependent setup times or costs. For example, we can think of situations where the supplier wants to group several activities requiring the same setup state of a resource, whereas the OEM prefers the steady delivery of an unvarying mix of input products.

We illustrate this idea with several small examples. Figure 8.1 depicts the situation: One supplier is delivering three goods A1, A2 and B1 to an OEM. In the figure, the size of the rectangles relates to the duration of the production activities. Sequence-dependent setup times occur on the OEM's resource R3, where producing A3 after B2 needs twice the setup time of B2 after A3. The supplier is assumed to need a setup on his resource R1, if A1 is followed by A2 or vice versa. The duration of setup intervals are depicted by the shaded areas in the respective setup matrix. Precedence relations are shown by arrows. In the example, A1 and A2 need to be finished before A3 can start, and B1 is a predecessor of B2. The picture shows the most relevant part of data and will henceforth be denoted as the *problem design*.

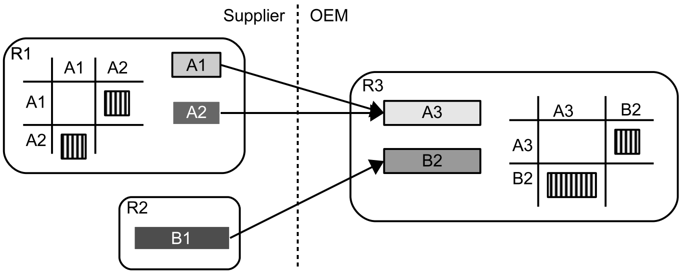


Figure 8.1: Design of setup conflicts—example 1.

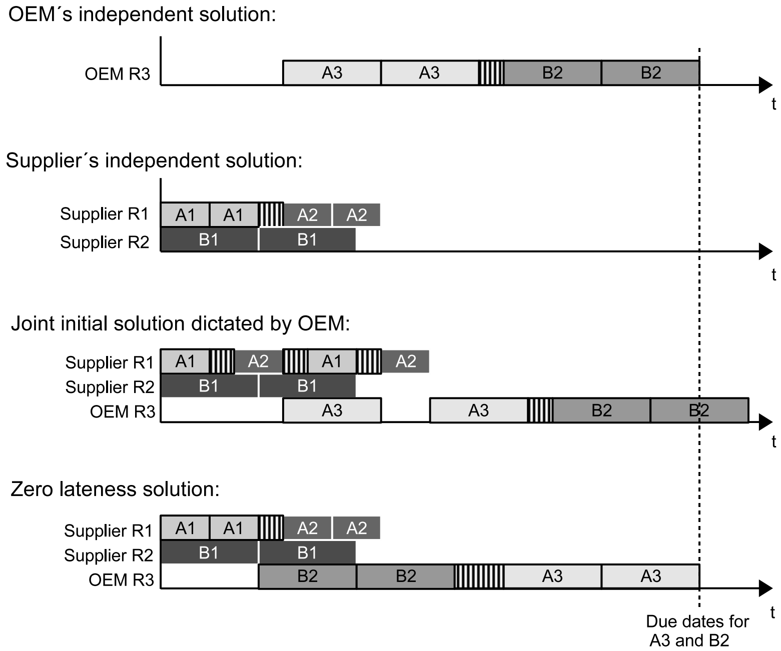


Figure 8.2: Coordination of setup conflicts—example 1.

Let us assume that an RCPSP instance is generated by taking each of the above activities two times. Figure 8.2 shows several possibilities to come to a joint solution for such a problem instance.[1] The first two Gantt charts show the independent solution of each partner, the sequences of activities with minimum lateness[2], setup times and makespan if there is *no* connection to the other partner. Obviously, these schedules (and the implied delivery sequences) do not match each other. While the OEM prefers A3, A3, B2, B2 as a production sequence, the supplier's independent solution implies B2, A3, B2, A3 as a production sequence for the OEM.

The conflict can be "solved" by one partner taking over the leadership or by finding a compromise. Let us suppose that the OEM dictates the joint initial solution. That is, upstream planning is applied at the beginning. As discussed in Section 7.3.1, the OEM generates his initial proposed dates by applying the DS solving module followed by the right-alignment heuristic. Not knowing the supplier's scheduling problem, the OEM communicates his preferred delivery sequence (A3, A3, B2, B2) by proposing related delivery dates. Trying to fulfill the OEM's desire, the supplier schedules his production as close as possible to the imposed sequence.[3] Sticking to the sequence requires the supplier to produce A1 and A2 in an alternating fashion on his resource R1. The alternating production entails setup times that finally lead to an idle time in the OEM's schedule (between the two A3 activities). It is remarkable that the OEM's preferred order can cause idle times at his *own* resources as soon as it comes to an interplay with the supplier. The still-in-practice prevailing tenet, "it is the best for the whole SC if the OEM decides and suppliers execute," is clearly disproved even by this small example. The last part of Figure 8.2 shows the zero-lateness solution. The due dates for A3 and B2 have been set such that DEAL can just manage to achieve zero lateness by generating the depicted schedules. Setting the due dates more right

[1] For better legibility, the pegging net is only drawn in the problem designs, but not in the Gantt charts henceforth.

[2] For the supplier, the lateness criterion is not of importance for the independent solution, as due dates have not been set by the OEM.

[3] As discussed, we assume that the penalties for total and maximum lateness are a relevant factor in the supplier's objective function when calling the DS Optimizer.

would leave more freedom for different schedules to achieve zero lateness as well, i.e. the need for coordination would decrease. Setting the due dates more left would just add a constant amount to the lateness objective that cannot be decreased further. As can be seen, the zero-lateness solution avoids unnecessary idle times. In contrast to the uncoordinated schedule, the OEM runs the production out of the sequence of the supplier's independent solution (not true for the general case). For the OEM, the larger setup time between B2 and A3 is more than compensated by the reduction of overall lateness. Moreover, less setup time is wasted at the supplier's side. It is an interesting question if and how fast the coordination mechanism will find such allocations that are against the OEM's myopic perspective in initial upstream planning.

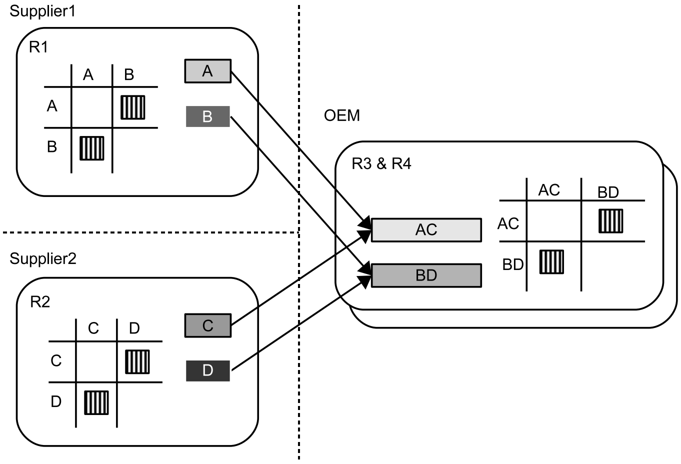


Figure 8.3: Design for setup conflicts—example 2.

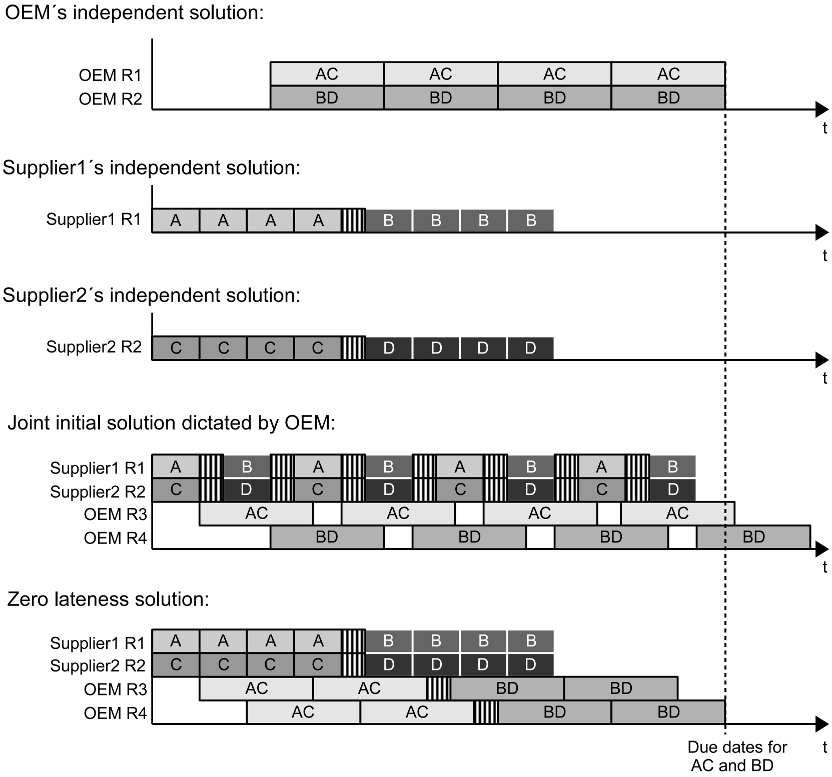


Figure 8.4: Coordination of setup conflicts—example 2.

The next example extends the first example to a three-partner scenario, where two suppliers deliver to one OEM. Figure 8.3 shows the related problem design. The two overlapping resources, R3 and R4, indicate that activities AC and BD can be scheduled on both resources (i.e., by using a different mode). Again, delivery sequences of the independent solutions contradict each other as highlighted in Figure 8.4. If the OEM takes over the leadership and implements upstream planning as an initial strategy, setup and idle times result. Obviously, the complexity of the coordination task increases if more partners are involved. Problems are imaginable where a single partner′s scheduling decision forbids the implementation of an overall promising solution. However, an increased number of partners does not necessarily imply that the coordinated results get worse. As the number of partners increases, also the intradomain problems get smaller—and might hence be solved more effectively.

For the above two examples, the need for coordination depends on the initial strategy. In fact, downstream planning would immediately lead to optimal results. However, the success of such simple successive planning strategies is not generally guaranteed.

The third example sketches a situation where neither the suppliers nor the OEM know the optimal production sequence in the beginning and a zero-lateness solution can only be achieved by central planning or—hopefully—by the DEAL framework. The problem design of the scenario is shown in Figure 8.5. Figure 8.6 shows the independent, uncoordinated and coordinated solutions. The worst initialization strategy is upstream planning. Downstream planning already leads to a smaller delay to the ultimate customer. However, in the coordinated solution, Supplier 1 and the OEM need to make mutual concessions with respect to their independent solutions in order to reach a schedule with less delay.

Similar to setup times, setup cost leads to a certain delivery sequence if included in the objective function. However, even if the fulfillment of a delivery sequence requires a schedule with high setup cost, setup activities do not waste resource capacity if setup times are negligible. In the computational tests, test instances with setup costs were not explicitly tested.

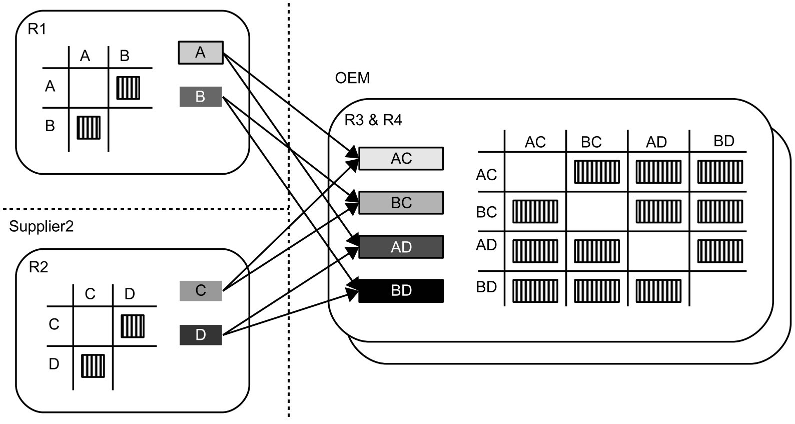


Figure 8.5: Design for setup conflicts—example 3.

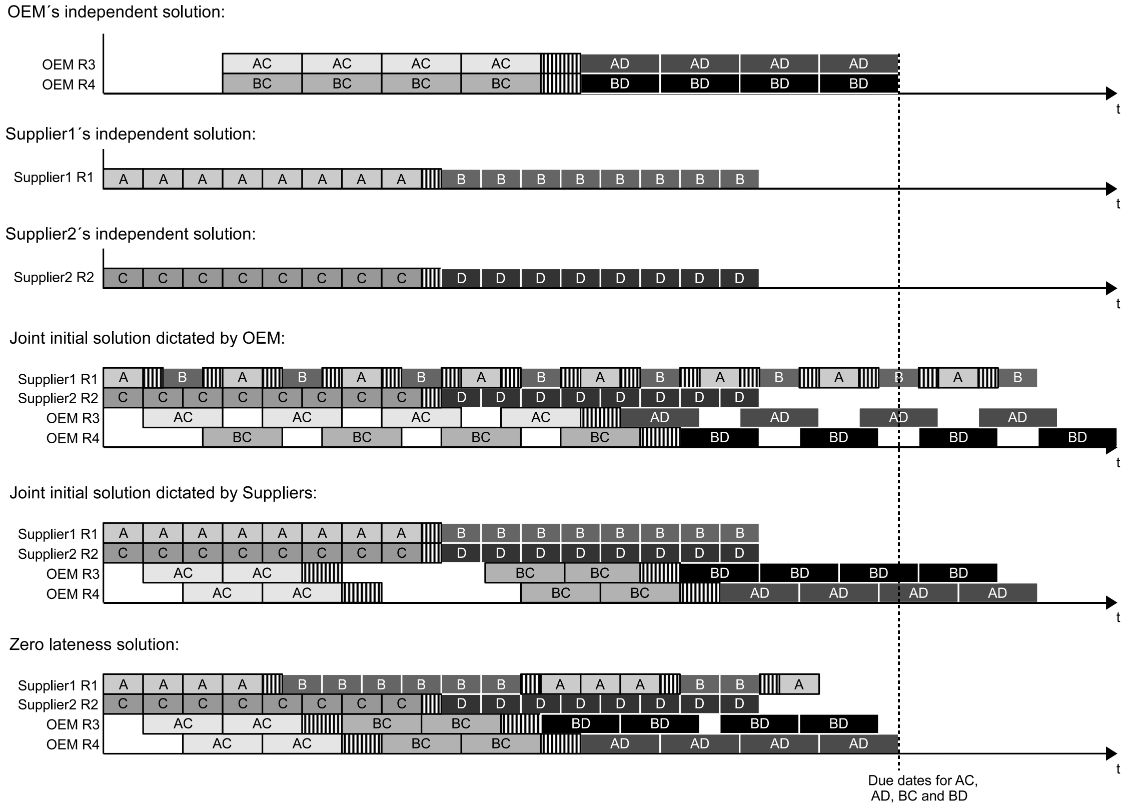


Figure 8.6: Coordination of setup conflicts—example 3.

8.1.1.2 Conflicting activity durations

Another cause for conflicts is different activity durations that imply a certain production sequence in order to avoid idle times. For every production stage, the activities must be arranged according to the lead time implied by their preceding activities. However, these sequences can be different from the perspective of OEM and supplier. This leads to many scattered idle slots if a successive planning strategy (e.g., upstream planning) is applied. Figure 8.7 exemplifies the cause for such conflicts for one supplier delivering to one OEM. The intermediate production activities A2 and B2 require A1 and B1, respectively. Again, the size of the rectangles refers to the activity durations.

The OEM has a certain preference for the sequence of activities on R2 and R3 from his myopic perspective. The independent optimal schedule is shown by the first Gantt chart in Figure 8.8. The supplier is initially indifferent about his production sequence. Suppose the OEM's independent solution serves as a cornerstone for the first joint solution. Since the OEM is not aware of the durations of the supplier's activities, he dictates a delivery sequence (by transmitting proposed dates) that ultimately leads to idle times on his own resources. As for setup problems, upstream

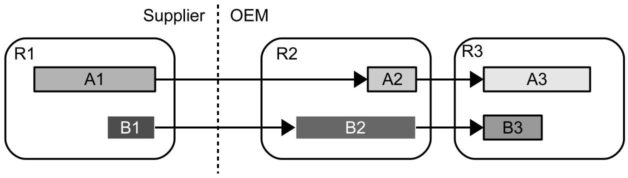


Figure 8.7: Design for conflicting activity durations.

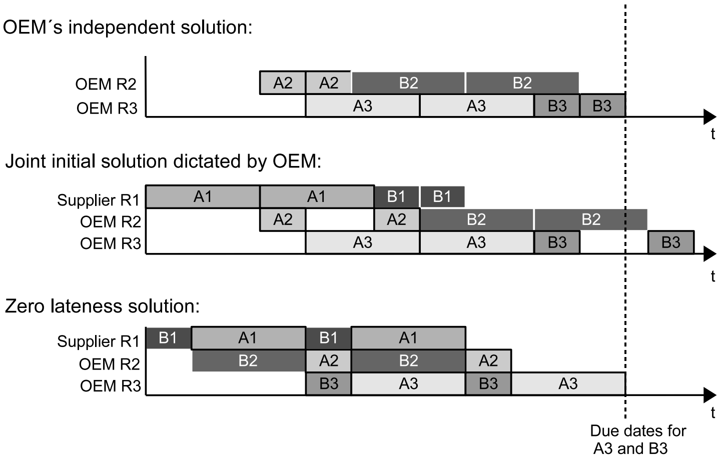


Figure 8.8: Coordination of conflicting activity durations.

planning can cause more harm than benefit. The last Gantt chart of Figure 8.8 shows the optimal schedule. This schedule can be realized without forcing any partner to make concessions apart from changing the order of activities on his resources. Interestingly, no partner knows the optimal sequence in the beginning. Moreover, all partners can only win, since idle times only decrease through coordination. However, the chance for building the initial schedule in this optimal sequence at random is very small, especially if more activities are considered.

Of course, if the supplier is also delivering to other customers, he is not completely indifferent to his production sequence. However, depending on the fraction of activities delivered to other customers, he is indifferent to a smaller or larger set of possible schedules.

8.1.1.3 Conflicting mode durations and cost

Another cause of conflicts is different durations of modes. Figure 8.9 exemplifies the design of an idealized coordination problem. One supplier is delivering two items A1 and B1, relating to the OEM’s two succeeding activities, A2 and B2. Each domain has two production resources. Though the production of the items can be scheduled on each of the two resources, it is preferable to schedule A1 on R1, B1 on R2, A2 on R3, and B2 on R4, otherwise larger mode durations need to be taken into account (indicated by the dashed boxes in the figure). Under such conditions, both partners actually prefer a steady flow of both items. However, we suppose that the resources of the OEM are not available all the time, as depicted in Figure 8.10 (a lightning bolt is drawn on slots with zero capacity).

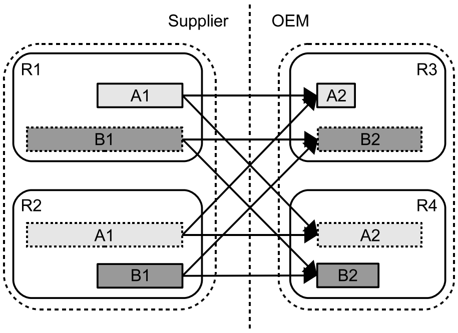


Figure 8.9: Design for mode-duration conflicts.

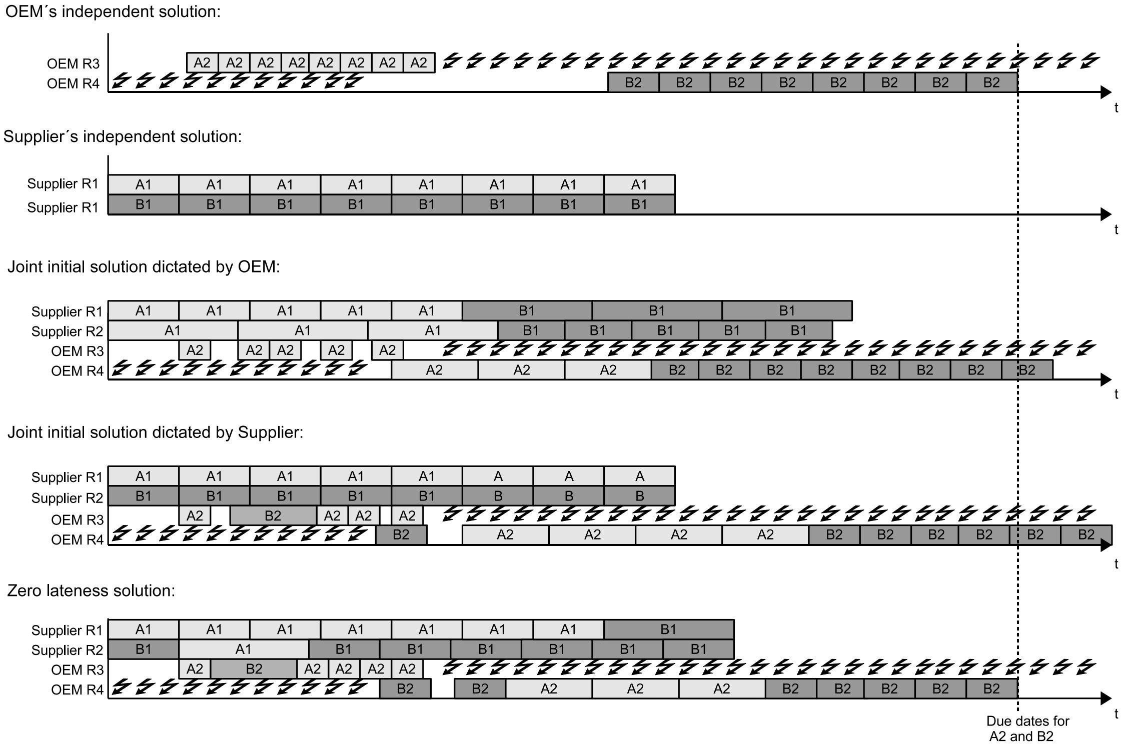


Figure 8.10: Coordination for mode durations conflicts.

Under such conditions, the OEM wants to receive first all A items, whereas the supplier still prefers a steady delivery of both items. The figure shows that both independent solutions lead to a delay to the ultimate customer if imposed to the other partner during successive planning. It can be seen that a compromise is again the best result. Also, mode costs influence the structure of the objective function, leading to certain delivery sequences that are problematic for adjacent planning domains. For simplicity we did not consider them in the computational tests, however.

8.1.1.4 Summary

Concluding, it can be said that several local properties have an influence on the delivery sequence. If the local properties of adjacent planning domains are in conflict, initial settings with a considerable coordination potential result. Not surprisingly, local properties can relate to either the constraints (e.g., precedence constraints, resource capacity or activity durations) or to the objective function (respectively the related penalty values) of an RCPSP instance. The above examples are by no means exhaustive; they are intended to give a flavor of the causes of uncoordinated initial global solutions. It is remarkable that, even though suppliers intend to cooperate, a delivery sequence imposed by the OEM can harm the OEM in the end by creating unnecessary idle times, setup times or activity durations in all planning domains. In such cases, coordination is advantageous for all parties: for the suppliers because they have more resource capacity available to schedule orders belonging to other customers and for the OEM because of reduced lateness of activities. It can be argued that—depending on the problem design—a set of advantageous schedules exists that can be explored even without exchanging side payments. Taking into account the difficulties such payments introduce, these findings underpin our approach of not considering side payments.

For the sake of simplicity, the above examples did not contain any activities belonging to external customers orders with prioritized deadlines. The test instances presented in sections 8.2 and 8.3 do contain such orders. As the computational tests will show, even in these cases suppliers have some freedom for rescheduling their production and the initial situation can be improved greatly.

8.1.2 Interface properties

The above building blocks cause idle times, setup times or larger mode-dependent activity durations in uncoordinated schedules. These idle times lead to a order delay with respect to ultimate-customer due-dates and hence result in scenarios with a huge potential for coordination. Properties of the interface between the partners can even further intensify the chance for and the length of such uncoordinated idle times. Assume that the supplied products are delivered overnight. We can model this property by introducing an artificial *transport resource* on the suppliers side. The resource has infinite capacity during the night, but no capacity during the day. As a new sink for each supplier's final activity a transport activity is introduced in each case. The transport activities restrict the delivery of products to happen only overnight. Orders that are too late for the current slot can only be delivered at the next transportation possibility. This discontinuity can lead to a larger overall makespan, as shown in Figure 8.11. Moreover, the transport activities "disrupt" the coordinated results. The coordination mechanism has to shift delivery dates to give as little disruption as possible.

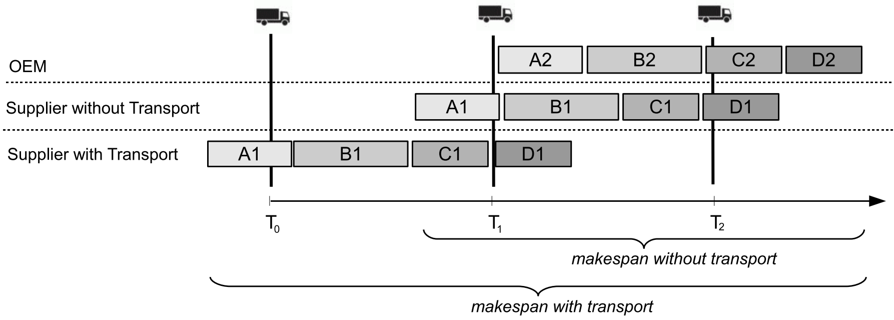


Figure 8.11: Illustration of transport slots and implication on overall makespan. The transport of a good must start at T_0 or T_1 to be delivered at T_1 or T_2, respectively.

As with transport slots, the production at the supplier or OEM can be modeled to have non-preemptive activities. For example, an activity cannot be paused over the weekend, during the night or must not be performed by two different shifts. Again, the coordination mechanism must avoid delivery dates for which activities only fit badly between the breaks.

8.1.3 Unforeseen events

From a practical perspective, a common source of conflicts is machine breakdowns. Assume the case where—due to the breakdown of a machine—a supplier has insufficient capacity to fulfill all orders of the OEM in time. The task is then to *limit the damage*, to delay those orders that are of minor importance to the OEM. The other way round, the OEM might experience a machine breakdown and need important orders of input products even earlier, whereas less important orders can be delayed. A common problem is that it is not always clear at first sight which orders are more or less important because of the complexity of the planning problem and the interdependency of production activities. This has already been illustrated by the conflicting modes examples, cf. Figures 8.9 and 8.10. When users manually shift those orders earlier that *seem to be* important, they might open the floodgates for other bottlenecks that are even more severe than the initial setting.

As with a change in the capacity profile, preferences of final customers can change. For example, customers might ask the OEM to deliver some orders earlier. The question is then how well the DEAL framework can propagate such changes through the SC. Finally, contingents of prenegotiated frame contracts may be outdated, and additionally production capacity is required from the OEM and his suppliers. In such cases, the DEAL framework should support the planning domains in jointly allocating resource capacity without delaying present delivery to the ultimate customer.

8.2 Deterministically generated test instances

The first test data generator (TDG) mimics the main functionality of a SAP module called Production Planning (PP). Given material quantity targets, this procedure determines the required production lots. In order to satisfy ultimate customer demand, the procedure traverses the different production stages and constructs replenishment and demand elements according to the BOM (similar to classical Material Requirements Planning). Related lot sizes are calculated according to predefined rules, such as the lot-for-lot rule. Moreover, a pegging net between replenishment and corresponding demand elements is constructed. Afterwards, the pegging net is frozen, which implies a fixed precedence relation between production activities that produce the material and those that require it. The production activities are then scheduled by the DS Optimizer, which tries to find the optimal sequence of the activities on the resources while respecting the precedence relations implied by the frozen pegging net and the capacity restrictions. Briefly summarized, PP generates an intradomain RCPSP instance out of master data and current demand information. Hence, the structure of the activity network is not totally random but a repetition of activity precedences, depending on the demand of output items, the BOMs used and the lot-sizing rules. We will henceforth refer to this generator as deterministic test data generator (DTDG).

8.2.1 Test data generation

In order to generate reproducible test data, a Simplified Production Planning procedure (called SPP) was implemented. As input, the user can specify the maximum lot size per product as a standard lot-sizing rule. The SPP algorithm then tries to construct demand and replenishment elements that have exactly this maximum lot size.[4] The SPP run does not consider resource capacity or sequence-dependent setup times, generating, instead, an "infinite" schedule.

Throughout the SPP procedure, the master data (e.g., the BOM for single production steps, related capacity requirements, maximum lot sizes and duration of modes) remain constant. Essentially, the whole process is triggered by the amount and distribution of the initial demand to be satisfied. An example of a demand matrix is shown in Table 8.1. The coefficients of the matrix denote the demand to be distributed per bucket and product. This facilitates the modeling of different seasonal, equal or random demand series for every product. In the current implementation,

[4] In some situations the lot sizes might be smaller. For example, at the end or beginning of the planning horizon.

	Bucket I	Bucket II	Bucket III	Bucket IV
Product A	10	15	20	5
Product B	8	10	5	5
Product C	3	2	7	10

Table 8.1: Example of a demand matrix with three products and four buckets.

the DTDG sets the RCPSP due dates at the end of the respective bucket. Before continuing with a detailed discussion of the DTDG, the relevant master data is presented.

8.2.1.1 Master data

For modeling the master data, several C++ classes are available to the user of the DTDG. For the sake of simplicity, the DTDG master data has less power of expression than real-world SAP master data. Thus, the generated RCPSP instances do not demand all the capabilities of the DS Optimizer. Nevertheless, it is possible to model problem properties leading to previously discussed interorganizational conflicts. The following list gives an overview of the available C++ classes for modeling master data. In order to distinguish from standard SAP master data we use the prefix *simple*.

- A **simple resource** defines a resource needed for production. Simple resources can be unary or multicapacitated. Except for breaks (production pauses), varying capacity cannot be modeled. Unary resources can further be distinguished as those with sequence dependent setup times and those without.

- A **simple mode** defines the resource required by a production activity and the related capacity consumption. The name simple mode stems from the fact that such a mode connects an activity to exactly one resource. Normally, a mode can have one primary and several secondary resources. For simplicity, secondary resources are not supported by the DTDG.

- A **simple link** defines an input or output relation between a product and a production step: how many input products are required or produced by a simple production process model (see below). Together with the computed pegging net, simple links determine the precedence constraints in the generated RCPSP instance.

- A **simple production process model (SPPM)** defines a single production step and an optional setup activity. It serves as compound entity to connect modes, resources, links and products (see below). Again, the word simple relates to a simplification of actual possibilities in the original SAP procedure, where several production steps can be modeled by one production process model. If an SPPM is run, it produces replenishment elements according to the output-link specifications and demand elements according to the input-link specifications. An activity of the RCPSP model describes a continued use of one SPPM during a time interval in order to produce the required lot size. Each SPPM can also trigger a separate setup activity for resources with sequence-dependent setup times.

- A **simple product** defines an input, intermediate, or output product. Each product has a maximum lot size and can be assigned as input to different SPPMs but can only be the output of one single SPPM. The SPP run tries to find pegging assignments such that the maximum lot size is reached for as many lots as possible.

- The **setup matrix** stores the time and costs associated with the transitions from one setup state to another. To each SPPM, a setup key can be assigned referring to a row and a column in the setup matrix. Thus, using the setup matrix, the user can define sequence-dependent setup times and costs.

Figure 8.12 illustrates the relationships between objects of the above classes.

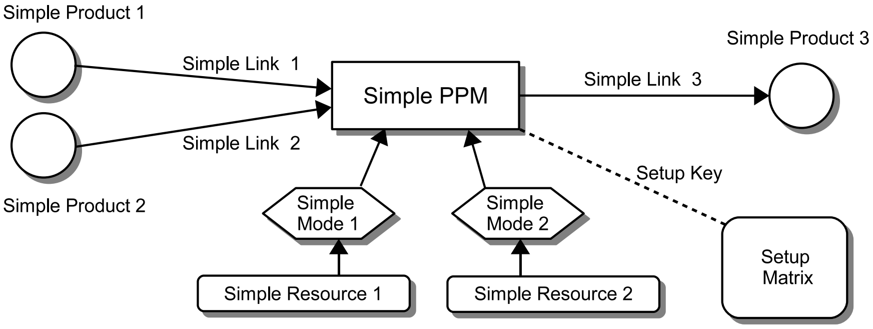


Figure 8.12: Master data for the test-data generator.

8.2.1.2 Generation algorithm

For given master data in the form of objects of the above classes and a demand matrix, the SPP run works as follows. First, demand elements are constructed according to the maximum lot size and the demand matrix. Then, a stage-numbering algorithm calculates production-stage numbers for each SPPM. The production-stage number of a SPPM equals the maximum number of predecessor SPPMs, whereas predecessors are determined over product input and output relations (defined by the simple links and products). Traversing the SPPMs in direction of descending production-stage numbers, each output product of the current SPPM is checked for yet unsatisfied demands. If there are any, several activities, respecting maximum lot sizes, are generated. On the one hand, information is stored for each activity on the duration required for the respective SPPM to generate the desired number of replenishment elements to (partly) satisfy the demand. On the other hand, each activity triggers new demand (according to the SPPM's required input products) that has to be covered by preceding productions stages. Demand and replenishment elements are dynamically grouped by a pegging net. For each demand element, the pegging net defines the related replenishment elements to cover the demand, leading to a precedence relation over all activities. Finally, all original and dependent demand is satisfied by a number of activities. A detailed description of the algorithm, `simplePPrun`, can be found in Appendix A.

Moreover, the DTDG allows an automated consideration of transport slots. If enabled, it assigns a transportation resource to each supplier, that "opens" for transport after a specified time interval and "closes" one second later. When "open", capacity is infinite, otherwise zero. Transport activities are modeled to take zero seconds. This allows the introduction of problematic "interrupted" delivery without requiring an adaptation of a scenario's due dates because of additional transportation times.

Still, the DTDG creates only one intradomain RCPSP instance that needs to be split up further. In a supplier–manufacturer relationship, planning domains commonly neither share resources nor exchange information concerning resource usage as discussed in Chapter 4. Hence, starting point of splitting up the generated RCPSP instance is an assignment of resources to planning domains. It is important that this resource assignment does not lead to cyclic dependencies between the planning domains (the suppliers are only delivering to the OEM and not vice versa). Moreover, all modes of an activity must relate to resources in the same planning domain. We assume a feasible resource assignment as further input from the user. For such an assignment, we can split up the remaining data (activities, release dates, due dates, calendars, and so forth) accordingly. During this process we can automatically gather the information relating to interorganizational precedence constraints, i.e. the information on how activities of a planning domain precede or succeed other activities of adjacent planning domains. This information is needed during the coordination process for translating proposed dates to due dates and finish dates to release dates, cf. Chapter 7.

8.2.2 Test instances

The building blocks discussed in the previous section were combined into a single benchmark scenario with structured complexity. This scenario comprises three subscenarios. 13 test instances were generated by supplying the DTDG with different demand matrices.[5] We start by describing the scenario before discussing the demand matrices.

8.2.2.1 The benchmark scenario

We assume the OEM to be an idealized producer of lawnmowers. Each mower consists of 7 components: A chassis, a motor, a tank, a blade, a handle, wheels and a control on the handle to control the motor. While the OEM assembles the components, several suppliers are responsible for providing them. Handles, chassis and blades are delivered by a supplier of steel material, henceforth called "Supplier Steel". "Supplier Plastic" is responsible for tanks and wheels. Finally, motors and controls are supplied by "Supplier Motor". All in all, four different kinds of lawnmowers (small, medium, large and extra large) are produced in a converging production process. The design of the scenario is depicted in Figure 8.13. The benchmark scenario can be divided into three subscenarios concerned with the setup-time, activity-duration and machine-breakdown conflicts discussed in Section 8.1.

The *setup subscenario* consists of two parts. The first part covers the assembly of chassis and motors. Supplier Steel is supposed to use a moulding press for forming the chassis. Two types of chassis are produced: a large one (lChass) and a small one (sChass). Each type requires a different setup state of the moulding press. Next to the chassis, two types of motors of different size (sMotor, lMotor) are delivered by Supplier Motor. Again, the production of each motor type requires its own setup state. Having received the items, the OEM assembles large motors with large chassis (lMlC) and small motors with small chassis (sMsc) on two parallel unary resources. Also this assembly is subject to sequence-dependent setup times. The resulting setup conflict has already been discussed in Section 8.1.1, see also Figures 8.3 and 8.4. The second part of the setup subscenario is about producing and assembling handles and controls and is related to the building block shown in Figures 8.5 and 8.6. Here too, OEM, Supplier Steel and Supplier Motor produce the goods on unary resources with sequence-dependent setup states.

The *breakdown subscenario* defines another production subprocess. Suppose that stock of several intermediate mowers are at the OEM's disposal (they are output of another subscenario discussed later). There exist two variants of intermediate mowers, where handles, controls, blades and wheels are still missing, however. One variant is a combination of a large motor, a large

[5] Two more test instances were tested with a reduced set of operators, cf. Section 8.4.

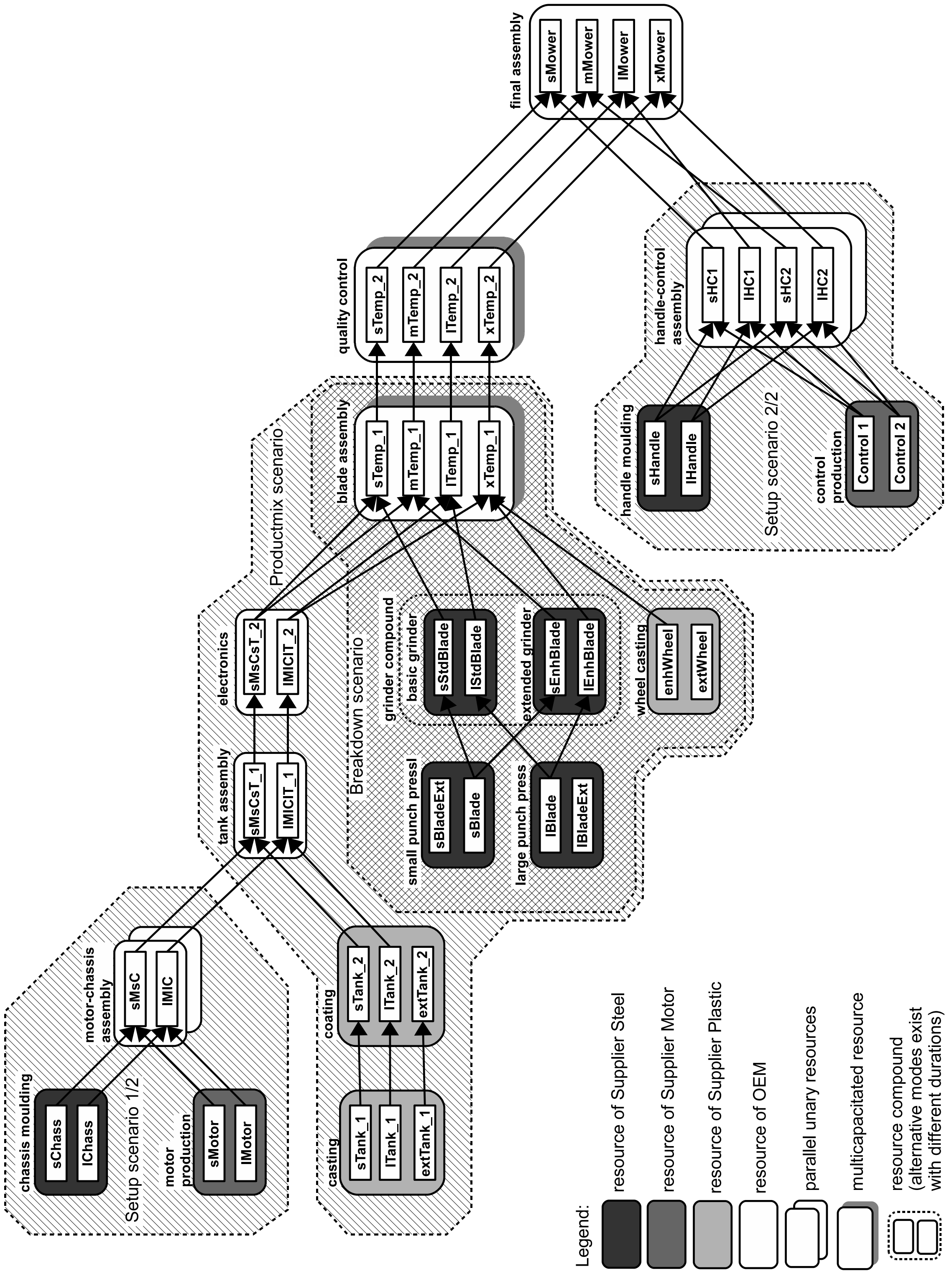


Figure 8.13: Design for the benchmark scenario. The hatched areas represent the subscenarios "product mix", "setup" and "breakdown". The durations of activities or modes are not depicted (all activities are drawn in equal length) and setup matrices are not drawn. For the grinder compound only the preferred modes with shorter duration are depicted. Each activity of the basic grinder can also be scheduled on the extended grinder and vice versa.

chassis and a large tank (lMlClT) and the other consists of related small components (sMsCsT). A further assembly with different wheels and blades leads to 4 variants of unfinished mowers: sTemp_1 is a combination of sMsCsT with a small, standard blade (sStdBlade); mTemp_1 consists of sMsCsT with a small, extra-sharp blade (sEnhBlade); lTemp_1 combines lMlClT and a large standard blade (lStdBlade); and xTemp_1 is composed of lMlClT and a large, extra-sharp blade (lEnhBlade). The blades are delivered by Supplier Steel. In addition, variant xTemp_1 requires a special set of enhanced wheels (enhWheel), delivered by Supplier Plastic (the other three variants are supposed to take standard wheels available from stock). The OEM assembles the components on a multicapacity resource called blade assembly. Supplier Steel owns four resources for producing the blades. Two punch presses, one for each size and two grinders for sharpening the blades, one basic and one extended grinder. Both grinders can be used to produce standard and extra-sharp blades. However, the basic grinder takes considerably longer to produce extra-sharp blades than the extended grinder (similar to the mode conflicts shown by Figures 8.9 and 8.10). No sequence-dependent setup times occur in this subscenario. The potential for coordination arises as some machines are breaking down. Details on repair times and resulting conflicts are given later.

In the *product mix subscenario*, production steps for casting and coating a tank (Supplier Plastic) and assembling it with the motor–chassis components (OEM) are added. However, machine breakdowns do not occur in this subscenario. The uncoordinated setting has a coordination potential because of conflicting activity durations for producing and assembling the tanks, similar to the building block relating to Figures 8.7 and 8.8. For an equal distributed demand, the best sequence is to produce sMsCsT and lMlClT alternatingly. In addition, different mode durations for sharpening the blades require a subordinated sequence (to produce standard and extra-sharp blades in parallel). Hence, the optimal solution demands a certain sequence of activities, a certain product mix.

In addition to delivering to the OEM, Supplier Steel and Supplier Plastic also have external customers and related activities (extTank_1, extTank_2, extWheel, lBladeExt and sBladeExt). As already indicated in Section 7.1, external customers of suppliers are prioritized by modeling related delivery dates as deadlines. The OEM's final assembly (sMower, mMower, lMower, xMower) connects all subscenarios. Moreover, an additional quality control is introduced here, requiring another multicapacitated resource. Table 8.2 lists all activity durations. Note that the maximum lot size was set to 20 items and that each SPPM has an input and output quantity of one for each item. Related sequence-dependent setup times are summarized in Table 8.3.

8.2.2.2 Generated instances of the setup subscenario

For the setup subscenario, four test instances were generated, a small one, two medium ones and a large one. The small instance of the setup subscenario consists of 576 activities to be scheduled in total, with 192 activities in the OEM's domain and 192 activities in each supplier's domain, respectively. At the end of the first day, the following demand q (in lot-sizes of 20 items) shall be satisfied: $q(sMsc) = 24$, $q(lMlC) = 24$, $q(sHC1) = 12$, $q(lHC1) = 12$, $q(sHC2) = 12$, and $q(lHC2) = 12$. The demand was chosen such that it can be fulfilled in time; there exists an ideal solution with zero lateness. According to Table 8.2, each activity producing one lot of sMsC or lMlC has a duration of 3,300s, respectively. Thus, in total, the production of both goods takes 22 hours on the parallel motor–chassis assembly resources. Since the lead time of the preceding

				Duration per item
Activity	Resource	Domain	Subscenario	(per max. lot size)
sChass	chassis moulding *	Supplier Steel	setup	80 (1,600)
lChass	chassis moulding *	Supplier Steel	setup	80 (1,600)
sMotor	motor production *	Supplier Motor	setup	80 (1,600)
lMotor	motor production *	Supplier Motor	setup	80 (1,600)
sMsC	motor–chassis assembly **	OEM	setup	165 (3,300)
lMlC	motor–chassis assembly **	OEM	setup	165 (3,300)
sHandle	handle moulding *	Supplier Steel	setup	80 (1,600)
lHandle	handle moulding *	Supplier Steel	setup	80 (1,600)
Control 1	control production *	Supplier Motor	setup	80 (1,600)
Control 2	control production *	Supplier Motor	setup	80 (1,600)
sHC1	handle–control assembly **	OEM	setup	160 (3,200)
sHC2	handle–control assembly **	OEM	setup	160 (3,200)
lHC1	handle–control assembly **	OEM	setup	160 (3,200)
lHC2	handle–control assembly **	OEM	setup	160 (3,200)
sTank_1	casting	Supplier Plastic	product mix	60 (1,200)
lTank_1	casting	Supplier Plastic	product mix	100 (2,000)
extTank_1	casting	Supplier Plastic	product mix	80 (1,600)
sTank_2	coating	Supplier Plastic	product mix	40 (800)
lTank_2	coating	Supplier Plastic	product mix	120 (2,400)
extTank_2	coating	Supplier Plastic	product mix	80 (1,600)
sMsCsT_1	tank assembly	OEM	product mix	120 (2,400)
lMlClT_1	tank assembly	OEM	product mix	60 (1,200)
sMsCsT_2	tank assembly	OEM	product mix	120 (2,400)
lMlClT_2	tank assembly	OEM	product mix	60 (1,200)
sBlade	small punch press	Supplier Steel	breakdown	90 (1,800)
sBladeExt	small punch press	Supplier Steel	breakdown	90 (1,800)
lBlade	large punch press	Supplier Steel	breakdown	90 (1,800)
lBladeExt	large punch press	Supplier Steel	breakdown	90 (1,800)
sStdBlade	basic grinder	Supplier Steel	breakdown	160 (3,200)
	extended grinder	Supplier Steel	breakdown	160 (3,200)
lStdBlade	basic grinder	Supplier Steel	breakdown	160 (3,200)
	extended grinder	Supplier Steel	breakdown	160 (3,200)
sEnhBlade	basic grinder	Supplier Steel	breakdown	260 (5,200)
	extended grinder	Supplier Steel	breakdown	200 (4,000)
lEnhBlade	basic grinder	Supplier Steel	breakdown	260 (5,200)
	extended grinder	Supplier Steel	breakdown	200 (4,000)
enhWheel	wheel casting	Supplier Plastic	breakdown	90 (1,800)
extWheel	wheel casting	Supplier Plastic	breakdown	90 (1,800)
sTemp_1	blade assembly ***	OEM	breakdown	180 (3,200)
mTemp_1	blade assembly ***	OEM	breakdown	180 (3,200)
lTemp_1	blade assembly ***	OEM	breakdown	180 (3,200)
xTemp_1	blade assembly ***	OEM	breakdown	180 (3,200)
sTemp_2	quality control ***	OEM	complete	300 (6,000)
mTemp_2	quality control ***	OEM	complete	360 (7,200)
lTemp_2	quality control ***	OEM	complete	380 (7,600)
xTemp_2	quality control ***	OEM	complete	400 (8,000)
sMower	final assembly	OEM	complete	90 (1,800)
mMower	final assembly	OEM	complete	90 (1,800)
lMower	final assembly	OEM	complete	90 (1,800)
xMower	final assembly	OEM	complete	90 (1,800)

Table 8.2: Activity durations for the benchmark scenario. Resources marked with a * indicate setup resources, ** indicates parallel setup resources and *** indicates multicapacitated non-setup resources. Duration in seconds. Standard maximum lot size is 20 items.

Chassis moulding

	sChass	lChass
sChass	0	3,200
lChass	3,200	0

Motor production

	sMotor	lMotor
sMotor	0	3,200
lMotor	3,200	0

Motor–chassis assembly

	sMsC	lMlC
sMsC	0	2,400
lMlC	2,400	0

Handle moulding

	sHandle	lHandle
sHandle	0	3,200
lHandle	3,200	0

Control production

	Control 1	Control 2
Control 1	0	3,200
Control 2	3,200	0

Handle–control assembly

	sHC1	sHC2	lHC1	lHC2
sHC1	0	1,600	1,600	1,600
sHC2	1,600	0	1,600	1,600
lHC1	1,600	1,600	0	1,600
lHC2	1,600	1,600	1,600	0

Table 8.3: Setup matrices for the benchmark scenario, all times in seconds, independent of lot size.

products is 3,200s, 4,000s are left to make the day complete: a setup can easily take place. Similar calculations can be set up for the suppliers' resources. If all planning domains schedule their production with as few setups as possible (cf. Figures 8.4 and 8.6) an undelayed production is possible.

The medium instance of the setup subscenario is four times the size of the small instance. It consists of 2,304 activities to be scheduled in total with 768 activities in each domain, respectively. Again, the demand was chosen such that it can be fulfilled: a solution with zero delay is theoretically possible by construction. Instead of demanding parts of the output every day, all items are demanded at the end of Day 4, whereas each item is demanded four times the amount of the small setup instance. It should be noted that the medium instance allows more "freedom" for placing the setup activities due to the increased planning horizon. Instead of 4,000s, 16,000s are now available on each resource for placing the setup activities. In fact, a few redundant setup activities do not immediately imply lateness—in contrast to the small instance. A second medium instance of the setup subscenario was created where Supplier Steel and Supplier Motor have been merged to a single planning domain. Our aim here was to study the effects a fragmentation of suppliers can imply. Demand matrix, capacities and so forth did not change, however.

The large instance is three times the medium instance and 12 times the small instance. It consists of 6,912 activities to be scheduled in total, with 2,304 activities in each domain, respectively. As for the other instances, schedules with zero lateness are theoretically possible by construction. The related demand matrix[6] is shown in Table 8.4.

[6] For better legibility, the demand matrices of medium and large instances always span one or two weeks and may contain a demand of zero in the most right columns.

Item	Demand (in lots a 20 items)													
sMsC	0	0	0	96	0	0	0	96	0	0	0	96	0	0
lMlC	0	0	0	96	0	0	0	96	0	0	0	96	0	0
sHC1	0	0	0	48	0	0	0	48	0	0	0	48	0	0
lHC1	0	0	0	48	0	0	0	48	0	0	0	48	0	0
sHC2	0	0	0	48	0	0	0	48	0	0	0	48	0	0
lHC2	0	0	0	48	0	0	0	48	0	0	0	48	0	0

Table 8.4: Biweekly demand matrix for the large instance of the setup subscenario.

8.2.2.3 Generated instances of the product mix subscenario

For the product mix subscenario, six test instances were generated: a small one, four mediums and a large one. The small instance of the product mix subscenario consisted of 403 activities in total, where 132 lie in the OEM's domain, 135 are handled by Supplier Plastic and 136 by Supplier Steel. At the end of Day 1, the following demand q (in lots of 20 items) needs be satisfied: $q(sTemp\_1) = 11$, $q(mTemp\_1) = 11$, $q(lTemp\_1) = 11$, $q(xTemp\_1) = 11$, $q(sBladeExt) = 24$, $q(lBladeExt) = 24$, $q(extWheel) = 24$, $q(extTank\_2) = 6$.

The medium instance of the product mix subscenario is approximately four times the size of the small instance. At the end of Day 4, the following demand q (in lots of 20 items) needs to be satisfied: $q(sTemp\_1) = 47$, $q(mTemp\_1) = 47$, $q(lTemp\_1) = 47$, $q(xTemp\_1) = 47$, $q(sBladeExt) = 96$, $q(lBladeExt) = 96$, $q(extWheel) = 96$, $q(extTank\_2) = 24$. In total, 1,699 activities were generated, where 564 activities lie in the OEM's domain, Supplier Steel handles 568 activities and Supplier Plastic is responsible for 567 activities. A second medium instance was generated as an exact copy with Supplier Steel and Supplier Plastic merged to a single domain. Two more medium instances with a similar number of activities were generated. For the third instance, additional transport slots were introduced at the interface between the suppliers and the OEM. Every day at midnight, a single transport from the suppliers to the OEM takes place (i.e., the transport resource is modeled having infinite capacity for one second). An activity of transporting has a duration of zero. As the first transport starts *at the end* of Day 1, reducing the lateness to zero is not possible because the OEM will experience a delay of at least one day. The fourth medium instance does not include transport activities, but is based on a much finer grained demand matrix. Instead of a full delivery after four days, the OEM is supposed to delivery a quarter of the total demand every single day. The related demand matrix can be found in Appendix G (Table G.1). Here, we wanted to investigate which effect a fine-grained demand distribution might have on central and coordinated solutions.

The large instance of the product mix subscenario was, regarding the number of activities and the length of the planning horizon, three times the medium instances and consisted of 5,155 activities in total (OEM: 1,716, Supplier Steel: 1,720, Supplier Plastic: 1,719). The related demand matrix can be found in Appendix G (Table G.2). As for the setup subscenario, schedules with a lateness of zero are—by construction—theoretically possible for all instances (excluding the one with transport slots).

8.2.2.4 Generated instances of the breakdown subscenario

The breakdown subscenario requires further explanation regarding the upstream planning solution and the coordination potential. Instead of allowing the OEM complete freedom in calculating his initial local solution, we suppose that a certain frame contract exists limiting the OEM in his initial decision. The general proceeding has already been discussed in Section 7.3.2. To simulate a frame contract, we first apply the DS Optimizer to the global RCPSP instance and take the resulting start dates of upstream-related activities as release dates for the OEM's initial solution. A coordination potential arises as the planning problem changes—for example, by machine breakdowns—and the release dates pertaining to the simulated frame contract do not fit any longer.

As can be seen in Figure 8.13, Supplier Steel has two grinders for sharpening the standard and the enhanced blades. In general, both grinders can be used for both types of blades. However, due to different mode durations, the extended grinder is the preferable tool for producing enhanced blades. Thus, the supplier can save resource capacity by sharpening blades of types sEnhBlade and lEnhBlade on the extended grinder. This favors a steady production of sEnhBlade and lEnhBlade on the extended grinder and sStdBlade and lStdBlade on the basic grinder. The situation has elements of the previously discussed building block of conflicting modes, see also Figure 8.9.

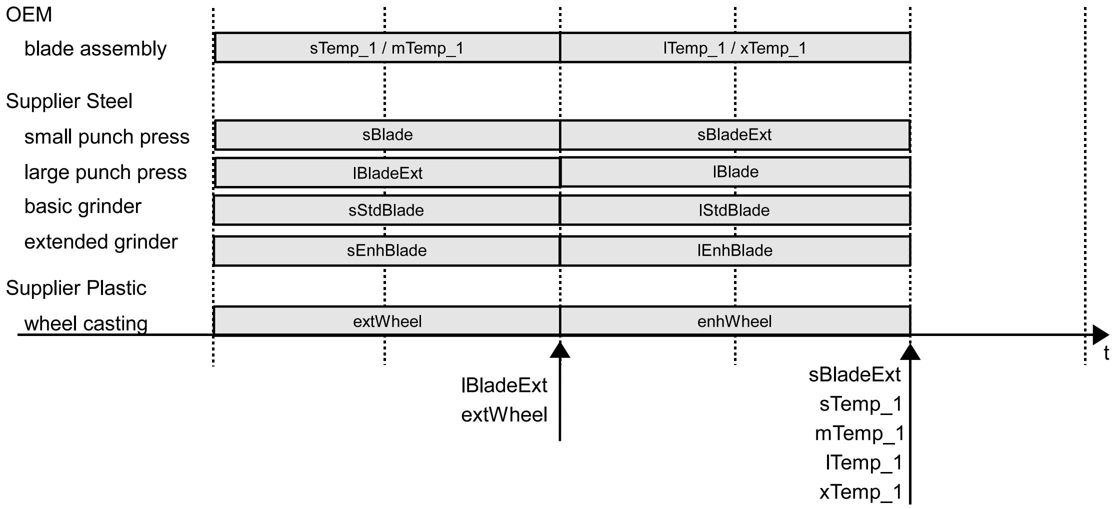


Figure 8.14: Due dates and ideal production sequence before machine breakdown.

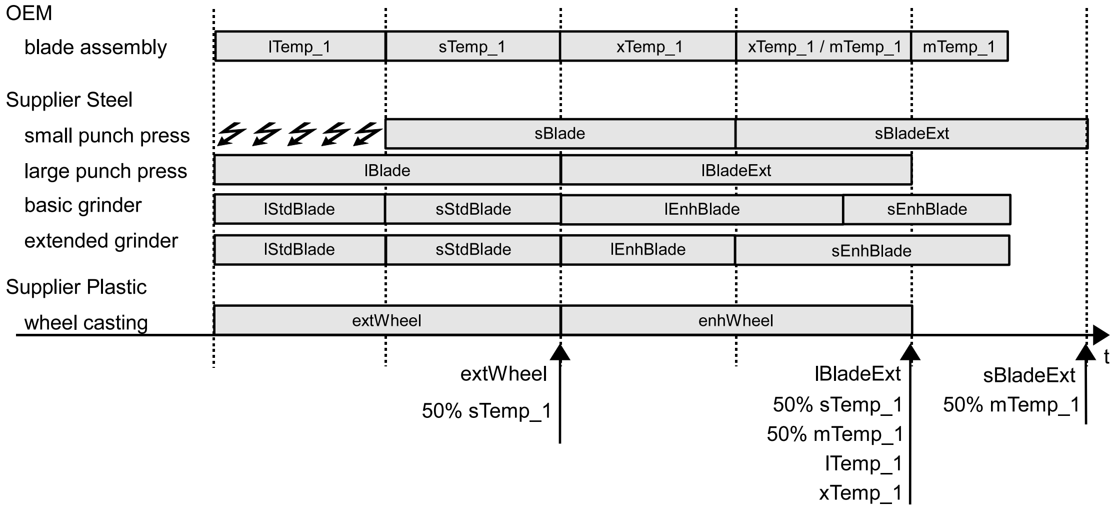


Figure 8.15: Due dates and ideal production sequence after machine breakdown.

In addition, Supplier Steel and Supplier Plastic produce items for external customers not taking part in the coordination. In the breakdown subscenario, the deadlines of these external activities have been set in such a way that a certain production sequence is required to avoid late delivery. The due dates and deadlines are shown by arrows and the frame contract's ideal production sequence is shown by boxes in Figure 8.14.[7] The planning horizon is divided into five buckets and the due dates are always set at the end of the related bucket. (For the frame contract, only the first four buckets are of importance.) It can be seen that Supplier Steel is parallel producing standard and enhanced blades in the preferable modes in order to save production capacity and time. Moreover, external unsharpened large and small blades have to be delivered at the end of Buckets 2 and 4. In addition, Supplier Plastic has to deliver wheels to an external customer at the end of bucket two. These three aspects determine the production sequence of the OEM: In the first two buckets, small standard and enhanced blades are assembled alternatingly and in the last two buckets large standard and enhanced blades are assembled. The DS Optimizer is used to compute the delivery dates of the frame contract by solving the complete (not yet divided) RCPSP instance using the due dates of Figure 8.14. Being a metaheuristic, the DS Optimizer does not guarantee to find the illustrated production sequence. However, the results suggest that the derived delivery dates cause the problems we intended to study.

Coordination potential arises, as capacity shortages are introduced and due dates to external customers are changed *after* the frame contract was computed. In our scenario, we assume that the small punch press of Supplier Steel is not available in the first day. The supplier was already able to negotiate a delay of external-related blades, as shown by the changed due dates of lBladeExt and sBladeExt in Figure 8.15. However, the parallel delivery of standard and enhanced blades to the OEM cannot be realized as promised. In addition, the OEM's customer demands 50% of the sTemp_1 items already at the end of Bucket 2. As setoff, the delivery of 50% of mTemp_1 can be delayed to the end of Bucket 5. Figure 8.15 shows a production sequence to respond to these developments. It can be seen that the required production sequence is entirely different from the initial solution in Figure 8.14.

Item	Demand (in lots a 20 items)						
sTemp_1	0	0 (24)	0	48 (24)	0	0	0
mTemp_1	0	0	0	48 (24)	0 (24)	0	0
lTemp_1	0	0	0	48	0	0	0
xTemp_1	0	0	0	48	0	0	0
sBladeExt	0	0	0	96 (0)	0 (96)	0	0
lBladeExt	0	96 (0)	0	0	0 (96)	0	0
extWheel	0	96	0	0	0	0	0

Table 8.5: Weekly demand matrix for the medium instance of the breakdown subscenario. Values in parentheses are the renegotiated demands after the breakdown occurred.

Note that this is not the only possible production sequence. Slight deviations can also lead to an overall solution with zero lateness. The RCPSP instance provides a certain amount of freedom for finding an alternative production and delivery sequence under what seems to be realistic assumptions. However, a new sequence can hardly be found manually, because the delivery of several highly interdependent items needs to be rearranged *concurrently*.

Table 8.5 shows the demand matrix that led to the generation of 192 activities in the OEM's domain. Supplier Steel received 576 activities and Supplier Plastic 144 activities, summing up to 912

[7] Here, a box does not relate to a single production activity but is a high-level visualization of several production activities of the same type.

activities to be scheduled in total. We will denote this instance medium—a small instance does not exist for the breakdown subscenario. However, two large variants of the breakdown subscenario have been tested with a total of 2,736 activities. The variants differ in the way demands are set and the breakdown occurs. In the first variant, the small punch press of Supplier Steel breaks down for the first three days. The demand matrix (Appendix G, Table G.3) still allows the rescheduling of the production such that no lateness results. In the second variant, the small punch press breaks down on Days 1, 4, and 8. Though a new demand distribution could be negotiated with customers (Appendix G, Table G.4), it is impossible to totally reduce the lateness. Here, the task of DEAL is to reduce the lateness as much as possible.

8.2.2.5 Generated instances of the complete scenario

The large instance generated from the complete scenario comprises 9,792 activities in total (OEM 5,188, Supplier Steel 4,034, Supplier Motor 2,306, and Supplier Plastic 1,728). The demand matrix can be found in Table 8.6. Again a total lateness of zero is theoretically possible by construction.

Item	Demand (in lots a 20 items)													
sMower	0	0	0	12	36	0	0	48	0	0	0	48	0	0
mMower	0	0	0	12	36	0	0	48	0	0	0	48	0	0
lMower	0	0	0	12	36	0	0	48	0	0	0	48	0	0
xMower	0	0	0	12	36	0	0	48	0	0	0	48	0	0
extWheel	0	0	0	96	0	0	0	96	0	0	0	96	0	0
sBladeExt	0	0	0	96	0	0	0	96	0	0	0	96	0	0
lBladeExt	0	0	0	96	0	0	0	96	0	0	0	96	0	0
extTank_2	0	0	0	24	0	0	0	24	0	0	0	24	0	0

Table 8.6: Biweekly demand matrix for the large instance of the complete scenario.

8.2.3 Test program

We applied a standard test program to all above described test instances. Because of the dependency of the results on the chosen initial random seed, a single run per test instance does not suffice. To be statistically more confident, 10 runs with different initial seeds for each coordination specification and centrally applied DS solving module were evaluated per instance.

First, for each test instance several **central runs** were performed where we applied the DS Solving module to each undivided test instance. The resulting central results serve as benchmarks to evaluate the performance of DEAL (the gap between the central and coordination runs was measured). Second, different specifications of DEAL were tested multiple times for each generated instance: a sequential, an asynchronous and two parallel **coordination runs** (with different population sizes). The related parameter settings are summarized in Table 8.7. For the complete instance, suppliers have one solving unit (i.e. an independent CPU for computing solutions, cf. Section 4.10) less than in the standard specification.

Depending on the size of an instance, different settings for the available runtime were chosen. For all small instances, central and coordination runs are aborted after 1,800s. During coordination, the DS solving module is terminated after 13s. This time interval will be denoted as **local runtime** henceforth. For the medium instances, these time limits were set to 3,600s and 25s, respectively. Large instances were allowed to run 7,200s and 50s, respectively. For computing the intermediate schedule needed for the frame contract, the DS Optimizer was run for 250s in the

Specification	Number of domain processes	Number of solving units	Population size	Queuing size
Sequential	1	n.a.	1	0
Asynchronous	3	n.a.	1	4
Parallel, pop. size 1	3	Suppliers 3 (2), OEM 4	1	12
Parallel, pop. size 12	3	Suppliers 3 (2), OEM 4	12	12

Table 8.7: Standard specifications for the computational tests. Values in parentheses are valid for the complete scenario.

medium and 500s in the large breakdown instance. The complete instance was computationally evaluated with the following time limits: 14,400s overall runtime and 100s for calling the DS solving module during coordination. For some instances, other time limits for calling the solving module were evaluated. A detailed discussion is provided later.

Some instances involve suppliers delivering to external customers not taking part in the coordination. As discussed in Section 7.1, the related due dates are prioritized (treated as deadlines) by DEAL. A difficult question is whether external customer due dates should also be prioritized in the central runs. On one hand, such a prioritization could accidentally support the central runs in finding good solutions. On the other hand, the results could worsen as the GA has less freedom in exploring the search space. For the sake of completeness, both alternatives have been computationally evaluated: **central hard-constrained runs** including a prioritization and **central soft-constrained runs** without any deadlines.[8]

As standard objective weights, we chose the following settings (for instances including setup times): weight of maximum lateness objective $wml = 98.9\% \ (98.8\%)$, weight of total lateness objective $wtl = 1\% \ (1\%)$, weight of makespan objective $wms = 0.1\% \ (0.1\%)$, and weight of total setup-time objective $wtst = 0\% \ (0.1\%)$. For simplicity, mode costs and setup costs have not been considered in the computational tests. It is worth mentioning, that the total lateness of a schedule is usually much larger than the maximum lateness. As a rule of thumb, it is said to be a good choice if total and maximum lateness get approximately the same influence in the objective function. As instances with a size from 500 to 10,000 activities have been generated, a tradeoff of $wml : wtl \approx 100 : 1$ seemed appropriate. Additionally, the makespan was considered with a very low weight. Preliminary experiments suggested that including the makespan in the objective function helps to steer the search process. The setup weight of $wtst = 0.1\%$ was calibrated according to results of central and coordination runs, as will be discussed later.

As discussed in Section 7.8, there exist two methods for the self-adaption of operators: a deterministic and a stochastic one. For each specification (sequential, asynchronous or parallel with population size 1 or 12) the deterministic self-adaptation has been tested with a predefined list of operator specifications. Operators that support a crude alignment were given a higher priority in the initial list, based on the intuitive idea that crude alignments are more useful at the beginning of the coordination. The initial list can be found in Appendix E (Table E.1).

Apart from the standard test program, additional tests have been applied to some of the test instances. These include different local runtime settings, the evaluation of different operator lists and the evaluation of the different objective weights. The stochastic operator selection has been tested as well, but only for parallel coordination runs with a population size of 12 and for a subset of test instances. All results related to the stochastic operator selection are discussed later

[8] Obviously, the terms *hard-constrained* and *soft-constrained* are used to abbreviate the above explanation but should not be taken in a strict mathematical sense in this chapter.

in Section 8.4.

Three computers were available for conducting the computational tests. An Intel Pentium 4 (3.6 GHz, 2 GB memory, 2 cores, Windows XP), an Intel Xeon (3.4 GHz, 8 GB memory, 8 cores, hyperthreaded, Windows Server 2003) and an AMD Opteron 842 (1.6 GHz, 11.5 GB memory, 4 cores, Windows Server 2003). From the 8 CPUs of the Xeon machine only six were available, the other two were used by other processes. For the central, sequential and asynchronous coordination, the Intel Xeon machine was used. In the parallel coordination, all computers are connected to a grid. The global distributor and all planning domain processes run on the Pentium 4, sharing the two available CPUs (these processes do not require much computational power and the two CPUs were not a bottleneck). The solving units are placed on the two other machines. First, the 6 available cores of the Xeon machine are distributed evenly between the planning domains, each planning domain getting approximately the same number of Xeon CPUs for its solving units. In a second round, the 4 cores of the Opteron machine are distributed. Hence, planning domains with more solving units will get more processes of the Opteron machine (which is slower than the Xeon machine). For each solving unit, one CPU is available.[9]

8.2.4 Test results

We start by presenting some overall results. A detailed analysis is presented subsequently.

8.2.4.1 Overall results

Summarizing the main results, three fundamental observations can be made. First, the coordination mechanism was able to significantly improve on the upstream planning solution.[10] Second, the parallel coordination showed a better performance than the asynchronous and the sequential coordination. Third and most surprisingly, the coordinated results were, in many cases, an improvement on the centrally computed solutions. It seems that the DEAL framework as a top-level metaheuristic was able to introduce new information that steered the overall search process in the right manner. Adjusting the due dates of the suppliers' RCPSP instances by an evolutionary process and repeatedly calling the DS solving modules finally led to the good results. For some instances, the upstream planning solution already achieved an improvement on the centrally computed solutions. By closely examining the schedules, we were able to identify two reasons for this astonishing result. On one hand, the initialization routines of the centrally applied DS solving module could not deduce good activity sequences at the beginning in some cases. On the other hand, precedence constrains between the activities were not "aligned." That is, for different resources, activity sequences were often contradicting, leading to idle times.

One might argue that the results presented in the following do not prove the effectiveness of DEAL but only the ineffectiveness of the DS solving module. This is not true. First of all, it has to be mentioned that, for NP-hard planning problems, an optimal solution can usually not be

[9] The scheduling of computational tasks is done by the operating system, however.

[10] As discussed in Section 7.3, upstream planning is regarded as the standard initialization method of the DEAL framework. The upstream planning solution is actually the first outcome of a coordination process. We also speak of upstream planning if the OEM's first schedule is additionally constrained by a simulated frame contract, centrally computed release dates. To calculate the upstream planning solution in the standard test program, the same time bounds as for calculating regular proposals have been applied to the DS solving module. Naturally, alternative time bounds are also conceivable. A brief computational evaluation regarding this particular issue is presented later in the detailed analysis.

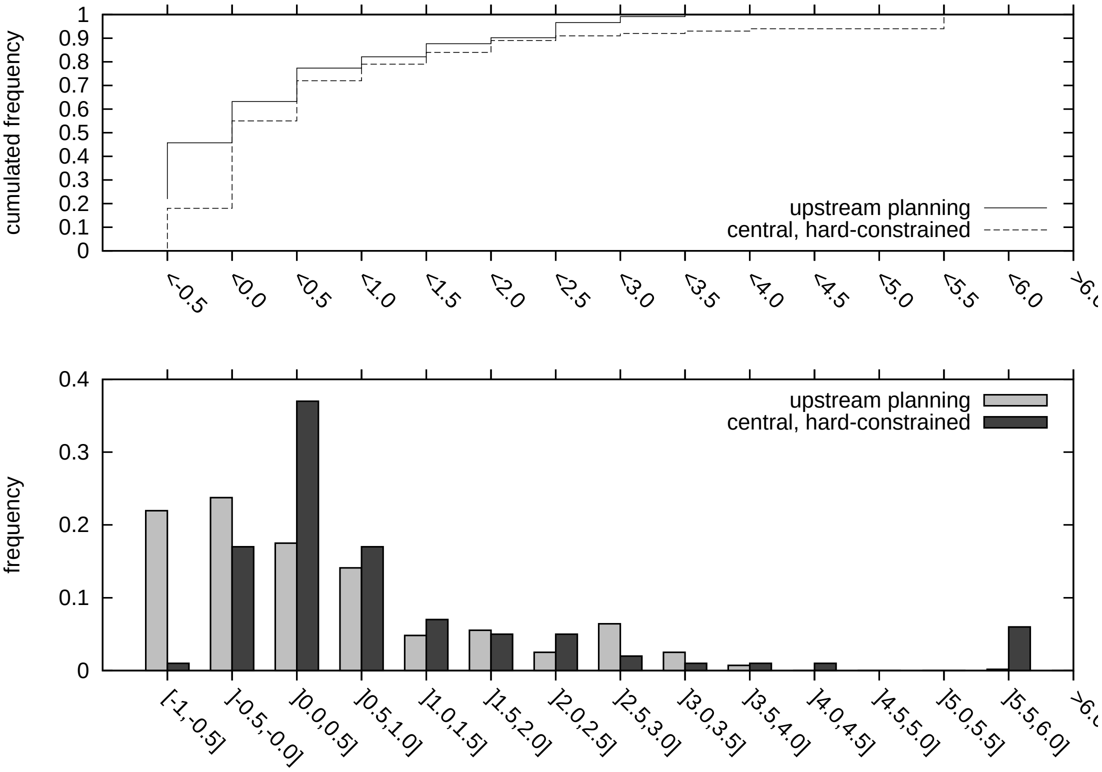


Figure 8.16: Relative difference of upstream planning and central, hard-constrained solutions from central, soft-constrained solutions.

computed in reasonable time.[11] Moreover, one has to keep in mind that the DS Optimizer was constructed to tackle RCPSPs of *general type*. For evaluating DEAL, we only consider a subset of all possible scheduling problems: problems, that exhibit regions (planning domains) that are only interdependent in a noncyclic manner. DEAL can be regarded as a specialized heuristic working on such problem structures. Hence, by decomposing the central scheduling problem, superior knowledge over the planning domains becomes available. According to the *No Free Lunch Theorem* of computer science, we have to *expect* that the results of a specialized heuristic are better than those of a general one.

In the following, we will present some condensed figures that underpin the above claims. It should be noted that these figures only represent the standard test program (not including the evaluation of different operators or different objective weights). Figure 8.16 shows a histogram and cumulated frequencies of the relative differences between the upstream planning and the central soft-constrained solution. That is, for each upstream solution u belonging to test instance H, the relative difference

$$\frac{wml \cdot ml_u + wtl \cdot tl_u - \sum_{c \in H} \frac{(wml \cdot ml_c + wtl \cdot tl_c)}{|H|}}{\sum_{c \in H} \frac{(wml \cdot ml_c + wtl \cdot tl_c)}{|H|}}$$

was measured, where c denotes a centrally computed result.[12] Additionally, the relative differ-

[11] This also holds for other coordination mechanisms working on such complex planning problems, e.g. the approach of Dudek (2007).

[12] It is questionable if an upstream solution can actually be related to the central solution of the same random seed.

ences of each hard-constrained central run to average soft-constrained central results were computed analogously and are also depicted in the figure. It can be seen that about 45% of the upstream planning solutions are already better than the average central soft-constrained result (the relative difference is smaller than zero). Obviously, not even being a lower bound for the upstream planning solutions, the central runs can actually not be taken as a true benchmark for the coordination mechanism. Nevertheless, we use the centrally computed results as another "point of reference." For some test instances, the weighted lateness caused by upstream planning is about five times larger than the average weighted lateness of the related central soft-constrained results.

The figure also shows that hard-constrained central runs (if existent) are better than the soft-constrained runs in only 20% of the cases. An explanation might be that the prioritization of deadlines accidently supports the DS Solving module in finding good solutions (more details are provided later). However, for the majority of cases, hard-constrained central results are much worse than the average soft-constrained solution of the same instance. Thus, we are on the safer side when taking the soft-constrained results as a point of reference for subsequent comparisons. To what degree can upstream planning solutions be improved by the coordination mechanism? For all runs, we measured the relative difference between the upstream planning solution u and coordination result of the DEAL framework d, the fraction

$$\frac{wml \cdot ml_d + wtl \cdot tl_d - (wml \cdot ml_u + wtl \cdot tl_u)}{wml \cdot ml_u + wtl \cdot tl_u}.$$

As zero-lateness solutions are possible by construction for almost all test instances generated by the DTGD, the best obtainable outcome of coordination is a relative difference of -100%.[13]

A histogram and cumulated frequencies can be found in Figure 8.17. It can be seen that nearly all upstream planning solutions were improved. Obviously, coordinated solutions are—in most cases—significantly better than upstream planning results. A few coordinated results are worse than the upstream solution. Recall, however, that upstream solutions are not necessarily feasible. That is, they may violate deadlines of external customers' orders. In order to achieve feasibility during coordination, the weighted lateness incurred by OEM activities can increase. This has been the case for coordinated results that are "worse" than the upstream planning solution.[14]

Overall, the parallel runs showed better final results than sequential or asynchronous runs. This is not surprising, since a parallel run has most resources at its disposal. Figure 8.18 compares parallel and asynchronous results with the sequentially computed results.[15] It can be seen that

Alternatively, it could also be related to any other central solution belonging to the same test instance. That is why we decided to work with average values of central results. The same strategy is followed for comparing outcomes of coordination runs with central results. For a given random seed, only the final coordinated result can be related to the upstream planning solution because both are outcomes of the very same computational process!

[13] In the approach of Dudek and Stadtler (2005), another performance measure, the so-called **gap closure** is additionally used. The gap closure is defined as $\frac{\text{upstream solution}-\text{coordinated solution}}{\text{upstream solution}-\text{central solution}}$. For our case this measure is too condensed, as both numerator and denominator can become negative. However, for most of the deterministically generated test instances the optimal solution is known to be zero (by construction). Substituting the central result by the optimal solution would lead to $\frac{\text{upstream solution}-\text{coordinated solution}}{\text{upstream solution}}$ for these instances. In fact, this is the negative value of our relative difference. For the sake of congruency to other measures we employed the negative relative difference instead of additionally using the gap closure for those instances, where the optimal solution is known to be zero.

[14] The violation of deadlines is not explicitly represented in the figure because it occurs so rarely.

[15] For each parallel run r, the relative difference from the average sequential results s of the same test instance H is considered, i.e. the number

$$\frac{wml \cdot ml_p + wtl \cdot tl_p - \sum_{s \in H} \frac{(wml \cdot ml_s + wtl \cdot tl_s)}{|H|}}{\sum_{s \in H} \frac{(wml \cdot ml_s + wtl \cdot tl_s)}{|H|}}$$

is measured. If all sequential runs of an instance achieved a lateness of zero, the instance was not considered. In such

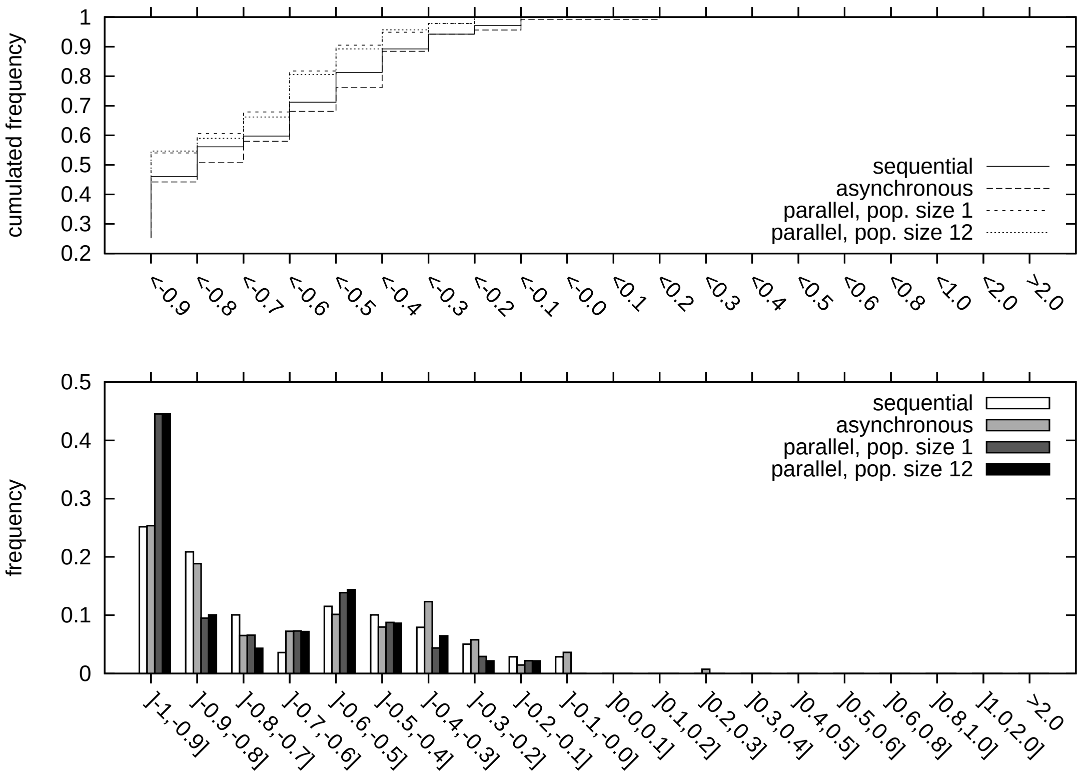


Figure 8.17: Relative difference of coordinated to upstream planning solutions.

the asynchronous coordination achieved a better result than the sequential coordination in 75% of the cases. For both parallel runs, this percentage rises to 97%. Moreover, it can be observed that the parallel corodination with a population size of 1 is—for some runs—slightly better than the parallel coordination with a population size of 12. This can be explained by the higher selection pressure: the lower the number of individuals in a population, the faster the convergence (although the risk of getting stuck in a local optimum increases). More information on this topic is provided in the subsequent detailed analysis.

Finally, Figure 8.19 compares central soft-constrained and coordinated runs. Again, the average weighted lateness of central runs of a test instance was considered for computing the relative difference.[16] It can be seen that the coordinated results are better than the average central, soft-constrained solution in 90% of the cases.

In above we compared mean values. It is worth investigating whether similar observations can be made for the deviation of values. For each test instance we calculated the variation coefficient[17] for parallel, asynchronous and sequential coordination and compared it with the coefficient of

cases, the parallel and asynchronous runs also achieved zero lateness.

[16] That is, the number

$$\frac{wml \cdot ml_d + wtl \cdot tl_d - \sum_{c \in H} \frac{(wml \cdot ml_c + wtl \cdot tl_c)}{|H|}}{\sum_{c \in H} \frac{(wml \cdot ml_c + wtl \cdot tl_c)}{|H|}}$$

is computed for each coordination run d and all central runs c belonging to the same test instance H. If all central runs of an instance achieved a lateness of zero, the instance was not considered. In such cases, the coordination runs also achieved zero lateness.

[17] The coefficient of variation is the normalized standard deviation, i.e. the standard deviation divided by the mean of a sample. For a mean of zero we set the coefficient of variation to zero as well. As mean values between coordination and central runs deviate heavily it seems more appropriate not to consider the standard deviation but its normalized version.

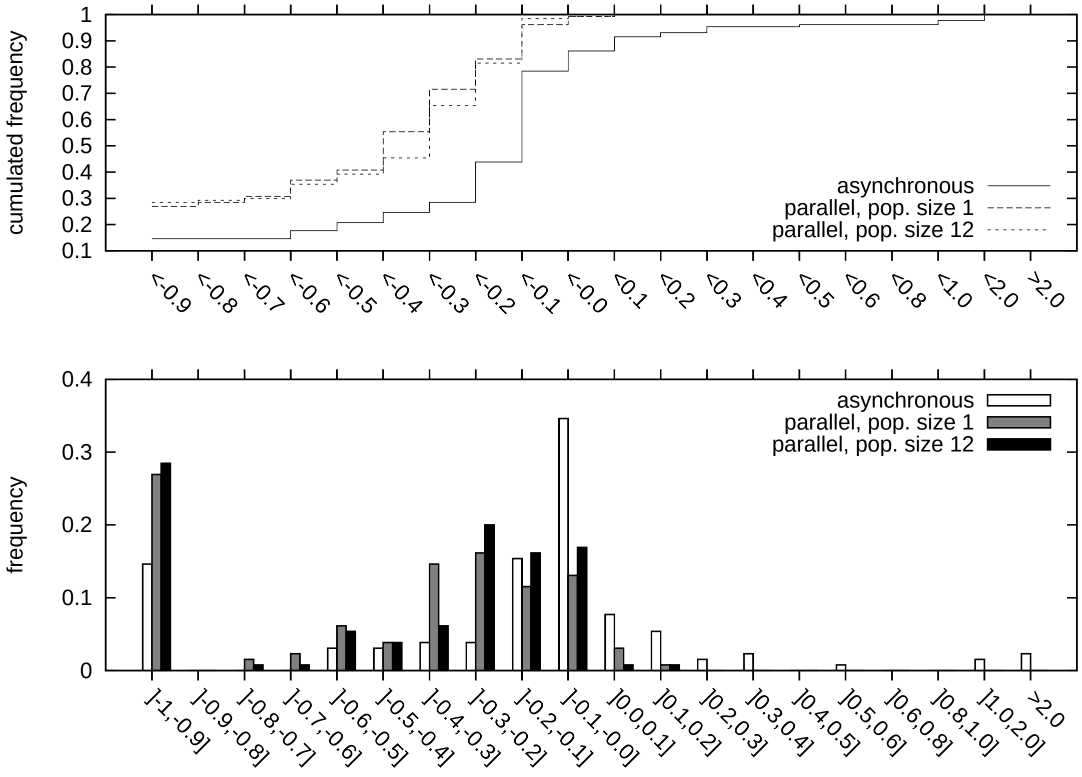


Figure 8.18: Relative difference of parallel and asynchronously coordinated to sequentially coordinated solutions.

variation of the central, soft-constrained solution. In average, the coefficient of variations of the coordination runs are higher than those of the central solutions. However, no clear pattern could be observed and the situation differs from test instance to test instance. Averaged over all test instances, the coefficient of variation is 0.3 for the central runs, 0.52 for the sequential runs, 0.55 for the asynchronous runs, 0.36 for the parallel runs with population size 1 and 0.63 for the parallel runs with population size 12.

Another interesting question is how runtime is distributed in the parallel coordination. Table 8.8 shows the portion of time a process was idle, communicating (i.e., sending and receiving messages) and computing (i.e., constructing and evaluating messages, updating the population and so forth). The averages of parallel runs with populations size of 12 over all instances (not including the complete scenario since it defines a different fraction of solving units to planning domains) were taken.

Three central observations can be made. First, there is some minimum idle time of approximately 10% caused by the distribution logic. For example, a planning domain sends a new solving request to a solving unit only after the previous request is finished, after the unit's reply was received. Since both messages are routed over the local and global distributors, some seconds pass between sending a reply and receiveing a new request where the solving unit is idle.[18]

Second, as expected, the solving units do most of the computing, although the OEM spends some time computing since left- and right-alignments are calculated locally (not outsourced to

[18] A speed increase could be possible if we allowed the domains to send solving requests even to busy solving units (the messages would then be buffered by the global distributor).

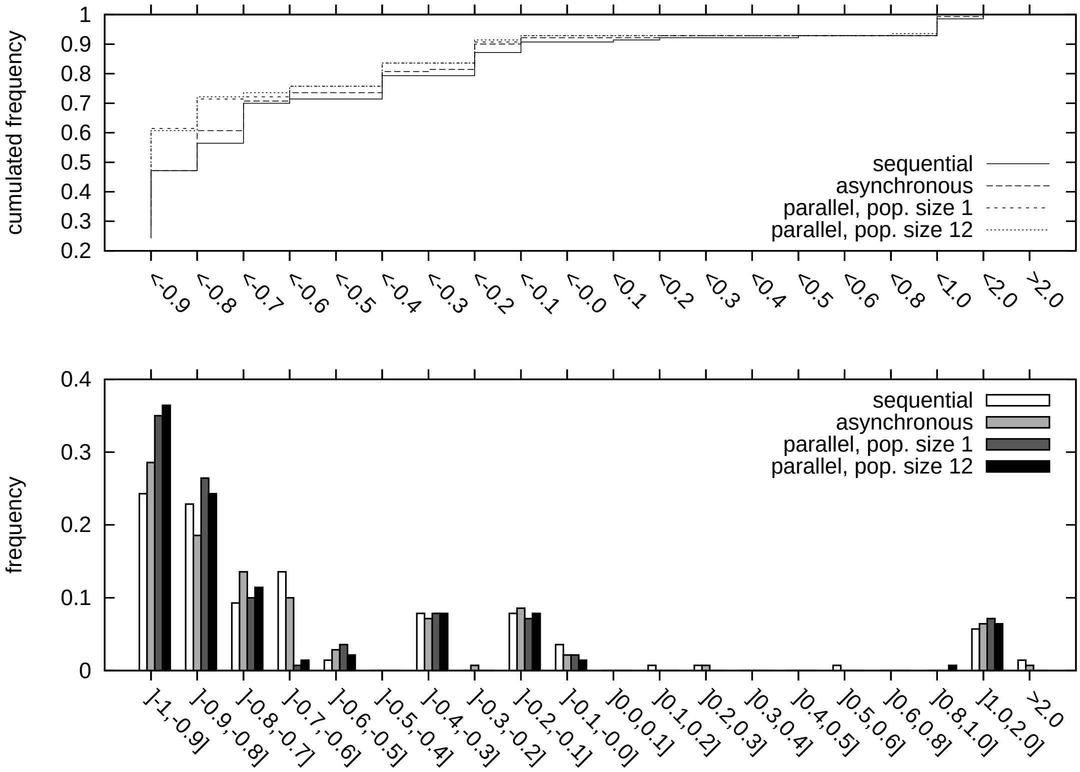


Figure 8.19: Relative difference of coordinated to central, soft-constrained solutions.

		Idle	Communicating	Computing
OEM	Small	37%	51%	12%
	Medium	41%	51%	7%
	Large	40%	50%	12%
OEM Solving Unit	Small	10%	13%	77%
	Medium	15%	17%	66%
	Large	8%	11%	80%
Supplier	Small	47%	50%	0%
	Medium	47%	50%	0%
	Large	47%	49%	2%
Supplier Solving Unit	Small	22%	20%	57%
	Medium	31%	27%	40%
	Large	23%	21%	55%

Table 8.8: Distribution of runtime for the parallel coordination with population size 12, averaged over all instances.

a solving unit). It can be observed that a Supplier's solving unit is on average more idle than an OEM's solving unit. This can be explained by the fixed assignments of units to domains (see Table 8.7 for the detailed specification) before the coordination run. Possibly, runtime could be used more efficiently if the OEM were assigned more solving units than in the used specification. Since the coordination results were already of considerable quality, this aspect was not researched further. In general, a more dynamic allocation of solving units could be a topic for future research.

Third, it can be seen that domains and solving units spend a large amount of their time communicating. Several factors might contribute to this communication overhead: the time for sending a message in the grid framework, the time for serializing and deserializing a message and colluding messages. Collusions occur if the global distributor cannot immediately process a local distributor's request as it is busy handling another local distributor. Recall that the local distributors iteratively poll the global distributor for new messages as discussed in Section 7.10. If the related domain or solving unit is idle, the local distributors poll every second. Many of these polls conflict with the transfer of "real" messages at the global distributor. Thus, some of actual idle time is expressed as communication overhead. However, communication is not a *real* bottleneck. This also explains why solving units have less communication overhead than domains since they are more busy with computational tasks.

Different tasks contribute to the "computing" times in Table 8.8: the construction of proposals, evaluation of messages, duplicate elimination, updating the population and so forth. A closer look reveals that detecting and eliminating redundant individuals or proposals, evaluating and updating the population together required little runtime (below 1 %). In total, redunancy checks filtered out about 20% of all proposals. On average, 13% of all constructed OEM individuals were identified as redundant and eliminated before the related proposals were sent to the suppliers. For all nonredundant individuals, 5% of proposals could be saved by connecting the individuals to preexisting communication threads in the OEM's domain. Additionally, 4% of all proposals received by the suppliers were identified as redundant, the suppliers thus resent an existing counterproposal instead of calculating a new one. As coordinated results are of considerable quality, we believe that our proposed duplicate-elimination strategy does really help in saving runtime.

8.2.4.2 Detailed analysis of the setup instances

We start by looking at the results for the small instance of the setup subscenario. Figure 8.20 shows a convergence plot of 10 central runs. However, not all objectives are considered. As explained in Chapter 7, only the maximum and total lateness are considered as important in the DEAL framework. Thus, only the weighted sum of both values is drawn over the overall runtime of the central optimization. The following observations can be made: First, the outcome of a run is highly dependent on the initial random seed. Second, there exist several discrete plateaus of weighted lateness, whereas a plateau's height is determined by the number of setup activities. In general, Evolutionary Algorithms have difficulty coping with plateaus in the fitness landscape. For the DS solving module, the problem is that many mutation operators lead to similar or worse fitness (as, for example, the setup is just positioned at another slot). Especially towards the end of a run, finding a schedule with lower setup times is achieved by chance and is not the result of evolutionary learning. Third, no run attained the optimum of zero lateness.

The results of sequential coordination with 13s runtime for every local optimization is shown in Figure 8.21. As mentioned in Section 7.10, the planning domains send messages to the controller for debugging the coordination process. The controller stores the time when a DEAL-individual is completed from the OEM's perspective and its fitness as perceived by the different partners. Figure 8.21 shows the weighted lateness from the OEM's perspective, which is, in this instance, equal to the overall lateness, since suppliers don't have external customers. It is important to realize that the figure does not depict the convergence of local optimization, showing only the fitness of the best complete DEAL-individual known so far. As a consequence, the interval between two steps of a graph is at least 39s, the minimum time needed to sequentially construct an individual with three partners and 13s local runtime plus additional overhead time. In longer sections with equal lateness, no improvement could be realized. Moreover, the coordinated runs start later, as

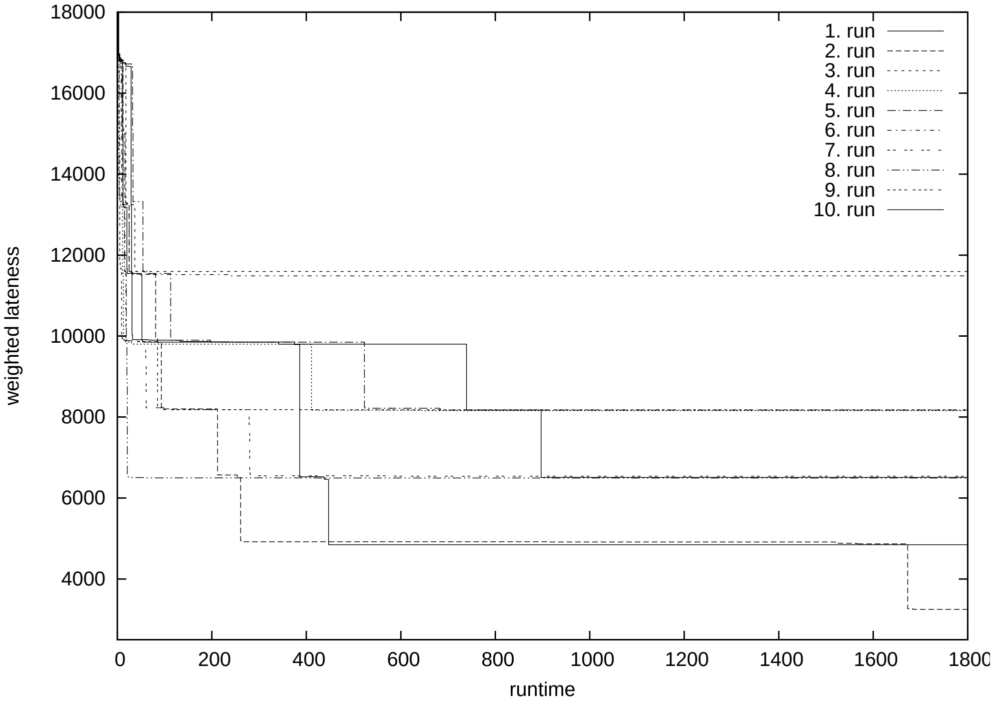


Figure 8.20: Weighted lateness of 10 different central runs for the small instance of the setup subscenario.

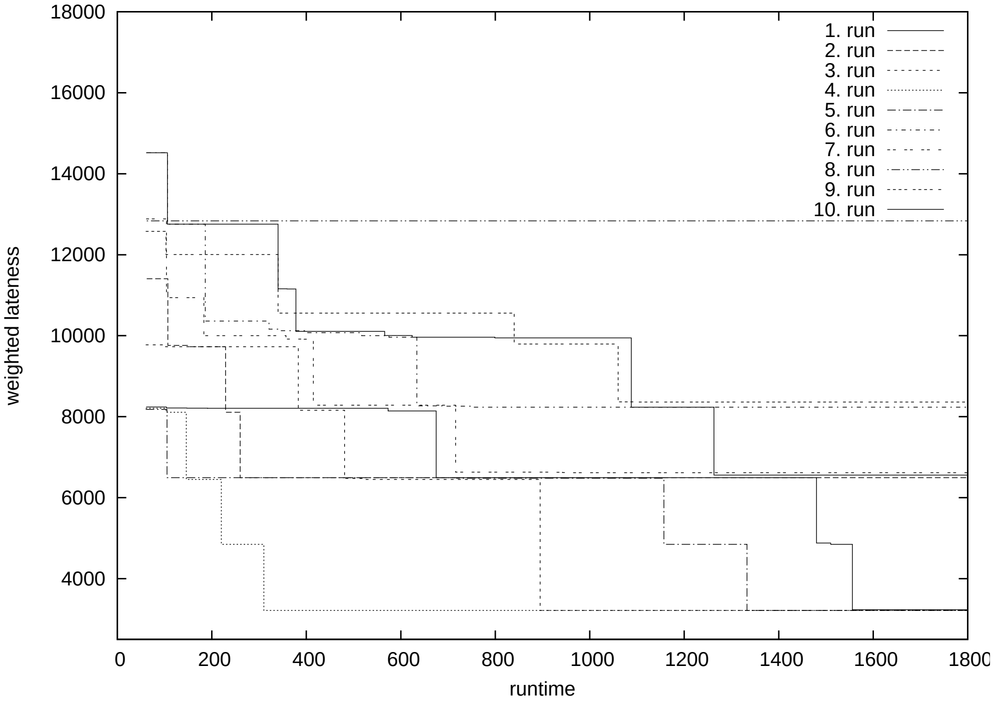


Figure 8.21: Weighted lateness of 10 different runs of sequential coordination, small instance of the setup subscenario, 13s local runtime, OEM perspective.

the initial DEAL-individual needs to be constructed first.

In comparison to Figure 8.20, many of the sequential runs start and end better than the central runs. An upstream planning superior to central planning seems to be irrational at first thought. However, recall that the underlying DS solving module is a metaheuristic that does not guarantee optimal results. Instead, results depend on the realization of random variables and the test instance. If we neglect the random effects, we can still examine in which aspects the test instances used by central optimization and coordination differ.

First, the coordination is based on three test instances of smaller size and one could argue that the size of a test instance might have a disproportionate influence on the GA's solution quality. Second, the three test instances comprise more due dates than the central test instance. Let us focus on the GA's initialization phase, described in Section 6.1.2. Recall that activity lists of GA-individuals are initially sorted according to propagated due dates, earliest start dates, slack or purely randomly. After the sorting, rudimentary campaigns are built. Finally, the lists are repaired by either pulling too late activities forward or by pushing too early activities backward. For the given instance, due dates, start dates and the slack are equal for all activities located on the same production stage. Hence, during initialization, there is no information available for sorting the activities deterministically; lists are only distorted randomly before building the campaigns. The problem is now that the (random) sequence on one production stage does not necessarily match the sequence on another production stage. For example, two activities a and b on resource "chassis moulding" are positioned a before b, but their successors a' and b' on resource "motor–chassis assembly" are sorted the other way round b' before a'. This has two implications: On one hand, different campaigns might be built for different production stages. On the other hand, preceding activities are not aligned; if drawn on a Gantt chart (see Figure 8.22), precedence constraints overlap.

What is happening in the DEAL framework during upstream initialization? For the OEM, the initialization process is similar to central planning, activity lists are a random output. As explained in Section 7.3, the OEM proposes the start dates of the right-aligned initialized schedule to his suppliers. Now for a supplier, the proposed due dates for downstream-related activities *do differ* and a deterministic sorting according to due dates is possible. In turn, the suppliers' counterproposals allow an initial sorting according to the release dates in the OEM's domain. Consequently, activities between the domains are better aligned than for the central run. This behavior is schematically illustrated in Figure 8.23. Apparently, the suppliers' initialization routines are also enabled to build better campaigns right from the beginning.

In general, overlapping precedence constraints seem to pose a problem for mutation operators, since inserting and swapping activities can have no effect. For example, in Figure 8.22, inserting B (and the predecessors of B) right after A (and the predecessors of A) does not change the makespan, whereas it does in Figure 8.23. Of course, these overlapping precedence constraints occur in both runs, but they occur in the coordination run with lower frequency.[19]

When using the DEAL framework, the alignment between suppliers' downstream and OEM's upstream activities is maintained during the whole coordination. In other words, the DEAL framework introduces additional information (due dates for suppliers) that "helps" the local optimization methods to compute better results.

[19] Setting up a universal measure in this regard is difficult, since the multimode RCPSP sometimes *requires* overlapping precedence constraints in order to efficiently use resource slots with sufficient capacity. However, we were able to verify our claim manually by closely examining the related Gantt charts. A detailed example is provided in the course of this chapter.

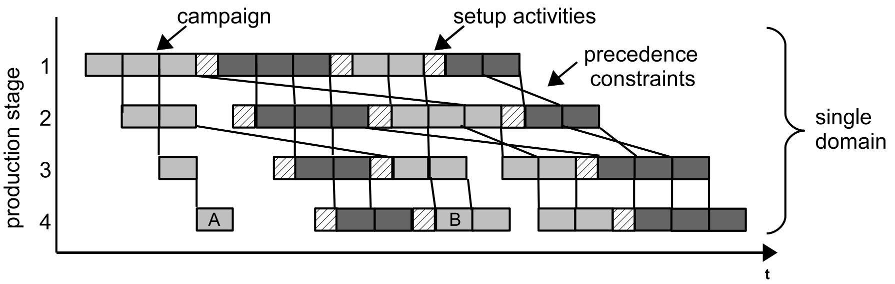


Figure 8.22: Illustration of overlapping precedence constraints in a central test instance.

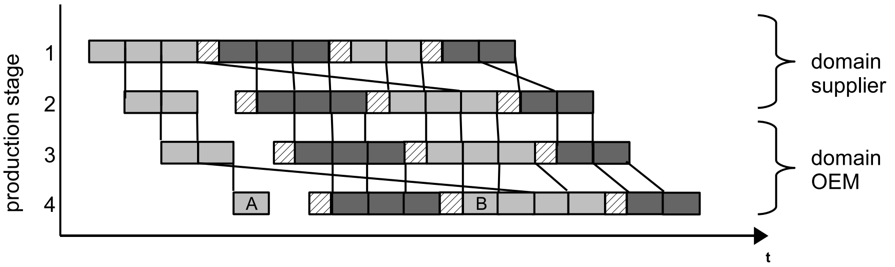


Figure 8.23: Illustration of overlapping precedence constraints in two interdependent test instances during coordination.

It should also be emphasized that, instead of a fixed due-date setting as in the central run, the coordination mechanism tries several, slightly modified, due-date combinations when calling the local metaheuristics. Good due date combinations are favored in the evolutionary process of the DEAL framework. The effects can outweigh the drawbacks of separated problems and might—depending on the test instance—eventually lead to better results.

Instead of comparing single runs, the minimum, the average and the maximum of 10 runs are considered henceforth. That is, for every time point the minimum, the average or the maximum of all 10 runs to that time is taken. It is important to note that the minimum value over time does not relate to a single run, but must rather be seen as the "best composition" of several runs. We will also consider populations that comprise several individuals, but only the best individual is considered for the figures. In other words, the average value of 10 runs is the average of the best found individual of every run at a given point in time.

First of all, a good adjustment of the weight for the setup time was sought (the other weights are assumed to be fixed by the planner). Figure 8.24 compares the weighted lateness for different setup weights. It turned out that a setup weight one *10th* of the total lateness weight yields the best results regarding the weighted lateness objective.

A similar comparison was carried out for the sequential runs, see Figure 8.25. Again, a setup weight one 10th of the total lateness weight yields the best results. However, we can see that the deviation between the sequential runs is smaller than for the central runs. In other words: the objective weights of the underlying DS solving module clearly have an effect on the performance of the coordination mechanism. However, as the DEAL framework imposes an additional ranking, the effect is less significant than for the central runs. A very high setup weight was also tested, triggering semi-active planning. The related results are the worst for both central planning and coordination mechanism. The results are mostly similar if not only the weighted sum of total and maximum lateness but of all objectives is considered. As an exception, upstream planning using semi-active scheduling yields the best results. The related figures F.1 and F.2 can be found in

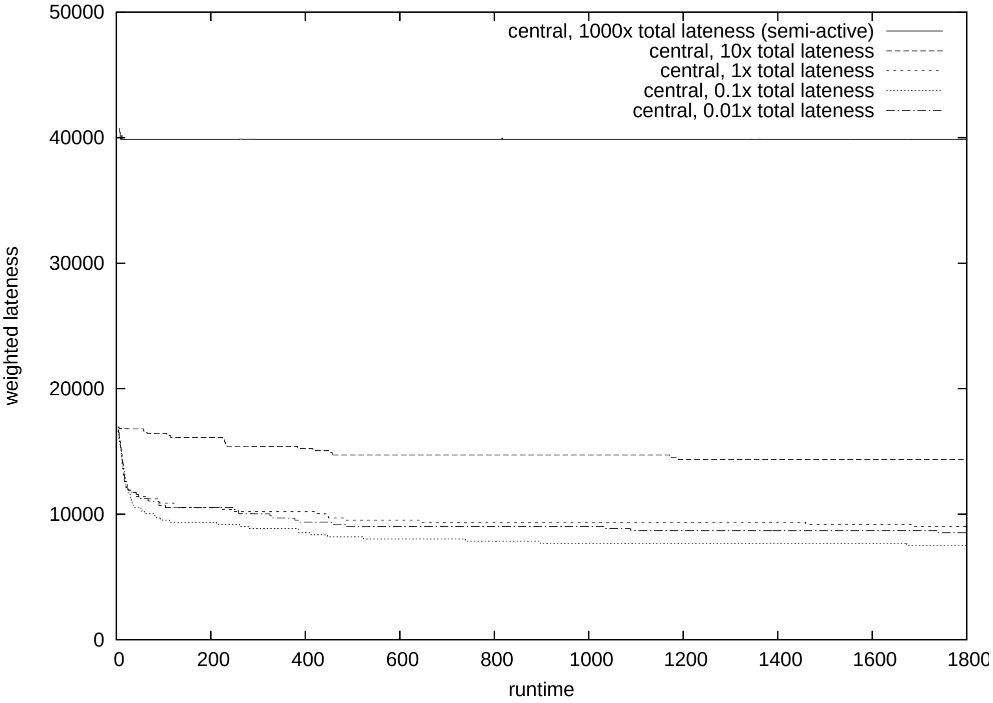


Figure 8.24: Average weighted lateness of 10 central runs with different setup weights.

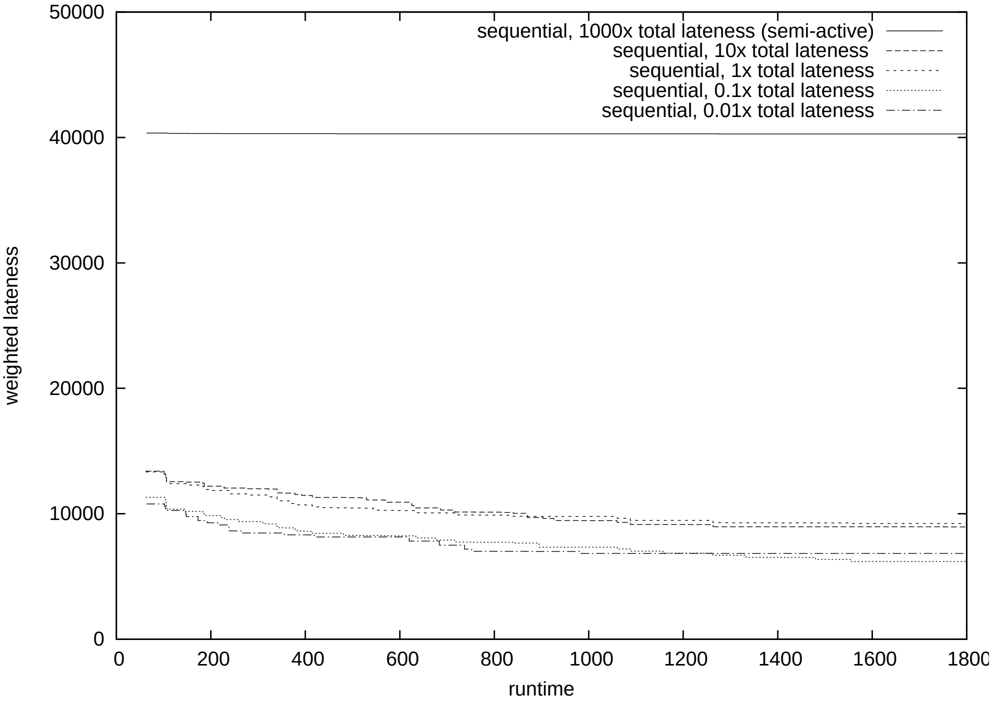


Figure 8.25: Average weighted lateness of 10 sequential runs with different setup weights, 13s local runtime, deterministic self-adaption.

Appendix F.

With the best found weight distribution (the one mentioned in Section 8.2.3), parallel, sequential and asynchronous coordinations are compared with central runs using convergence graphs in Figure 8.26. The central runs converge quickly on average to a weighted lateness of 7,500, apparently marking a local optimum that cannot be further improved (or only very slowly). As can be seen in the figure, sequential, asynchronous and parallel coordinations start with almost similar average initial lateness. On average, the sequential runs outperform central planning but also get stuck at a weighted lateness of 6,500. Recall, that the sequential coordination is actually a hill climbing procedure. It tries to improve a single (DEAL) individual by sequentially applying the different operators. As soon as an improvement is realized, the old individual is discarded.

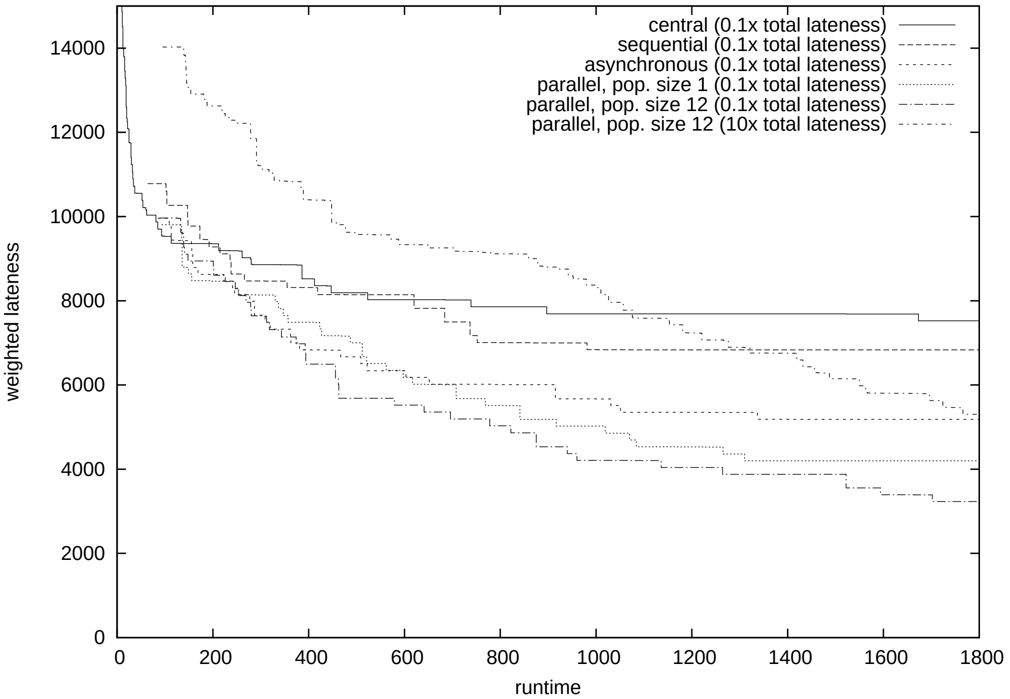


Figure 8.26: Comparison of weighted lateness for parallel, asynchronous and sequential coordination and central planning, small instance of the setup subscenario, average of 10 runs each, deterministic self-adaption, 13s of local runtime.

The asynchronous coordination is apparently not only faster, but even better than the sequential coordination. Besides the higher computational power, this behavior can be explained by the inherent "buffering" of individuals. In the standard specification (cf. Table 8.7), the asynchronous coordination has a queuing size of 4. That is, as soon as less than 4 individuals are in the mating pool (or uncompleted), a new evaluation (or construction) is started. Hence, even if a better individual has been found, already started construction processes continue or "old information" might be contained in the mating pool. This decreases the chance of getting stuck at a local optimum. The effect gets stronger with an increasing queuing size.

The parallel coordination comes with a queuing size of 12, which leads to even better results even with a population size of one. Larger population sizes allow the storage of even more information. The figure shows that the parallel coordination with a population size of 12 performs best, not showing an end of convergence yet. In addition, the convergence graph for a parallel coordination based on a higher setup weight is plotted. Though the results are worse, the slope of

			Second priority criteria		External lateness		
	Domain	Statistic	Makespan	Setup time	Max	Total	Weighted
	Central	min	89,601	30,400	3,200	8,400	3,251
		avg	**93,761**	**33,920**	**7,360**	**23,830**	**7,523**
		max	97,601	40,000	11,200	51,200	11,596
parallel, pop. size 12	OEM	min	86,401 (−10%)	14,400 (−23%)	0 (−100%)	0 (−100%)	0 (−100%)
		avg	**89,601 (−7%)**	**17,280 (−8%)**	**3,200 (−67%)**	**6,240 (−80%)**	**3,230 (−68%)**
		max	92,801 (−3%)	20,800 (−11%)	6,400 (−34%)	16,000 (−48%)	6,495 (−35%)
	Supplier Steel	min	80,001 (−8%)	6,400 (−53%)	n.a.	n.a.	n.a.
		avg	**82,881 (−5%)**	**9,280 (−33%)**	**n.a.**	**n.a.**	**n.a.**
		max	89,601 (+3%)	16,000 (+16%)	n.a.	n.a.	n.a.
	Supplier Motor	min	80,001 (−6%)	6,400 (−46%)	n.a.	n.a.	n.a.
		avg	**83,841 (−2%)**	**10,240 (−14%)**	**n.a.**	**n.a.**	**n.a.**
		max	89,601 (+5%)	16,000 (+35%)	n.a.	n.a.	n.a.
parallel, pop. size 1	OEM	min	88,001 (−8%)	14,400 (−25%)	1,600 (−83%)	3,200 (−89%)	1,615 (−84%)
		avg	**90,561 (−6%)**	**17,440 (−8%)**	**4,160 (−57%)**	**8,000 (−74%)**	**4,168 (−57%)**
		max	92,801 (−3%)	20,800 (+10%)	16,000 (−33%)	6,400 (−47%)	6,495 (−34%)
	Supplier Steel	min	80,001 (−8%)	6,400 (−53%)	n.a.	n.a.	n.a.
		avg	**83,841 (−4%)**	**10,240 (−26%)**	**n.a.**	**n.a.**	**n.a.**
		max	89,601 (+3%)	16,000 (+16%)	n.a.	n.a.	n.a.
	Supplier Motor	min	80,001 (−6%)	6,400 (−46%)	n.a.	n.a.	n.a.
		avg	**83,841 (−2%)**	**10,240 (-14%**	**n.a.**	**n.a.**	**n.a.**
		max	89,601 (+5%)	16,000 (+35%)	n.a.	n.a.	n.a.
asynchronous	OEM	min	89,601 (−7%)	16,000 (−15%)	3,200 (−67%)	4,800 (−84%)	3,216 (−68%)
		avg	**91,521 (−5%)**	**19,520 (+3%)**	**5,120 (−48%)**	**11,520 (−62%)**	**5,183 (−48%)**
		max	92,801 (−3%)	20,800 (+10%)	6,400 (−34%)	16,000 (−47%)	6,495 (−35%)
	Supplier Steel	min	80,001 (−8%)	6,400 (−53%)	n.a.	n.a.	n.a.
		avg	**85,761 (−2%))**	**12,160 (−12%)**	**n.a.**	**n.a.**	**n.a.**
		max	89,601 (+3%)	16,000 (+16%)	n.a.	n.a.	n.a.
	Supplier Motor	min	80,001 (−6%)	6,400 (−46%)	n.a.	n.a.	n.a.
		avg	**83,201 (-3%**	**9,600 (−19%)**	**n.a.**	**n.a.**	**n.a.**
		max	89,601 (+5%)	16,000 (+35%)	n.a.	n.a.	n.a.
sequential	OEM	min	92,801 (−6%)	8,000 (−58%)	4,800 (−55%)	9,600 (−71%)	4,848 (−55%)
		avg	**93,121 (−4%)**	**18,400 (−4%)**	**6,720 (−37%)**	**18,080 (−45%)**	**6,382 (−37%)**
		max	97,601 (+1%)	22,400 (+17%)	11,200 (+6%)	44,800 (+36%)	11,532 (+7%)
	Supplier Steel	min	80,001 (−9%)	6,400 (−56%)	n.a.	n.a.	n.a.
		avg	**86,401 (−2%)**	**12,800 (−11%)**	**n.a.**	**n.a.**	**n.a.**
		max	92,801 (+5%)	19,200 (+33%)	n.a.	n.a.	n.a.
	Supplier Motor	min	80,001 (−7%)	6,400 (−49%)	n.a.	n.a.	n.a.
		avg	**84,801 (−1%)**	**11,200 (−10%)**	**n.a.**	**n.a.**	**n.a.**
		max	92,801 (+8%)	19,200 (+54%)	n.a.	n.a.	n.a.

Table 8.9: Comparison of average (avg), minimum (min) and maximum (max) values of 10 runs of parallel and sequential coordination with 10 runs of central optimization after 1,800s, local runtime 13s, small instance of the setup subscenario. The numbers in parantheses denote the relative difference to the average upstream planning solution

the curve is similar to that of the coordination with adjusted setup weight. The parallel coordination with bad setup weights beats the sequential coordination with best-adapted setup weights. Apparently, the parallel coordination is able to counteract poorly adjusted parameters of the base optimization engines. A comprehensive overview of the results is given in Table 8.9, including minimum and maximum values and other objectives[20] of OEM and suppliers.

The table shows several interesting details. First, the parallel coordination with a population size of 12 was able to reduce the lateness of external customers' orders completely to zero. Second, the random seed has a relatively large impact on the result of the scheme. The worst solution of the parallel coordination with population size 12 is not better than the best solution of the sequential coordination. Moreover, depending on the initial seed, the suppliers and the OEM experience increases or decreases in setup time and makespan. Two exemplary Gantt charts for the central and coordinated final results (parallel coordination, population size 12) are shown in Figure 8.27.

[20] Note that the makespan is not additive. That is, $ms_{OEM} + ms_{SupplierMotor} + ms_{SupplierSteel} \neq ms_{Central}$!

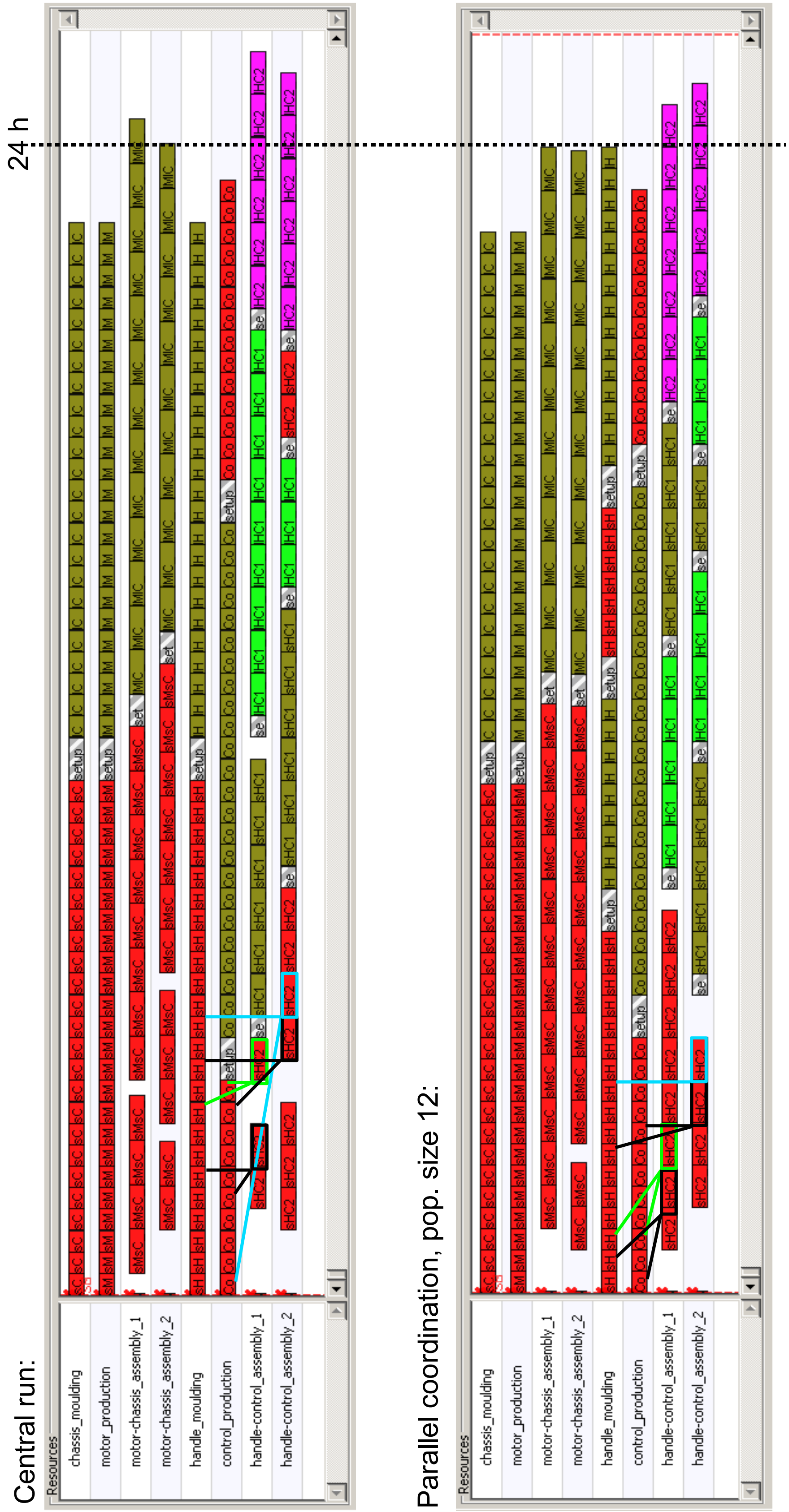


Figure 8.27: Gantt charts of coordinated and centrally computed schedule for the small instance of the setup subscenario.

Specification	Rel. diff. from upstream planning	Rel. diff. from central solution	No. of called operators (OEM only)
Parallel, pop. size 12	−68%	−57%	261
Parallel, pop. size 1	−57%	−44%	324
Asynchronous	−48%	−31%	78
Sequential	−31%	−15%	45

Table 8.10: Summary of average values for 10 runs for the small instance of the setup subscenario after 1,800s, 13s local runtime.

Production activities of the same type are drawn in similar color.[21] Both charts exhibit a more or less similar amount of setup activities (the parallel coordination has two more setups on the resource "control production"). We observe that the centrally computed schedule is delayed by about one lHC2 activity. The cause for this delay lies in the larger idle times between the sHC2 activities at the beginning of the schedule. How do these idle times arise? At this location in the schedule, there is neither a setup required, nor is the number of already scheduled predecessors too small! However, the problem is that the sequence of predecessors is not in line with the sequence of successors (as already discussed previously on page 194). The resulting overlapping precedence constraints are drawn for 4 activities in each chart. Though parallel coordination also exhibits such problems, they apparently occur less than in the centrally computed result.

Table 8.10 gives a brief summary of the average results, including the relative difference from upstream and central planning and the number of tried operators from the OEM's perspective. We see that upstream planning results were improved by at least 31% and central planning results by 15%. Regarding the number of constructed individuals, the following observations can be made. In the asynchronous coordination, approximately twice as many individuals were constructed as in the sequential coordination. However, as the test instance involves three partners, the asynchronous coordination has the potential of evaluating three times as many individuals. This potential could not be realized because of the overhead of interprocess communication and unavoidable idle times.[22] Being able to find the "optimum" of zero lateness in some runs (which leads to a termination of the coordination mechanism), the parallel coordinaton with population size of 12 requires on average fewer operator calls than the parallel coordination with a population size of 1.

Another parameter the user has to specify is the amount of local runtime, the time bound for calling the DS Optimizer's GA. On one hand, more runtime granted to the GA generates the better local results. On the other hand, if local results take more computation time, fewer DEAL-individuals can be constructed. Thus, the local runtime defines the trade-off between constructed DEAL-individuals in total and GA-individuals per local run. If we neglect the overhead of message exchange and processes not related to constructing DEAL-individuals, the *total* number of GA-individuals is obviously independent of the local runtime. In order to exchange more proposals, we need to restrict the local runtime to be short. As the RCPSP is a strong combinatorial problem, it seems promising to exchange many proposals. Especially if our proposed strategy of

[21] The following colors have been chosen:

- sChass, sMotor, sMsC, sHandle, Control1 and sHC1 are red,
- lChass, lMotor, lMlC, lHandle, Control2 and sHC2 are olive,
- lHC1 is green, and
- lHC2 is pink.

[22] Recall that rescheduling-based operators additionally call the DS solving module. During this time, suppliers are idle.

carrying over GA solutions from previous DEAL-individuals works successfully, the overall solution quality should increase with shorter local runtime. Figure 8.28 shows the average values for four different local runtimes: 7s, 13s, 25s, and 50s.

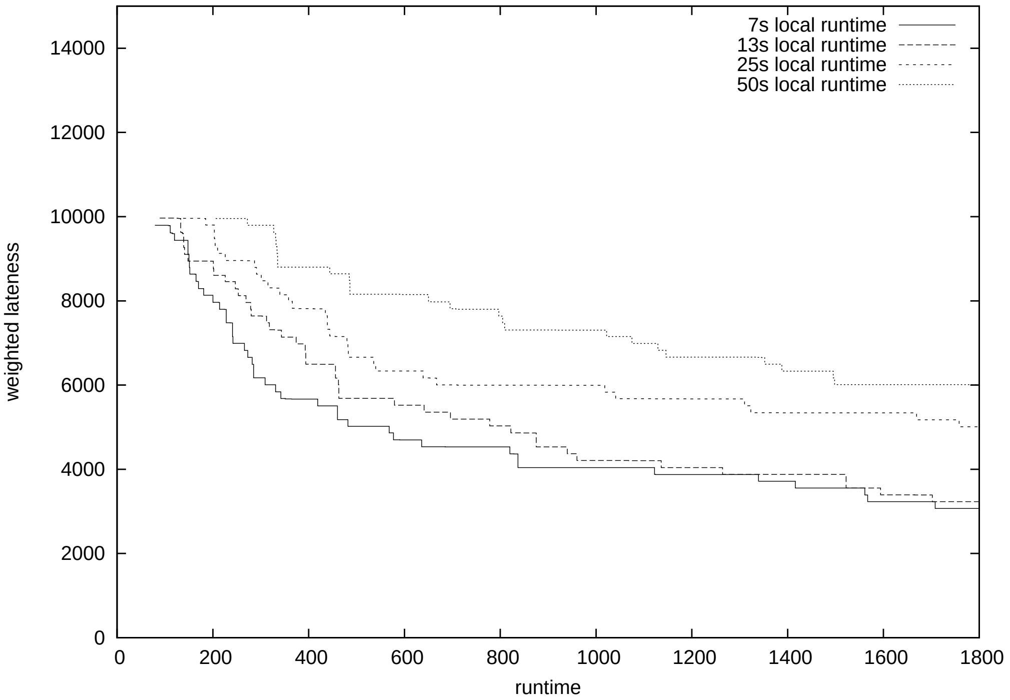


Figure 8.28: Comparison of weighted lateness for different local runtimes for the parallel coordination with population size 12, small instance of setup subscenario, deterministic self-adaption.

The 7s and 13s runs lie very close to each other and apparently define a good trade-off between local optimization and information exchange. The overall coordination process does not profit from longer local runtimes. It can be seen that runs with 25s and 50s take twice and four times the time to reach the values of the 13s runs. It is remarkable that the central runs (see Figure 8.26) take 50 times longer, around 700s, to converge to a local optimum. The graphs exhibit a classical convergence behavior: a fast decrease of lateness in the beginning that gets slower and slower towards the end. However, in contrast to classical EAs, we have a dynamic adjustment of local runtime as a further screw. It might turn out advantageous to employ the following strategy: use small local runtimes in the beginning to quickly realize a reduction of lateness and increase them during the course of coordination. For example, we could think of a self-adaption of runtime: if more than $x\%$ of the last proposals were an improvement, reduce local runtime. Otherwise, increase it. However, this issue is a topic for future research.

As discussed in Section 7.7, three main classes of operators have been developed. Propagating operators apply backward passes for relating lateness to upstream-related release dates. Alignment-based operators first relax some of the upstream-related release dates and then apply a left-right-alignment (LA | RA) to derive new proposed dates. Rescheduling operators employ a reoptimization (FWD) within certain bounds derived from the current schedule.[23] Figure 8.29 shows the mean values for coordination runs based only on a subset of operators. Propagating

[23] For the sake of simplicity, crossover operators involving a rescheduling are considered as rescheduling operators as well. The remaining crossover operators are considered to belong to the class of propagating operators.

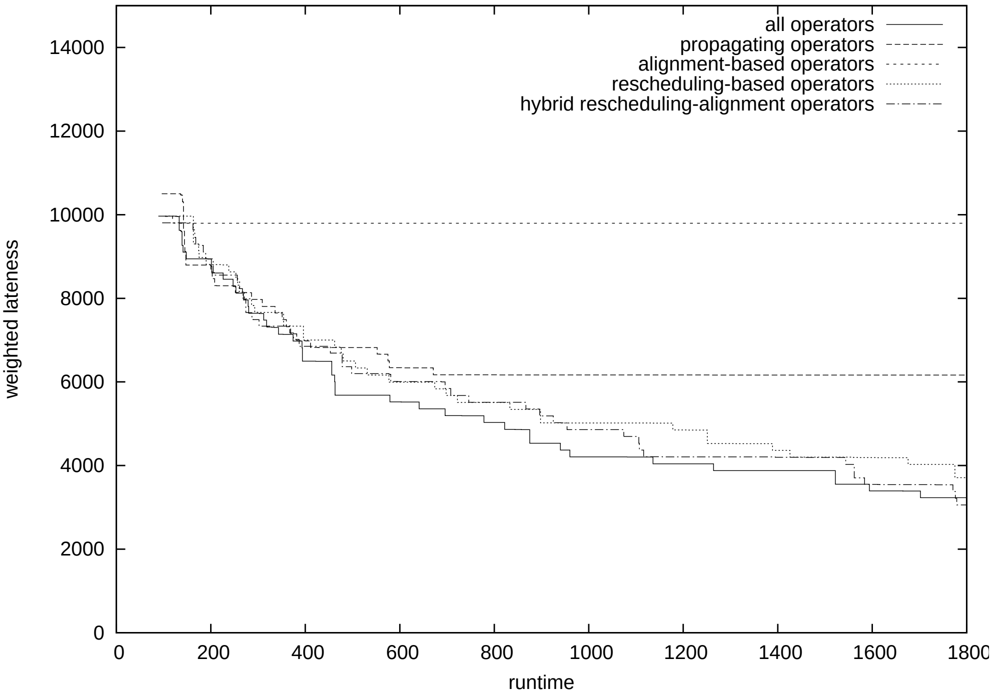


Figure 8.29: Comparison of weighted lateness for different operators for the parallel coordination with population size 12, small instance of the setup subscenario, deterministic self-adaption.

operators lead to some improvement at the beginning, but get stuck soon at a weighted lateness of 6,000. This is in line with our discussion in Section 7.7, where it was argued that propagating operators provide a fast but crude alignment of schedules. Rescheduling operators show a good overall performance, from the beginning until the end of a coordination run. Surprisingly, alignment-based operators get stuck very early. A closer look at the detailed information each run provides led to the following insight. As the current implementation of the left-alignment heuristic does not allow a temporary degradation of the objective value, left shifts of activities were not possible in many cases as they implied an increase of setup times. This led ultimately to proposed dates that did not differ substantially from the release dates of the parent DEAL-individual. Such a behavior was unique to the setup subscenario and was not be observed in any other subscenario.

Interestingly, this poor performance of alignment-based operators can be compensated by the other operators. This gives rise to the supposition that not all of our operators might be required to achieve good results. We will explore means to reduce the operator set in Section 8.4. Figure 8.29 also includes a graph for "all operators" (including propagating, alignment-based and rescheduling-based operators)—these runs clearly outperform each of the above subsets of operators. To counteract the shortcomings of the alignment-based operators, a fourth set of operators was tested. It is denoted as hybrid rescheduling–alignment operators. These operators equal the alignment-based operators, whereas the left–alignment (LA) is replaced by a temporal reoptimization (FWD, see Section 7.7 for more details on this topic). As can be seen in Figure 8.29 the hybrid operators showed a considerable good performance. For all operator subsets, the deterministic self-adaption was used. The related operator lists are derived from Table E.1 with all

Specification	Rel. diff. from upstream planning	Rel. diff. from central Solution	No. of called operators (only OEM)
Parallel, pop. size 12	−100%	−100%	46
Parallel, pop. size 1	−100%	−100%	43
Asynchronous	−89%	−94%	36
Sequential	−97%	−98%	29

Table 8.11: Summary of average values for 10 runs for the medium setup subscenario after 3,600s, 25s local runtime, deterministic self-adaption.

unavailable operators removed from the table.[24] In the remainder of this section, the standard setting of propagation, alignment-based and rescheduling-based operators was kept for the medium and large variants, however.

For the medium setup instance, the convergence graphs are shown in Figure 8.30. Both parallel runs are able to produce schedules without any late activity even before the overall coordination time of 3,600s was reached. A population size of 1 is slightly faster, which can be explained by the higher selection pressure (see Section 4.2).[25] The sequential and asynchronous coordination are on average able to almost reach zero lateness after 3,600s. The centrally applied GA performs significantly worse. Even an upstream solution is on average better than the central solution after one hour. Table 8.11 gives a brief overview, a detailed summary of results can found in Appendix G (Table G.5).

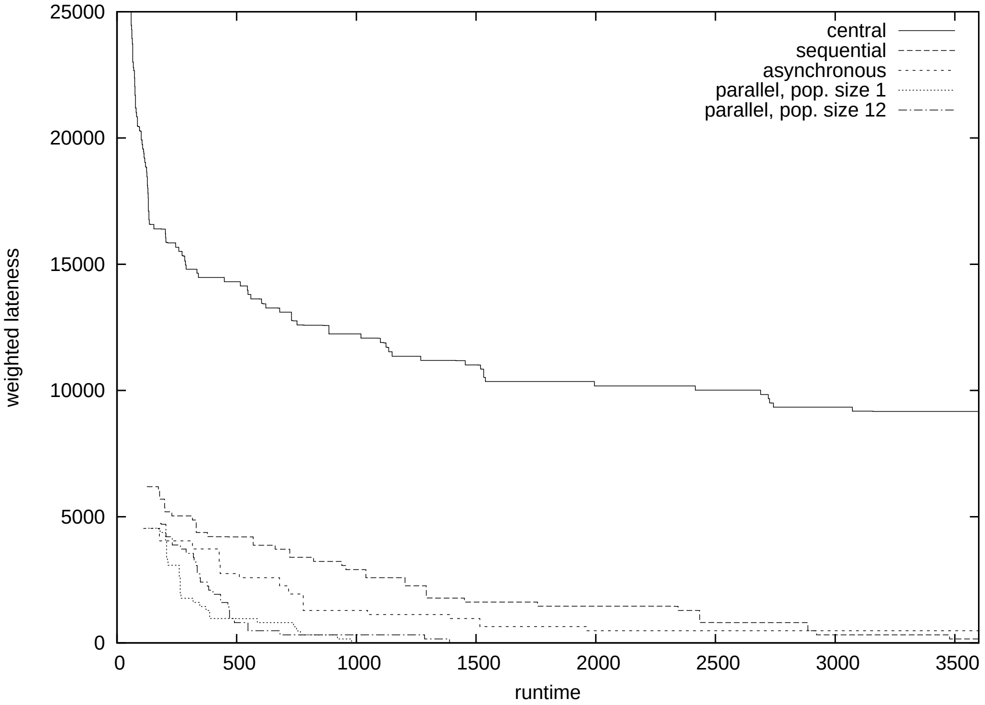


Figure 8.30: Comparison of weighted lateness for parallel, asynchronous and sequential coordination and central run for the medium instance of the setup subscenario, 25s local runtime, deterministic self-adaption.

As for the small variant, the coordination is able to better align the activities according to

[24] For an overview of abbreviations of operator subprocedures see also Table 7.3.

[25] Examining the distribution of runtime, it became apparent that the mere overhead for managing a larger population, such as detecting duplicates and redundant proposals, was relatively small and may not be the only reason for performance losses.

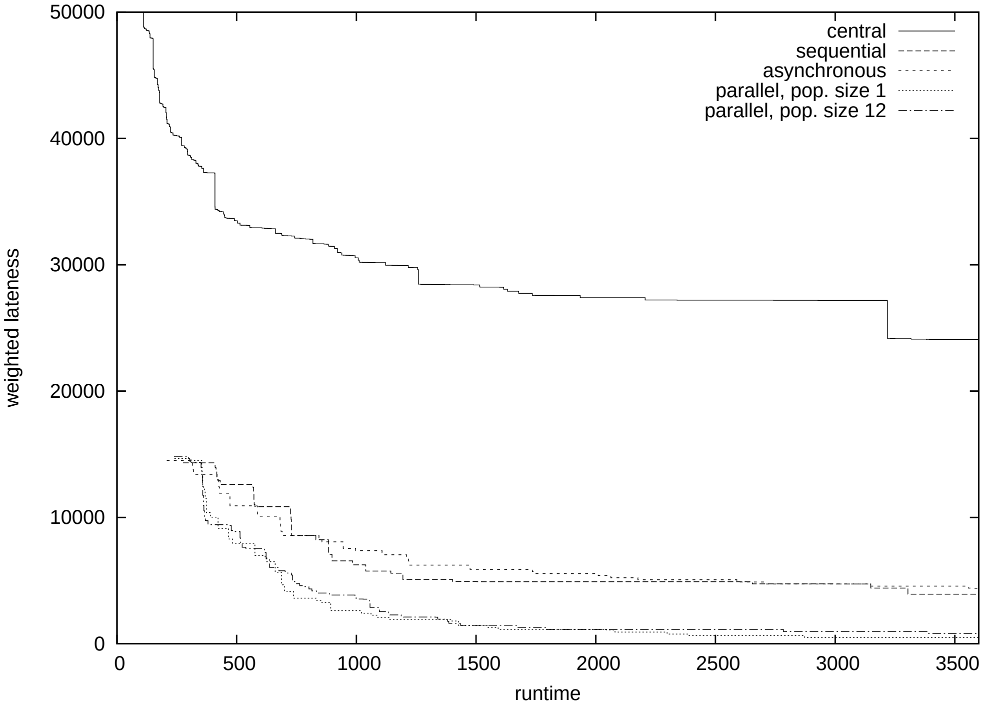


Figure 8.31: Comparison of weighted lateness for parallel, asynchronous and sequential coordination and central run for the large instance of the setup subscenario, 50s local runtime, deterministic self-adaption.

Specification	Rel. diff. from up-stream planning	Rel. diff. from central solution	No. of called operators (only OEM)
Parallel, pop. size 12	−96%	−98%	330
Parallel, pop. size 1	−96%	−98%	113
Asynchronous	−80%	−88%	88
Sequential	−78%	−87%	38

Table 8.12: Summary of average values for 10 runs for the large setup subscenario after 7,200s, 50s local runtime, deterministic self-adaption.

their precedence constraints (cf. Figures 8.22 and 8.23), which could be manually confirmed by closely examining the related Gantt charts. Another variation of the medium setup subscenario was tested, with Supplier Steel and Supplier Motor merged to a single planning domain. Interestingly, the results are very close to the two-supplier scenario, even a slightly bit worse, which might be explainable by the intuition that test instances of decreased size can be solved more efficiently by the DS solving module. The related convergence plot can be found in Appendix F (Figure F.3).

Figure 8.31 shows the convergence graphs for the large setup instance. Again, the coordination runs outperform the centrally applied DS solving module straight from the beginning. On average, the parallel coordination performs better than the asynchronous or sequential coordination. Table 8.12 gives a short overview of the results. More detail can be found in Table G.6 in Appendix G.

Comparing the small, medium and large instances of the setup subscenario, the observation can be made that the performance of the coordination mechanism is nearly independent of the problem size. The average weighted lateness for the parallel run with population size 12 reaches a final value of 3,230 for the small variant, 0 for the medium variant and 640 for the large vari-

Specification	Rel. diff. from upstream planning	Rel. diff. from central solution	No. of called operators (only OEM)
Parallel, pop. size 12	−63%	−81% (−81%)	276
Parallel, pop. size 1	−65%	−82% (−83%)	285
Asynchronous	−57%	−79% (−79%)	77
Sequential	−55%	−77% (−78%)	42

Table 8.13: Summary of average values for 10 runs for the small product mix subscenario after 1,800s, 13s local runtime, deterministic self-adaption. Values in parentheses denote the difference from the soft-constrained solution.

ant. Results tend to get better as larger instances allow more freedom of restructuring. Similar behavior can also be observed for other coordination specifications including even the sequential coordination, although it apparently gets stuck earlier at a local optimum. It is astonishing that the central runs did not scale as well; the weighted lateness increases with larger problems (the related average values are 7,253 for the small variant, 9,168 for the medium variant, 23,725 for the large variant). A different scaling can also be observed for initial upstream (central) solutions, from 10,056 (15,771) for the small variant over 4,732 (51,224) to 14,839 (76,123) for the large variant. We can argue that the additional due dates introduced by the coordination mechanism have a significant effect right from the beginning.

8.2.4.3 Detailed analysis of the product mix instances

The product mix instances differ from the setup instances in two important aspects. First, activity durations rather than setup times require a certain production and delivery sequence. Second, suppliers also have commitments to external customers not participating in the coordination. As explained in Chapter 7, due dates pertaining to such external customers are prioritized by considering them as deadlines during local optimization. This allows the computation of acceptable interorganizational solutions without exchanging side payments between the parties involved in coordination. The interesting question is to what degree this prioritization hampers the calculation of good interorganizational solutions.

Figure 8.32 shows a significant difference between coordinated and central runs. As discussed previously, central, soft-constrained runs have also been evaluated. However, both types of central runs get stuck at a local optimum very early. Detailed values can be found in Appendix G (Table G.7) and Table 8.13 provides a brief summary. Even initial upstream planning yields results that are much better. As for the setup subscenarios we can argue that the artificial due dates introduced by coordination have a very positive effect.

Since the values are already very good after initialization by upstream planning, there is no big difference between sequential, asynchronous and parallel coordination, although the parallel runs are on average slightly better. It should be emphasized that coordination results are such that no external customer’s deadline was violated.

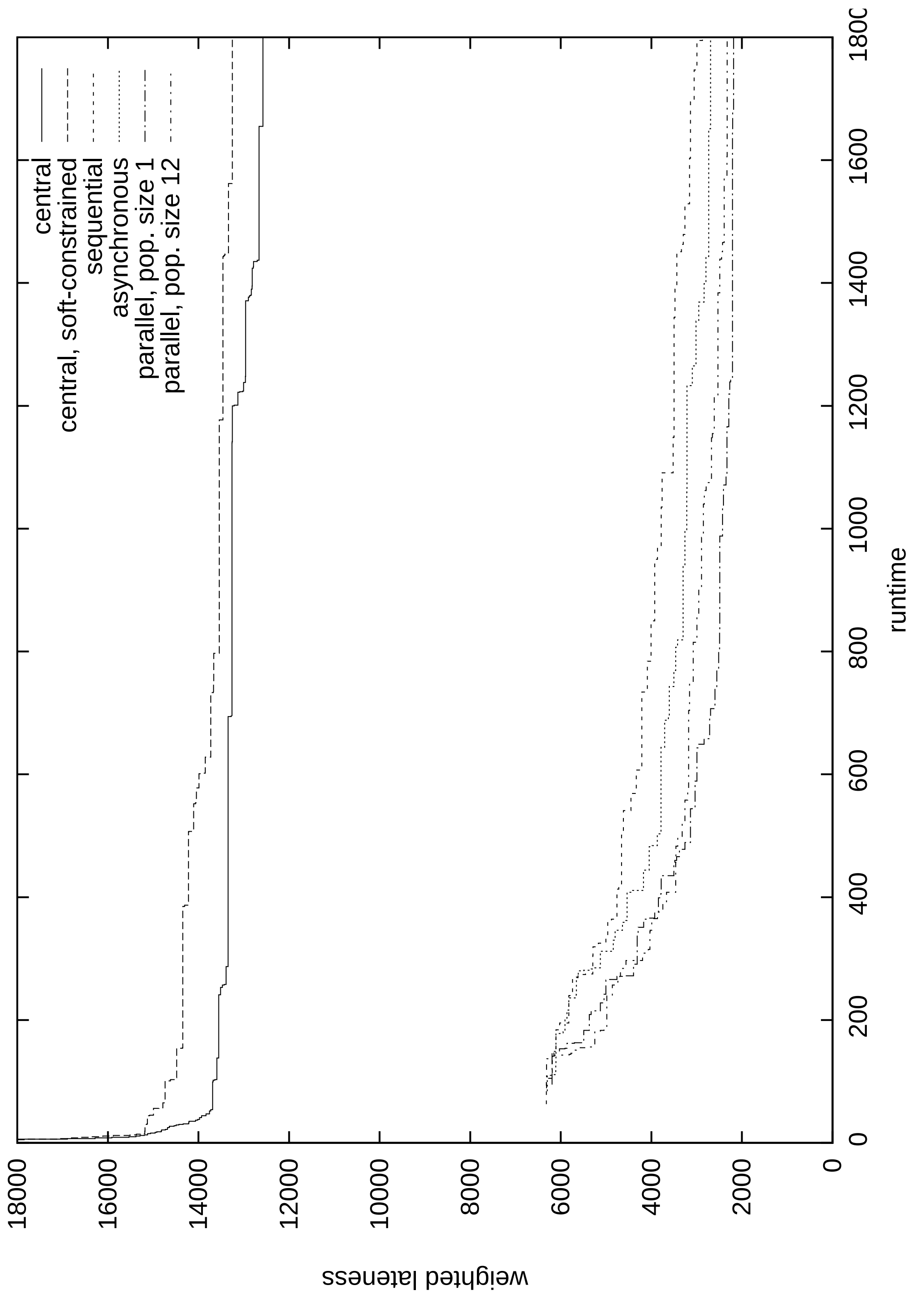


Figure 8.32: Comparison of weighted lateness for parallel, asynchonous and sequential coordination and central run for the small instance of product mix subscenario.

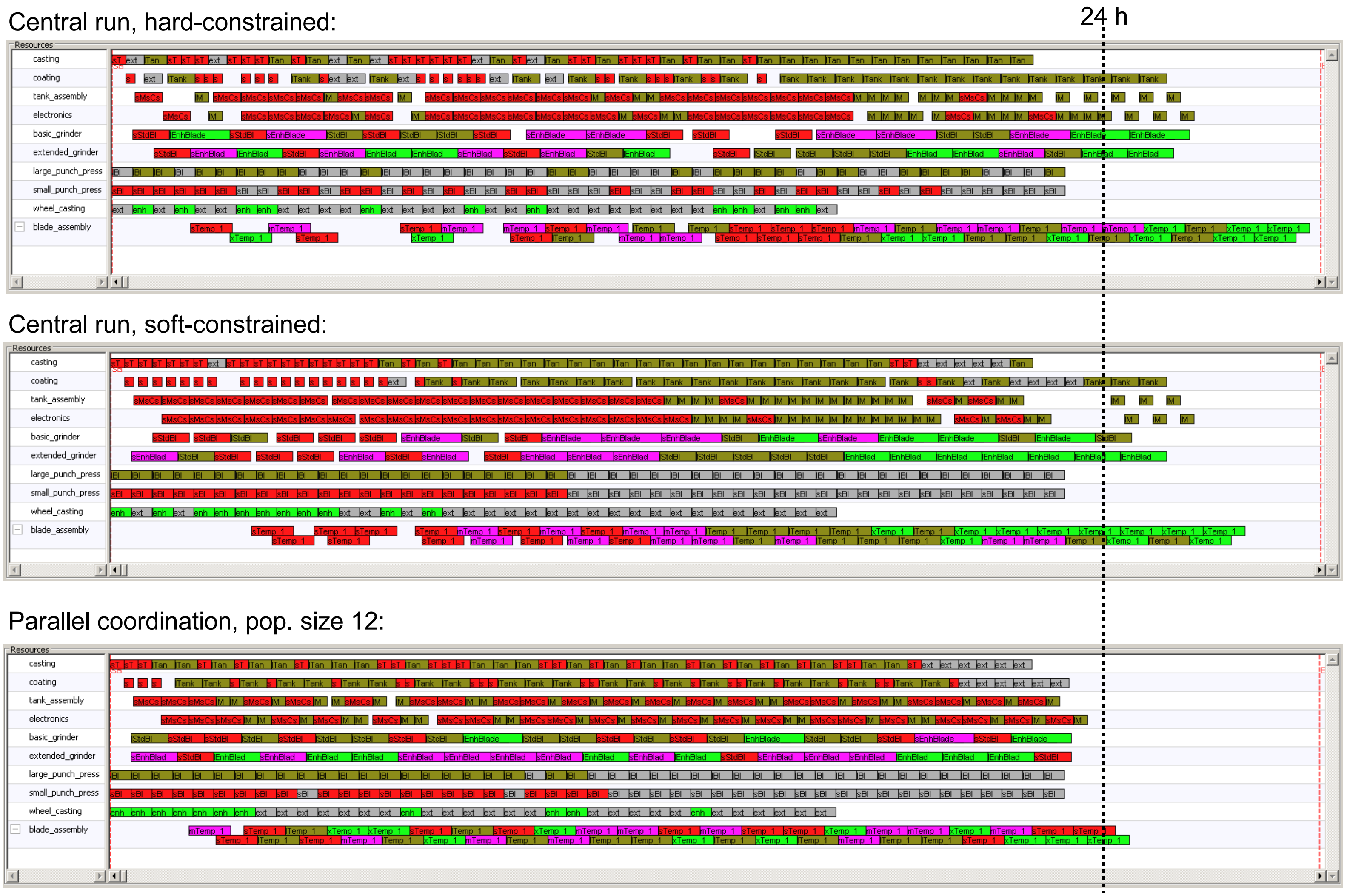


Figure 8.33: Gantt charts of soft- and hard-constrained centrally computed and coordinated schedule for the small instance of the product mix subscenario.

Figure 8.33 shows three related Gantt charts, two centrally computed hard- and soft-constrained schedules and one schedule resulting from parallel coordination with population size 12. Again, production activities of the same type are drawn in similar colors.[26] Comparing the centrally computed, soft-constrained schedule with the outcome of parallel coordination, we can observe that the centrally applied GA has difficulty finding the right sequence on the casting and coating resources. First, all the small tanks are produced and then all the large tanks. This leads to idle times on the coating resource that could be avoided if small and large tanks were produced alternatingly. At the end, these idle times are the cause for lateness. Second, we see that many enhanced blades are produced on the basic grinder. Although using this grinder implies a larger activity duration, this mode selection does not directly affect the lateness because the wrong sequence on the casting and coating resources delays the production anyway. To resolve this situation, a GA mutation operator would need to select an lTank activity placed between two other lTank activities and place it between two other sTank activities or vice versa. Moreover, also the sequence of succeeding or preceding activities and wrong mode selections might need to be changed such that the lateness really reduces. Otherwise the GA-individual does not survive the update operation. Apparently the present mutation operators have difficulty achieving all these tasks at once. We can argue that the situation illustrated by the Gantt chart marks a local optimum.

The parallel coordination is able to find a better allocation of production activities. On one hand, the already mentioned alignment effect by the additionally introduced due dates in the suppliers' subproblems might contribute to this. On the other hand, the different DEAL operators provide additional information to steer the search process in the right direction.

For the hard-constrained central run, the production of sTemp_1, mTemp_1, lTemp_1 and xTemp_1 is even more delayed. On the coating resource, the external tanks need to be produced at the end of the schedule in order to avoid idle times on the tank assembly and subsequent resources (see the other Gantt charts). Assume a schedule at the beginning of the GA where external tanks are placed at the end, but the alternating production of sTank and lTank activities is not fully established yet. The GA's mutation operator will hence try to reduce first-priority lateness and shift external tank activities earlier. However, as just discussed, this move introduces idle times on other resources. We can argue that a prioritization of external-related activities changes the fitness landscape, leaving the GA less freedom in finding good activity sequences. Apparently, this prioritization causes less harm during a coordination run.

As for the small variant of the setup subscenario, different runtimes have been compared for the parallel coordination with a population size of 12. Again, short local runtimes yielded the best result. The related convergence plot can be found in the Appendix F (Figure F.4). Moreover, the performance of different operators was compared. As for the small instance of the setup subscenario, the coordination process gets stuck quickly if only propagating operators are available. Focusing only on rescheduling-based operators yields good results. The best results are achieved by alignment-based operators (in contrast to the setup subscenario the required left-alignment worked without problems). Figure F.5 in Appendix F provides a graphical comparison.

[26] The following colors have been chosen:

- sBlade, sStdBlade, sTank_1, sTank_2, sMsCsT_1, sMsCsT_2, and sTemp_1 are red,
- lBlade, lStdBlade, lTank_2, lTank_2, lMlClT_1, lMlClT_2, and lTemp_1 are olive,
- extTank_1, extTank_2, sBladeExt_1, sBladeExt_2, and extWheel are gray,
- sEnhBlade, enhWheel, and xTemp_1 are green, and
- lEnhBlade and mTemp_1 are pink.

	Central		Sequential	
Max:total	Max lateness	Total Lateness	Max lateness	Total lateness
100:1	12,160	50,080	2,860	4,200
10:1	11,360	43,380	3,180	4,540
1:1	11,600	45,360	2,200	3,000
1:10	11,200	41,200	2,360	3,200

Table 8.14: Averaged results for different weights for maximum and weighted lateness, 10 runs each, total runtime 1,800s, local runtime 13s (in case of the sequential coordination).

In addition, we studied the effect of different weights of total and maximum lateness for the small instance of the product mix subscenario. Table 8.14 shows the average results for 10 runs of central optimization and sequential coordination with different lateness weights. For the central runs it seems to be advantageous if the weight for total lateness is set rather high. However, the sample size is too small for conducting statistical tests. According to SAP Consultancy however, users are usually more concerned about large deviations from due dates and thus set a huge maximum lateness weight. Moreover, for larger instances, a high maximum lateness weight seems to be more justified. Hence, we kept a ratio of 100:1 for maximum to total lateness for all scenarios of the deterministic test data generator.

For the medium product mix instance, Figure 8.34 graphically compares the average weighted lateness values for the coordination runs and central results, and Table 8.15 summarizes the related values. (More detailed information can be found in Table G.8 in Appendix G.)

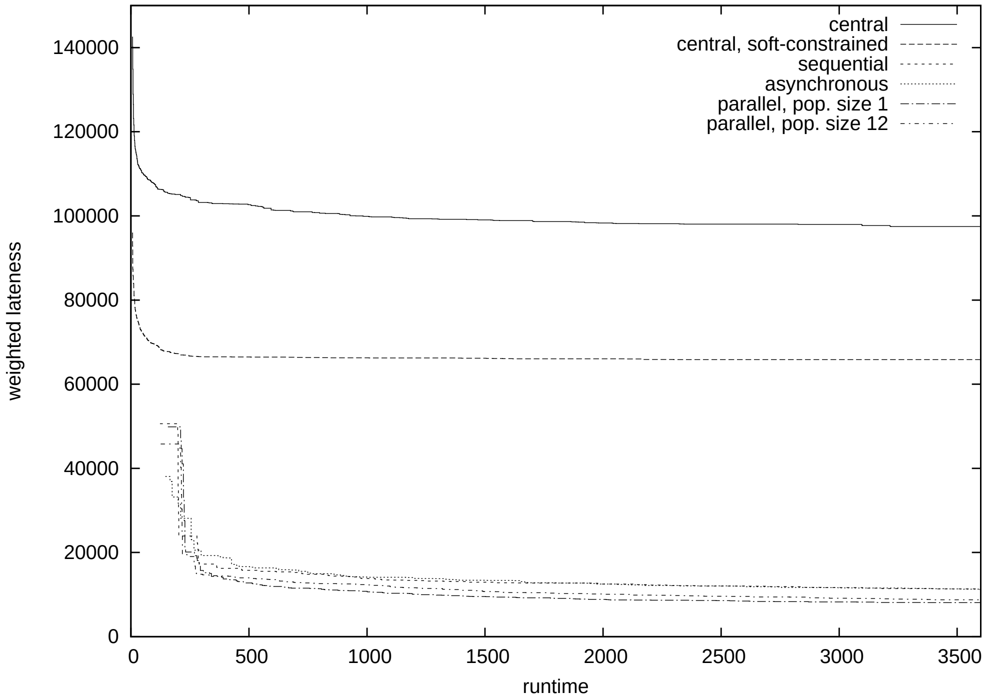


Figure 8.34: Comparison of weighted lateness for parallel and sequential coordination and central run for the medium instance of the product mix subscenario.

Several observations can be made. First, the results of the soft-constrained central runs are much better than those of the hard-constrained central runs but cannot beat those of the upstream planning solutions (also containing deadlines). However, despite the high quality of upstream planning solutions, the coordination mechanism is able to quickly improve results even further.

Specification	Rel. diff. from upstream planning	Rel. diff. from central solution	No. of called operators (only OEM)
Parallel, 12 parents	−81%	−91% (−86%)	307
Parallel, 1 parent	−83%	−91% (−88%)	339
Asynchronous	−70%	−88% (−83%)	81
Sequential	−78%	−88% (−83%)	43

Table 8.15: Summary of average values for 10 runs for the medium product mix subscenario after 3,600s, 25s local runtime, deterministic self-adaption. Values in parentheses denote the difference from the soft-constrained solution.

Most of the improvement is achieved within the first 500s.

In addition to the standard three-domain scenario, a setting was evaluated where Supplier Steel and Supplier Plastic were merged to a single supplier. It turned out that the quality of coordinated results was not influenced by this change. The related figure can be found in Appendix F (Figure F.6).

Moreover, the effect of transport disruptions was evaluated. As discussed in Subsection 8.1.2, transport resources were introduced at the interface between suppliers and OEM. Every day at midnight, a single transport from the suppliers to the OEM takes place (i.e., the transport resource is modeled to have infinite capacity for one second). An activity of transporting has a duration of zero. Figure F.7 in Appendix F shows a comparison of coordination and central runs. As the due dates pertaining to ultimate customer orders do not change, but the OEM receives the first delivery at the end of the first day, we can expect that lateness of each final activity will increase by at least one day. This increase of lateness can be observed for both central and coordinated results. However, the coordinated results are still significantly better.

In addition, the instance was evaluated where demand is more frequent (e.g. daily). Table G.1 shows the related demand matrix. Instead of demanding all items at the end of Day 4, part of the demand has to be satisfied every single day. Several observations can be made. First, the average weighted lateness at the beginning of central and coordination runs increases. One the one hand, if due dates are distributed over the planning horizon more lateness can simply be measured than if everything must be delivered at the end of the horizon. On the other hand, the planning domains have less freedom in their decisions, e.g. the building of setup campaigns becomes more difficult. Second, the soft-constrained central runs achieve better results if finer-grained demand information is available. Apparently the GA-individuals can be evaluated and compared more precisely in this setting. However, the situation is different for the hard-constrained central runs. Distributing prioritized demand over the planning horizon seems to change the fitness landscape for the worse. Surprisingly, the coordinated result do not change. The related convergence plot can be found in Appendix F (Figure F.8).

Results for the large product mix instance are very similar, demonstrating again the good scaling properties of the coordination mechanism. Related figures and tables can be found in Appendix G (Tables G.9 and G.10 and Figure F.9).

8.2.4.4 Detailed analysis of the breakdown instances

Figure 8.35 shows a convergence plot of coordination runs and the centrally applied DS Optimizer for the medium breakdown instance. The time for computing the intermediate schedule needed for the frame contract (250 seconds) is not shown in the picture. Table 8.16 summarizes the related values, more details can be found in Appendix G (Table G.11).

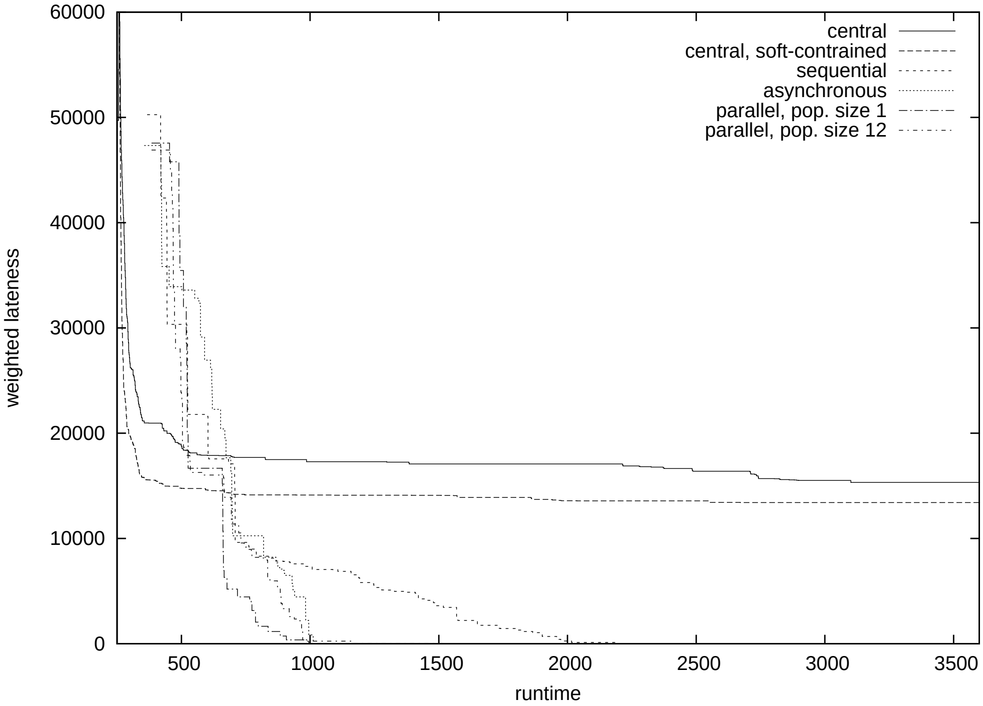


Figure 8.35: Comparison of weighted lateness for parallel and sequential coordination and central runs for the medium instance of the breakdown subscenario.

Specification	Rel. diff. from upstream planning	Rel. diff. from central solution	No. of called operators (OEM only)
Parallel, 12 parents	−100%	−100%	57
Parallel, 1 parent	−100%	−100%	44
Asynchronous	−100%	−100%	17
Sequential	−100%	−100%	14

Table 8.16: Summary of average values for 10 runs for the medium breakdown subscenario after 3,600s, 25s local runtime, deterministic self-adaption.

It can be seen that the related start dates lead to a high weighted lateness at the beginning of the coordination runs. All coordination schemes were able to compute a solution with zero lateness, whereas the parallel coordination schemes were approximately twice as fast as the sequential one. Though the DS Optimizer was occasionally able to find central solutions with zero lateness, local optima result in an average performance that is below that of the coordinated scheme, regardless of considering external-related due dates as hard or soft constraints.

For the first large breakdown instance, the parallel run with a population size of 12 was on average able to improve the upstream planning solution by 59% and central planning results by 37%. The related result, Tables G.12 and G.13, and the convergence plot, Figure F.10, can be found in Appendices G and F, respectively. It is worth mentioning that for this instance, suppliers were not able to adhere to all the deadlines of activities related to external customer orders in the upstream planning solution right from the beginning of coordination. Even though deadlines are prioritized during local optimization, the local runtime was too short to reduce all the related lateness to zero. During coordination, these undesirable violations of deadlines could gradually be decreased to zero which underpins the usefulness of the implemented double ranking scheme (cf. Section 7.4).

The impact of prioritized deadlines is even more dramatic for the second large breakdown instance. Figure 8.36 plots the convergence of weighted lateness for the different runs. More details can be found in tables G.14 and G.15 in Appendix G. All coordinated runs and the central

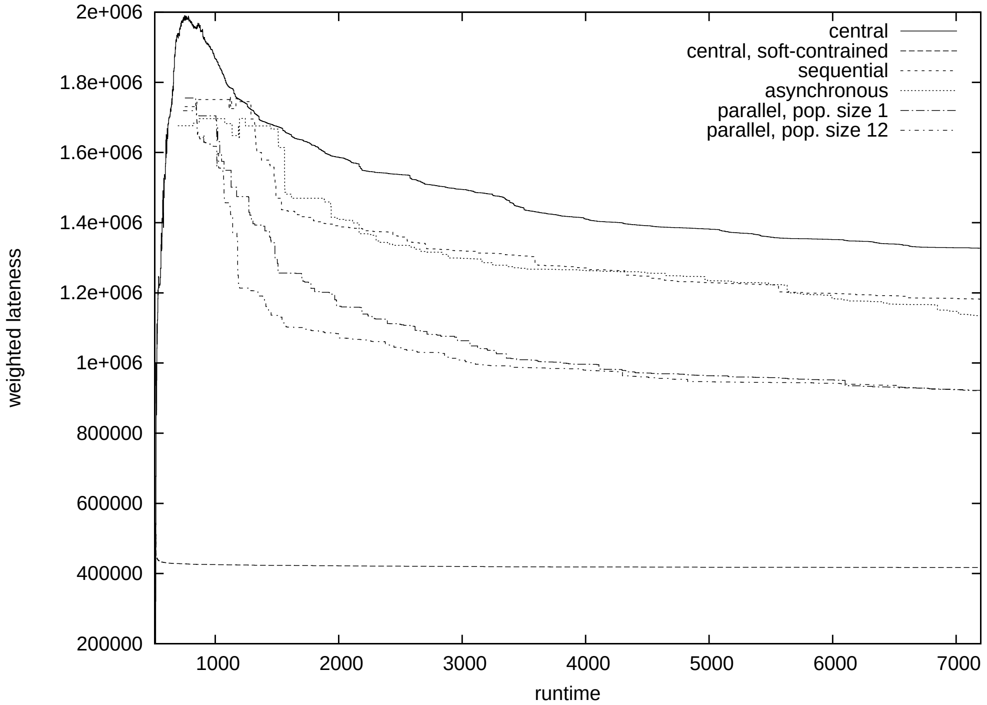


Figure 8.36: Comparison of weighted lateness for parallel and sequential coordination and central runs for the second large instance of the breakdown subscenario.

hard-constrained run exhibit an increase of weighted lateness at the beginning. This increase is related to a decrease of prioritized deadline violation not shown in the Figure. Between 1,000 and 1,500s all runs are able to decrease deadline violation to zero, and after that point they work on decreasing the weighted (unprioritized) lateness. All coordination runs produce better results than the central, hard-constrained runs. Again, parallel coordination outperforms asynchronous and sequential coordination. However, central soft-constrained runs are the clear winner for this instance. This raises the question whether the absolute prioritization of external customer orders is in general the right means to achieve global acceptability of results. Regarding the current implementation of the DS Optimizer, it is apparently the most convenient approach. However, more flexible approaches are necessary. For example, we could imagine that deadline-violation was prioritized less at the beginning of coordination to find a good overall solution, that might not yet be accepted by all domains. Then, in the course of coordination, prioritization could get a growing influence until the final outcome is globally accepted.

8.2.4.5 Detailed results for the complete scenario

Figure 8.37 compares coordination and central runs for the large instance of the complete scenario. The related values are summarized in tables G.16 and G.17. There is no huge difference between hard- and soft-constrained central results. Again, all coordination runs are much better than the central solution.

There is another, so far uncovered point. Until now, we implicitly assumed that there is only one way to compute the upstream solution: using the same bounds for local runtime that applied

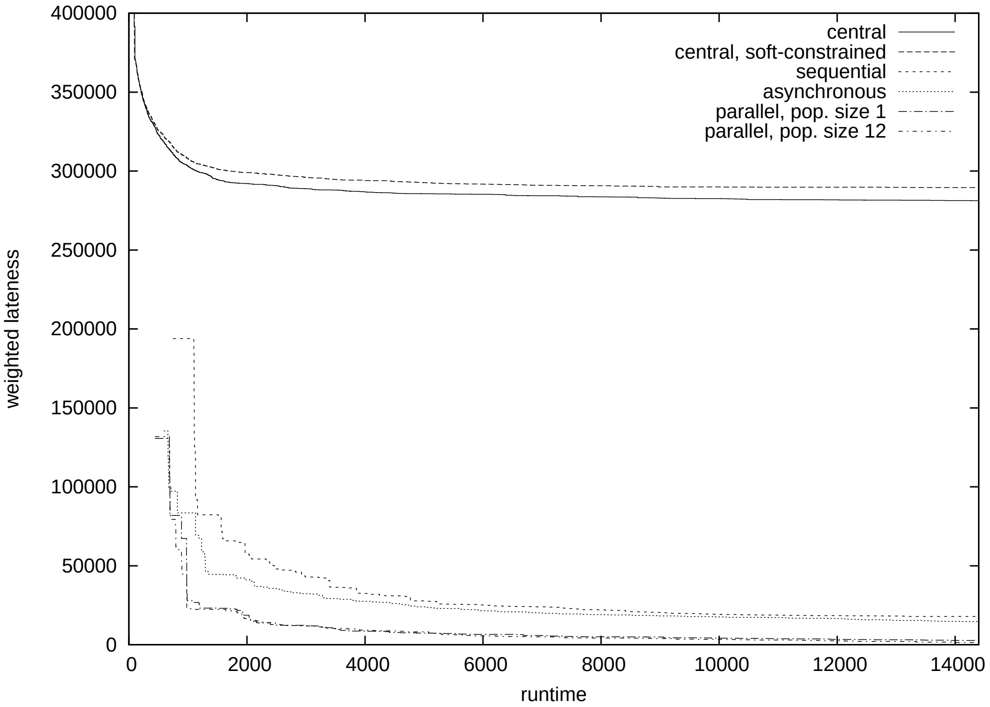


Figure 8.37: Comparison of weighted lateness for parallel and sequential coordination and central runs for the large instance of the complete scenario.

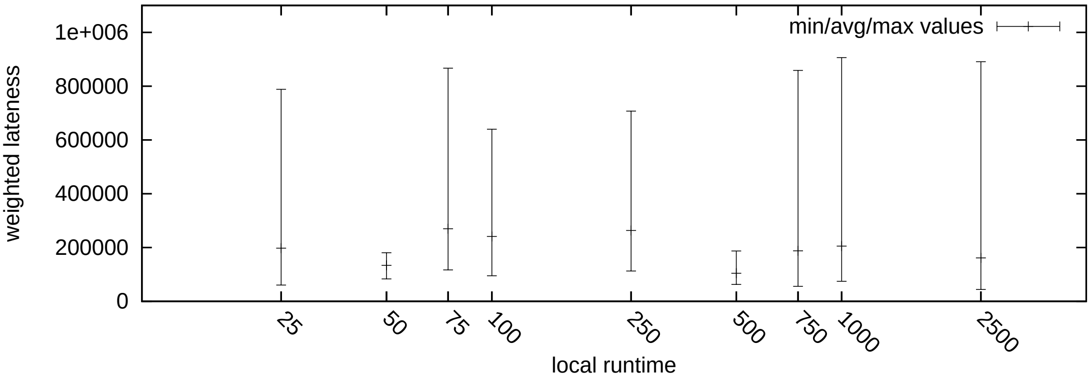


Figure 8.38: Upstream planning results for different total runtimes and the large instance of the complete scenario.

to proposals and counterproposals. However, the above results show that the amount of local runtime permitted to the DS solving module has an impact on the coordination results. Might the same be true for the upstream planning solutions? Perhaps granting a very long or very short local runtime to upstream planning might dramatically worsen or improve upstream planning results. If this was the case, which upstream planning solution should be taken as the reference point? Figure 8.38 compares average upstream planning runs for different local runtimes (10 runs per setting).

First of all, it should be emphasized that local runtimes apply to planning domains but not to the overall process. In sequential coordination, a local runtime of $2,500s$ implies a total runtime of $6 \cdot 2,500s = 15,000s$ (twice the OEM and four times the suppliers). The figure shows minimum, mean and maximum values for each local runtime setting. It can be seen that the minimum values are approximately equal, however, maximum values deviate heavily. Apparently, local runtimes of 50s and 500s support the finding of good upstream solutions, but the outliers could also be just missing by accident. Overall, no obvious pattern can be observed and the mean values deviate only slightly. Apparently, the influence of local runtimes during upstream planning is negligible. In preliminary experiments, similar results could be observed for the other instances, though not presented here.

8.3 Randomly generated test instances

The TDG of Section 8.2 creates test instances deterministically. This comes with the advantage that readers can quickly comprehend the underlying business case. A drawback is that all generated test instances have the characteristics of a flow-shop problem: The generated activity network is a repetition of BOM structures on the master data level. The critique that the generated instances consist of only a subset of the RCPSP complexity is justified. Hence, a second randomized test data generator (RTDG) was implemented (generating random network structures and resource assignments). The RTDG creates network structures that have more characteristics of job-shop problems. Before going into the details. we give a brief overview of relevant literature first.

Several randomized TDGs for generating RCPSPs already exist in literature (cf. Demeulemeester et al., 1993; Kolisch et al., 1992, 1995; Schwindt, 1995; Drexl et al., 2000). Usually, the construction of test instances follows a two-step procedure. First, several complexity measures are defined and related target values are set. Then, randomized schedules are generated that fulfill the target values. Demeulemeester et al. (1993) argue that for a comprehensive evaluation of solution methods, not only the parameters of a network (e.g., the durations of activities), but also the structure and size of benchmarking instances must be altered. From their point of view, a good TDG is one that supports an unbiased random creation of the network. In their work, they present two methods for generating dense and nondense networks, respectively. Besides network parameters, the density is regarded as the most important complexity measure.

For the TDG called *progen*, Kolisch et al. (1992, 1995) consider activity-on-node networks. As the name indicates, activities are represented as nodes in such networks and precedence constraints as directed arcs. The authors regard three complexity measures as important: the network complexity, the resource factor and the resource complexity. Exact definitions of these figure will be given in subsequent subsections. After the user has specified upper and lower bounds for activities, number of modes per activity, activity durations, number of resources, resource utilization and target values for the complexity measures, a test instance is generated. First, activities and resources are randomly generated according to the specified bounds. Then, some activities are randomly selected as start and finish activities; these are activities without predecessor or successor, respectively. All nonstart activities are assigned a random predecessor and all nonend activities are assigned a random successor. Additional precedence constraints are added until the network complexity has been reached. Moreover, activity-mode-resource combinations are randomly selected until the desired resource factor has been reached. For the selected combinations, a positive resource utilization is randomly generated according to the specified bounds. Finally, the capacities of the resources are set such that the resource strength has been reached.

Schwindt (1995) extends the progen implementation by considering minimum and maximum time lags. Another extension is provided by Drexl et al. (2000), who include mode compatibility constraints, minimum time lags, sequence-independent setup times and other constraints.

8.3.1 Test data generation

Our RTDG bases on the TDG of Kolisch et al. (1992, 1995) which has to be modified to match our coordination problem. As precedence constraints and resource assignments are randomly generated, the TDG of Kolisch et al. (1992, 1995) is usually not able to construct multidomain test instances due to cyclic dependencies between activities or resources, respectively. As an example, assume a generated network of $n = 30$ activities, two available resources A and B, one mode per activity, and one resource-assignment per mode. Assume further that the activities have been numbered according to a topological sorting respecting the precedence constraints. We can split up the network into two domains, e.g. one supplier and one OEM, if there are no cyclic dependencies between the domains (otherwise the OEM would also deliver to the supplier, which the current implementation of DEAL does not support). That is, the first i activities have been assigned to Resource A (belonging to the supplier) and the second $n - i$ activities have been assigned to Resource B (belonging to the OEM). The chance for generating such a setting randomly is $0.5^i \cdot 0.5^{n-i}$. The overall probability is thus $\sum_{i=1}^{29} 0.5^i \cdot 0.5^{n-i} = 2,6 \cdot 10^{-8}$. The extremely low probabilities for finding a suitable test instance is the reason for not considering the available standard test sets of Kolisch and Sprecher (1996), generated by the TDG of Kolisch et al. (1992, 1995).

The generator was thus modified to produce only noncyclic dependencies between planning domains. However, only the construction of two-domain scenarios (consisting of one supplier and one manufacturer) is supported. Moreover, sequence-dependent activities, maximum and minimum time lags with a distance greater than zero, and varying capacity profiles or calendars are not considered. As in Kolisch et al. (1992, 1995), the generation consists of four phases: network generation, resource-utilization generation, resource availability generation and due date generation. With regard to the original work, the construction has been slightly simplified, omitting some input parameters. We first give an introduction to the complexity measures and then describe the four phases of generating a test instance.

8.3.1.1 Complexity measures

Network complexity, C, is defined as the average number of nonredundant links (minimum time lags with distance zero) per activity. According to Kolisch et al. (1992, 1995), a link between two activities is called redundant if there exist other paths (including more than two activities) in the directed graph connecting the two activities.[27] If the activity network does not contain redundant links, the network complexity can be computed as

$$C = \frac{\sum\limits_{j \in J} |P_j| + |A| + |\Omega|}{|J| + 2},$$

where $A, \Omega \subset J$ denote the set of start and finish activities without predecessors and successors, respectively.[28] Upper and lower bounds for C can be found in Kolisch et al. (1992, 1995).

If the network complexity increases, the problem is usually easier to solve. More time lags determine to a greater extend the sequence of activities on the resources, decreasing the number

[27] This definition does not explicitly consider maximum and minimum time lags with a distance greater than zero.

[28] Kolisch et al. (1992, 1995) include a super-source and a super-sink in their model formulation. Thus, for our formulation, the number of links to super-source ($|A|$) and -sink ($|\Omega|$) need to be added explicitly in the numerator, and the denominator is increased by two.

$J^{\max}(J^{\min})$	Maximal (minimal) number of activities
$A^{\max}(A^{\min})$	Maximal (minimal) number of start activities without predecessors
$\Omega^{\max}(\Omega^{\min})$	Maximal (minimal) number of finish activities without successors
δ_{Supplier}	Supplier share of the network
$\delta^{\Omega}_{\text{Supplier}}$	Share of finish activities in supplier domain
$S^{\max}(S^{\min})$	Maximal (minimal) number of successors per activity
$P^{\max}(P^{\min})$	Maximal (minimal) number of predecessors per activity
C^*	target network complexity

Table 8.17: Parameters for generating the network in the RTDG.

of feasible sequences.

Resource factor and resource strength relate to resource scarceness. The resource factor is defined as

$$RF = \frac{1}{|J|}\frac{1}{|R|}\sum_{j\in J}\frac{1}{|M_j|}\sum_{r\in R}\begin{cases}1, \text{ if } u_{jrm} > 0\\ 0, \text{ otherwise,}\end{cases} \tag{8.1}$$

and relates to the density of the resource-utilization coefficient matrix. The resource strength, RS_r, expresses the average availability of resource r and is defined as

$$a_r = U_r^{\min} + RS_r \cdot \left(U_r^{\max} - U_r^{\min}\right), \tag{8.2}$$

where $U_r^{\max}$ and $U_r^{\min}$ denote the maximum and minimum resource utilization. The number $U_r^{\min}$ is computed as

$$U_r^{\min} = \max_{j\in J}\left(\min_{m\in M_j} u_{jmr}\right). \tag{8.3}$$

If the constant resource capacity drops below this number, no feasible schedule can be computed. After having performed a forward pass, the maximum resource utilization is calculated as

$$U_r^{\max} = \max_{t\in T}\left(\sum_{j\in J: es_j \le t \le ef_j} u_{jm^*r}\right), \tag{8.4}$$

where m^* denotes the mode with shortest duration. For $a_r = U_r^{\max}, \forall r \in R$, the schedule relating to the forward pass (where each activity j starts at ef_j) is already feasible. In other words, even if the resource capacity were larger, no better schedule could be computed with respect to the makespan or lateness criterion. In general it can be said that test instances with a high resource factor, a low resource strength and a low network complexity are difficult to solve because the actual sequence of activities becomes more and more important.

8.3.1.2 Network generation

The randomized generation of the network can be controlled by the parameters given in Table 8.17 and consists of four steps.

1. A random number of activities between $J^{\min}$ and $J^{\max}$ is generated. Moreover, the sets of start activities, A, and finish activities, Ω, are randomly selected, such that $|A| \in [A^{\min}; A^{\max}]$ and $|\Omega| \in [\Omega^{\min}; \Omega^{\max}]$. The activities are indexed in an ascending way, $j = 1, \ldots, |A|$ for all start activities and $j = |J| - \left\lceil \left(1 - \delta^{\Omega}_{\text{supplier}}\right) \cdot |\Omega| \right\rceil + 1, \ldots, |J|$ for the OEM's finish activities. Later, resource assignment will happen in such a way that the first $\lfloor \delta_{\text{supplier}} \cdot |J| \rfloor$ activities

$M^{\max}(M^{\min})$	Maximal (minimal) number of modes per activity
$d^{\max}(d^{\min})$	Maximal (minimal) duration of an activity
$R^{\max}(R^{\min})$	Maximal (minimal) number of resources
$RM^{\max}$	Maximal number of resources per mode
$u^{\max}(u^{\min})$	Maximal (minimal) resource utilization

Table 8.18: Parameters for generating the resource utilization in the RTDG.

will be in the supplier's domain. Thus, the set of the supplier's finish activities, $\Omega_{\text{supplier}} \subset \Omega$, is situated in the middle of the whole activity network, with

$$\Omega_{\text{supplier}} = \left\{ j \in J | j \leq \lfloor \delta_{\text{supplier}} \cdot |J| \rfloor \wedge j > \lfloor \delta_{\text{supplier}} \cdot |J| \rfloor - \lfloor \delta^{\Omega}_{\text{supplier}} \cdot |\Omega| \rfloor \right\}.$$

2. Beginning with the lowest indexed nonstart activity, each activity is assigned a predecessor (an activity with lower index) at random. However, no finish activity of any domain must be chosen as predecessor and the maximal numbers of successors of a preceding activity must not be violated.

3. Each nonfinish activity with no successor (an activity with higher index) is assigned one. The assignment has to respect the values $S^{\max}(S^{\min})$ and $P^{\max}(P^{\min})$. Moreover, per definition, no start activity may be predecessor of another start activity. Direct links between finish activities are prohibited as well. It has also to be taken care of that only nonredundant links are introduced. It can happen that no candidate can be found with respect to the above requirements, and even if all successors could be retrieved successfully, the network complexity might be already above its target value. In such cases the procedure aborts and restarts with a different random seed.

4. Further nonredundant links are added until the complexity has been reached, respecting the same requirements as in the previous step. Again, cases are imaginable where the target network complexity has not been reached yet but no further nonredundant links are possible and the procedure aborts and restarts with a different random seed.

Due to the way the activity network is constructed, no cyclic dependencies exist between the supplier's and the OEM's activities.

8.3.1.3 Resource-utilization generation

In this phase, activity–mode–resource assignments are computed. Table 8.18 lists the related input parameters. At the beginning, the total number of resources, $|R|$, is randomly drawn from the interval $[R^{\min}; R^{\max}] \subset \{2, 3, \ldots, \infty\}$. Then the supplier gets assigned the first $\lfloor \delta_{\text{supplier}} \cdot |J| \rfloor$ resources and the OEM the remaining ones. Respecting the above decisions, the generation of activity–mode–resource assignments now consists of two steps for each planning domain, respectively.

1. For each activity j, a random number of modes is drawn between $M^{\min}$ and $M^{\max}$. Then, for each mode m of each activity j, a duration $d_{jm} \in [d^{\min}; d^{\max}]$ is randomly drawn, a primary resource r (belonging to the current planning domain) is randomly selected, and the related resource utilization $u_{jrm} \in [u^{\min}; u^{\max}]$ is randomly chosen.

RS	Target resource strength
δ_{dd}	Due date factor

Table 8.19: Parameters for generating the resource availability and due dates in the RTDG.

2. Secondary resources with random resource utilization in $[u^{\min}; u^{\max}]$ are randomly added until the desired resource factor (see Equation 8.1) or the maximum number of resources per mode, $RM^{\max}$, is reached. If the target resource factor could not be reached, a warning message is given.

It should be noted that at least one mode with one resource assignment is constructed for each activity, $M^{\min}, R^{\min} \geq 1$.

8.3.1.4 Resource availability and due date generation

Having generated activities, precedence constraints, mode durations and resource utilizations, the resource capacities, a_r, are computed. For each resource, r, minimum and maximum utilizations, $U_r^{\min}$ and $U_r^{\max}$, are calculated as defined by equations 8.3 and 8.4. Finally, a_r is chosen such that equation 8.1 is approximately fulfilled by computing $a_r = \left\lfloor U_r^{\min} + RS \cdot \left(U_r^{\max} - U_r^{\min}\right)\right\rfloor$, where RS is the target resource strength, see also Table 8.19. Finally, due dates are set for each finish activity $j \in \Omega$. For doing this, the maximal horizon $\overline{T}^{\max}$ is computed as

$$\overline{T}^{\max} = \sum_{j \in J} \left(\max_{m \in M_j} (d_{jm}) \right).$$

Then each due date is set to

$$dd_j := ef_j + \left\lfloor \delta_{dd} \cdot \left(\overline{T}^{\max} - ef_j \right) \right\rfloor, \forall j \in \Omega,$$

where δ_{dd} denotes a predefined due date factor. The earliest finish dates $ef_j, j \in J$ are computed by an initial forward pass. Finally, the test data is split up according to the generated domain–resource assignment.

8.3.2 Test program and test instances

For each instance generated with the RTDG, 10 runs of central optimization were compared with 10 runs of parallel coordination and a population size of 12. Since only two domains are present, we changed the distribution of the 10 CPUs available for the solving units: Six solving units were assigned to the OEM and four units to the supplier. Moreover, the coordination employs the stochastic operator self-adaption scheme discussed in Section 7.8. However, the same type of operators as for the DTDG (cf. Table E.1) are used.

For the full operator set, we tested six RDTG-generated test instances: three small, two medium and one large.[29] Table 8.20 summarizes input parameters for the first small instance. The other two small instances differ in the parameter $\delta^{\Omega}_{\text{supplier}}$, which was set to values of 0.3 and 0.7, respectively. The parameters of the first medium instance can be found in Table 8.21. For the second medium instance, $\delta^{\Omega}_{\text{supplier}}$ was set to 0.5. Table 8.22 shows the input parameters for generating the large test instance.

[29] Two more test instances were tested with a reduced set of operators, cf. Section 8.4.

	Parameter	Value	Parameter	Value	Parameter	Value
Network parameters	$J^{\min}, J^{\max}$	200	$A^{\min}, A^{\max}$	20	$\Omega^{\min}, \Omega^{\max}$	20
	δ_{Supplier}	0.5	$\delta^{\Omega}_{\text{supplier}}$	0.0	C^*	1.5
	$S^{\min}$	1	$S^{\max}$	3		
	$P^{\min}$	1	$P^{\max}$	3		
Resource parameters	$M^{\min}$	1	$M^{\max}$	2	$RM^{\max}$	3
	$d^{\min}$	10	$d^{\max}$	1,000	δ_{dd}	0.1
	$R^{\min}$	6	$R^{\max}$	6	RS	0.5
	$u^{\min}$	1	$u^{\max}$	2		

Table 8.20: Parameters for the first small instance of the RTDG.

	Parameter	Value	Parameter	Value	Parameter	Value
Network parameters	$J^{\min}, J^{\max}$	1,000	$A^{\min}, A^{\max}$	50	$\Omega^{\min}, \Omega^{\max}$	50
	δ_{Supplier}	0.5	$\delta^{\Omega}_{\text{supplier}}$	0.0	C^*	1.5
	$S^{\min}$	1	$S^{\max}$	4		
	$P^{\min}$	1	$P^{\max}$	4		
Resource parameters	$M^{\min}$	1	$M^{\max}$	3	$RM^{\max}$	10
	$d^{\min}$	10	$d^{\max}$	1,000	δ_{dd}	0.1
	$R^{\min}$	20	$R^{\max}$	20	RS	0.5
	$u^{\min}$	1	$u^{\max}$	2	RF	0.5

Table 8.21: Parameters for the first medium instance of the RTDG.

For the small and medium test instances, the same local runtimes as for the deterministically generated instances were chosen, respectively. Due to the number of generated modes and involved resources, the large instance required more time as preliminary experiments suggested: 7,200s of total runtime and a limit of 240s for calling the solving module. As fewer activities were involved, we reduced the weighting for the maximum lateness. The following weights were used: $wml = 90.1\%$, $wtl = 9\%$, $wms = 0.9\%$.

8.3.3 Test results

As for the deterministically generated instances, coordination always improved the upstream planning solution. Moreover, for all randomly generated test instances, coordination was also able to beat hard- and soft-constrained centrally computed results.

Figure 8.39 compares centrally computed and coordinated results for the small instance with $\delta^{\Omega}_{\text{supplier}} = 0.0$. We can observe that parallel coordination quickly reduces the lateness of the upstream planning solution. On average, coordination also achieves better results than the centrally applied GA, which apparently gets stuck at a local optimum of 6,500. Detailed results are listed in Table 8.23. Calling 299 times the operators, DEAL was in average able to decrease the central

	Parameter	Value	Parameter	Value	Parameter	Value
Network parameters	$J^{\min}, J^{\max}$	5,000	$A^{\min}, A^{\max}$	500	$\Omega^{\min}, \Omega^{\max}$	500
	δ_{Supplier}	0.5	$\delta^{\Omega}_{\text{supplier}}$	0.0	C^*	1.6
	$S^{\min}$	1	$S^{\max}$	3		
	$P^{\min}$	1	$P^{\max}$	3		
Resource parameters	$M^{\min}$	1	$M^{\max}$	3	$RM^{\max}$	15
	$d^{\min}$	10	$d^{\max}$	1,000	δ_{dd}	0.2
	$R^{\min}$	50	$R^{\max}$	50	RS	0.2
	$u^{\min}$	1	$u^{\max}$	2	RF	0.4

Table 8.22: Parameters for the large instance of the RTDG.

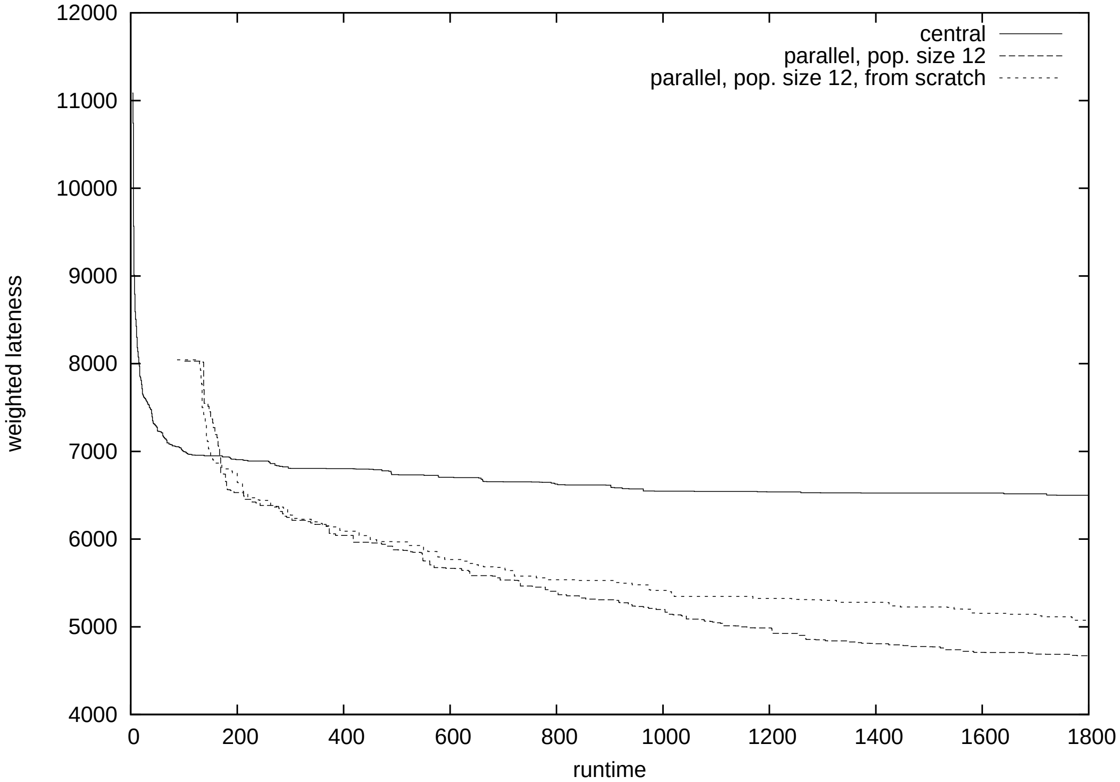


Figure 8.39: Comparison of weighted lateness of parallel coordination and central runs for the small randomly generated test instance with $\delta^{\Omega}_{\text{supplier}} = 0.0$.

	Domain	Statistic	Second priority Makespan	Max	External lateness Total	Weighted
		min	19,802	4,660	18,522	5,920
	Central	**avg**	**20,137**	**4,909**	**22,406**	**6,500**
		max	20,833	5,655	25,582	7,467
standard		min	18,697 (−12%)	3,609 (−41%)	8,899 (−68%)	4,090 (−49%)
	OEM	**avg**	**19,134 (−9%)**	**3,984 (−35%)**	**11,532 (−58%)**	**4,670 (−42%)**
		max	19,827 (−6%)	4,743 (−22%)	15,033 (−45%)	5,678 (−29%)
		min	17,703 (−12%)	0 (±0%)	0 (±0%)	0 (±0%)
	Supplier	**avg**	**18,073 (−10%)**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	18,941 (−6%)	0 (±0%)	0 (±0%)	0 (±0%)
from scratch		min	19,085 (−9%)	4,001 (−34%)	10,568 (−62%)	4,598 (−43%)
	OEM	**avg**	**19,350 (−8%)**	**4,258 (−30%)**	**13,241 (−52%)**	**5,074 (−37%)**
		max	19,755 (−6%)	4,671 (−23%)	16,532 (−40%)	5,749 (−29%)
		min	18,130 (−10%)	0 (±0%)	0 (±0%)	0 (±0%)
	Supplier	**avg**	**18,423 (−8%)**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	18,869 (−6%)	0 (±0%)	0 (±0%)	0 (±0%)

Table 8.23: Comparison of average (avg), minimum (min) and maximum (max) values of 10 runs of parallel coordination with 10 centrally computed runs after 1,800s, local runtime 13s, full set of operators, small randomly generated instance, $\delta^{\Omega}_{\text{supplier}} = 0.0$. The numbers in parentheses denote the relative difference from the average upstream planning solution.

solution by 28%. Additionally a parallel coordination was tested, where the possibility to carry over previous GA-solutions[30] when initializing a local optimization was switched off. We will denote such a coordination as **coordination from scratch**. Figure 8.39 and Table 8.23 show that the coordinaton from scratch has a small decrease in average performance. However, results do not

[30] As discussed in Section 7.9, carrying over previous solutions was done in two ways: by initializing the first GA-individual by means of back-encoding an existing schedule of an existing DEAL-solution and by injecting additional GA-individuals into the initial GA-population.

differ dramatically for the small instance.

Results are similar if $\delta^{\Omega}_{\text{supplier}}$ is increased to 0.3 or 0.7 respectively. The related figures F.11 and F.12 and tables G.18 and G.18 can be found in the Appendix. Due to the randomized generation of modes and resource assignments, it is difficult to figure out *why* DEAL is able to beat the centrally computed results. It can be argued that the same causes as for the DTDG will hold: artificially introduced due dates and information of DEAL operators that contribute to steering the search process in the right direction. For the medium-sized instance with $\delta^{\Omega}_{\text{supplier}} = 0.0$ similar observations can be made, cf. Figure F.13 and Tables G.20 in the Appendix. Given the runtime settings, DEAL calls in average 410 times an operator for coordinating a medium instance. For the medium-sized instance with $\delta^{\Omega}_{\text{supplier}} = 0.5$, we additionally tested the performance of the coordination from scratch. Results are shown by Figure 8.40 and Table G.21 (in the Appendix).

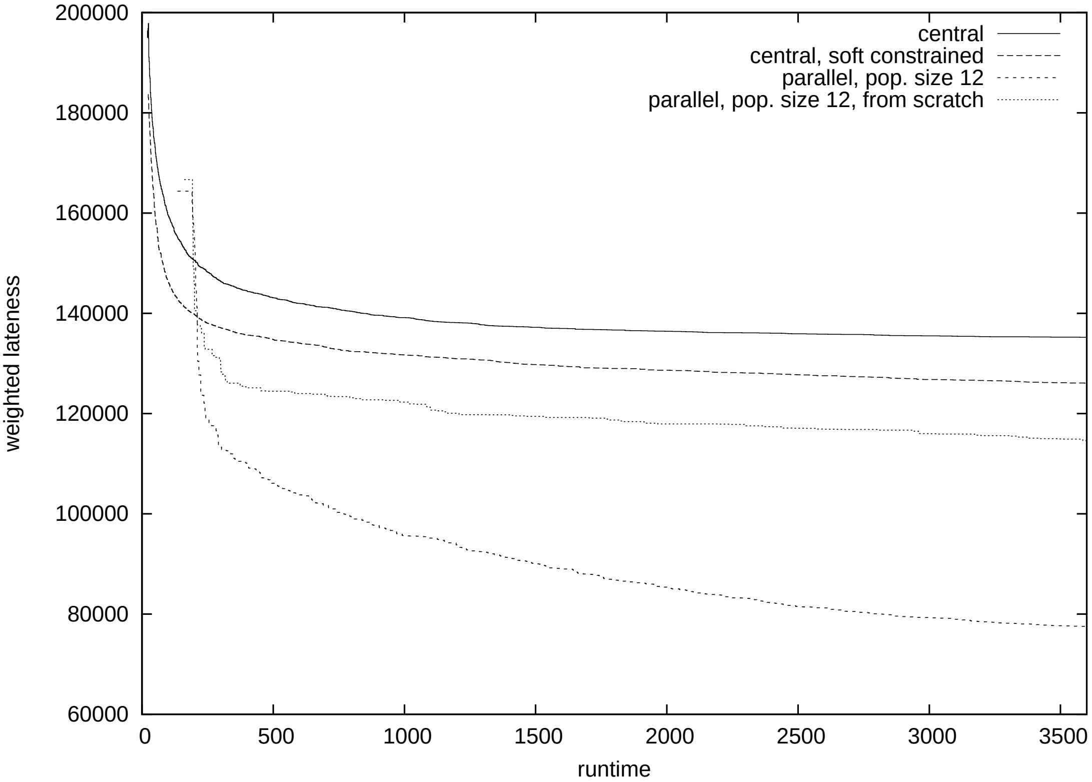


Figure 8.40: Comparison of weighted lateness of parallel coordination and central runs for the medium-sized randomly generated instance with $\delta^{\Omega}_{\text{supplier}} = 0.5$.

It can be seen that results get significantly worse, if previous results are not carried over for initializing a new GA-population. Apparently, with a growing problem size the gap between standard coordination and coordination from scratch gets larger. This can be explained intuitively. On the one hand, the problem complexity increases. The GA simply takes more time for finding good solutions. Hence, reusing results has a larger effect if test instances are larger. On the other hand, larger test instances might provide more flexibility: if activity lists are longer, probability might increase that at least parts of a previous result can be reused successfully.

Figure 8.41 shows a convergence plot of central computation standard and from scratch coordination for the large randomly generated instance. Related results can be found in Table G.22 in the Appendix. Due to the large local runtime, only few individuals can be constructed during coordination. In average, the OEM called operators 77 times. The figure shows that in average the

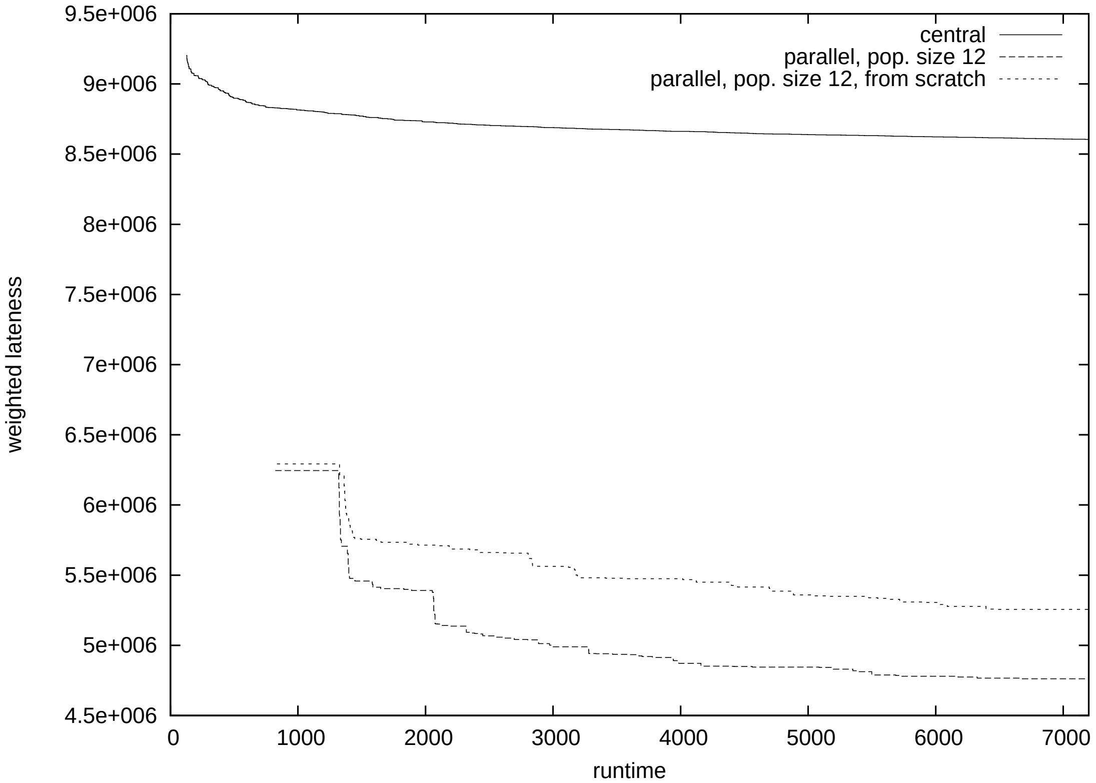


Figure 8.41: Comparison of weighted lateness of parallel coordination and central runs for the large randomized scenario, $\delta^{\Omega}_{\text{supplier}} = 0.0$.

coordination process quickly improves the lateness after initialization but gets then gets slower and slower in finding improved solutions. As for the medium-sized instance, a coordination from scratch exhibits a significantly worse convergence.

8.4 Additional tests for the stochastic self-adaption of operators

8.4.1 Reducing the set of operators

In Chapter 7, we discussed several operators that enable the OEM to compute new proposals. Moreover, two self-adaption schemes were presented: a deterministic one relying on an initial list of operator specifications that is subsequently re-sorted and a stochastic one employing techniques from Ant Colony Optimization. The above sections showed that both schemes are able to reduce the lateness of the joined schedules during coordination. However, two question remain: Which self-adaption scheme is advantageous and are all operators actually needed? To tackle the first question, most of the DTDG instances were solved again using the stochastic self-adaption in a parallel coordination with a population size of 12. Moreover, we tried to reduce the operator set using deterministically and randomly generated instances. To do so, the operators were ranked according to their performance within the stochastic self-adaption in the chosen test instances. For each operator, performance was measured as the fraction of successful operator applications (the constructed individual was not classified as redundant and the constructed child achieved a better global ranking than the parent) over all applications of this operator. Based on the performance information, we derived a reduced set of operators. More details on retrieving this set

	Instance	Deterministic self-adaption	Stochastic self-adaption		
		Full set	Full set	Reduced set	More reduced set
DTDG	Small setup	−53%	−48%	−24%	−37%
	Medium setup	−100%	−100%	−100%	−100%
	Large setup	−98%	−99%	−97%	−97%
	Small product mix	−83%	−84%	−84%	−82%
	Medium product mix	−87%	−87%	−84%	−83%
	Large product mix	−79%	−71%	−66%	−65%
	Medium breakdown	−100%	−100%	−100%	−100%
	Large breakdown, 1st variant	−13%	−8%	−8%	−7%
	Large breakdown, 2nd variant	+120%	+120%	+153%	+154%
RTDG	Small, $\delta^{\Omega}_{\text{supplier}} = 0.0$	n.a.	−28%	−23%	−23%
	Small, $\delta^{\Omega}_{\text{supplier}} = 0.3$	n.a.	−35%	−27%	−26%
	Small, $\delta^{\Omega}_{\text{supplier}} = 0.7$	n.a.	−47%	+27%	+23%
	Medium, $\delta^{\Omega}_{\text{supplier}} = 0.0$	n.a.	−56%	−58%	−55%
	Medium, $\delta^{\Omega}_{\text{supplier}} = 0.5$	n.a.	−39%	−37%	−37%
	Large, $\delta^{\Omega}_{\text{supplier}} = 0.0$	n.a.	−45%	−45%	−42%

Table 8.24: Average relative differences of weighted lateness from central, soft-constrained solutions for different test instances and operator sets.

Ranking	DTDG		RTDG
	Setup instances	Product mix & breakdown instances	
1.	SSB | FWD | RA (57%)	SSB | FWD | RA (85%)	PLX (78%)
2.	FS | PDD (50%)	IDD | FWD | RA (74%)	SSB | FWD | RA (73%)
3.	IDD | FWD | RA (47%)	LX (72%)	RA | FDS | PDD (66%)
4.	RMD | FWD | RA (41%)	RRD | LA | RA (71%)	FS | ICR | FWD | RA (60%)
5.	RA | FDS | PDD (38%)	FS | RMD | LA | RA (67%)	IDD | FWD | RA (59%)
6.	RRD | FWD | RA (37%)	PLX | FWD | LA | RA (66%)	RMD | LA | RA (59%)

Table 8.25: The six most successful operators for the stochastic self-adaption and the full operator set. The values in parentheses denote the operator performance.

and its composition are presented later. After reevaluating the test instances with the reduced set we were able to reduce the set of operators even further. Table 8.24 gives an overview of the chosen test instances and the relative deviation of each operator set from the average central, soft-constrained solution. We can observe that performance on average declines a bit if the operator set gets smaller, but still is of considerable strength. Moreover, it can be seen, that there is no large deviation between deterministic and stochastic operator self-adaption for the test instances generated by the DTDG.

To evaluate the performance of the full operator set, we divided the test instances into three classes: deterministically generated test instances with setup times, deterministically generated test instances without setup times and randomly generated test instances. This division was motivated by the intuition that each class has different characteristics. The previous sections showed that alignment-based operators perform poorly on instances including setup times. Hence, for instances deterministically generated from the setup subscenario, we replaced all alignment-based operators with their hybrid versions (performing a reoptimization instead of a left-alignment). Table 8.25 shows the six best operators[31] for each set.[32]

It can be seen that some operators occur in every class, for example the operators *SSB | FWD | RA*

[31] Cf. Table 7.3 for an overview of the abbreviations (for hybrid sorting–alignment operators the left-alignment step (LA) is replaced by a rescheduling (FWD)).

[32] Regarding the performance of crossover operators, the following caveat should be mentioned. Crossover combines a parent individual with other individuals of the population. However, to measure the improvement, only the parent individual is considered. Hence, if the selected parent is an individual with low fitness, an improvement might have some natural probability, possibly leading to biased performance values. It is not clear to what degree this effect influences the performance values. We kept the crossover operators since we believe that combining several individuals of the population helps in steering the search process in the right direction.

Ranking	DTDG		RTDG
	Setup instances	Product mix & breakdown instances	
1.	SSB ∣ FWD ∣ RA (62%)	SSB ∣ FWD ∣ RA (78%)	PLX (78%)
2.	IDD ∣ FWD ∣ RA (47%)	RRD ∣ LA ∣ RA (72%)	IDD ∣ FWD ∣ RA (66%)
3.	RMD ∣ FWD ∣ RA (40%)	IDD ∣ FWD ∣ RA (71%)	RRD ∣ LA ∣ RA (60%)

Table 8.26: The three most successful operators for the stochastic self-adaption and the reduced operator set. The values in parentheses denote the operator performance.

	parameter	value	parameter	value	parameter	value
network parameters	$J^{\min}, J^{\max}$	1,000	$A^{\min}, A^{\max}$	500	$\Omega^{\min}, \Omega^{\max}$	500
	δ_{Supplier}	0.5	$\delta^{\Omega}_{\text{supplier}}$	0.0	C^*	1.498
	$S^{\min}$	1	$S^{\max}$	1		
	$P^{\min}$	1	$P^{\max}$	1		
resource parameters	$M^{\min}$	1	$M^{\max}$	1	$RM^{\max}$	1
	$d^{\min}$	10	$d^{\max}$	1,000	δ_{dd}	0.1
	$R^{\min}$	2	$R^{\max}$	2	RS	0.5
	$u^{\min}$	1	$u^{\max}$	2	RF	1.0

Table 8.27: Parameters for the first additional medium-sized scenario of the RTDG.

or *IDD ∣ FWD ∣ RA*. Other operators seem to work well just for a specific class. We chose the union of the six best operators for every class as the **reduced operator set**. This reduced set consisted of 12 operators in total.

Using this reduced operator set, the test instances were again evaluated using the parallel coordination employing a population size of 12 and the stochastic self-adaption scheme. For these results, the three best operators for every class are listed in Table 8.26. Again, the union of the three best operators for every class was chosen as the **more reduced operator set**, which consists of five operators in total. Finally, the test instances were again evaluated using the latter set.

Regarding all test classes and the parallel coordination employing this more reduced operator set and a population size of 12, the operators showed the following performance. Linear crossover of the whole population without rescheduling (PLX) was able to improve a solution in 66% of all cases. Rescheduling within hard bounds (SSB ∣ FWD ∣ RA) succeeded in 64% of cases. The performance for rescheduling with penality (IDD ∣ FWD ∣ RA) was 61%. Relaxing all activities followed by a realignment (RRD ∣ LA ∣ RA) was able to improve 48% of the parent solutions. The hybrid version of relaxing the most delayed activities (RMD ∣ FWD ∣ RA) also had a performance of 48%.

Obviously, we can reduce the operator set further and further. However, since we are evaluating only a nonrepresentative set of test instances, the question arises to what degree this proceeding is useful. We decided to stop here and recommend the above five operators for future practical implementations.

8.4.2 Additional tests using the reduced set of operators

Using the more reduced operator set, four medium-sized test instances were additionally evaluated. We generated two of them with the RTDG and the other two with the DTDG. Table 8.27 shows the input parameters for the RTDG of the first test instance.

According to the parameters, a test instance with a very simple structure is created. Supplier and OEM have a single resource each with constant capacity. Activities utilize either the half or the full capacity. Precedence constraints only exist between the planning domains, but not within a planning domain. In other words, each OEM activity has exactly one predecessor in the supplier domain and each supplier activity has exactly one successor in the OEM domain. By using this test instance we wanted to investigate the performance of DEAL in environments with only a few production stages. Figure 8.42 shows the related convergence plot, detailed numbers can be found in the Appendix (Table G.23).

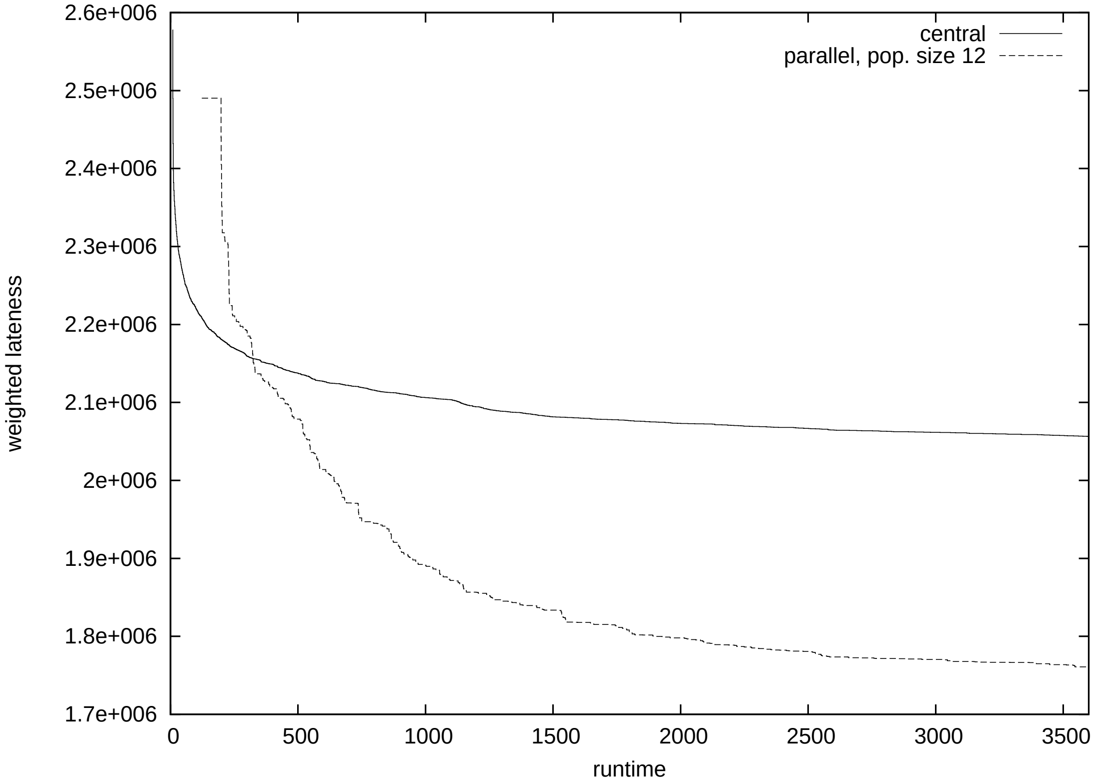


Figure 8.42: Comparison of weighted lateness of parallel coordination and central runs for the first additional medium-sized randomized scenario, more reduced operator set.

In the figure it can be seen that DEAL shows a slightly better performance (with respect to the scaling of the y-axis). After 1 hour, the coordination exhibits in average a 14 % lower weighted lateness than the central solution. We were then interested in investigating if this behaviour changes if more production stages are added. Other parameters remaining equal, we changed $A^{\min} = A^{\max} = 50$ as well as $\Omega^{\min} = \Omega^{\max} = 50$. Hence, for 1,000 activities and only a single allowed predecessor and successor per activity, the generated instance exhibits not only two, but in average twenty production stages, distributed evenly between the planning domains. Figure 8.43 shows the related convergence plot, detailed numbers can be found in the Appendix (Table G.24). The values of weighted lateness are smaller, since only the last activity in each connected component of the activity network is assigned a due date. Again, the parallel coordination outperforms centrally computed results. This can be regarded as indication that DEAL does not only scale well with respect to the number of involved activities, but also with respect to the number of involved production stages.

Until now, calendars have not been considered. Thus, we also tested the behavior of DEAL for test instances involving calendars. For this purpose, two additional medium-sized instances of the complete benchmark scenario were generated using the DTDG. We assume that production is possible workdays during 6 am to 10 pm but not on the weekends (days 6 and 7). As discussed in chapters 5 and 7 the breaks are assumed to be preemptive, i.e. the last activities in

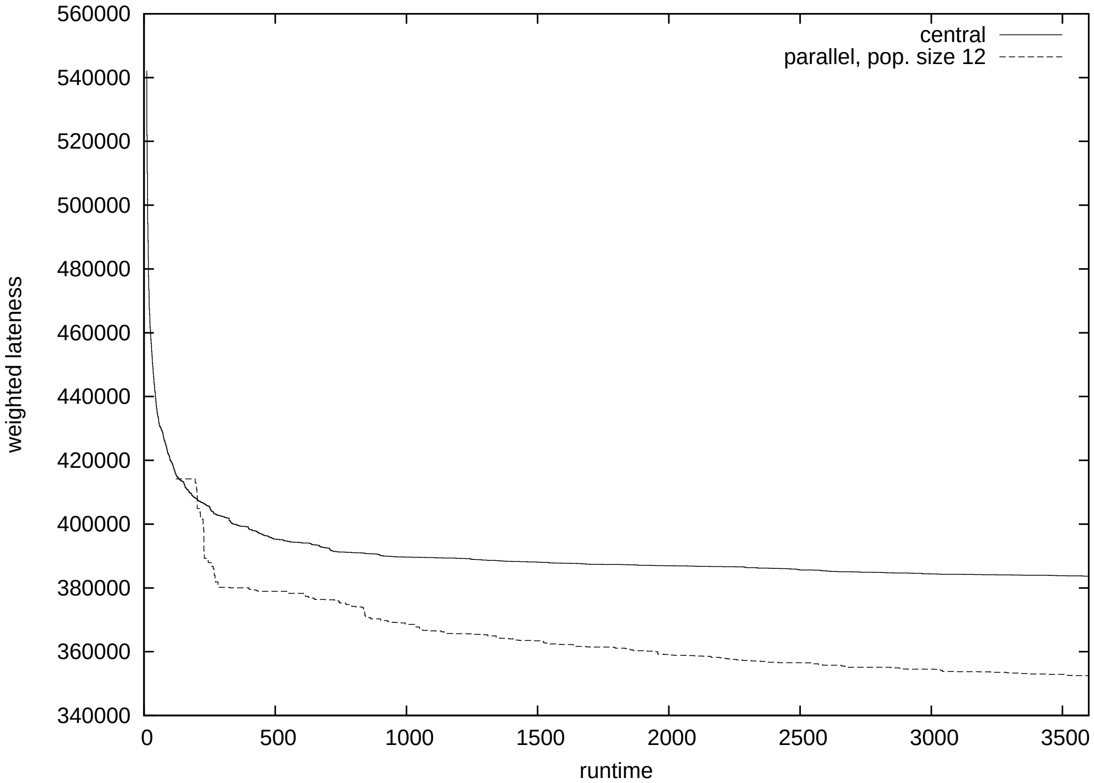


Figure 8.43: Comparison of weighted lateness of parallel coordination and central runs for the second additional medium-sized randomized scenario, , more reduced operator set.

the productive shift are stretched by the breaks and are continued by the next shift. The related demand matrices can be found in the Appendix, Table G.25 and Table G.26. Demands are no longer equally distributed as previously but exhibit irregular patterns. For the first instance, a transportation resource was additionally modelled, that allowed a transport at midnight every day. Transportation introduces a nonpreemptive component into the test instance. For both instances, DEAL outperforms hard- and soft-constrained central runs. Related figures can be found in the Appendix (figures F.14 and F.15 and tables G.27 and G.28). We can regard these results as an indication that calendars do not hamper the performance of DEAL.

CHAPTER 9

Conclusion

In this thesis we presented a Decentralized Evolutionary Algorithm (DEAL) for reconciling interdependent detailed schedules of a manufacturer and multiple suppliers. After the introduction in Chapter 1, Chapter 2 gives an overview about Advanced Planning Systems and their modules. In Chapter 3, we defined the term Collaborative Planning and reviewed state-of-the-art literature. It became obvious that for designing a coordination mechanism, three fundamental requirements have to be considered: Complex planning problems have to be supported, sensitive data must not be disclosed and incentive compatibility has to be guaranteed, preventing partners from supplying biased input to the mechanism. To ensure the latter, most present approaches rely on side payments, payments made in an agreement by one or more parties to other parties to induce them to join the agreement. Though this idea seems to be logical at first sight, side payments are not without problems. As argued, such payments may - if not properly designed within the mechanism - provide possibilities for cheating and probing the other partner to infer sensitive information instead of preventing them from doing so. Thus, we did not integrate such payments into DEAL, but restricted it to generating solutions that are *acceptable* to all partners. Naturally, this decision prunes the search space, possibly making the global optimum unattainable. However, it is questionable if this is a real drawback. Usually, side payments are hardly deducible from control costs of objective functions and for many practical problems the optimal solution is usually not attained anyway. For short-term planning it was argued that it might be rather *advantageous* to shift the focus from cash flows in accountancy (optimizing monetary values) to availability in logistics (optimizing material flow and resource utilization by means of control cost).

It appears to be difficult to consider all of above three requirements to the fullest in general. Our intention was to focus on a setting of limited scope and to develop a coordination mechanism that is *practically* acceptable in such a setting. Practical acceptability adds further requirements: The mechanism should handle multiple domains, different optimization engines and different optimization models. It should be scalable to large problems and able to handle degraded, suboptimal solutions. Moreover, due to the limited time before escalation, short-term scheduling most urgently demands coordination.

We believe Evolutionary Algorithms to be the logical choice under such conditions. They are not restricted to a particular type of problem requiring only a black-box fitness computation and are, as a population-based approach, easy to parallelize. In Chapter 4, we discussed how these properties can be used for Collaborative Planning. Essentially, the idea is to adjust the data of underlying planning problems by an evolutionary process, using local optimization engines for decoding. As only the local data, but not the related optimization models are changed, different optimization engines can be connected to our framework. For evaluating different proposals, we disguised sensitive data and limited possibilities for cheating by transmitting only ordinal rankings (reflecting a proposal's quality) between the partners.

Chapter 6 presents SAP's tool for short-term production planning, the Detailed Scheduling (DS) Optimizer. Moreover, the underlying mathematical problem, the Resource-Constrained Project Scheduling Problem (RCPSP), which belongs to the class of NP-hard problems, was formulated in Chapter 5. To tackle the problem's combinatorial complexity, the DS Optimizer itself employs an Evolutionary Algorithm.

In Chapter 7, a customization of DEAL to the DS Optimizer was presented. First, we clarified how planning domains using the DS Optimizer interact with each other. Essentially, the domains are interdependent on chosen delivery sequences that determine the constraints of local optimization problems. These delivery sequences are determined by proposed dates of delivery transmitted from one planning domain to the other. In particular, DEAL was customized to handle multiple suppliers delivering to one manufacturer. A possible extension to three tiers was additionally discussed. Second, several operators were presented for analyzing current and past interorganizational solutions and deriving new promising proposals. Third, further means were discussed to ensure efficiency of the algorithm: methods to detect and eliminate duplicates and to support a self-adaption of operators. In fact, DEAL's customization is a *metametaheuristic*. The top-level metaheuristic is concerned with aligning constraints (by calibrating the models' data) of the local optimization problems, while the low-level metaheuristic of the DS Optimizer is used to calculate the related detailed schedules. This opens interesting opportunities, for example carrying over low-level solutions from one constraint setting to the other.

Chapter 8 provides a computational evaluation. We first give examples for typical building blocks that lead to conflicting delivery sequences, even though planning domains share the same objectives. Two test data generators (TDGs) were developed. A deterministic one that is a simplified simulation of the standard SAP planning procedure and a randomized one, allowing the construction of RCPSPs to certain complexity targets. The TDGs also differ in the type of problem they produce: The deterministic TDG generates problems with a flow-shop characteristic, while the randomized one generates problem characteristica typical for job-shop problems.

Various degrees of freedom exist to set up the test instances and the test program, such as the size of problem instances, the amount and distribution of runtime or different population sizes. Due to the numerous possibilities, we were able to give a first impression about the effectiveness of DEAL rather than to provide a "complete" computational evaluation. Nevertheless, several interesting observations can be made.

First, it is worth highlighting that DEAL showed an overall good performance, always improving the upstream planning solution and sometimes even improving centrally computed results. We argued that several factors contribute to this. Proposed dates of delivery imply certain tradeoffs in the local optimization problem. Rather than the dates' absolute values, these tradeoffs steer the local optimization methods. Hence, a crude alignment of delivery dates by DEAL seems to be sufficient—the exact setting of delivery dates is done subsequently by the local optimization methods themselves. Moreover, DEAL can be regarded as a decomposition method working on multidomain problems that exhibit specific properties. For example, there are no cyclic dependencies between the different planning domains. We argued that DEAL makes use of these problem structures and hence is able to even beat the performance of the centrally applied DS Optimizer.

Second, according to the computational evaluation, DEAL does not require more overall runtime to achieve the results than the centrally applied DS Optimizer. This is due to the fact that considerably short *local* runtimes for computing and evaluating proposals proved to be sufficient. One feature of DEAL is seen as an essential precondition in this regard: When initializing a local optimization, DEAL carries over previous local results pertaining to other proposals. Thus, a whole coordination run can be regarded as a single optimization process. Whenever a proposal is exchanged, the process is interrupted and the model data are slightly changed. Shorter local runtimes hence allow more proposals to be exchanged. Due to the combinatorial complexity of the RCPSP, a frequent exchange of proposals is advantageous.

Third, DEAL showed a very good scaling. Even for larger problem instances a good coordinated solution was found in reasonable time. Although complexity increases, larger problem instances usually allow more freedom of rescheduling.

Fourth, it could be observed that larger population sizes lead to better results. This is a common property of Evolutionary Algorithms: Larger populations have a better chance of overcoming local optima, at the cost of a slower convergence, however. By computing and evaluating proposals in parallel, DEAL allows larger population sizes without prolonging runtime. As computational power is today available at low costs, the parallel coordination seems to be recommendable.

Fifth, we showed that DEAL is able to successfully tackle complex SC structures with multiple suppliers that are also delivering to other customers, not taking part in the coordination process. By using a proper ranking scheme and calibrating the local optimization methods, DEAL works fully automated, generating only solutions that are acceptable by all involved partners.

Sixth, we were able to reduce the set of operators, leaving only the five high-performance ones. Last but not least, further complicating problem properties, such as break or transport calendars do not seem to provide any problem to DEAL.

However, there exist several caveats that should be mentioned as well. First of all, the number of problem instances used for computational evaluation is limited. Certain problem ingredients, such as maximum time lags or mode or setup costs have not been considered at all. Though we tried to simulate the standard SAP planning procedure as realistically as possible, there is no guarantee that the problem instances exhibit all those properties found in reality. Moreover, DEAL has only been verified for the DS Optimizer. Though the connection of other optimization tools is theoretically possible, there is no empirical evidence for the hypothesis that DEAL will actually work with them. Also the presented operators for constructing proposals are specially adapted to the DS Optimizer—again, there is no guarantee that they will work if other solution methods are used. In general, it is difficult to characterize the fitness landscape and operators have been build on intuition and the on the results of preliminary experiments. Similar uncertainties hold for three and more tiers. Possibilities for tackling larger supply networks were only discussed very briefly—again there is no validation that the presented ideas will work.

The computational evaluation is not without problems, as all results are computed by metaheuristics. As we have seen, it is hard to define any reference points for evaluating DEAL. Centrally computed results are sometimes better, sometimes worse. For some instances, upstream planning results are already better than the central ones. However, the performance of upstream planning itself is apparently dependent on local runtimes. Moreover, objective weights have a large influence on all results.

In addition, some of our assumptions are critical: First, costs for buffering intermediate items delivered from supplier to manufacturer have not been considered at all. However, these costs are relevant in a practical setting as they lock up capital. Including such holding costs into the calculus introduces a new objective, possibly hampering the exploration of different delivery sequences.

Moreover, an immediate transportation in zero time at no cost from one plant to the other is a high abstraction from the real world. In reality, orders to be delivered obviously need to be grouped together as transportation is costly. Such an intermediate transportation planning will prolong the coordination and might further distort coordination results.

It should also be noted that the computation of the upstream planning solution does not match reality. Usually, an initial setting is bound to frame contracts and prenegotiated quantities and not achieved by calling different scheduling metaheuristics in a certain sequence.

Finally, it should be emphasized that the proposed coordination mechanism must not be regarded as a replacement for hierarchical coordination employing Master Planning. Essentially, Master Planning efficiently allocates quantities over a medium-term planning interval and then generates the orders for detailed scheduling. DEAL is only able to change the sequence of orders.

Concluding, we believe that DEAL is the right means for resolving short-term bottlenecks. As only limited time remains until execution, a readjustment of quantities is hardly possible in such a setting. Moreover, sufficient safety stock or production capacity should usually be available for most intermediate goods. Only some orders are critical and holding costs should be negligible for a short time period. We believe that in many real-world cases production plans provide enough flexibility for a rearrangement. However, due to the interdependency of production activities, a manual adjustment is a very tedious process. This is the point where DEAL can provide decision support in facilitating the rearrangement by an automated process. Obviously, this requires that all partners employ planning tools that support such automation.

Within the thesis, possibilities for further research have occasionally been highlighted. There are several details that could be improved. For example, a dynamic allocation of runtime—short local runtimes at the beginning of coordination to exchange many proposals and long runtimes at the end for a good exploitation—seems very promising. To introduce the concept into practice, it might also be advantageous to further simplify it. Instead of complex coordination processes, more simple strategies might already enhance today's successive planning dramatically. For example, it might already be sufficient if a manufacturer proposes several delivery sequences only once and the suppliers vote for the best one. Finally, an extension to Supply Chains with more than three tiers provides a challenging future topic.

Appendices

APPENDIX A

The simple Production Planning run algorithm

The following tables and algorithms define the procedure for deterministically generating test instances as discussed in Section 8.2.1.2.

g	index of production stage
s	index of SPPM
p	index of product
P	set of all products
I_s	set of input products for SPPM s
O_s	set of output products for SPPM s
d	index of demand
λ	index of replenishment elements
v	index of bucket (starting with 1)
V	set of all buckets

Table A.1: Additional indexes and index sets of the simple Production Planning algorithm.

$\mathcal{S}_g$	set of SPPMs on production stage g
g_{max}	maximum number of production stages
b_p	maximum lot size of product p
q	initial total demand
d_{pv}	demand of p in bucket v
$\overline{M}_s$	set of modes for PPM s
l_{sm}	length (duration) if SPPM is executed once in mode m

Table A.2: Additional data of the simple Production Planning algorithm.

$\overline{\Lambda}_d$	Associated replenishment elements to satisfy demand d
Λ_p	Nonassociated replenishment elements of product p
Λ	Set of all replenishment elements
s_j	SPPM of activity j
z_j	Numbers of executions of the SPPM relating to activity j
$\hat{\Lambda}_j$	Replenishment elements produced by j
$\hat{D}_j$	Demand elements needed by j
t_λ	Time after which replenishment element λ can be used
qd_d	(still uncovered) Quantity of demand d
q_λ	(still available) Quantity of replenishment λ
n_{ps}	Number of required / produced products if SPPM is run once
D_p	Set of demands for product p
td_d	Time of demand
p_d	Product of demand
j_d	Activity of demand d
j_λ	Activity of replenishment λ

Table A.3: Additional variables of the simple Production Planning algorithm.

Algorithm 9: simplePPrun(q, $\overline{T}$)

```
input  : total initial demand $q$
         length of planning horizon $\overline{T}$
$J := \Lambda := \emptyset$
forall products $p \in P$ do
    $\Lambda_p := \emptyset$
end
createDemands ($q, \overline{T}$))
for production stage $g = g_{max}$ to $0$ do
    forall SPPMs $s \in \mathcal{S}_g$ do
        createReplenishmentElements ($s$)
    end
    aggregateDemands ()
    createAdditionalPegging ()
end
```

Algorithm 10: createDemands($q, \overline{T}$)

```
input          : total initial demand $q$
                 length of planning horizon $\overline{T}$
local variables: temporal demand $x$ contained in lots with maximum size
                 temporal demand $y$ contained in other lots
d := 0
forall buckets $v \in V$ do
    forall products $p \in P$ do
        $D_p := \emptyset$
        $x := 0$
        for $i := 1$ to $\left\lfloor \frac{p_{pv}}{b_p} \right\rfloor$ do
            $d := d + 1$
            $qd_d := b_p$
            $td_d := \left\lfloor \overline{T} \cdot \frac{v}{|V|} \right\rfloor$
            $x := x + b_p$
        end
        $y := d_{pv} - x$
        if $y > 0$ then
            $d := d + 1$
            $qd_d := y$
            $td_d := \left\lfloor \overline{T} \cdot \frac{v}{|V|} \right\rfloor$
        end
        $D_p := D_p \cup \{d\}$
        $p_d := p$
    end
end
```

Algorithm 11: createReplenishmentElements(s)

```
input: $s$: selected SPPM
forall products $p \in O_s$ do
    forall demands $d \in D_p$ do
        $\overline{\Lambda}_d := \emptyset$
        if $qd_d > 0$ then
            $j := |J| + 1$
            $J := J \cup \{j\}$
            $\overline{\hat{\Lambda}}_j := \emptyset$
            $z_j := \min\left\{\min_{p' \in I_s}\left(\frac{b_{p'}}{n_{p's}}\right), \min_{p' \in O_s}\left(\frac{b_{p'}}{n_{p's}}\right), \left\lceil\frac{qd_d}{n_{ps}}\right\rceil\right\}$
            forall modes $m \in \overline{M}_s$ do
                $d_{jm} := l_{sm} \cdot z_j$
            end
            /* Setting start and finish dates already creates a crude
               schedule, not respecting capacity                        */
            $fd_j := td_d$
            $sd_j := fd_j - \min_{m \in M_j} (d_{jm})$
            forall products $p' \in O_s$ do
                $\lambda := |\Lambda| + 1$
                $t_\lambda := fd_j$
                if $p = p'$ then
                    $\overline{\Lambda}_d := \overline{\Lambda}_d \cup \{\lambda\}$
                    $qd_p := qd_p - z_a \cdot n_{p's}$
                else
                    $q_\lambda := z_a \cdot n_{p's}$
                    $\Lambda_{p'} := \Lambda_{p'} \cup \{\lambda\}$
                end
                $\Lambda := \Lambda \cup \{\lambda\}$
                $\hat{\Lambda}_j := \hat{\Lambda}_j \cup \{\lambda\}$
                $j_\lambda := j$
            end
            forall products $p' \in I_s$ do
                $d := \left(\sum_{p \in P} |D_p|\right) + 1$
                $p_d := p$
                $td_d = sd_j$
                $qd_d := z_j \cdot n_{p's}$
                $D_{p'} := D_{p'} \cup \{d\}$
                $\hat{D}_j := \hat{D}_j \cup \{d\}$
                $j_d := j$
            end
        end
    end
end
```

Algorithm 12: aggregateDemands

```
local variables: Temporary demand sets $D^*$ and $D^{**}$
forall products $p \in P$ do
    $D^* = D_p$
    Sort demand elements in $D_p$ according to ascending dates
    while $D^* \neq \emptyset$ do
        $d$ = earliest element of $D^*$
        $D^* = D^* \setminus \{d\}$
        $D^{**} = D^*$
        while $qd_d < b_p$ and $D^{**} \neq \emptyset$ do
            $d'$ = earliest element of $D^{**}$
            $D^{**} = D^{**} \setminus \{d'\}$
            if $qd_d + qd_{d'} \leq b_{p_d}$ and $td_d \leq td_{d'}$ then
                $qd_d := qd_d + qd_{d'}$
                $\hat{D}_{j_{d'}} := \hat{D}_{j_{d'}} \setminus \{d'\}$
                $D_p := D_p \setminus \{d'\}$
                $\hat{D}_{j_{d'}} := \hat{D}_{j_{d'}} \cup \{d\}$
            end
        end
    end
end
```

Algorithm 13: createAdditionalPegging

```
local variables: Temporary sets $D^*$ and $\Lambda^{**}$
                 Temporary quantity $x$
Sort demand elements according to ascending dates
Sort replenishment elements according to ascending dates
forall products $p \in P$ do
    $\Lambda^* := \Lambda_p$
    $D^* = D_p$
    while $D^* \neq \emptyset$ and $\Lambda^* \neq \emptyset$ do
        $d$ = earliest element of $D^*$
        $D^* = D^* \setminus \{d\}$
        while $qd_d > 0$ and $\Lambda^* \neq \emptyset$ do
            $\lambda$ = earliest element of $\Lambda^*$
            $\Lambda^* := \Lambda^* \setminus \{\lambda\}$
            if $t_\lambda \leq td_d$ then
                $x := \min\{qd_d, q_\lambda\}$
                $q_\lambda = q_\lambda - x$
                $qd_d = qd_d - x$
                $\overline{\Lambda}_d := \overline{\Lambda}_d \cup \{\lambda\}$
            end
        end
    end
end
```

Algorithm 14: createRCPSPInstance

```
local variables: Temporary set of activities $J^*$
create resources, calendars and setup matrices according to master data
$J_S := \emptyset$
$J^* := J$
forall activities $j \in J^*$ do
    forall modes $m \in M_j$ do
        create primary resource utilization $u_{jmr}$ according to master data
    end
    $S_j := P_j := \emptyset$
    forall $d \in \hat{D}_j$ do
        forall $\lambda \in \overline{\Lambda}_d$ do
            $k := j_\lambda$
            $S_j := S_j \cup \{\}$
            $P_k := P_k \cup \{j\}$
        end
    end
    if related SPPM $s_j$ has setup activity then
        create new setup activity $k$
        $J_S := J_S \cup \{k\}$
        $J := J \cup \{k\}$
        $S_k := \{j\}$
        $P_j := P_j \cup \{k\}$
    end
end
```

APPENDIX B

The OEM's control procedure

The algorithms presented in the following define the OEM's coordination logic in the customized DEAL framework, cf. Section 7.10.

Algorithm 15: ControlProcedureOfOEM

```
ProcessIncomingMessagesOfOEM ()
PorcessIdleIndividualsOfOEM ()
if Number of waiting and idle individuals is below queuing size then
    if Mating pool does not contain enough parents to construct desired number of children then
        if Evaluation procedure finished then
            if Update procedure not started yet then
                Determine deletable individuals
                Send appropriate list of pertaining communication threads to each supplier
            end
        else
            if Evaluation procedure not started yet then
                Send request_rank message to each supplier
            end
        end
    end
    if Mating pool not empty then
        Take oldest individual of mating pool
        Construct empty individual as new child and set state to idle
    end
end
```

Algorithm 16: ProcessIncomingMessagesOfOEM

```
if Incoming start message from controller then
    Set up empty initial individual
    if Parallel mode then
        if Free solving unit available then
            Send solving_request to solving unit
            Set individual state to waiting
            Mark solving unit as busy
        end
    else
        Call DS Solving module
        Set individual state to idle
    end
end
if Termination criterion fulfilled then
    if Final solution not broadcasted yet then
        Select and broadcast final solution
    end
    if No further idle individuals then
        Send message with terminated to controller
        Terminate
    end
end
forall Incoming messages from solving units do
    Attach to related communication thread
    Store solution in individual
    Delete communication thread
    Mark solving unit as available
    Set individual state to idle
end
forall Incoming messages from suppliers do
    Update communication thread
    if Command: confirm_update then
        Delete related communication threads
        Delete all individuals that have no threads left and whose children have been constructed
        Set state of update procedure to finished if all suppliers have responded
    else
        if Evaluation message then
            Map ranking of threads to individuals
            Set state of evaluation procedure to finished if all suppliers have responded
            Select new parents to mating pool until queuing size has been reached
        else
            Update thread
            If counterproposals of all suppliers have arrived, set individual state to idle
        end
    end
end
```

Algorithm 17: ProcessIdleIndividualsOfOEM

```
if Population contains idle individuals then
    Select oldest idle individual
    if Solution available then
        if Solution pertaining to counterproposal then
            Set state to complete
            Send check_feasibility message containing solution to controller
        else
            Finish construction of proposed dates
            if Proposed dates are redundant then
                Delete solution and reinitialize individual as empty child
            else
                Construct proposal for each supplier with appropriate proposed dates
                forall Suppliers do
                    if Proposal is redundant then
                        Discard new proposal and add child to existing thread with similar
                        proposal
                    end
                    Connect child to new proposal
                    Set state to waiting
                end
            end
        end
    else
        if Counterproposals have arrived then
            if Parallel mode then
                if Free solving unit available then
                    Send solving_request to solving unit
                    Set individual state to waiting
                    Mark solving unit as busy
                end
            else
                Call DS Solving module
            end
        else
            Select operator to construct new proposed dates
            Analyze parent individual
            if Operator requires intermediate solution then
                if Parallel mode then
                    if Free solving unit available then
                        Send solving_request to solving unit
                        Set individual state to waiting
                        Mark solving unit as busy
                    end
                else
                    Call DS Solving module
                end
            end
        end
    end
end
```

APPENDIX C

The supplier's control procedure

The algorithms presented in the following define a supplier's coordination logic in the customized DEAL framework, cf. Section 7.10.

Algorithm 18: ControlProcedureOfSupplier

```
if Termination requested then
    if No further idle individuals then
        Send message with command terminated to controller
        Terminate
    end
end
forall Incoming messages from solving units do
    Update communication thread
    Store solution in individual
    Delete communication thread
    Mark solving unit as available
    Set individual state to idle
end
forall Incoming messages from OEM do
    if Update request then
        Delete designated communication threads
        Delete all individuals that have no threads left and whose children have been constructed
        Send reply message with command: confirm_update
    else
        if Final solution then
            Implement final solution
            Set state to terminated requested
        else
            if Evaluation request then
                Rank all individuals
                Map ranking to communication threads
                Send reply message with ranking
            else
                if Redundant proposal then
                    Connect thread to existing individual
                    Reply according to existing counterproposal
                else
                    Construct new individual
                    Connect individual to thread
                    Set individual state to idle
                end
            end
        end
    end
end
ProcessIdleIndividualsOfSupplier()
```

Algorithm 19: ProcessIdleIndividualsOfSupplier

```
if Population contains idle individuals then
    Select oldest idle individual
    if Proposal received then
        if Parallel mode then
            if Free solving unit available then
                Send solving_request to solving unit
                Set individual state to waiting
                Mark solving unit as busy
            end
        else
            Call DS Solving module
        end
    else
        Send downstream-related finish dates of individual as counterproposal to OEM
        Set individual state to complete
    end
end
```

APPENDIX D

The controller's control procedure

The algorithm presented in the following defines the message exchange and debug functionality in the customized DEAL framework, cf. Section 7.10.

Algorithm 20: ControlProcedureOfController

```
if Coordination not started yet then
    Send message with start command to OEM;
else
    forall Incoming messages from planning domains do
        if Domain terminated then
            if All domains termintated then
                if Parallel mode then
                    Stop coordination mode and terminate all processes;
                else
                    Terminate process;
                end
            end
        else
            Store feasibility_check message in pool;
            if Pool of stored feasibility_checks contains messages belonging to one interor-
            ganizational solution then
                Combine partial solutions to central solution;
                Check feasibility of central solution;
                Remove related messages from pool;
            end
        end
    end
end
```

APPENDIX E

Operator list for the deterministic self-adaption

Table E.1 defines the operator list used for the deterministic self-adaption of operators during the computational evaluation, cf. Chapter 8.

Initial		
position	Operator name	α
1	FS | PDD	n.a.
2	FDS | PDD	1.0
3	PLX	n.a.
4	LX	n.a.
5	CX	n.a.
6	RRD | LA | RA	n.a.
7	RNB | LA | RA	0.75
8	RMP | LA | RA	0.75
9	RMD | LA | RA	1.0
10	FS | RMD | LA | RA	1.0
11	RC | FWD | RA	n.a.
12	ICR | FWD | RA	n.a.
13	FS | ICR | FWD | RA	n.a.
14	ICB | FWD | RA	1.0
15	FS | ICB | FWD | RA	1.0
16	SSB | FWD | RA	1.0
17	IDD | FWD | RA	1.0
18	PLX | FWD | LA | RA	n.a.
19	LX | FWD | LA | RA	n.a.
20	CX | FWD | LA | RA	n.a.
21	FDS | PDD	0.75
22	RMS | LA | RA	0.75
23	RNB | LA | RA	0.5
24	RMP | LA | RA	0.5
25	RMD | LA | RA	0.75
26	FS | RMD | LA | RA	0.75
27	ICB | FWD | RA	0.75
28	FS | ICB | FWD | RA	0.75
29	SSB | FWD | RA	0.75
30	IDD | FWD | RA	0.75
31	FDS | PDD	0.5
32	RMS | LA | RA	0.5
33	RNB | LA | RA	0.25
34	RMP | LA | RA	0.25
35	RMD | LA | RA	0.5
36	FS | RMD | LA | RA	0.5
37	ICB | FWD | RA	0.5
38	FS | ICB | FWD | RA	0.5
39	SSB | FWD | RA	0.5
40	IDD | FWD | RA	0.25
41	FDS | PDD	0.25
42	RMS | LA | RA	0.25
43	RMD | LA | RA	0.25
44	FS | RMD | LA | RA	0.25
45	ICB | FWD | RA	0.25
46	FS | ICB | FWD | RA	0.25
47	SSB | FWD | RA	0.25

Table E.1: Standard operator list for the deterministic self-adaption.

APPENDIX F

Additional figures

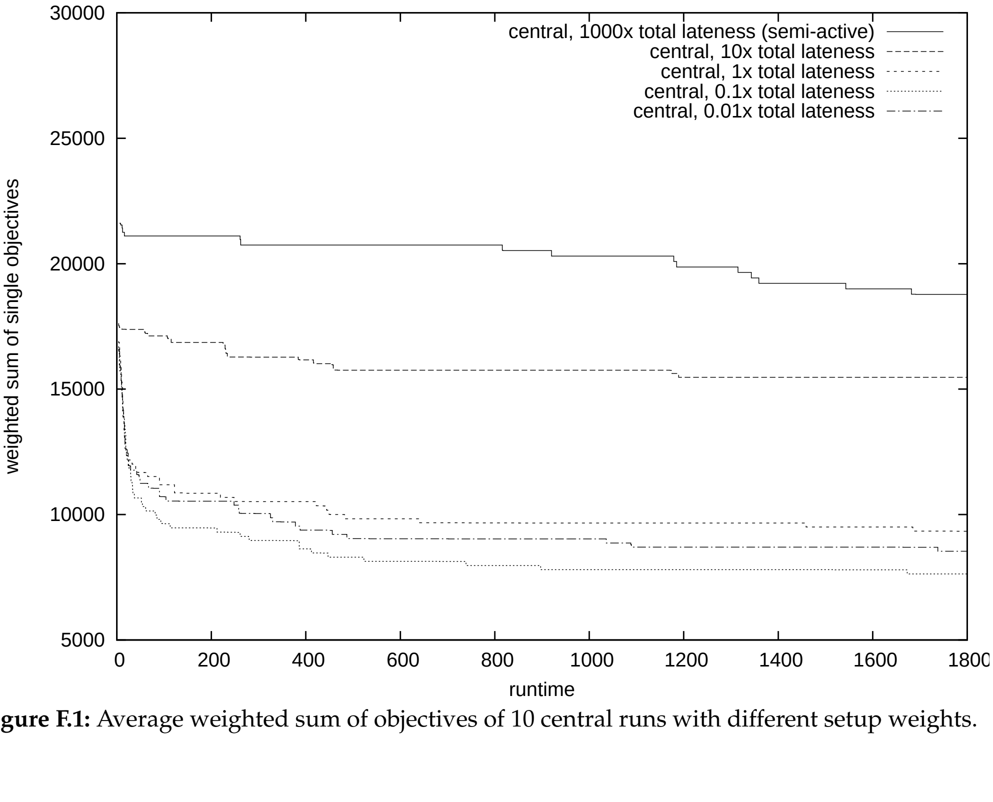


Figure F.1: Average weighted sum of objectives of 10 central runs with different setup weights.

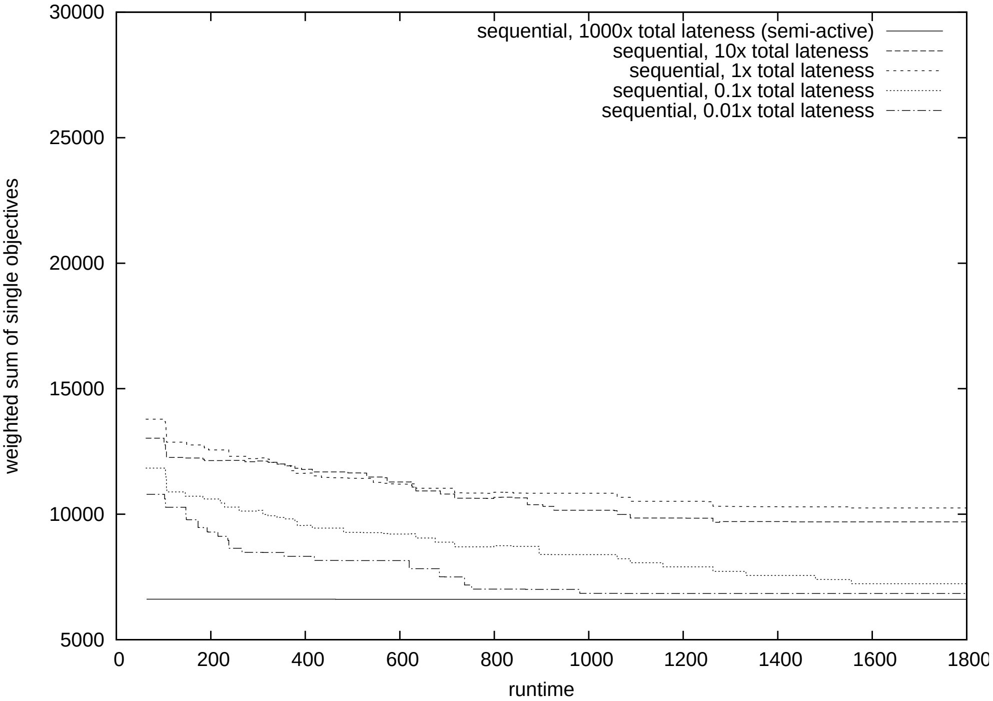


Figure F.2: Average weighted sum of objectives of 10 sequential runs with different setup weights, 13s local runtime, deterministic self-adaption.

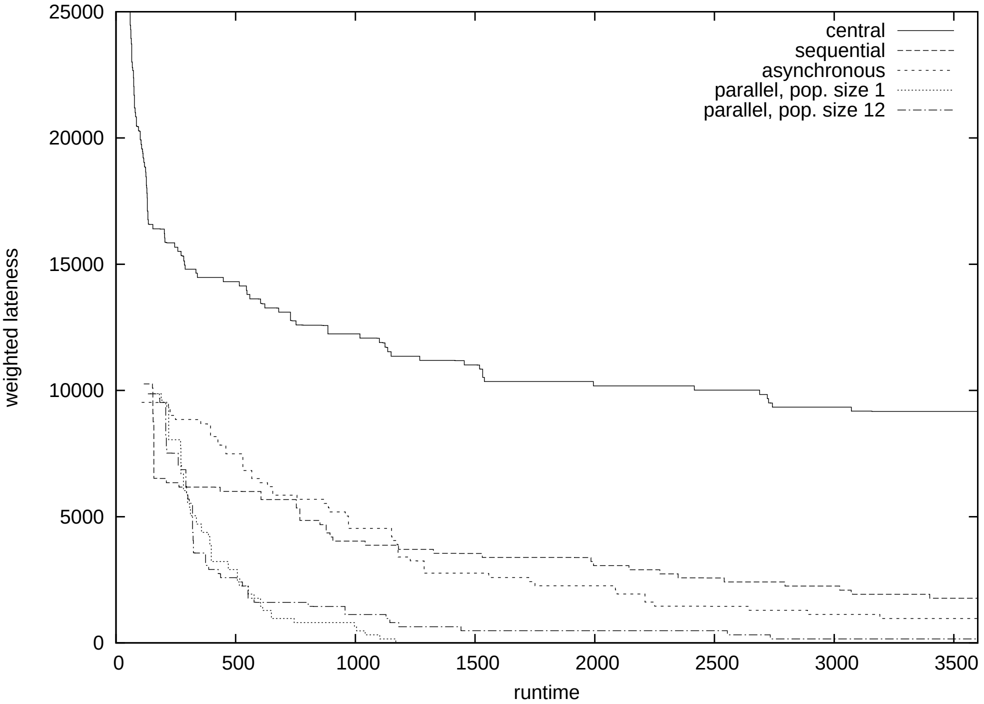


Figure F.3: Comparison of weighted lateness for parallel and sequential coordination and central run for the medium instance of setup subscenario, two planning domains, 25s local runtime, deterministic self-adaption.

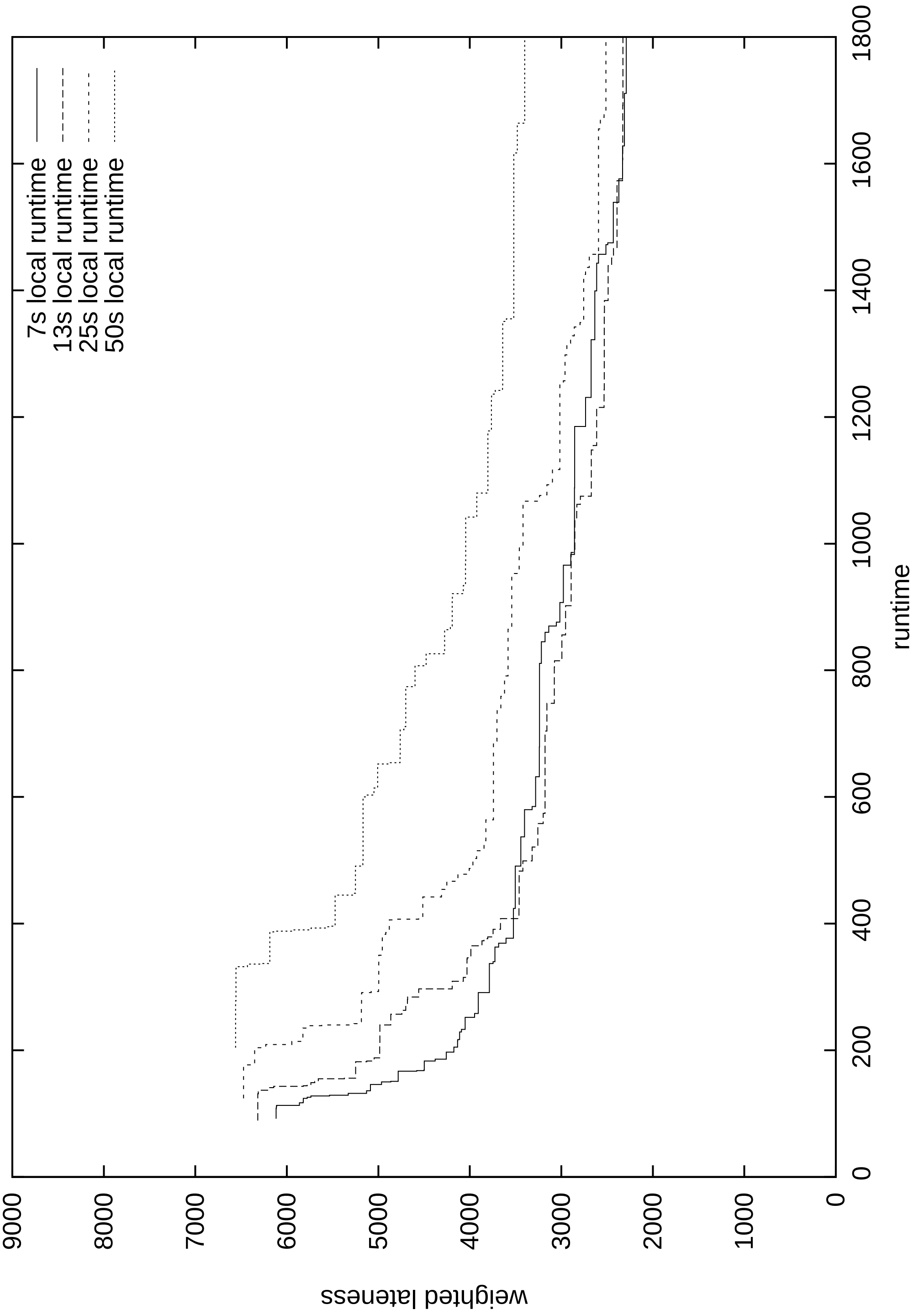


Figure F.4: Comparison of weighted lateness for different local runtimes for the parallel coordination with population size 12 for the small instance of the product mix subscenario.

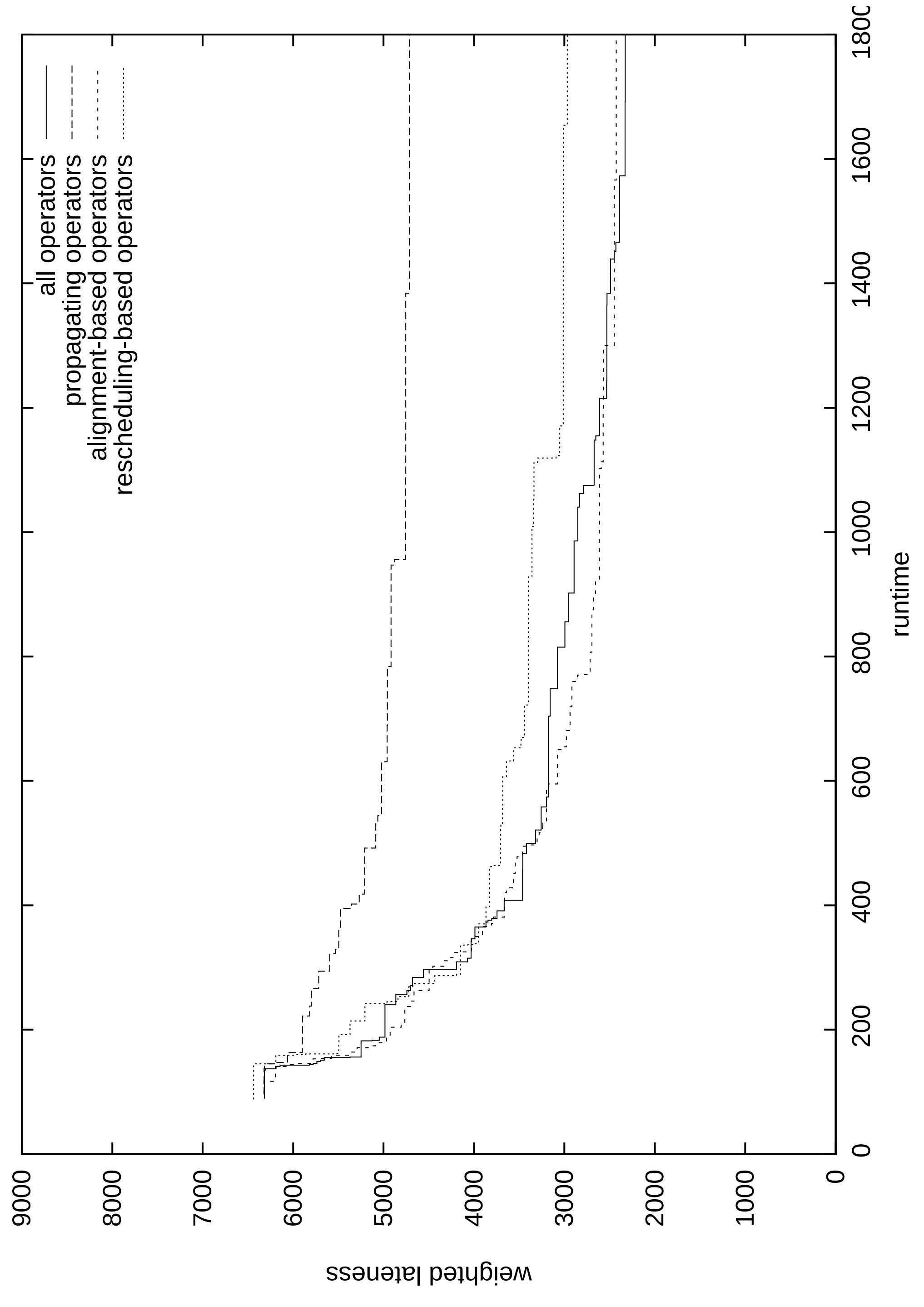


Figure F.5: Comparison of weighted lateness for different operators for the parallel coordination with population size 12 for the small instance of the product mix subscenario.

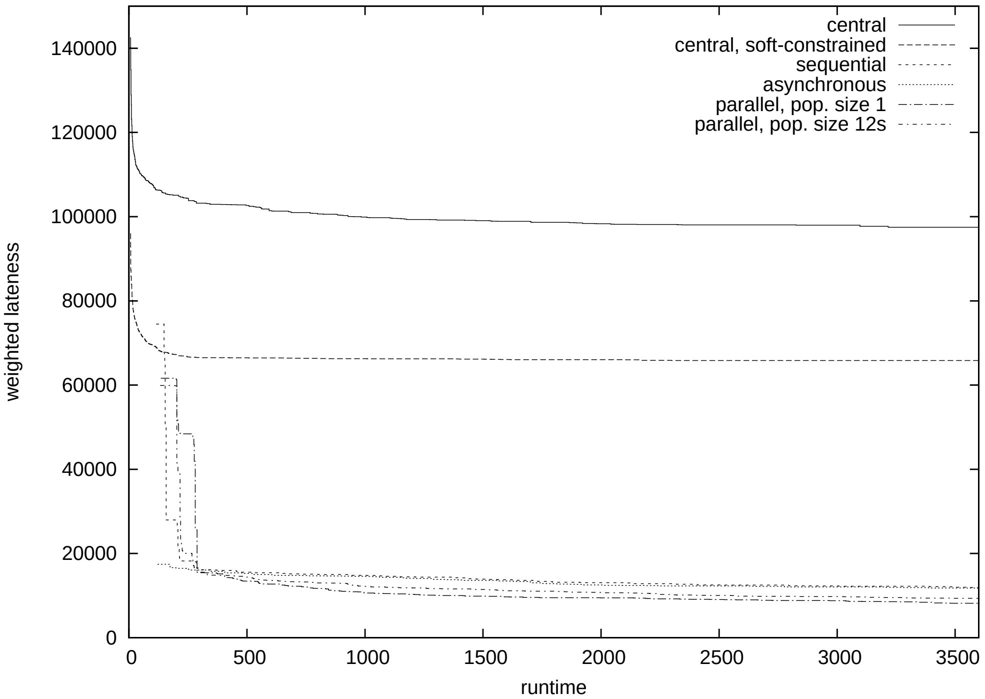


Figure F.6: Comparison of weighted lateness for parallel and sequential coordination and central run for the medium instance of the product mix subscenario, two-partner case.

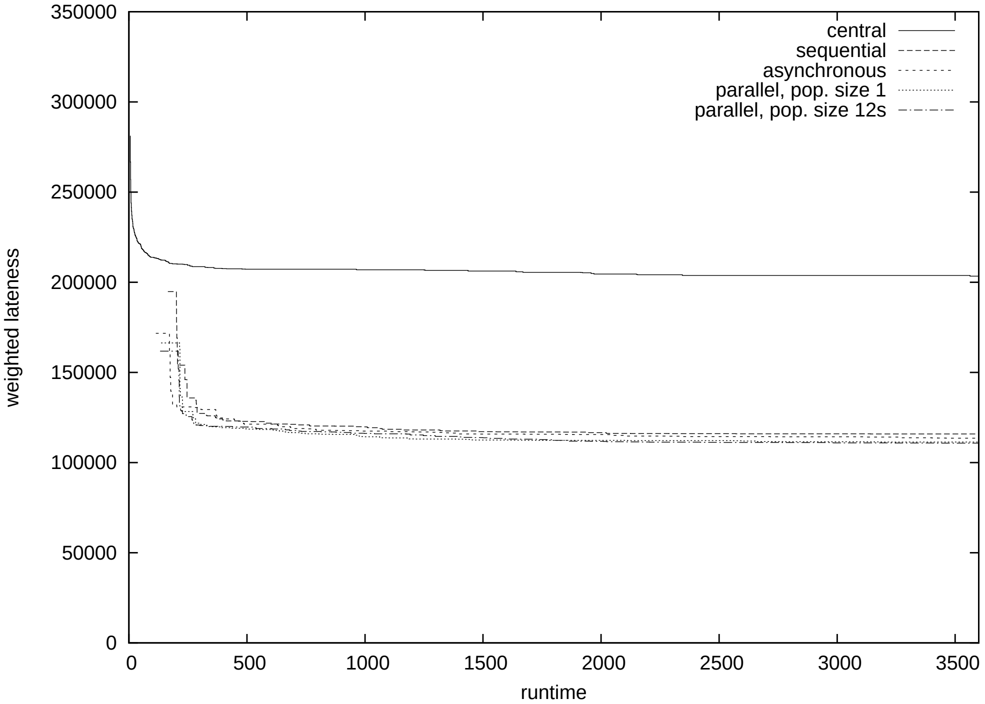


Figure F.7: Comparison of weighted lateness for parallel and sequential coordination and central run for the medium instance of the product mix subscenario, including transport disruptions.

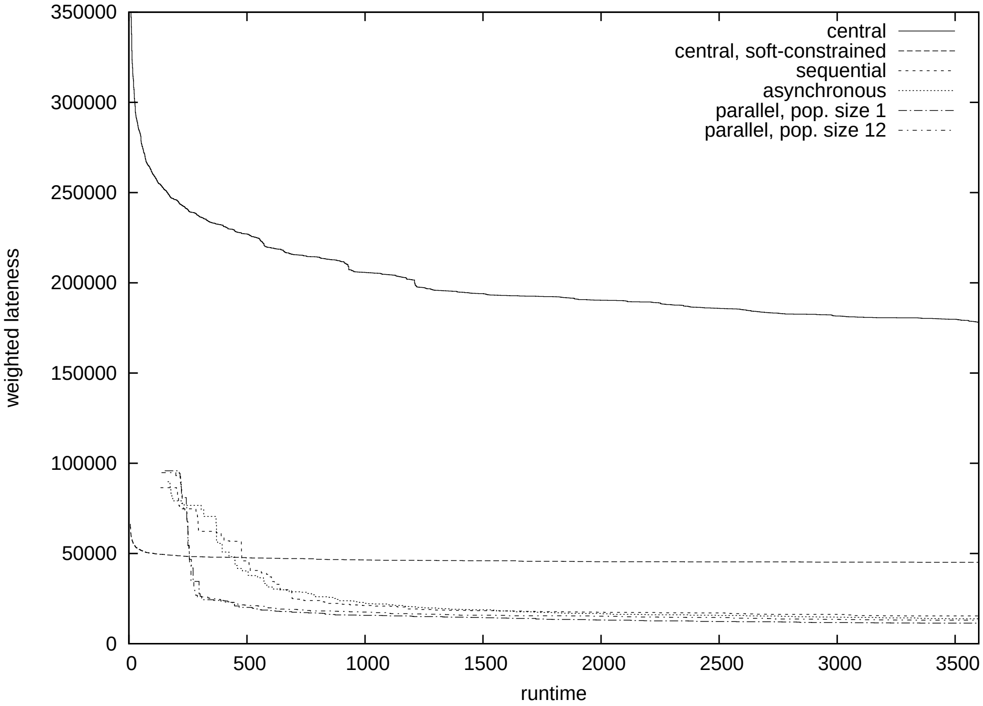


Figure F.8: Comparison of weighted lateness for parallel and sequential coordination and central run for the medium instance with fine-grained demand of the product mix subscenario.

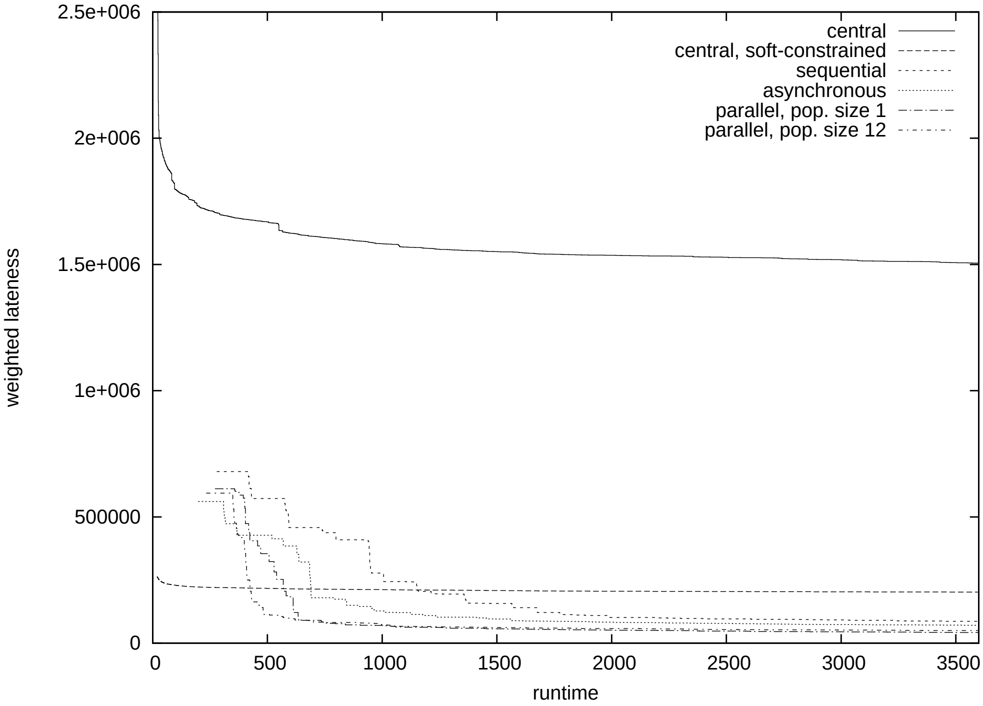


Figure F.9: Comparison of weighted lateness for parallel and sequential coordination and central run for the large instance of the product mix subscenario.

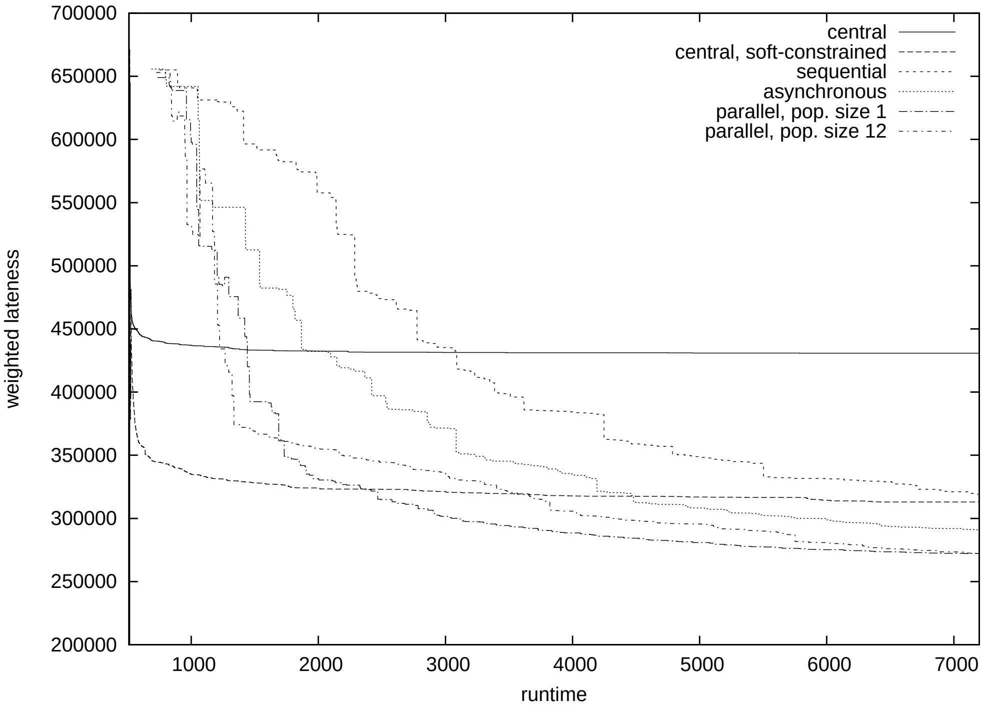


Figure F.10: Comparison of weighted lateness for parallel and sequential coordination and central runs for the large instance of the breakdown subscenario.

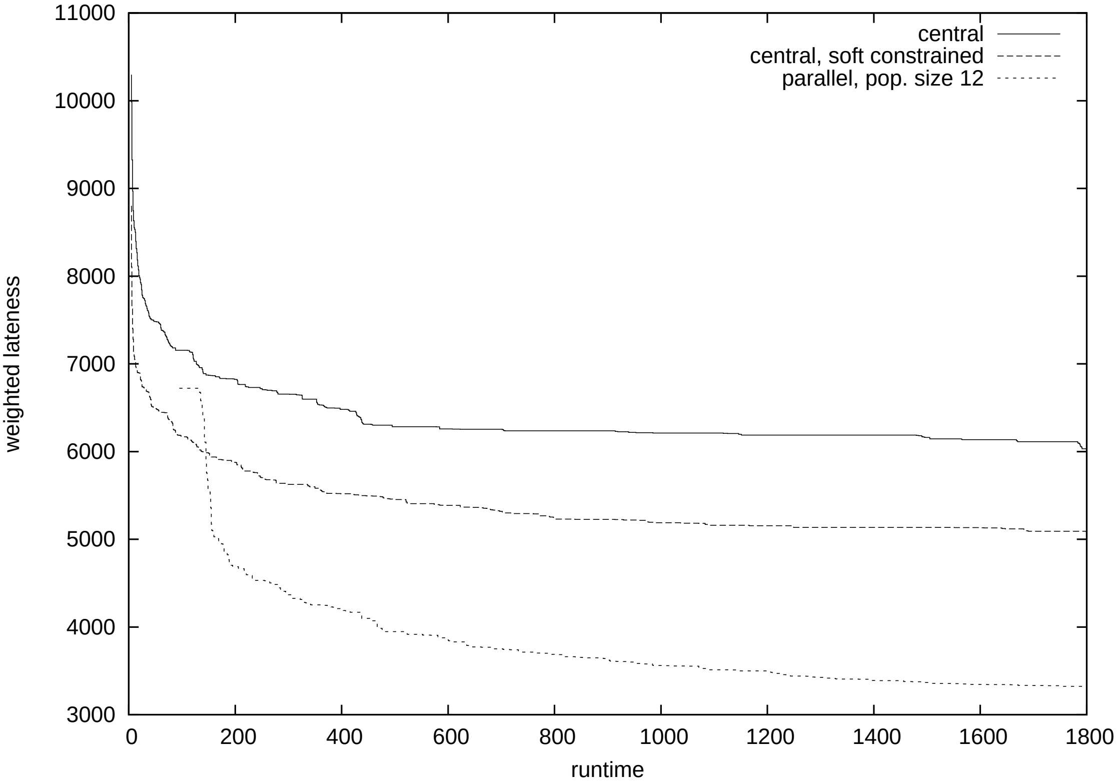


Figure F.11: Comparison of weighted lateness of parallel coordination and central runs for the small randomly generated instance with $\delta^{\Omega}_{\text{supplier}} = 0.3$.

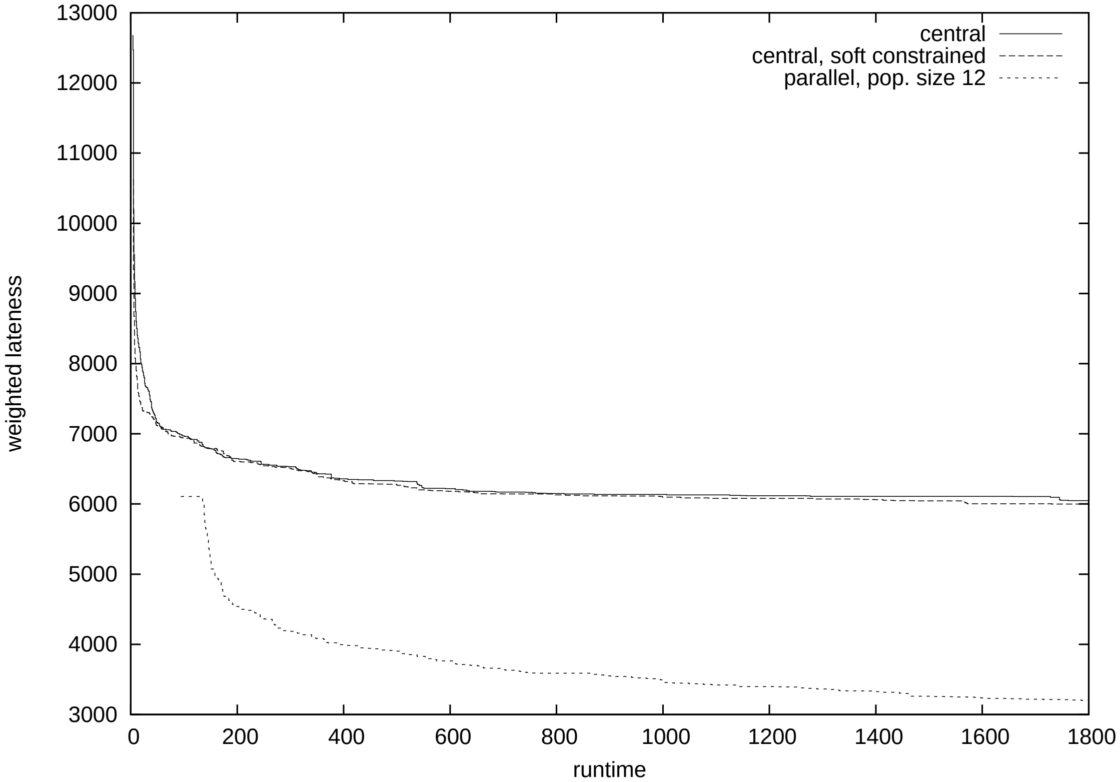


Figure F.12: Comparison of weighted lateness of parallel coordination and central runs for the small randomly generated instance with $\delta^{\Omega}_{\text{supplier}} = 0.7$.

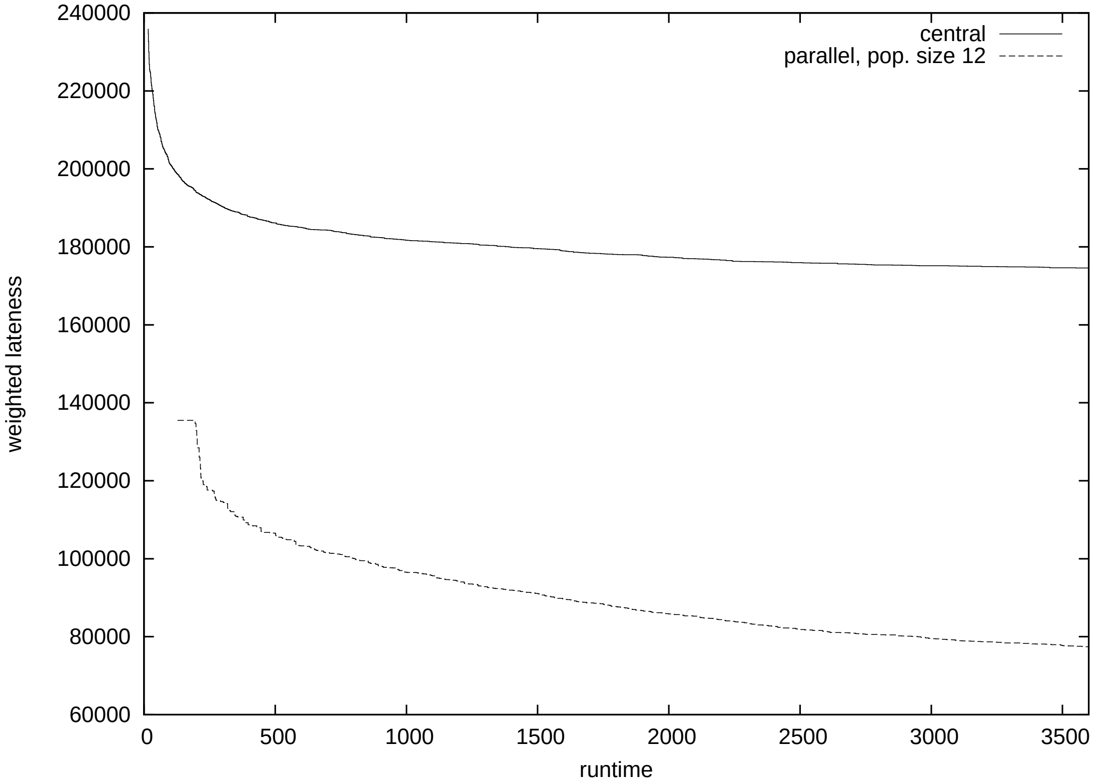


Figure F.13: Comparison of weighted lateness of parallel coordination and central runs for the medium randomly generated instance with $\delta^{\Omega}_{\text{supplier}} = 0.0$.

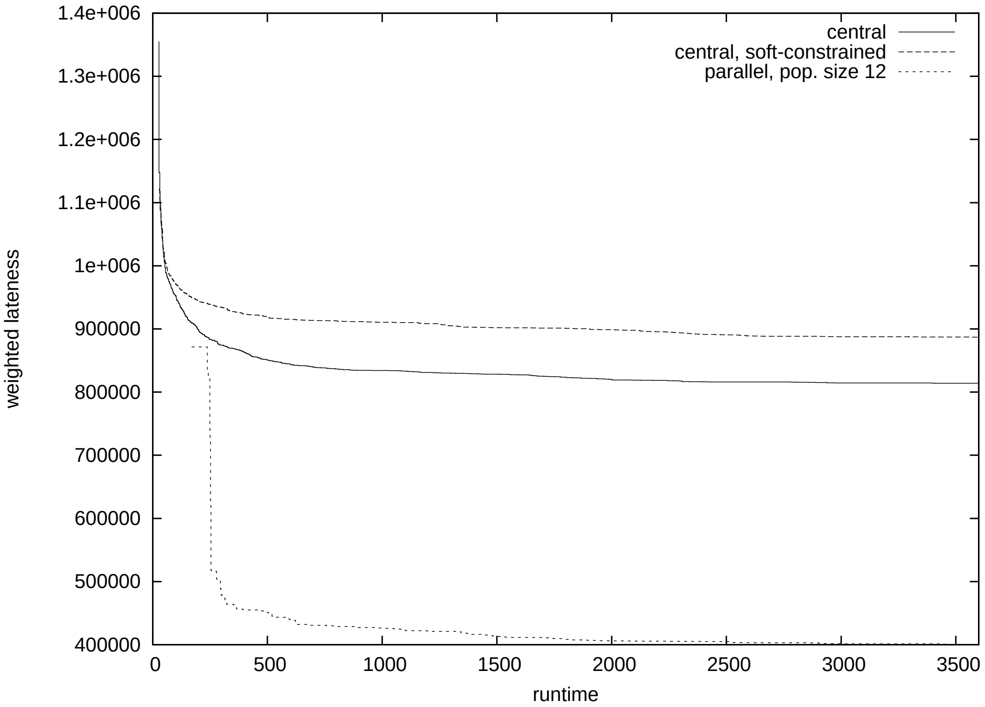


Figure F.14: Comparison of weighted lateness for parallel and central runs for the first additional medium instance of the complete scenario .

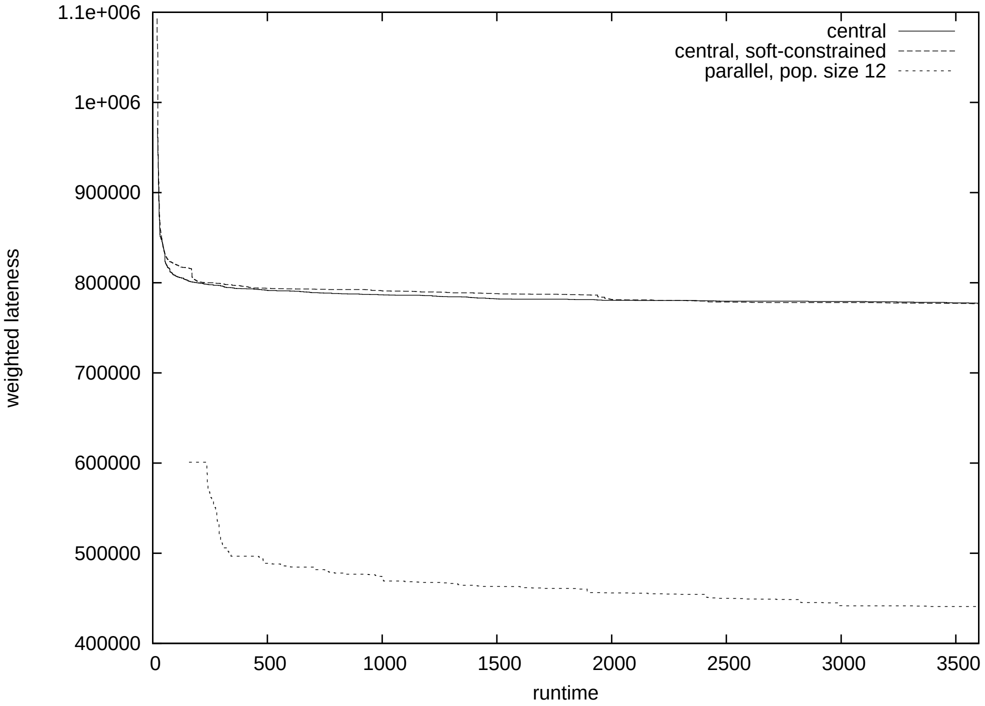


Figure F.15: Comparison of weighted lateness for parallel and central runs for the second additional medium instance of the complete scenario .

APPENDIX G

Additional tables

Item	Demand (in lots of 20 items)						
sTemp_1	11	12	12	12	0	0	0
mTemp_1	11	12	12	12	0	0	0
lTemp_1	11	12	12	12	0	0	0
xTemp_1	11	12	12	12	0	0	0
sBladeExt	24	24	24	24	0	0	0
lBladeExt	24	24	24	24	0	0	0
extWheel	24	24	24	24	0	0	0
extTank_2	6	6	6	6	0	0	0

Table G.1: Weekly demand matrix for the medium instance of the of the product mix subscenario with fine-grained demand.

item	demand (in lots of 20 items)													
sTemp_1	0	0	0	47	0	0	0	48	0	0	0	48	0	0
mTemp_1	0	0	0	47	0	0	0	48	0	0	0	48	0	0
lTemp_1	0	0	0	47	0	0	0	48	0	0	0	48	0	0
xTemp_1	0	0	0	47	0	0	0	48	0	0	0	48	0	0
sBladeExt	0	0	0	96	0	0	0	96	0	0	0	96	0	0
lBladeExt	0	0	0	96	0	0	0	96	0	0	0	96	0	0
extWheel	0	0	0	96	0	0	0	96	0	0	0	96	0	0
extTank_2	0	0	0	24	0	0	0	24	0	0	0	24	0	0

Table G.2: Biweekly demand matrix for the large instance of the product mix subscenario.

item	demand (in lots of 20 items)														
sTemp_1	0	0	0	0	0	0 (72)	0	0	0	0	144 (72)	0	0	0	0
mTemp_1	0	0	0	0	0	0	0	0	0	0	144 (72)	0	0	0 (72)	0
lTemp_1	0	0	0	0	0	0	0	0	0	0	144	0	0	0	0
xTemp_1	0	0	0	0	0	0	0	0	0	0	144	0	0	0	0
sBladeExt	0	0	0	0	0	0	0	0	0	0	288 (0)	0	0	0	0 (288)
lBladeExt	0	0	0	0	0	288	0	0	0	0	0	0	0	0	0
extWheel	0	0	0	0	0	288	0	0	0	0	0	0	0	0	0

Table G.3: 15-day demand matrix for the first large instance of the breakdown subscenario. Values in brackets are the renegotiated demands after the breakdown occurred.

item	demand (in lots of 20 items)													
sTemp_1	0	0 (24)	0	48 (24)	0	0 (24)	0	48 (24)	0	0 (24)	0	48 (24)	0	0
mTemp_1	0	0	0	48 (24)	0 (24)	0	0	48 (24)	0 (24)	0	0	48 (24)	0 (24)	0
lTemp_1	0	0	0	48	0	0	0	48	0	0	0	48	0	0
xTemp_1	0	0	0	48	0	0	0	48	0	0	0	8	0	0
sBladeExt	0	0	0	96 (0)	0 (96)	0	0	96 (0)	0 (96)	0	0	96 (0)	0 (96)	0
lBladeExt	0	96 (0)	0	0	0 (96)	96 (0)	0	0	0 (96)	96 (0)	0	0	0 (96)	0
extWheel	0	96	0	0	0	96	0	0	0	96	0	0	0	0

Table G.4: 14-day demand matrix for the second large instance of the breakdown subscenario. Values in brackets are the renegotiated demands after the breakdown occurred.

			Second priority criteria		External lateness		
	Domain	Statistic	Makespan	Setup time	Max	Total	Weighted
	Central	min	340,801	32,000	0	0	0
		avg	**353,761**	**39,680**	**8,800**	**45,920**	**9,168**
		max	368,001	46,400	22,400	179,200	23,952
Parallel, pop. size 12	OEM	min	344,001 (−2%)	14,400 (−9%)	0 (−100%)	0 (−100%)	0 (−100%)
		avg	**345,281 (−1%)**	**20,000 (+26%)**	**0 (−100%)**	**0 (−100%)**	**0 (−100%)**
		max	345,601 (−1%)	27,200 (+73%)	0 (−100%)	0 (−100%)	0 (−100%)
	Supplier Steel	min	310,401 (−0%)	6,400 (+1%)	n.a.	n.a.	n.a.
		avg	**320,321 (±0%)**	**16,640 (+2%)**	**n.a.**	**n.a.**	**n.a.**
		max	339,201 (+6%)	38,400 (+135%)	n.a.	n.a.	n.a.
	Supplier Motor	min	310,401 (−7%)	6,400 (−79%)	n.a.	n.a.	n.a.
		avg	**329,601 (−1%)**	**26,560 (−12%)**	**n.a.**	**n.a.**	**n.a.**
		max	339,201 (+1%)	38,400 (+26%)	n.a.	n.a.	n.a.
Parallel, pop. size 1	OEM	min	344,001 (−2%)	12,800 (−26%)	0 (−100%)	0 (−100%)	0 (−100%)
		avg	**345,391 (−1%)**	**16,160 (−6%)**	**0 (−100%)**	**0 (−100%)**	**0 (−100%)**
		max	345,601 (−1%)	20,800 (+20%)	0 (−100%)	0 (−100%)	0 (−100%)
	Supplier Steel	min	310,401 (−3%)	6,400 (−61%)	n.a.	n.a.	n.a.
		avg	**319,041 (±0%)**	**15,040 (−8%)**	**n.a.**	**n.a.**	**n.a.**
		max	339,201 (+6%)	35,200 (+116%)	n.a.	n.a.	n.a.
	Supplier Motor	min	310,401 (−7%)	6,400 (−79%)	n.a.	n.a.	n.a.
		avg	**330,561 (−1%)**	**26,560 (−14%)**	**n.a.**	**n.a.**	**n.a.**
		max	342,401 (−2%)	38,400 (+25%)	n.a.	n.a.	n.a.
Asynchronous	OEM	min	344,001 (−2%)	11,200 (−31%)	0 (−100%)	0 (−100%)	0 (−100%)
		avg	**345,921 (−1%)**	**14,800 (−9%)**	**480 (−89%)**	**800 (−92%)**	**483 (−89%)**
		max	348,801 (±0%)	20,800 (+27%)	3,200 (−29%)	6,400 (−37%)	3,232 (−29%)
	Supplier Steel	min	310,401 (−3%)	6,400 (−61%)	n.a.	n.a.	n.a.
		avg	**319,361 (±0%)**	**16,000 (−2%)**	**n.a.**	**n.a.**	**n.a.**
		max	339,201 (+6%)	35,200 (+116%)	n.a.	n.a.	n.a.
	Supplier Motor	min	310,401 (−7%)	6,400 (−79%)	n.a.	n.a.	n.a.
		avg	**331,841 (−1%)**	**28,480 (−6%)**	**n.a.**	**n.a.**	**n.a.**
		max	342,401 (+2%)	38,400 (+26%)	n.a.	n.a.	n.a.
Sequential	OEM	min	340,801 (−3%)	11,200 (−33%)	0 (−100%)	0 (−100%)	0 (−100%)
		avg	**344,961 (−2%)**	**21,040 (+26%)**	**160 (−97%)**	**160 (−99%)**	**160 (−97%)**
		max	347,201 (−1%)	46,400 (+179%)	1,600 (−74%)	1,600 (−91%)	1,600 (−74%)
	Supplier Steel	min	310,401 (−4%)	6,400 (−66%)	n.a.	n.a.	n.a.
		avg	**324,801 (+1%)**	**21,440 (+16%)**	**n.a.**	**n.a.**	**n.a.**
		max	339,201 (−5%)	35,200 (+90%)	n.a.	n.a.	n.a.
	Supplier Motor	min	310,401 (−7%)	6,400 (−79%)	n.a.	n.a.	n.a.
		avg	**327,041 (−2%)**	**23,680 (−22%)**	**n.a.**	**n.a.**	**n.a.**
		max	342,401 (+2%)	38,400 (+26%)	n.a.	n.a.	n.a.

Table G.5: Comparison of average (avg), minimum (min) and maximum (max) values of 10 runs of parallel, asynchronous and sequential coordination with 10 centrally computed runs after 3,600s, local run time 25s, full set of operators, medium instance of the setup subscenario. The numbers in parentheses denote the relative difference from the average upstream planning solution.

			Second priority criteria		External lateness		
	Domain	Statistic	Makespan	Setup time	Max	Total	Weighted
	Central	min	1,017,601	112,000	0	0	0
		avg	**1,022,741**	**130,800**	**21,300**	**266,180**	**23,725**
		max	1,032,201	142,400	44,800	723,200	51,517
Parallel, pop. size 12	OEM	min	1,014,400 (−2%)	39,200 (−46%)	0 (−100%)	0 (−100%)	0 (−100%)
		avg	**102,760 (−1%)**	**58,080 (−20%)**	**480 (−97%)**	**640 (−99%)**	**482 (−97%)**
		max	1,036,800 (±0%)	94,400 (+31%)	3,200 (−77%)	3,200 (−97%)	3,200 (−78%)
	Supplier Steel	min	995,201 (−2%)	102,400 (−11%)	n.a.	n.a.	n.a.
		avg	**1,009,920 (−1%)**	**112,960 (−2%)**	**n.a.**	**n.a.**	**n.a.**
		max	1,020,800 (±0%)	134,400 (+17%)	n.a.	n.a.	n.a.
	Supplier Motor	min	940,801 (−3%)	35,200 (−44%)	n.a.	n.a.	n.a.
		avg	**971,841 (±0%)**	**72,320 (+15%)**	**n.a.**	**n.a.**	**n.a.**
		max	1,014,400 (+5%)	124,800 (+99%)	n.a.	n.a.	n.a.
Parallel, pop. size 1	OEM	min	1,016,300 (−2%)	42,400 (−45%)	0 (−100%)	0 (−100%)	0 (−100%)
		avg	**1,027,280 (±0%)**	**60,000 (−22%)**	**480 (−97%)**	**880 (−99%)**	**483 (−97%)**
		max	1,035,200 (±0%)	78,400 (+2%)	3,200 (−77%)	6,400 (−93%)	3,232 (−78%)
	Supplier Steel	min	988,801 (−3%)	89,600 (−22%)	n.a.	n.a.	n.a.
		avg	**1,009,920 (−1%)**	**115,200 (±0%)**	**n.a.**	**n.a.**	**n.a.**
		max	1,020,800 (±0%)	140,800 (+23%)	n.a.	n.a.	n.a.
	Supplier Motor	min	937,601 (−3%)	28,800 (−54%)	n.a.	n.a.	n.a.
		avg	**970,240 (±0%)**	**76,800 (+22%)**	**n.a.**	**n.a.**	**n.a.**
		max	1,004,800 (+4%)	137,600 (+119%)	n.a.	n.a.	n.a.
Asynchronous	OEM	min	1,010,900 (−2%)	36,800 (−55%)	0 (−100%)	0 (−100%)	0 (−100%)
		avg	**1,030,530 (±0%)**	**63,040 (−24%)**	**2,880 (−80%)**	**7,840 (−91%)**	**2,929 (−80%)**
		max	1,036,800 (±0%)	89,600 (+9%)	8,000 (−42%)	28,800 (−68%)	8,206 (−6%)
	Supplier Steel	min	982,401 (−3%)	76,800 (−49%)	n.a.	n.a.	n.a.
		avg	**1,013,120 (±0%)**	**115,840 (+2%)**	**n.a.**	**n.a.**	**n.a.**
		max	1,027,200 (+1%)	134,400 (+18%)	n.a.	n.a.	n.a.
	Supplier Motor	min	937,601 (−3%)	32,000 (−49%)	n.a.	n.a.	n.a.
		avg	**965,761 (±0%)**	**65,600 (+5%)**	**n.a.**	**n.a.**	**n.a.**
		max	998,401 (+3%)	99,200 (+58%)	n.a.	n.a.	n.a.
Sequential	OEM	min	1,017,601 (−2%)	40,000 (−46%)	0 (−100%)	0 (−100%)	0 (−100%)
		avg	**1,031,681 (−0%)**	**67,200 (−9%)**	**3,040 (−78%)**	**7,680 (−91%)**	**3,086 (−78%)**
		max	1,036,801 (±0%)	100,800 (+36%)	11,200 (−18%)	36,800 (−58%)	11,453 (−20%)
	Supplier Steel	min	982,401 (−3%)	76,800 (−34%)	n.a.	n.a.	n.a.
		avg	**1,014,081 (±0%)**	**115,520 (±0%)**	**n.a.**	**n.a.**	**n.a.**
		max	1,033,601 (+2%)	153,600 (+32%)	n.a.	n.a.	n.a.
	Supplier Motor	min	944,001 (−2%)	38,400 (−40%)	n.a.	n.a.	n.a.
		avg	**963,841 (±0%)**	**60,800 (−5%)**	**n.a.**	**n.a.**	**n.a.**
		max	992,001 (+3%)	86,400 (+35%)	n.a.	n.a.	n.a.

Table G.6: Comparison of average (avg), minimum (min) and maximum (max) values of 10 runs of parallel, asynchronous and sequential coordination with 10 centrally computed runs after 7,200s, local run time 50s, full set of operators, large instance of the setup subscenario. The numbers in parentheses denote the relative difference from the average upstream planning solution.

			Second priority criteria		External lateness		
	Domain	Statistic	Makespan	Setup time	Max	Total	Weighted
	Central	min	95,601	n.a.	9,200	26,800	9,374
		avg	**98,561**	**n.a.**	**12,160**	**50,080**	**12,535**
		max	104,001	n.a.	17,600	98,000	18,396
	Central soft-constr.	min	96,801	n.a.	10,400	42,800	10,721
		avg	**99,241**	**n.a.**	**12,480**	**54,560**	**12,897**
		max	100,401	n.a.	14,000	64,000	14,495
Parallel, pop. size 12	OEM	min	88,401 (−5%)	n.a.	2,000 (−68%)	2,200 (−84%)	2,002 (−68%)
		avg	**88,127 (−4%)**	**n.a.**	**2,320 (−63%)**	**3,020 (−78%)**	**2,327 (−63%)**
		max	89,601 (−3%)	n.a.	3,200 (−49%)	4,000 (−71%)	3,208 (−49%)
	Supplier Steel	min	83,401 (−1%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**84,001 (−1%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	85,801 (+2%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
	Supplier Plastic	min	82,801 (+1%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**83,041 (±0%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	83,601 (±0%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
Parallel, pop. size 1	OEM	min	88,401 (−4%)	n.a.	2,000 (−67%)	2,000 (−85%)	2,000 (−68%)
		avg	**88,581 (−4%)**	**n.a.**	**2,180 (−64%)**	**2,880 (−79%)**	**2,187 (−65%)**
		max	89,601 (−3%)	n.a.	3,200 (−48%)	4,400 (−68%)	3,209 (−48%)
	Supplier Steel	min	83,401 (−1%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**84,041 (−1%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	85,401 (+1%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
	Supplier Plastic	min	82,401 (−1%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**83,041 (±0%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	83,601 (±0%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
Asynchronous	OEM	min	88,401 (−5%)	n.a.	2,000 (−68%)	2,800 (−80%)	2,008 (−68%)
		avg	**89,801 (−4%)**	**n.a.**	**2,680 (−57%)**	**3,880 (−73%)**	**2,692 (−57%)**
		max	90,401 (−2%)	n.a.	4,000 (−36%)	5,800 (−59%)	4,018 (−36%)
	Supplier Steel	min	83,401 (−1%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**83,801 (−1%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	85,801 (+2%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
	Supplier Plastic	min	82,401 (−1%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**83,041 (−9%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	83,601 (+1%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
Sequential	OEM	min	88,401 (−5%)	n.a.	2,000 (−68%)	2,800 (−80%)	2,008 (−68%)
		avg	**89,261 (−4%)**	**n.a.**	**2,860 (−54%)**	**4,200 (−70%)**	**2,873 (−55%)**
		max	90,001 (−3%)	n.a.	3,600 (−42%)	6,000 (−58%)	3,624 (−43%)
	Supplier Steel	min	82,801 (−2%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**83,501 (−1%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	85,001 (+1%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
	Supplier Plastic	min	82,001 (−1%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**82,641 (−1%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	83,201 (±0%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)

Table G.7: Comparison of average (avg), minimum (min) and maximum (max) values of 10 runs of parallel and sequential coordination with 10 centrally computed runs after 1,800s, local runtime 13s, full set of operators, small instance of the product mix subscenario. The numbers in parentheses denote the relative difference from the average upstream planning solution.

			Second priority criteria		External lateness		
	Domain	Statistic	Makespan	Setup time	Max	Total	Weighted
	Central	min	399,201	n.a.	53,600	816,000	61,149
		avg	**425,501**	**n.a.**	**79,900**	**1,853,540**	**97,461**
		max	437,201	n.a.	91,600	2,402,400	114,479
	Central soft-constr.	min	396,801	n.a.	51,200	745,600	58,075
		avg	**402,681**	**n.a.**	**57,080**	**940,840**	**65,831**
		max	408,801	n.a.	63,200	1,140,400	73,865
Parallel, pop. size 12	OEM	min	353,401 (−8%)	n.a.	7,800 (−81%)	21,600 (−96%)	7,941 (−83%)
		avg	**354,161 (−8%)**	**n.a.**	**8,560 (−79%)**	**25,260 (−96%)**	**8,725 (−81%)**
		max	354,601 (−8%)	n.a.	9,000 (−78%)	27,600 (−95%)	9,174 (−80%)
	Supplier Steel	min	348,601 (−2%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**349,761 (−2%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	350,601 (−1%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
	Supplier Plastic	min	342,401 (−3%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**343,401 (−2%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	345,601 (−2%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
Parallel, pop. size 1	OEM	min	352,401 (−10%)	n.a.	6,800 (−84%)	17,200 (−97%)	6,903 (−86%)
		avg	**353,541 (−9%)**	**n.a.**	**7,940 (−82%)**	**21,700 (−97%)**	**8,076 (−84%)**
		max	354,801 (−9%)	n.a.	9,200 (−79%)	26,800 (−96%)	9,374 (−81%)
	Supplier Steel	min	347,001 (−2%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**348,561 (−2%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	349,801 (−2%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
	Supplier Plastic	min	342,001 (−3%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**343,801 (−3%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	345,601 (−2%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
Asynchronous	OEM	min	354,401 (−7%)	n.a.	8,800 (−74%)	25,600 (−94%)	8,966 (−76%)
		avg	**356,581 (−6%)**	**n.a.**	**10,980 (−68%)**	**40,560 (−90%)**	**11,272 (−70%)**
		max	358,401 (−6%)	n.a.	12,800 (−63%)	52,000 (−88%)	13,188 (−65%)
	Supplier Steel	min	350,601 (−1%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**351,521 (−1%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	353,801 (−1%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
	Supplier Plastic	min	342,401 (−3%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**345,481 (−2%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	347,601 (−1%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
Sequential	OEM	min	355,601 (−9%)	n.a.	10,000 (−77%)	34,800 (−95%)	10,246 (−80%)
		avg	**356,541 (−9%)**	**n.a.**	**10,940 (−75%)**	**39,380 (−94%)**	**11,221 (−78%)**
		max	358,401 (−8%)	n.a.	12,800 (−71%)	50,000 (−93%)	13,168 (−74%)
	Supplier Steel	min	349,801 (−2%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**351,381 (−1%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	353,001 (−1%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
	Supplier Plastic	min	344,001 (−3%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**345,801 (−2%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	348,001 (−2%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)

Table G.8: Comparison of average (avg), minimum (min) and maximum (max) values of 10 runs of parallel and sequential coordination with 10 centrally computed runs after 3,600s, local runtime 25s, full set of operators, medium instance of the product mix subscenario. The numbers in parentheses denote the relative difference from the average upstream planning solution.

Specification	Rel. diff. to upstream planning	Rel. diff. to central solution	No. of called operators (only OEM)
Parallel, 12 parents	−93%	−97% (−78%)	300
Parallel, 1 parent	−94%	−98% (−82%)	335
Asynchronous	−89%	−96% (−70%)	101
Sequential	−90%	−95% (−64%)	39

Table G.9: Summary of average values for 10 runs for the large instance of the product mix subscenario after 7,200s, 50s local runtime, deterministic self-adaption. Values in parentheses denote the difference from the soft-constrained solution.

			Second priority criteria		External lateness		
	Domain	Statistic	Makespan	Setup time	Max	Total	Weighted
	Central	min	1,177,601	n.a.	261,200	25,989,800	515,938
		avg	**1,391,381**	**n.a.**	**466,420**	**97,675,660**	**1,428,887**
		max	1,601,201	n.a.	892,000	196,231,400	2,826,053
	Central soft-constr.	min	1,148,801	n.a.	112,000	7,841,600	188,530
		avg	**1,152,461**	**n.a.**	**115,660**	**8,428,980**	**197,970**
		max	1,155,601	n.a.	118,800	8,841,600	205,164
Parallel, pop. size 12	OEM	min	1,065,800 (−12%)	n.a.	29,000 (−91%)	452,600 (−98%)	33,194 (−94%)
		avg	**1,071,500 (−12%)**	**n.a.**	**34,700 (−89%)**	**804,180 (−97%)**	**42,319 (−93%)**
		max	1,078,600 (−11%)	n.a.	41,800 (−87%)	1,180,400 (−96%)	52,691 (−91%)
	Supplier Steel	min	1,060,200 (−4%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**1,066,700 (−3%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	1,075,000 (−3%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
	Supplier Plastic	min	1,037,200 (−9%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**1,043,480 (−8%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	1,048,800 (−8%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
Parallel, pop. size 1	OEM	min	1,060,000 (−13%)	n.a.	23,200 (−93%)	347,600 (−99%)	26,411 (−96%)
		avg	**1,066,660 (−13%)**	**n.a.**	**29,860 (−91%)**	**588,000 (−98%)**	**35,386 (−94%)**
		max	1,073,800 (−12%)	n.a.	37,000 (−88%)	1,020,400 (−97%)	46,737 (−92%)
	Supplier Steel	min	1,056,200 (−4%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**1,061,420 (−4%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	1,069,800 (−3%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
	Supplier Plastic	min	1,040,000 (−8%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**1,043,040 (−8%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	1,046,000 (−8%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
Asynchronous	OEM	min	1,071,800 (−12%)	n.a.	35,000 (−88%)	792,800 (−97%)	42,808 (−92%)
		avg	**1,081,400 (−11%)**	**n.a.**	**44,600 (−85%)**	**1,512,100 (−94%)**	**59,130 (−89%)**
		max	1,087,400 (−10%)	n.a.	50,600 (−83%)	2,100,200 (−92%)	70,893 (−87%)
	Supplier Steel	min	1,066,200 (−3%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**1,075,820 (−3%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	1,083,800 (−2%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
	Supplier Plastic	min	1,041,200 (−8%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**1,055,200 (−7%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	1,062,400 (−6%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
Sequential	OEM	min	1,082,201 (−12%)	n.a.	45,400 (−87%)	1,423,400 (−96%)	59,043 (−91%)
		avg	**1,089,261 (−11%)**	**n.a.**	**52,700 (−85%)**	**1,937,500 (−94%)**	**71,361 (−90%)**
		max	1,092,801 (−11%)	n.a.	56,400 (−84%)	2,382,200 (−93%)	79,428 (−88%)
	Supplier Steel	min	1,077,801 (−2%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**1,084,461 (−2%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	1,088,601 (−2%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
	Supplier Plastic	min	1,050,001 (−8%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**1,059,321 (−7%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	1,065,601 (−7%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)

Table G.10: Comparison of average (avg), minimum (min) and maximum (max) values of 10 runs of parallel and sequential coordination with 10 centrally computed runs after 7,200s, local runtime 50s, full set of operators, large instance of the product mix subscenario. The numbers in parentheses denote the relative difference from the average upstream planning solution.

			Second priority criteria		External lateness		
	Domain	Statistic	Makespan	Setup time	Max	Total	Weighted
	Central	min	432,000	n.a.	0	0	0
		avg	**432,000**	**n.a.**	**14,380**	**89,095**	**15,120**
		max	432,000	n.a.	30,600	275,400	33,024
	Central soft-constr.	min	432,000	n.a.	0	0	0
		avg	**432,000**	**n.a.**	**11,980**	**96,396**	**12,816**
		max	432,000	n.a.	19,800	217,800	21,760
Parallel, pop. size 12	OEM	min	389,400 (−8%)	n.a.	0 (−100%)	0 (−100%)	0 (−100%)
		avg	**392,260 (−8%)**	**n.a.**	**0 (−100%)**	**0 (−100%)**	**0 (−100%)**
		max	395,200 (−7%)	n.a.	0 (−100%)	0 (−100%)	0 (−100%)
	Supplier Steel	min	432,000 (±0%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**432,000 (±0%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	432,000 (±0%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
	Supplier Plastic	min	259,201 (±0%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**259,201 (±0%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	259,201 (±0%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
Parallel, pop. size 1	OEM	min	388,200 (−9%)	n.a.	0 (−100%)	0 (−100%)	0 (−100%)
		avg	**390,420 (−8%)**	**n.a.**	**0 (−100%)**	**0 (−100%)**	**0 (−100%)**
		max	394,200 (−7%)	n.a.	0 (−100%)	0 (−100%)	0 (−100%)
	Supplier Steel	min	432,000 (±0%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**432,000 (±0%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	432,000 (±0%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
	Supplier Plastic	min	259,201 (±0%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**259,201 (±0%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	259,201 (±0%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
Asynchronous	OEM	min	387,400 (−9%)	n.a.	0 (−100%)	0 (−100%)	0 (−100%)
		avg	**391,160 (−8%)**	**n.a.**	**0 (−100%)**	**0 (−100%)**	**0 (−100%)**
		max	394,400 (−7%)	n.a.	0 (−100%)	0 (−100%)	0 (−100%)
	Supplier Steel	min	432,000 (±0%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**432,000 (±0%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	432,000 (±0%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
	Supplier Plastic	min	259,201 (±0%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**259,201 (±0%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	259,201 (±0%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
Sequential	OEM	min	387,200 (−9%)	n.a.	0 (−100%)	0 (−100%)	0 (−100%)
		avg	**390,740 (−8%)**	**n.a.**	**0 (−100%)**	**0 (−100%)**	**0 (−100%)**
		max	396,200 (−7%)	n.a.	0 (−100%)	0 (−100%)	0 (−100%)
	Supplier Steel	min	432,000 (±0%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**432,000 (±0%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	432,000 (±0%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
	Supplier Plastic	min	259,201 (±0%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**259,201 (±0%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	259,201 (±0%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)

Table G.11: Comparison of average (avg), minimum (min) and maximum (max) values of 10 runs of parallel and sequential coordination with 10 centrally computed runs after 3,600s, local runtime 25s, full set of operators, medium instance of the breakdown subscenario. The numbers in parentheses denote the relative difference from the (min/avg/max) upstream planning solution.

Specification	Rel. diff. to upstream planning	Rel. diff. to central, solution	No. of called operators (only OEM)
Parallel, 12 parents	−59%	−37% (−13%)	375
Parallel, 1 parent	−58%	−37% (−13%)	428
Asynchronous	−56%	−33% (−7%)	99
Sequential	−54%	−30% (−4%)	58

Table G.12: Summary of average values for 10 runs for the large instance of the breakdown subscenario, first variant, after 7,200s, 50s local runtime, deterministic self-adaption. Values in parentheses denote the difference from the soft-constrained solution.

	Domain	Statistic	Second priority criteria		External lateness		
			Makespan	Setup time	Max	Total	Weighted
	Central	min	1,404,600	n.a.	208,000	20,434,800	408,265
		avg	**1,407,940**	**n.a.**	**217,080**	**21,804,357**	**430,815**
		max	1,411,800	n.a.	220,999	22,617,028	442,742
	Central soft-constr.	min	1,296,000	n.a.	18,999	9,488,900	112,760
		avg	**1,352,260**	**n.a.**	**168,820**	**14,559,862**	**311,306**
		max	1,409,201	n.a.	219,200	22,215,800	436,988
Parallelt, pop. size 12	OEM	min	1,349,400 (−2%)	n.a.	152,599 (−58%)	10,857,100 (−63%)	258,584 (−60%)
		avg	**1,354,840 (−2%)**	**n.a.**	**157,499 (+57%)**	**11,569,240 (−61%)**	**270,486 (−59%)**
		max	1,364,800 (−1%)	n.a.	166,799 (−54%)	12,207,800 (−59%)	285,480 (−56%)
	Supplier Steel	min	1,345,800 (−2%)	n.a.	0 (−100%)	0 (−100%)	0 (−100%)
		avg	**1,351,240 (−2%)**	**n.a.**	**0 (−100%)**	**0 (−100%)**	**0 (−100%)**
		max	1,361,200 (−1%)	n.a.	0 (−100%)	0 (−100%)	0 (−100%)
	Supplier Plastic	min	777,601 (±0%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**777,601 (±0%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	777,601 (±0%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
Parallel, pop. size 1	OEM	min	1,344,200 (−2%)	n.a.	148,599 (−59%)	10,329,900 (−65%)	249,404 (−62%)
		avg	**1,351,880 (−2%)**	**n.a.**	**159,079 (−56%)**	**11,532,020 (−61%)**	**271,683 (−58%)**
		max	1,366,800 (−1%)	n.a.	179,600 (−50%)	13,213,500 (−55%)	303,646 (−52%)
	Supplier Steel	min	1,340,600 (−2%)	n.a.	0 (−100%)	0 (−100%)	0 (−100%)
		avg	**1,348,280 (−2%)**	**n.a.**	**0 (−100%)**	**0 (−100%)**	**0 (−100%)**
		max	1,363,200 (−1%)	n.a.	0 (−100%)	0 (−100%)	0 (−100%)
	Supplier Plastic	min	777,601 (±0%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**777,601 (±0%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	777,601 (±0%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
Asynchronous	OEM	min	1,350,200 (−2%)	n.a.	154,599 (−58%)	10,883,400 (−63%)	260,824 (−60%)
		avg	**1,356,160 (−1%)**	**n.a.**	**167,439 (−55%)**	**12,528,270 (−57%)**	**289,824 (−56%)**
		max	1,364,600 (−1%)	n.a.	216,599 (−41%)	16,858,400 (−43%)	381,369 (−42%)
	Supplier Steel	min	1,346,600 (−2%)	n.a.	0 (−100%)	0 (−100%)	0 (−100%)
		avg	**1,352,560 (−1%)**	**n.a.**	**0 (−100%)**	**0 (−100%)**	**0 (−100%)**
		max	1,361,000 (−1%)	n.a.	0 (−100%)	0 (−100%)	0 (−100%)
	Supplier Plastic	min	777,601 (±0%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**777,601 (±0%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	777,601 (±0%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
Sequential	OEM	min	1,342,400 (−2%)	n.a.	154,799 (−58%)	10,702,252 (−64%)	259,220 (−60%)
		avg	**13,591,220 (−1%)**	**n.a.**	**174,800 (−52%)**	**12,815,254 (−57%)**	**299,953 (−54%)**
		max	1,366,001 (−1%)	n.a.	232,000 (−37%)	17,406,000 (−41%)	402,040 (−39%)
	Supplier Steel	min	1,338,800 (−2%)	n.a.	0 (−100%)	0 (−100%)	0 (−100%)
		avg	**1,355,520 (−1%)**	**n.a.**	**0 (−100%)**	**0 (−100%)**	**0 (−100%)**
		max	1,362,401 (−1%)	n.a.	0 (−100%)	0 (−100%)	0 (−100%)
	Supplier Plastic	min	777,601 (±0%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**777,601 (±0%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	777,601 (±0%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)

Table G.13: Comparison of average (avg), minimum (min) and maximum (max) values of 10 runs of parallel and sequential coordination with 10 centrally computed runs after 7,200s, local runtime 50s, full set of operators, large instance of the breakdown subscenario, first variant. The numbers in parentheses denote the relative difference from the average upstream planning solution.

Specification	Rel. diff. to upstream planning	Rel. diff. to central solution	No. of called operators (only OEM)
Parallel, 12 parents	−47%	−31% (+120%)	323
Parallel, 1 parent	−48%	−30% (+120%)	340
Asynchronous	−33%	−15% (+167%)	93
Sequential	−32%	−11% (180%)	50

Table G.14: Summary of average values for 10 runs for the large instance of the breakdown subscenario, second variant, after 7,200s, 50s local runtime, deterministic self-adaption. Values in parentheses denote the difference from the soft-constrained solution.

			Second priority criteria		External lateness		
	Domain	Statistic	Makespan	Setup time	Max	Total	Weighted
	Central	min	1,315,400	n.a.	634,999	46,190,956	1,086,048
		avg	**1,334,340**	**n.a.**	**749,819**	**58,562,069**	**1,322,218**
		max	1,369,400	n.a.	1,000,399	75,003,164	1,733,100
	Central soft-constr.	min	1,296,000	n.a.	172,799	24,454,655	413,213
		avg	**1,296,000**	**n.a.**	**172,799**	**24,839,941**	**417,199**
		max	1,296,000	n.a.	172,799	25,129,440	419,894
Parallel, pop. size 12	OEM	min	1,316,400 (−2%)	n.a.	370,199 (−65%)	45,406,100 (−32%)	816,099 (−53%)
		avg	**1,318,500 (−2%)**	**n.a.**	**442,299 (−58%)**	**48,349,480 (−28%)**	**916,628 (−47%)**
		max	1,323,000 (−2%)	n.a.	465,399 (−56%)	52,165,800 (−22%)	977,284 (−43%)
	Supplier Steel	min	1,309,400 (−2%)	n.a.	0 (−100%)	0 (−100%)	0 (−100%)
		avg	**1,314,540 (−2%)**	**n.a.**	**0 (−100%)**	**0 (−100%)**	**0 (−100%)**
		max	1,319,400 (−2%)	n.a.	0 (−100%)	0 (−100%)	0 (−100%)
	Supplier Plastic	min	777,601 (±0%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**777,601 (±0%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	777,601 (±0%)	n.a.	0 (±0%)	0 (±0%)	0 0 (±0%))
Parallel, pop. size 1	OEM	min	1,312,000 (−2%)	n.a.	361,599 (−67%)	47,611,400 (−29%)	829,419 (−53%)
		avg	**1,318,220 (−2%)**	**n.a.**	**433,919 (−61%)**	**49,314,930 (−26%)**	**917,889 (−48%)**
		max	1,323,000 (−1%)	n.a.	462,199 (−58%)	51,744,400 (−22%)	969,944 (−45%)
	Supplier Steel	min	1,308,400 (−2%),	n.a.	0 (−100%)	0 (−100%)	0 (−100%)
		avg	**1,313,940 (−2%)**	**n.a.**	**0 (−100%)**	**0 (−100%)**	**0 (−100%)**
		max	1,319,400 (−1%)	n.a.	0 (−100%)	0 (−100%)	0 (−100%)
	Supplier Plastic	min	777,601 (±0%)	n.a.	0 0 (±0%)	0 0 (±0%)	0 0 (±0%)
		avg	**777,601 (±0%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	777,601 (±0%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
Asynchronous	OEM	min	1,316,200 (−2%)	n.a.	452,199 (−57%)	51,259,600 (−21%)	955,243 (−43%)
		avg	**1,319,260 (−2%)**	**n.a.**	**577,619 (−44%)**	**54,832,860 (−16%)**	**1,114,800 (−33%)**
		max	1,328,200 (−1%)	n.a.	962,599 (−7%)	61,268,400 (−6%)	1,559,686 (−7%)
	Supplier Steel	min	1,309,800 (−2%)	n.a.	0 (−100%)	0 (−100%)	0 (−100%)
		avg	**1,315,060 (−2%)**	**n.a.**	**0 (−100%)**	**0 (−100%)**	**0 (−100%)**
		max	1,324,600 (−1%)	n.a.	0 (−100%)	0 (−100%)	0 (−100%)
	Supplier Plastic	min	777,601 (±0%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**777,601 (±0%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	777,601 (±0%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
Sequential	OEM	min	1,314,800 (−2%)	n.a.	472,599 (−56%)	48,527,551 (−28%)	948,391 (−45%)
		avg	**1,318,360 (−2%)**	**n.a.**	**639,939 (−41%)**	**54,077,421 (−19%)**	**1,169,024 (−32%)**
		max	1,322,800 (−2%)	n.a.	724,999 (−33%)	59,690,653 (−11%)	1,308,817 (−24%)
	Supplier Steel	min	1,310,200 (−2%)	n.a.	0 (−100%)	0 (−100%)	0 (−100%)
		avg	**1,314,140 (−2%)**	**n.a.**	**0 (−100%)**	**0 (−100%)**	**0 (−100%)**
		max	1,319,200 (−2%)	n.a.	0 (−100%)	0 (−100%)	0 (−100%)
	Supplier Plastic	min	777,601 (±0%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**777,601 (±0%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	777,601 (±0%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)

Table G.15: Comparison of average (avg), minimum (min) and maximum (max) values of 10 runs of parallel and sequential coordination with 10 centrally computed runs after 7,200s, local runtime 50s, full set of operators, large instance of the breakdown subscenario, second variant. The numbers in parentheses denote the relative difference from the average upstream planning solution.

Specification	Rel. diff. to upstream planning	Rel. diff. to central solution	No. of called operators (only OEM)
Parallel, 12 parents	−99%	−100%	1,418
Parallel, 1 parent	−98%	−99%	1,356
Asynchronous	−89%	−95%	846
Sequential	−91%	−94%	322

Table G.16: Summary of average values for 10 runs for the large instance of the complete scenario, after 14,400s, 100s local runtime, deterministic self-adaption.

	Domain	Statistic	Second priority criteria		External lateness		
			Makespan	Setup time	Max	Total	Weighted
	Central	min	1,242,201	141,600	122,000	10,939,800	254,899
		avg	**1,271,461**	**150,080**	**148,560**	**13,544,800**	**281,196**
		max	1,284,001	165,600	160,800	14,880,800	306,543
	Central, soft-constr.	min	1,253,601	145,600	130,400	11,326,900	241,256
		avg	**1,276,161**	**160,560**	**152,960**	**13,947,050**	**289,535**
		max	1,287,401	188,000	164,200	15,508,400	316,122
Parallel, pop. size 12	OEM	min	1,121,600 (−7%)	81,600 (+2%)	0 (−100%)	0 (−100%)	0 (−100%)
		avg	**1,124,170 (−7%)**	**24,640 (+55%)**	**1,340 (−98%)**	**4,930 (−100%)**	**1,376 (−99%)**
		max	1,130,200 (−7%)	161,600 (+101%)	7,000 (−92%)	34,400 (−99%)	7,271 (−94%)
	Supplier Steel	min	1,099,200 (−1%)	153,600 (+16%)	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**1,103,720 (−1%)**	**202,560 (+53%)**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	1,109,200 (−1%)	243,200 (+83%)	0 (±0%)	0 (±0%)	0 (±0%)
	Supplier Plastic	min	1,040,800 (−5%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**1,047,400 (−5%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	1,052,000 (−4%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
	Supplier Motor	min	1,001,600 (−0%)	112,000 (−22%)	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**1,014,400 (+1%)**	**159,680 (+11%)**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	1,030,400 (+3%)	198,400 (+38%)	0 (±0%)	0 (±0%)	0 (±0%)
Parallel, pop. size 1	OEM	min	1,121,900 (−7%)	59,200 (−30%)	0 (−100%)	0 (−100%)	0 (−100%)
		avg	**1,125,620 (−7%)**	**104,400 (+24%)**	**2,660 (−97%)**	**9,720 (−100%)**	**2,730 (−98%)**
		max	1,135,200 (−6%)	147,200 (+75%)	12,000 (−86%)	50,200 (−99%)	12,378 (−91%)
	Supplier Steel	min	1,098,800 (−1%)	108,800 (−17%)	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**1,102,900 (−1%)**	**186,880 (+42%)**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	1,108,400 (−1%)	275,200 (+109%)	0 (±0%)	0 (±0%)	0 (±0%)
	Supplier Plastic	min	1,039,200 (−5%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**1,046,400 (−5%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	1,059,600 (−3%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
	Supplier Motor	min	985,601 (−2%)	124,800 (−13%)	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**1,028,160 (+2%)**	**190,720 (+33%)**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	1,078,400 (+7%)	281,600 (+96%)	0 (±0%)	0 (±0%)	0 (±0%)
Asynchronous	OEM	min	1,122,800 (−7%)	64,800 (−23%)	0 (−100%)	0 (−100%)	0 (−100%)
		avg	**1,136,920 (−6%)**	**75,280 (−11%)**	**13,760 (−85%)**	**98,880 (−98%)**	**14,603 (−89%)**
		max	1,145,900 (−5%)	95,200 (+13%)	22,700 (−75%)	194,300 (−96%)	24,399 (−82%)
	Supplier Steel	min	1,098,200 (−1%)	92,800 (−28%)	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**1,102,500 (−1%)**	**122,560 (−5%)**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	1,106,200 (−1%)	166,400 (+29%)	0 (±0%)	0 (±0%)	0 (±0%)
	Supplier Plastic	min	1,046,800 (−5%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**1,053,880 (−4%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	1,064,400 (−3%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
	Supplier Motor	min	982,401 (−2%)	86,400 (−38%)	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**1,000,961 (−0%)**	**111,040 (−21%)**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	1,020,800 (+2%)	140,800 (±0%)	0 (±0%)	0 (±0%)	0 (±0%)
Sequential	OEM	min	1,124,601 (−9%)	64,800 (−4%)	1,400 (−99%)	1,400 (−100%)	1,400 (−99%)
		avg	**1,139,171 (−7%)**	**79,280 (+18%)**	**15,970 (−86%)**	**205,940 (−98%)**	**17,851 (−91%)**
		max	1,154,701 (−6%)	103,200 (+53%)	31,500 (−72%)	699,800 (−92%)	38,117 (−80%)
	Supplier Steel	min	1,095,401 (−2%)	115,200 (−13%)	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**1,100,081 (−1%)**	**132,160 (−0%)**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	1,104,401 (−1%)	156,800 (+18%)	0 (±0%)	0 (±0%)	0 (±0%)
	Supplier Plastic	min	1,044,801 (−5%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**1,052,681 (−5%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	1,062,801 (−4%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
	Supplier Motor	min	966,401 (−4%)	70,400 (−50%)	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**1,003,201 (−0%)**	**121,920 (−13%)**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	1,017,601 (+1%)	160,000 (+14%)	0 (±0%)	0 (±0%)	0 (±0%)

Table G.17: Comparison of average (avg), minimum (min) and maximum (max) values of 10 runs of parallel and sequential coordination with 10 centrally computed runs after 14,400s, local run time 100s, full set of operators, large instance of the complete scenario. The numbers in parentheses denote the relative difference from the average upstream planning solution.

		Second priority		External lateness	
Domain	Statistic	Makespan	Max	Total	Weighted
	min	19,258	3,698	19,633	5,146
Central	**avg**	**19,936**	**4,378**	**22,463**	**6,022**
	max	20,389	5,106	26,541	7,055
Central	min	19,100	3,512	14,635	4,253
soft-	**avg**	**19,370**	**3,796**	**17,906**	**5,079**
constr.	max	19,846	4,258	24,287	6,079
	min	18,024 (−11%)	2,436 (−49%)	6,649 (−75%)	2,819 (−58%)
OEM	**avg**	**18,362 (−9%)**	**2,777 (−42%)**	**8,760 (−67%)**	**3,321 (−51%)**
	max	18,917 (−7%)	3,317 (−30%)	10,991 (−58%)	4,015 (−40%)
	min	16,744 (−8%)	0 (±0%)	0 (±0%)	0 (±0%)
Supplier	**avg**	**17,036 (−7%)**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
	max	17,347 (−5%)	0 (±0%)	0 (±0%)	0 (±0%)

Table G.18: Comparison of average (avg), minimum (min) and maximum (max) values of 10 runs of parallel coordination with 10 centrally computed runs after 1,800s, local runtime 13s, full set of operators, small randomly generated instance, $\delta^{\Omega}_{\text{supplier}} = 0.3$. The numbers in parentheses denote the relative difference from the average upstream planning solution.

		Second priority		External lateness	
Domain	Statistic	Makespan	Max	Total	Weighted
	min	20,418	4,044	11,088	4,684
Central	**avg**	**21,374**	**4,957**	**16,936**	**6,046**
	max	21,910	5,536	21,838	7,018
Central	min	20,674	4,270	12,082	4,980
soft-	**avg**	**21,196**	**4,750**	**18,489**	**5,999**
constr.	max	21,772	5,398	25,350	7,212
	min	19,090 (−10%)	2,499 (−48%)	4,739 (−75%)	2,703 (−56%)
OEM	**avg**	**19,347 (−9%)**	**2,821 (−41%)**	**6,998 (−64%)**	**3,201 (−48%)**
	max	19,725 (−7%)	3,351 (−30%)	8,642 (−55%)	3,832 (−37%)
	min	16,532 (−7%)	0 (±0%)	0 (±0%)	0 (±0%)
Supplier	**avg**	**16,856 (−6%)**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
	max	17,255 (−3%)	0 (±0%)	0 (±0%)	0 (±0%)

Table G.19: Comparison of average (avg), minimum (min) and maximum (max) values of 10 runs of parallel coordination with 10 centrally computed runs after 1,800s, local runtime 13s, full set of operators, small randomly generated instance, $\delta^{\Omega}_{\text{supplier}} = 0.7$. The numbers in parentheses denote the relative difference from the average upstream planning solution.

		Second priority		External lateness	
Domain	Statistic	Makespan	Max	Total	Weighted
	min	129,546	58,618	1,210,677	163,350
Central	**avg**	**132,811**	**62,030**	**1,300,110**	**174,583**
	max	134,900	63,979	1,375,060	183,168
	min	105,907 (−14%)	34,979 (−34%)	432,629 (−55%)	71,129 (−48%)
OEM	**avg**	**108,146 (−12%)**	**37,225 (−29%)**	**479,337 (−50%)**	**77,417 (−43%)**
	max	110,454 (−11%)	39,526 (−25%)	527,028 (−45%)	83,844 (−38%)
	min	97,571 (−8%)	0 (±0%)	0 (±0%)	0 (±0%)
Supplier	**avg**	**98,653 (−7%)**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
	max	99,801 (−6%)	0 (±0%)	0 (±0%)	0 (±0%)

Table G.20: Comparison of average (avg), minimum (min) and maximum (max) values of 10 runs of parallel coordination with 10 centrally computed runs after 3,600s, local runtime 25s, full set of operators, medium randomly generated instance, $\delta^{\Omega}_{\text{supplier}} = 0.0$. The numbers in parentheses denote the relative difference from the average upstream planning solution.

			Second priority	External lateness		
	Domain	Statistic	Makespan	Max	Total	Weighted
	Central	min	135,495	63,568	733,286	124,451
		avg	**138,125**	**66,198**	**825,467**	**135,222**
		max	141,358	69,431	924,049	147,123
	Central soft-constr.	min	132,144	60,217	612,952	110,467
		avg	**134,711**	**62,955**	**757,358**	**126,083**
		max	138,669	66,742	897,644	142,279
standard	OEM	min	109,920 (−23%)	37,993 (−47%)	400,859 (−63%)	70,981 (−57%)
		avg	**111,940 (−22%)**	**40,013 (−44%)**	**452,097 (−59%)**	**77,475 (−53%)**
		max	113,755 (−20%)	41,828 (−41%)	525,552 (−52%)	85,803 (−48%)
	Supplier	min	98,175 (−7%)	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**99,722 (−5%)**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	100,571 (−5%)	0 (±0%)	0 (±0%)	0 (±0%)
from scratch	OEM	min	124,800 (−14%)	53,614 (−26%)	678,223 (−39%)	110,397 (−34%)
		avg	**126,332 (−13%)**	**54,932 (−24%)**	**711,642 (−36%)**	**114,633 (−31%)**
		max	127,870 (−11%)	56,158 (−23%)	753,296 (−32%)	119,534 (−28%)
	Supplier	min	109,384 (4%)	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**111,832 (6%)**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	113,871 (8%)	0 (±0%)	0 (±0%)	0 (±0%)

Table G.21: Comparison of average (avg), minimum (min) and maximum (max) values of 10 runs of parallel coordination with 10 centrally computed runs after 3,600s, local runtime 25s, full set of operators, medium randomly generated instance, $\delta^{\Omega}_{\text{supplier}} = 0.5$. The numbers in parentheses denote the relative difference from the average upstream planning solution.

			Second priority	External lateness		
	Domain	Statistic	Makespan	Max	Total	Weighted
	Central	min	727,285	400,799	88,018,488	8,366,043
		avg	**734,528**	**408,174**	**90,550,938**	**8,602,970**
		max	739,830	413,428	92,146,245	8,752,775
standard	OEM	min	616,159 (−6%)	290,153 (−12%)	46,887,000 (−28%)	4,526,230 (−28%)
		avg	**631,295 (−4%)**	**305,334 (−7%)**	**49,319,060 (−25%)**	**4,761,127 (−24%)**
		max	652,075 (−1%)	326,911 (−1%)	52,555,000 (−20%)	5,074,919 (−19%)
	Supplier	min	576,209 (±0%)	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**585,401 (2%)**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	593,519 (3%)	0 (±0%)	0 (±0%)	0 (±0%)
from scratch	OEM	min	637,979 (−2%)	314,065 (−4%)	52,381,100 (−21%)	5,047,432 (−20%)
		avg	**648,988 (−1%)**	**324,097 (−1%)**	**54,568,910 (−17%)**	**5,255,444 (−16%)**
		max	657,599 (1%)	332,569 (1%)	57,841,400 (−12%)	5,560,645 (−12%)
	Supplier	min	585,110 (2%)	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**592,447 (3%)**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	602,118 (5%)	0 (±0%)	0 (±0%)	0 (±0%)

Table G.22: Comparison of average (avg), minimum (min) and maximum (max) values of 10 runs of parallel coordination with 10 centrally computed runs after 7,200s, local runtime 240s, full set of operators, small randomly generated instance, $\delta^{\Omega}_{\text{supplier}} = 0.0$. The numbers in parantheses denote the relative difference to the average upstream planning solution:

		Second priority	External lateness		
Domain	Statistic	Makespan	Max	Total	Weighted
Central	min	203,560	150,482	19,903,251	1,946,188
	avg	**204,478**	**151,397**	**21,105,749**	**2,056,338**
	max	206,541	153,427	22,850,640	2,216,810
OEM	min	202,323 (−13%)	149,245 (−17%)	17,755,000 (−31%)	1,749,768 (−30%)
	avg	**203,219 (−13%)**	**150,149 (−16%)**	**17,867,250 (−30%)**	**1,760,795 (−29%)**
	max	204,120 (−12%)	151,042 (−16%)	17,976,600 (−30%)	1,771,547 (−29%)
Supplier	min	193,002 (±0%)	0 (±0%)	0 (±0%)	0 (±0%)
	avg	**193,268 (±0%)**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
	max	193,612 (±0%)	0 (±0%)	0 (±0%)	0 (±0%)

Table G.23: Comparison of average (avg), minimum (min) and maximum (max) values of 10 runs of parallel coordination with 10 centrally computed runs after 3,600s, local runtime 25s, more reduced set of operators, first additional medium randomly generated instance. The numbers in parentheses denote the relative difference from the average upstream planning solution.

		Second priority	External lateness		
Domain	Statistic	Makespan	Max	Total	Weighted
	min	203,646	139,007	2,632,109	365,653
Central	**avg**	**207,520**	**142,760**	**2,792,799**	**383,673**
	max	209,785	146,301	2,896,785	369,345
	min	206,345 (−6%)	141,025 (−9%)	2,389,510 (−20%)	345,433 (−17%)
OEM	**avg**	**210,211 (−5%)**	**144,926 (−7%)**	**2,428,342 (−19%)**	**352,509 (−15%)**
	max	212,853 (−4%)	147,533 (−5%)	2,479,120 (−17%)	359,495 (−13%)
	min	192,394 (−1%)	0 (±0%)	0 (±0%)	0 (±0%)
Supplier	**avg**	**193,758 (±0%)**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
	max	195,395 (±0%)	0 (±0%)	0 (±0%)	0 (±0%)

Table G.24: Comparison of average (avg), minimum (min) and maximum (max) values of 10 runs of parallel coordination with 10 centrally computed runs after 3,600s, local runtime 25s, more reduced set of operators, second additional medium randomly generated instance. The numbers in parentheses denote the relative difference from the average upstream planning solution.

item	demand (in lots a 20 items)													
sMower	0	0	0	0	40	0	0	0	0	0	10	40	0	0
mMower	0	0	0	0	30	0	0	0	0	0	0	50	0	0
lMower	0	0	0	0	0	0	0	10	20	20	0	0	0	0
xMower	0	0	0	0	50	0	0	0	0	0	0	30	0	0
extWheel	10	15	10	35	10	0	0	10	15	10	15	10	0	0
sBladeExt	0	30	0	30	30	0	0	0	0	50	0	20	0	0
lBladeExt	10	15	20	15	20	0	0	0	10	10	20	0	0	0
extTank_2	0	0	0	0	40	0	0	0	0	0	0	20	0	0

Table G.25: Biweekly demand matrix for the first additional medium instance of the complete scenario

item	demand (in lots a 20 items)													
sMower	0	0	0	0	40	0	0	0	0	0	0	0	0	0
mMower	0	0	0	0	60	0	0	0	0	0	0	0	0	0
lMower	0	0	0	0	0	0	0	0	0	0	0	80	0	0
xMower	0	0	0	0	0	0	0	0	0	0	0	70	0	0
extWheel	0	0	0	0	60	0	0	0	0	0	0	60	0	0
sBladeExt	0	0	0	0	60	0	0	0	0	0	0	60	0	0
lBladeExt	0	0	0	0	60	0	0	0	0	0	0	60	0	0
extTank_2	0	0	0	0	20	0	0	0	0	0	0	20	0	0

Table G.26: Biweekly demand matrix for the second additional medium instance of the complete scenario

			Second priority criteria		External lateness		
	Domain	Statistic	Makespan	Setup time	Max	Total	Weighted
	Central	min	1,357,602	83,200	322,080	32,320,957	638,901
		avg	**1,380,379**	**104,240**	**369,576**	**45,246,486**	**813,902**
		max	1,397,381	121,600	396,920	56,023,926	947,682
	Central, soft-constr.	min	1,383,102	86,400	367,801	42,807,434	787,995
		avg	**1,394,169**	**114,000**	**387,811**	**50,810,745**	**887,048**
		max	1,402,202	149,600	403,801	59,680,049	990,695
Parallel, pop. size 12	OEM	min	1,294,200 (−7%)	110,400 (+11%)	257,401 (−34%)	11,957,600 (−75%)	373,245 (−57%)
		avg	**1,306,690 (−6%)**	**144,800 (+46%)**	**269,891 (−31%)**	**13,472,050 (−72%)**	**400,605 (−54%)**
		max	1,318,200 (−5%)	156,000 (+57%)	281,401 (−28%)	14,238,700 (−71%)	419,592 (−52%)
	Supplier Steel	min	1,253,360 (−1%)	89,600 (−1%)	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**1,253,360 (−1%)**	**126,720 (+39%)**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	1,253,360 (−1%)	147,200 (+62%)	0 (±0%)	0 (±0%)	0 (±0%)
	Supplier Plastic	min	1,037,260 (−8%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**1,037,260 (−8%)**	**n.a.**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	1,037,260 (−8%)	n.a.	0 (±0%)	0 (±0%)	0 (±0%)
	Supplier Motor	min	1,037,260 (−8%)	64,000 (−22%)	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**1,231,750 (+10%)**	**93,440 (+14%)**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	1,253,360 (+12%)	118,400 (+45%)	0 (±0%)	0 (±0%)	0 (±0%)

Table G.27: Comparison of average (avg), minimum (min) and maximum (max) values of 10 runs of parallel coordination with 10 centrally computed runs after 3,600s, local run time 25s, full set of operators, first additional medium instance of the complete scenario. The numbers in parentheses denote the relative difference from the average upstream planning solution.

			Second priority criteria		External lateness		
	Domain	Statistic	Makespan	Setup time	Max	Total	Weighted
	Central	min	1,346,702	40,000	404,401	36,137,736	758,196
		avg	**1,353,008**	**55,120**	**408,887**	**37,629,431**	**777,407**
		max	1,359,502	66,400	416,701	39,617,540	804,828
	Central, soft-constr.	min	1,344,502	45,600	407,401	35,967,146	759,477
		avg	**1,351,926**	**60,240**	**409,051**	**37,538,779**	**776,672**
		max	1,362,302	66,400	411,001	39,341,332	796,450
Parallel, pop. size 12	OEM	min	1,026,800 (−19%)	33,600 (−46%)	330,601 (−18%)	9,462,640 (−54%)	421,017 (−30%)
		avg	**1,121,708 (−11%)**	**68,080 (9%)**	**346,409 (−14%)**	**9,886,212 (−52%)**	**440,862 (−27%)**
		max	1,220,900 (−3%)	84,000 (+34%)	371,401 (−7%)	10,768,900 (−48%)	474,347 (−21%)
	Supplier Steel	min	950,821 (±0%)	35,200 (−8%)	0 (±0%)	0 (±0%)	0 (±0%)
		avg	**950,821 (±0%)**	**59,200 (+55%)**	**0 (±0%)**	**0 (±0%)**	**0 (±0%)**
		max	950,821 (±0%)	92,800 (+144%)	0 (±0%)	0 (±0%)	0 (±0%)
	Supplier Plastic	min	950,821 (−3%)	(%)	(%)	(%)	(%)
		avg	**952,979 (−3%)**	**(%)**	**(%)**	**(%)**	**(%)**
		max	972,401 (−1%)	n.a.	(%)	(%)	(%)
	Supplier Motor	min	(%)	(+%)	(%)	(%)	(%)
		avg	**(%)**	**(%)**	**(%)**	**(%)**	**(%)**
		max	(%)	(%)	(%)	(%)	(%)

Table G.28: Comparison of average (avg), minimum (min) and maximum (max) values of 10 runs of parallel coordination with 10 centrally computed runs after 3,600s, local run time 25s, full set of operators, second additional medium instance of the complete scenario. The numbers in parentheses denote the relative difference from the average upstream planning solution.

List of figures

List of tables

Bibliography

Abdul-Jalbar, B., Guiterrez, J. M., and Sicilia, J. (2007): "An integrated inventory model for the single-vendor two-buyer problem." *International journal of production economics*, 108, pp. 246–258. (Cited on page 17.)

Alcaraz, J. and Maroto, C. (2001): "A Robust Genetic Algorithm for Resource Allocation in Project Scheduling." *Annals of Operations Research*, 102, pp. 83–109. (Cited on pages 80, 81, 83, and 84.)

Atallah, M. J., Deshpande, V., Elmongui, H. G., and Schwarz, L. B. (2003): "Secure Supply-Chain Protocols." In "Proceedings of the IEEE International Conference on E-Commerce," . (Cited on page 28.)

Banerjee, A. (1986): "A joint economic-lot-size model for purchaser and vendor." *Decision Sciences*, 17, no. 3, pp. 292–311. (Cited on pages 16 and 17.)

Banerjee, A., Kim, S. L., and Burton, J. (2007): "Supply chain coordination through effective mulitstage inventory linkages in a JIT environment." *International Journal of production economics*, 108, pp. 271–280. (Cited on page 17.)

Bhatnagar, R. and Chandra, P. (1993): "Models for multi-plant coordination." *European Journal of Operational Research*, 67, pp. 141–160. (Cited on pages 12, 14, and 16.)

Bierwirth, C., Schneider, S., and Kopfer, H. (2002): "Elektronische Transportmärkte: Aufgaben, Entwicklungsstand und Gestaltungsoptionen." *Wirtschaftinformatik*, 44, no. 4, pp. 335–344. (Cited on page 29.)

Bierwith, C. (1995): "A generalized permutation approach to job shop scheduling with genetic algorithms." *OR Spektrum*, 17, pp. 87–92. (Cited on page 82.)

Blazewicz, J., Domschke, W., and Pesch, E. (1996): "The job shop scheduling problem: conventional and new solution techniques." *European Journal of Operational Research*, 93, pp. 1–33. (Cited on page 80.)

Blazewicz, J., Lenstra, J., and Rinnoy Kan, A. H. (1983): "Scheduling projects to resource constraints: Classification and complexity." *Discrete Applied Mathematics*, 5, pp. 11–24. (Cited on pages 69 and 80.)

Bouleimen, K. and Lecocq, H. (2003): "A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version." *European Journal of Operational Research*, 149, pp. 268–281. (Cited on pages 81 and 82.)

Branke, J. and Schmidt, C. (2005): "Faster convergence by means of fitness estimation." *Soft Computing*, 9, pp. 13–20. (Cited on page 62.)

Braun, H. (1997): *Neuronale Netze - Optimierung durch Lernen und Evolution*. Springer. (Cited on page 39.)

Brucker, P., Drexl, A., Möhring, R. H., Neumann, K., and Pesch, E. (1999): "Resource-Constrained Project Scheduling: Notation, Classification, Models and Methods." *European Journal of Operational Research*, 112, no. 1, pp. 3–41. (Cited on page 80.)

Brucker, P. and Knust, S. (2000): "A linear programming and constraint propagation-based lower bound for the RCPSP." *European Journal of Operational Research*, 127, pp. 355–362. (Cited on page 80.)

Cachon, G. P. (2003): *Supply Chain Coordination with Contracts*, vol. 11 of *Handbooks in Operations Research and Management Science: Supply Chain Management*, chap. 6, pp. 229–340. North-Holland. (Cited on page 18.)

Cachon, G. P. and Lariviere, M. A. (2005): "Supply chain coordination with revenue-sharing contracts: strenghts and limitations." *Management Science*, 51, no. 1, pp. 30–44. (Cited on pages 2 and 18.)

Chatterjee, K. and Samuelson, W. (1983): "Bargaining under incomplete information." *Operations Research*, 31, no. 5, pp. 835–851. (Cited on page 19.)

Chen, F., Federgruen, A., and Zheng, Y.-S. (2001): "Coordination mechanisms for a distributed system with one supplier and multiple retailers." *Management Science*, 47, no. 5, pp. 693–708. (Cited on page 17.)

Chiou, C.-C., Yao, M.-J., and Tsai, J. (2007): "A mutually beneficial coordination mechanism for a one-supplier multi-retailers supply chain." *International journal of production economics*, 108, pp. 314–328. (Cited on page 17.)

Cho, J.-H. and Kim, Y.-D. (1997): "A Simulated Annealing Algorithm for Resource Constrained Project Scheduling Problems." *The Journal of the Operational Research Society*, 48, pp. 736–744. (Cited on page 84.)

Clark, A. J. and Scarf, H. (1960): "Optimal Policies for a Multi-Echelon Inventory Problem." *Management Science*, 6, no. 4, pp. 475–490. (Cited on page 18.)

Clark, E. (1971): "Multipart pricing of public goods." *Public Choice*, 11, pp. 17–33. (Cited on page 19.)

Conejo, A. J., Castillo, E., Minguez, R., and Garcia-Bertrand, R. (2006): *Decomposition Techniques in Mathematical Programming*. Springer, Berlin-Heidelberg. (Cited on pages 21 and 22.)

Corbett, C. J. and de Groote, X. (2000): "A Supplier's Optimal Quantity Discount Policy Under Asymmetric Information." *Management Science*, 46, no. 3, pp. 444–450. (Cited on page 17.)

Dahl, S. and Derigs, U. (2008): "Planning in Express Carrier Networks: A Simulation Study." In "OR Proceedings," . (Cited on pages 2 and 28.)

Damay, J., Quilliot, A., and Sanlaville, E. (2007): "Linear programming based algorithms for preemptive and non-preemptive RCPSP." *European Journal of Operational Research*, 182, pp. 1012–1022. (Cited on page 80.)

Darwin, C. (1859): *The origin of species by means of natural selection*. John Murray. (Cited on page 36.)

Dawkins, R. (1976): *The selfish gene*. Oxford University Press. (Cited on page 38.)

Deb, K. (1999): "Non-Linear Goal Programming using Multi-Objective Genetic Algorithms." Tech. Rep. CI-60/98, Dortmund: Department of Computer Science/LS11, University of Dortmund, Germany. (Cited on page 44.)

Deb, K. (2001): *Multi-objective optimization using evolutionary algorithms*. Wiley. (Cited on pages 37, 39, 43, 59, and 110.)

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002): "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II." *IEEE Transactions on Evolutionary Computation*, 6, no. 2, pp. 182–197. (Cited on page 43.)

Debels, D. and Vanhoucke, M. (2005): *Computational Science and Its Applications Ű ICCSA 2005*, chap. A Bi-population Based Genetic Algorithm for the Resource-Constrained Project Scheduling Problem, pp. 378–387. Springer, Berlin / Heidelberg. (Cited on pages 81 and 83.)

Demeulemeester, E., Dodin, B., and Herroelen, W. (1993): "A random activity network generator." *Operations Research*, 41, no. 5, pp. 972–980. (Cited on page 213.)

Dickersbach, J. T. (2003): *Supply Chain Management with APO*. Springer. (Cited on pages 8, 9, and 87.)

Diks, E. B., de Kok, A. G., and G., L. A. (1996): "Multi-echelon systems: A service measure perspective." *European Journal of Operational Research*, 95, pp. 241–263. (Cited on page 18.)

Domschke, W. and Scholl, A. (2008): *Grundlagen der Betriebswirtschaftslehre*. Springer, Berlin Heidelberg, 4 edn. (Cited on pages 41 and 43.)

Drexl, A. (1990): "Fließbandaustaktung, Maschinenbelegung und Kapazitätsplanung in Netzwerken." *Zeitschrift für Betriebswirtschaft*, 60, pp. 53–70. (Cited on page 69.)

Drexl, A., Nissen, R., Patterson, J., and S., F. (2000): "ProGen/πx - An instance generator for resource-constrained project scheduling problems with partially renewable reseouces and further extensions." *European Journal of Operational Research*, 125, pp. 59–72. (Cited on page 213.)

Dudek, G. (2007): *Collaborative planning in supply chains. A negotiation-based approach*. Springer, Berlin. (Cited on pages 2, 26, 29, 159, and 186.)

Dudek, G. and Stadtler, H. (2005): "Negotiation-based collaborative planning between supply chain partners." *European Journal of Operational Research*, 163, pp. 668–687. (Cited on pages 25, 26, 28, 116, 126, 137, and 187.)

Eiben, A. and Smith, J. (2003): *Introduction to Evolutionary Computing*. Springer. (Cited on pages 37 and 39.)

Engelmann, T. (1998): *Prozessflussoptimierung mit evolutionären Algorithmen*. Master's thesis, Institut für angewandte Informatik und formale Beschreibungsverfahren, Universität Karlsruhe (TH). (Cited on pages 76, 77, 81, 83, 94, 100, 104, and 150.)

Ertogal, K. and Wu, D. S. (2000): "Auction-theoretic coordination of production planning in the supply chain." *IIE Transactions*, 32, no. 10, pp. 931–940. (Cited on pages 2 and 24.)

Fan, M., Stallaert, J., and Whinston, A. B. (2003): "Decentralized mechanism design for supply chain organizations using an auction market." *Information Systems Research*, Vol. 14, no. 1, pp. 1–22. (Cited on page 19.)

Feltovich, N. (2000): "Reinforcement-based vs. belief-based learning models in experimental asymmetric-information games." *Econometrica*, 68, no. 3, p. 605Ű641. (Cited on page 20.)

Fink, A. (2004): "Supply Chain Coordinaton by Means of Automated Negotiations." In "Proceedings of the 37th Hawaii International Conference on System Science," . (Cited on pages 2, 12, 21, 28, 29, and 31.)

Fink, A. (2006): *Multiagent based Supply Chain Management*, chap. Supply chain coordination by means of automated negotiations between autonomous agents, pp. 351–372. (Cited on pages 28, 29, and 31.)

Fleischmann, B., Meyr, H., and Wagner, M. (2007): *Supply Chain Management and Advanced Planning*, chap. Advanced Planning, pp. 81–106. Springer, 4 edn. (Cited on page 5.)

Fleszar, K. and Hindi, K. S. (2004): "Solving the resource-constrained project scheduling problem by a variable neighbourhood search." *European Journal of Operational Research*, 155, pp. 402–413. (Cited on page 83.)

Fogel, D. B. (2000): *Evolutionary Computation: Advanced Algorithms and Operators*, chap. Introduction to evolutionary computation, pp. 1–3. CRC Press. (Cited on pages 36 and 37.)

Fogel, L., Owens, A., and M., W. (1966): *Artificial Intelligence Through Simulated Evoluion*. Wiley, New York. (Cited on page 39.)

Foster, I. and Kesselmann, C. (1998): *The Grid: Blueprint for a new computing infrastructure*. The Elsevier series in Grid Computing. Morgan Kaufmann. (Cited on page 154.)

Fransoo, J. C., F., W. M. J., and de Kok, T. G. (2001): "Multi-echelon multi-company inventory planning with limited information exchange." *The Journal of the Operantional Research Society*, 52, pp. 830–838. (Cited on pages 17 and 18.)

Goncalves, J. F., Mendes, J. J. M., and Resende, M. G. C. (2007): "A genetic algorithm for the resource constrained multi-project scheduling problem." *European Journal of Operational Research*, InPress. (Cited on page 84.)

Goyal, S. K. and Gupta, Y. P. (1989): "Integrated inventory modelas: The buyer-vendor coordination." *European Journal of Operational Research*, 41, pp. 261–269. (Cited on page 16.)

Groves, T. (1973): "Incentive in teams." *Econometrica*, 41, pp. 617–631. (Cited on page 19.)

Harsanyi, J. C. and Selten, R. (1972): "A Generalized Nash Solution for Two-Person Bargaining Games with Incomplete Information." *Mangement Science*, 18, no. 5, pp. 80–106. (Cited on page 20.)

Hartmann, S. (1998): "A competitive genetic algorithm for resource-constrained project scheduling." *Naval research logistics*, 45, no. 7, pp. 733–750. (Cited on pages 81 and 82.)

Hartmann, S. (2001): "Project Scheduling with Multiple Modes." *Annals of Operations Research*, 102, pp. 111–135. (Cited on page 83.)

Hartmann, S. (2002): "A Self-Adapting Genetic Algorithm for Project Scheduling under Resource Constraints." *Naval Research Logistics*, 49, pp. 433–448. (Cited on pages 81, 83, and 84.)

Hartmann, S. and Kolisch, R. (2000): "Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem." *European Journal of Operational Research*, 127, pp. 394–407. (Cited on pages 80, 81, and 85.)

Herroelen, W., De Reyck, B., and Demeulemeester, E. (1998): "Resource-constrained project scheduling: a survey of recent developments." *Computers and Operations Research*, 25, no. 4, pp. 279–302. (Cited on page 80.)

Hertz, A. and Kobler, D. (2000): "A framework for the description of evolutionary algorithms." *European Journal of Operational Research*, 126, pp. 1–12. (Cited on page 39.)

Hindi, K. S., Yang, H., and Fleszar, K. (2002): "An Evolutionary Algorithm for Resource-Constrained Project Scheduling." *IEEE Transactions on Evolutionary Computation*, 6, no. 5, pp. 512–518. (Cited on page 83.)

Holland, J. (1975): *Adaptation in Natural and Artificial Systems*. University of Michigan Press. (Cited on page 39.)

Jeong, I.-J. and Leon, V. J. (2002): "Decison-making and cooperative ineraction via coupling agents in organizationally distributed systems." *IIE Transactions*, 34, pp. 789–802. (Cited on page 25.)

Jeong, I.-J. and Leon, V. J. (2005): "A single-machine distributed scheduling methodology using cooperative-interaction via coupling-agents." *IIE Transactions*, 37, no. 2, pp. 136–152. (Cited on page 25.)

Jin, Y. (2002): "A Framework for Evolutionary Optimization With Approximate Fitness Functions." *IEEE Transactions on Evolutionary Computation*, 6, no. 5, pp. 481–494. (Cited on page 62.)

Jozefowska, J., Mika, M., Rozycki, R., Waligora, G., and Weglarz, J. (2001): "Simulated Annealing for Multi-Mode Resource-Constrained Project Scheduling." *Annals of Operations Research*, 102, pp. 137–155. (Cited on pages 81 and 83.)

Jung, H., Frank, C. F., and Jeong, B. (2005): "A production-distribution coordination model for third party logistics partnership." In "IEEE International Conference on Automated Science and Engineering," . (Cited on page 26.)

Kerschbaum, F. and Deitos, R. J. (2008): "Security against the Business Partner." In "Proceedings of the 2008 ACM workshop on Secure web services," . (Cited on pages 27 and 28.)

Kilger, C. and Reuter, B. (2007): *Supply Chain Management and Advanced Planning*, chap. Collaborative Planning, pp. 259–278. Springer, 4 edn. (Cited on pages 12 and 13.)

Kim, K. W., Gen, M., and Yamazaki, G. (2003): "A hybrid genetic algorithm with fuzzy logic for resource-constrained project scheduling." *Applied Soft computing*, 2, no. 3, pp. 174–188. (Cited on page 82.)

Klein, R. (1998): "Bidirectional planning: improving priority rule-based heuristics for scheduling resource-constrained projects." *European Journal of Operational Research*, 127, pp. 619–638. (Cited on page 85.)

Klein, R. (2000): *Scheduling of resource-constrained projects*. Kluwer Academics. (Cited on pages 72, 74, and 80.)

Kolisch, R. (1995): *Project scheduling under resource constraints: efficient heuristics for several problem classes*. Physica-Verlag, Heidelberg. (Cited on pages 71, 76, 80, and 82.)

Kolisch, R. (1996): "Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation." *European Journal of Operational Research*, 90, pp. 320–333. (Cited on pages 81 and 82.)

Kolisch, R. and Hartmann, S. (1999): *Project Scheduling: Recent models, algorithms and applications*, chap. Heuristic Algorithms for Solving the Resource-Constrained Project-Scheduling Problem: Classification and Computational Analysis, pp. 147–178. Kluwer, Amsterdam, the Netherlands. (Cited on page 80.)

Kolisch, R. and Hartmann, S. (2006): "Experimental investigation of heuristics for resource-constrained project scheduling: An update." *European Journal of Operational Research*, 174, pp. 23–37. (Cited on pages 80 and 85.)

Kolisch, R. and Sprecher, A. (1996): "PSPLIB - a project scheduling problem library." *European Journal of Operational Research*, 96, pp. 205–216. (Cited on page 214.)

Kolisch, R., Sprecher, A., and Drexl, A. (1992): "Characterization and Generation of a General Class of Resource-Constrained Project Scheduling Problems." Tech. Rep. 301, Insititut für Betriebswirtschaftslehre, Universität Kiel, Germany. (Cited on pages 213 and 214.)

Kolisch, R., Sprecher, A., and Drexl, A. (1995): "Characterization and Generation of a General Class of Resource-Constrained Project Scheduling Problems." *Management Science*, 41, pp. 1693–1703. (Cited on pages 213 and 214.)

Kutanoglu, E. and Wu, S. D. (1999): "On combinatorial auction and Lagrangean relaxation for distributed resource scheduling." *IIE Transactions*, 31, pp. 813–826. (Cited on pages 23 and 29.)

Kutanoglu, E. and Wu, S. D. (2006): "Incentive compatible, collaborative production scheduling with simple communication among ditributed agents." *International Journal of Production Research*, 44, no. 3, pp. 421–446. (Cited on pages 2, 19, and 23.)

Leon, V. J. and Ramamoorthy, B. (1995): "Strength and adaptability of problem space based neighborhoods for resource-constrained scheduling." *OR Spektrum*, 17, pp. 173–182. (Cited on page 84.)

Li, J. and Atallah, M. (2006): "Secure and Private Collaborative Linear Programming." In "International Conference on Collaborative Computing: Networking, Applications and Worksharing.", . (Cited on pages 27 and 28.)

Li, K. Y. and Willis, R. J. (1992): "An iterative scheduling technique for resource-constrained project scheduling." *European Journal of Operational Research*, 65, pp. 370–379. (Cited on page 85.)

Lu, L. (1995): "A one-vendor multi-buyer integrated inventory model." *European Journal of Operational Research*, 81, pp. 312–323. (Cited on page 17.)

Manne, A. S. (1960): "On the Job-Shop Scheduling Problem." *Operations Research*, 8, no. 2, pp. 219–223. (Cited on page 80.)

Mattfeld, D. C. (1995): *Evolutionary search and the job shop*. Springer Verlag. (Cited on page 82.)

Mattfeld, D. C. and Bierwith, C. (2004): "An efficient genetic algorithm for job shop scheduling with tardiness objective." *European Journal of Operational Research*, 155, pp. 616–630. (Cited on page 82.)

Merkle, D., Middendorf, M., and Schmeck, H. (2002): "Ant colony optimization for resource constrained project scheduling." *IEEE Transactions on Evolutionary Computation*, 6, no. 4, pp. 333–346. (Cited on pages 84 and 143.)

Meyr, H., Rhode, J., Wagner, M., and Wetternauer, U. (2007a): *Supply Chain Management and Advanced Planning*, chap. Architecture of Selected APS, pp. 341–353. Springer, 4 edn. (Cited on page 9.)

Meyr, H., Wagner, M., and Rhode, J. (2007b): *Supply Chain Management and Advanced Planning*, chap. Structure of Advanced Planning Systems, pp. 109–115. Springer, 4 edn. (Cited on pages 6 and 275.)

Monahan, J. P. (1984): "A quantity discount pricing model to increase vendor profits." *Management Science*, 30, no. 6, pp. 720–726. (Cited on page 16.)

Munson, C. L. and Rosenblatt, M. J. (2001): "Coordinating a three-level supply chain with quantity discounts." *IIE Transactions*, 33, pp. 371–384. (Cited on page 17.)

Myerson, R. B. (1979): "Incentive compatibility." *Econometrica*, 47, no. 1, pp. 61–73. (Cited on page 19.)

Myerson, R. B. and Satterthwaite, M. A. (1983): "Efficient Mechanisms for Bilateral Trading." *Journal of Economic Theory*, 29, pp. 265–281. (Cited on page 20.)

Naphade, K. S., Wu, S. D., and Storer, R. H. (1997): "Problem space search algorithms for resource-constrained project scheduling." *Annals of Operations Research*, 70, pp. 307–326. (Cited on page 84.)

Neumann, K., Schwindt, C., and Zimmermann, J. (2001): *Project Scheduling with Time Windows and Scarce Resources*. Springer. (Cited on pages 69, 72, 74, 80, 91, 93, and 94.)

Nie, L., Xu, X., and Zhan, D. (2006): "Collaborative planning in supply chains by Lagrangian relaxation." In "Proceedings of the First International Multi-Symposiums on Computer and Computational Science," . (Cited on pages 11 and 24.)

Perakis, G. and Roels, G. (2005): "The price of anarchy in supply chains: quantifying th efficiency of price-only contracts." (Cited on page 18.)

Pibernik, R. and Sucky, E. (2007): "An approach to inter-domain master planning in supply chains." *International journal of production economics*, 108, pp. 200–212. (Cited on page 26.)

Ponnambalam, S. G., Aravindan, P., and Rao, P. S. (2001): "Comparative evaluation of genetic algorithms for job-shop scheduling." *Production Planning & Control*, 12, no. 6, pp. 560–574. (Cited on page 80.)

Radoport, A., Daniel, T. E., and Seale, D. A. (1998): "Reinforcement-based adaptive learning in asymetric two-person bargaining with incomplete information." *Experimental Economics*, 1, pp. 221–253. (Cited on page 20.)

Radoport, A. and Fuller, M. A. (1995): "Bidding strategies in a bilateral monopoly with two-sided incomplete information." *Journal of Mathematical Psychology*, 39, pp. 179–196. (Cited on page 20.)

Ratle, A. (1998): *Lecture Notes In Computer Science*, chap. Accelerating the convergence of evolutionary algorithms by fitness landscape approximation, pp. 87–96. Springer, Berlin - Heidelberg. (Cited on page 62.)

Rechenberg, I. (1973): *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, Stuttgart. (Cited on page 39.)

Rohde, J. and Wagner, M. (2007): *Supply Chain Management and Advanced Planning*, chap. Master Planning, pp. 159–177. Springer, 4 edn. (Cited on pages 6 and 7.)

Samuelson, W. (1984): "Bargaining under asymmetric information." *Econometrica*, 52, no. 4, pp. 995–1005. (Cited on page 20.)

Schneeweis, C. (2003): *Distributed Decision Making*. Springer. (Cited on pages 11, 12, 13, 14, 15, 44, and 275.)

Schneeweis, C. and Zimmer, K. (2004): "Hierarchical coordination mechanisms within the supply chain." *European Journal of Operational Research*, 153, pp. 687–703. (Cited on page 11.)

Schwefel, H.-P. (1981): *Numerical Optimization for Computer Models*. Wiley, Chichester. (Cited on page 39.)

Schwindt, C. (1995): "Progen/max: a new problem generator for different resource constrained project scheduling problems with minimal and maximal time lags." Tech. rep., Fakultät für Wirtschaftswissenschaften (Fak. f. Wirtschaftswiss.), Institut für Wirtschaftstheorie und Operations Research (WIOR), Universität Karlsruhe. (Cited on page 213.)

Schwindt, C. (2005): "Resource Allocation in Project Management." In Burkard, R., Fleischmann, B., Inderfurth, K., Möhring, R., and Voß, S., editors, "GOR Publications," Springer. (Cited on pages 74 and 75.)

Spears, W. (2007): *Evolutionary Algorithms*. Springer. (Cited on pages 37 and 39.)

Spengler, J. (1950): "Vertical integration and antitrust policy." *Journal of Political Economy*, 4, no. 58, pp. 347–352. (Cited on page 18.)

Sprecher, A., Kolisch, R., and Drexl, A. (1995): "Semi-active, active and non-delay schedules for the resource-constrained project-scheduling problem." *European Journal of Operational Research*, 80, pp. 94–102. (Cited on pages 71, 72, 73, and 74.)

Stadtler, H. (2005a): "Multilevel capacitated lot-sizing problems and resource-constrained project scheduling: an integrating perspective." *International Journal of Productions Research*, 43, no. 24, pp. 5253–5270. (Cited on page 72.)

Stadtler, H. (2005b): "Supply chain management and advanced planning - basics, overview and challenges." *European Journal of Operational Research*, 163, pp. 575–588. (Cited on page 34.)

Stadtler, H. (2007a): "A framework for collaborative planning and state-of-the-art." *OR Spectrum*. (Cited on pages 2, 11, 12, 13, 14, and 19.)

Stadtler, H. (2007b): *Supply Chain Management and Advanced Planning*, chap. Production Planning and Scheduling, pp. 197–214. Springer, 4 edn. (Cited on page 7.)

Sucky, E. (2006): "A bargaining model with asymmetric information for a single supplier-single buyer problem." *European Journal of Operational Research*, 171, pp. 516–533. (Cited on pages 2 and 17.)

Thaler, R. H. (1988): "Anomalies: The Ultimatum Game." *The Journal of Economic Perspective*, 2, no. 4, pp. 195–206. (Cited on page 20.)

Thomalla, C. (1998): *Job Shop Scheduling mit Lagrangescher Relaxation*. VDI Verlag, Düsseldorf. (Cited on page 23.)

Thomas, P. R. and Salhi, S. (1998): "A Tabu Search Algorithm for the Resource Constrained Project Scheduling Problem." *Journal of Heuristics*, 4, pp. 123–139. (Cited on page 16.)

Tormos, P. and Lova, A. (2001): "A Competitive Heuristic Solution Technique for Resource-Constrained Project Scheduling." *Annals of Operations Research*, 102, pp. 65–81. (Cited on page 85.)

Tormos, P. and Lova, A. (2003): "An efficient multi-pass heuristic for project scheduling with constrained resources." *International Journal of Production Research*, 41, pp. 1071–1086. (Cited on page 85.)

Valls, V., Ballestin, F., and Quintanilla, S. (2005): "Justification and RCPSP: A technique that pays." *European Journal of Operational Research*, 165, pp. 375–386. (Cited on page 85.)

Valls, V., Quintanilla, S., and Ballestin, F. (2003): "Resource-constrained project scheduling: a critical activity reordering heuristic." *European Journal of Operational Research*, 149, pp. 282–301. (Cited on page 84.)

Van de Panne, C. (1991): "Decentralizaton for multidivison enterprises." *Operations Research*, 39, no. 5, pp. 786–797. (Cited on pages 22 and 25.)

van Houtum, G. J., Inderfurth, K., and Zijm, W. H. M. (1996): "Materials coordination in stochastic multi-echelon systems." *European Journal of Operational Research*, 95, pp. 1–23. (Cited on page 18.)

Voß, S. (2001): *Local Search for Planning and Scheduling*, chap. Meta-heuristics: The State of the Art, pp. 1–23. Springer. (Cited on page 39.)

Voß, S., Martello, S., Osman, I., and Roucairol, C., editors (1999): *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*. Kluwer, Boston. (Cited on page 39.)

Voß, S. and Woodruff, D. (2006): *Introduction to Computational Optimization Models for Production Planning in a Supply Chain*. Springer, Berlin Heidelberg. (Cited on page 112.)

Walther, G., Schmid, E., and Spengler, T. S. (2008): "Negotiation based coordination in product recovery." *Journal of Production Economics*, 111, no. 2, pp. 334–350. (Cited on pages 2 and 24.)

Wellman, M. P. and Walsh, W. E. (2001): "Auction protocols for decentralized scheduling." *Games and Economic Behaviour*, 35, pp. 271–303. (Cited on page 19.)

Whang, S. (1995): "Coordination in operations: A taxonomy." *Journal of Operations Management*, 12, pp. 413–422. (Cited on page 14.)

Williams, H. P. (1999): *Model Building in Mathematical Programming*. Wiley, 4 edn. (Cited on pages 21 and 22.)

Yao, A. (1982): "Protocols for secure computation." In "Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science," . (Cited on page 27.)

Özdamar, L. and Gündüz, U. (1996): "A note on an iterative forward/backward scheduling technique with reference to a procedure by Li and Willis." *European Journal of Operational Research*, 89, pp. 400–407. (Cited on page 85.)

Zhao, X., Luh, P., and Wang, J. (1999): "Surrogate Gradient Algorithm for Lagrangian Relaxation." *Journal of Optimization Theory and Applications*, 100, no. 3, pp. 699–712. (Cited on pages 22 and 24.)

Ziegler, J. and Banzhaf, W. (2003): "Decreasing the Number of Evaluations in Evolutionary Algorithms by Using a Meta-model of the Fitness Function." In "Proceedings of the 6th European Conference on Genetic Programming, EUROGP," . (Cited on page 62.)

Index