Self-Organizing Transport Networks – Decentralized Optimization based on Genetic Programming

Dissertation

zur Erlangung des akademischen Grades
Dr. rer. nat.
an der Fakultät für Mathematik, Informatik und Naturwissenschaften
der Universität Hamburg

eingereicht beim Fach-Promotionsausschuss Informatik von

Johannes Göbel

aus Hamburg

Einreichung: 03.12.2012
Disputation: 14.06.2013

Betreuer:

Prof. Dr.-Ing. Bernd Page
Modellbildung und Simulation
Fachbereich Informatik
MIN-Fakultät
Universität Hamburg
Deutschland

Prof. Anthony E. Krzesinski PhD
Computer Science Division
Department of Mathematical Sciences
University of Stellenbosch
South Africa

Zusammenfassung

Das Ziel dieser Arbeit ist die Optimierung so genannter *Transport-Netzwerke*, also von Netzwerk-Topologien, in denen Knoten Entitäten „bearbeiten“ und anschließend zum nächsten Knoten weiterleiten, wobei sowohl für das „Bearbeiten“ der Entitäten an den Knoten als auch für das Durchqueren der Kanten Kapazitätsbeschränkungen gelten können. Beispiele für solche Netzwerke sind Stadtverkehr (Autos, ampelgesteuerte Kreuzungen), IP-Netzwerke (IP-Pakete, Router) und automatische Produktionssysteme (Werkstücke, Maschinen). Als Teil von Steuerung und ggf. Optimierung solcher Netzwerke, also z.B. der Minimierung der Wartezeiten der Entitäten, muss insbesondere definiert werden, welche von potenziell mehreren zu einem Zeitpunkt an einem Knoten wartenden Entitäten als nächste „bearbeitet“ und weitergeleitet werden sollen. Diese Entscheidung kann von einem zentralen Server bzw. auf Basis von allgemeinen Regeln getroffen werden, die von einem zentralen Server festgesetzt und bei Bedarf geändert werden; hierfür muss der Server den Zustand des Netzwerkes (z.B. mittlere Flüsse, Verteilung der Entitäten) kennen. Bei Anwendung einer Zentralsteuerung erfolgt die Bearbeitung der Entitäten durch die Knoten typischerweise nicht beliebig flexibel: Für das Handeln der Knoten gelten Einschränkungen, z.B. in Form von sich zyklisch wiederholenden festen Ampelphasen im Stadtverkehr. Dies ist ein notwendiges Zugeständnis, um den Rechenaufwand der zentralen Instanz zu beschränken – andernfalls wäre eine dynamische, zentrale Steuerung nicht möglich.

Im Gegensatz dazu erfolgt die Priorisierung der „Bearbeitung“ der Entitäten in den Knoten eines in dieser Arbeit vorgeschlagenen *selbstorganisierenden Transport-Netzwerks* vollständig autonom und dezentral, also ohne Kommunikation mit einer zentralen Instanz. Im Rahmen seiner Programmlogik zur Priorisierung der Entitäten kann der Knoten somit ausschließlich auf lokal verfügbare Daten zurückgreifen, z.B. auf die Längen seiner Warteschlangen. Eine solche Netzwerksteuerung ist skalierbar und robust. Hinsichtlich der Wartezeiten der Entitäten lassen sich vergleichbare oder sogar bessere Ergebnisse als bei typischen zentralen Steuerungsansätzen erzielen: Die Synchronisation der Knoten erfolgt implizit und das Fehlen globaler Netzwerkzustandsdaten wird durch die zusätzliche Flexibilität der Knoten, die nicht z.B. an feste Schaltpläne gebunden sind, kompensiert.

Abstract

The goal of this thesis is *transport network optimization*. Transport networks are defined as network topologies in which entities are processed by nodes and successively forwarded to the next node on a link. Capacity restrictions on both nodes and links may apply. Examples include urban traffic (vehicles/signalized intersections), IP networks (packets/routers), and automated production systems (items/workstations). The control and optimization of such networks, e.g. minimizing entity waiting times, particularly requires determining which of potentially multiple entities waiting at a node should be processed next. Such decisions or general rules to base such decisions on can be imposed and altered if need be by a central authority based on global knowledge of the network state. In such centralized solutions, entity processing as conducted by the nodes typically is subject to constraints such as fixed signal plans, which are periodically applied in urban traffic; without implementing such constraints, dynamically determining node behaviour would be computationally infeasible.

In contrast, a *self-organizing transport network* as proposed in this thesis relies on decision rules that require local data like traffic densities and current queue lengths as their only input. Genetic programming is used to evolve such rules, which – once deployed – can be autonomously applied by the nodes to dynamically prioritize waiting entities without contacting a central server. Such a decentralized network control is scalable and robust. Furthermore, empirical results indicate that the performance is similar to or even better than centrally controlled systems: among multiple nodes, the synchronization of entity processing is achieved implicitly. The nodes may be unable to base entity prioritization on global traffic data; this, however, is compensated for by the absence of processing constraints such as fixed signal plans.

Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Description	Definition
DESMO-J	Discrete-Event Simulation and Modelling in Java	P. 97
FIFO	First in, first out	P. 11
GA	Genetic algorithm	P. 75
GP	Genetic programming	P. 79
IP	Internet Protocol	P. 2
JGAP	Java Genetic Algorithms Package	P. 78
MANET	Mobile ad-hoc network	P. 139
NaSch model	Nagel-Schreckenberg model	P. 24
OIIC	(Near-)optimal isolated intersection control	P. 67
ONC	(Near-)optimal network control	P. 68
OPAC	Optimization Policies for Adaptive Control	P. 48
Retr	Retrieve operator	P. 95
PI	Priority index	P. 62
RTS/CTS	Request to send/clear to send	P. 140
S1–S6	Scenarios 1–6	P. 107
SCOOT	Split Cycle Offset Optimisation Technique	P. 45
TRANSYT-7F	Traffic Network Study Tool, version 7F	P. 48
VANET	Vehicular ad-hoc network	P. 141

In all considerations referring to urban traffic, without loss of generality, right-hand traffic is assumed. The four points of the compass (N, W, S, and E) are figuratively used to refer to the top, left, bottom, and right of the graphical representation of a network node or topology.

For various transport network prioritization criteria, refer to Table 4.4 on p. 63 for an overview and to Appendix for details.

Acknowledgements

A doctoral thesis is the work of several years and cannot be conducted alone:

First of all, I would like to thank my advisors, Prof. Dr.-Ing. Bernd Page and Prof. Anthony E. Krzesinski. Without their help and knowledge , this work would not have been possible. I am grateful they offered inspirations, discussions, and support whenever needed, while at the same time permitting me to independently develop and pursue my own research approach.

For valuable suggestions and counsel I also thank the other members of the "Modelling and Simulation" research group and closely affiliated persons at the Department of Informatics at the University of Hamburg, including Prof. Dr. Andreas G. Fleischer, Prof. Dr.-Ing. Matthias Riebisch, Dr. Nicolas Denz, Dr. Julia Fix, Dr. Daniel Moldt, Milena Andonova, Philip Joschko, Arne Koors, Sven Kruse, and Claudia Wyrwoll. Finally, I thank Dennis Witt for proofreading and various corrections.

Chapter 1

Introduction

I believe there is no philosophical high-road in science, with epistemological signposts. No, we are in a jungle and find our way by trial and error, building our road behind us as we proceed.

— Max Born
verbally, 1954

This introductory chapter describes the motivation for the decentralized transport network optimization to be conducted in this thesis in Section 1.1, formulates the research targets in Section 1.2, and outlines the structure of the subsequent chapters in Section 1.3. Finally, Section 1.4 briefly mentions previous publications of parts of this thesis .

1.1 Motivation

Transport networks

This thesis focuses on the optimization of an abstract class of networks referred to as *transport networks*. These are defined as graph topologies formed by nodes and links. Entities "appear" at nodes, forming the traffic that the network has to process : to each entity, a route is assigned, determining a sequence of adjacent nodes to pass through ; the last node on the route is referred to as the destination of the entity. All network nodes or only a subset of them may qualify as origins and destinations. Upon arrival at a node, the entities are immediately processed, if the state of the node permits, or queued for processing otherwise. After an entity is processed, it advances to the next link on its route such that it is eventually processed at the subsequent node until it finally reaches its destination, where it is "removed" from the network. Both entity processing at the nodes and link

traversing typically are subject to capacity restrictions, i.e. the nodes and links can handle only a certain maximum number of entities per unit of time. Examples of such networks, later referred to as transport network *types*, include urban traffic in which vehicles advance from one intersection to the next, telecommunication networks in which IP (Internet Protocol) packets are forwarded from one router to the next, and conveyor-based manufacturing systems that process items at workstations.

Performance measures

The focus of this thesis is transport network *optimization*, which may, for example, involve *performance measures*, such as waiting or travel times, minimized or throughput maximized. Observe that optimization focusing on such a notion of performance is not always unconditionally desirable: while maximizing the throughput of an IP network may be a reasonable goal , reducing jams through the use of efficient urban traffic light programming may induce additional individual traffic otherwise deviating to metro transportation, leading to an increase in pollutant emissions. Thus, traffic optimization is no substitute for an appropriate transport policy that provides incentives for public transport, car pooling, walking, cycling, and – where individual transport cannot be avoided – reduction of fuel consumption and emissions. Minimizing waiting time as conducted in this thesis , therefore, represents an exemplarily chosen optimization target; the proposed optimization approach, however, is sufficiently flexible to facilitate other targets as well.

An optimization procedure can be applied only if the system offers a means of influencing its performance by suitably (re-)configuring or adjusting its components; apart from discarding entities or adjusting their routes (which may or may not be feasible, depending on the network type) and long-term improvements to the network topology itself (e.g. increasing the node capacities, establishing additional links), the only degree of freedom for pursuing optimization targets is

Node control

appropriately configuring the *node control logic* (or, in short, *node control*) to determine which entity, i.e. vehicle, IP packet, or item, to process next. The

Entity prioritization

decision of which node control to apply is referred to as *entity prioritization*, e.g. the phases of *traffic lights* (see Figure 1.1) indicating which entities are processed next at an intersection.

The proposed abstract view, i.e. interpreting seemingly unequal application domains, such as urban traffic, IP networks, and automated manufacturing systems, as differently parameterized, special cases of a *transport network*, permits analyzing, simulation modelling, and ultimately optimizing multiple transport networks using the *same approach* and, therefore, particularly transferring principles of optimization from one network type to another, provided that this approach satisfies the following crucial conditions:

- Means of modelling and optimization need be identified that are not restricted to the specific properties of one of the transport network types or,

Figure 1.1 – First traffic lights in Germany: Stephansplatz, Hamburg, 1922 (left, photo from Bar12) and Potsdamer Platz, Berlin, 1924 (right, replication from 1997)

assuming a fixed network type, to certain network topologies (i.e. node/link configurations) or traffic patterns.

- Furthermore, despite *not* exploiting such concrete properties of a transport network, these means have to offer sufficiently valid modelling of the network *and* optimization results that are comparable or even superior to existing optimization algorithms.

This thesis will show that modelling based on *queuing networks* and optimization by using *genetic programming* to evolve node control from its atomic components used by other optimization approaches (i.e. input data, mathmatical/logical functions, and program control operators) fulfills these conditions. Therefore, this approach to optimizing an abstract transport network is *feasible*, at least replicating the results obtained by specialized approaches on a more abstract level.

Queuing networks

Genetic programming

However, an argument still has to be made as to why a unified approach to optimizing a general transport network is *advantageous*: despite the simplicity of applying the same optimization approach to different transport networks, the proposed approach is *decentralized*, exhibiting favourable properties that centralized optimization does not: entity prioritization can traditionally be set up by a *central* authority, which is typically provided with global knowledge of the network state. For example, based on global (estimated) traffic density data, urban traffic signals can be coordinated such that platoons of vehicles are able to traverse the network

Decentralization

without stopping ("green waves") at least on some arterials. However, attempts to centrally optimize such networks typically imply exponential computational complexity [Hol95]; for this reason, the ability of a central authority to supervise all local conditions and to effectively and efficiently incorporate them in an optimization decision decreases with the size of the system and its behavioural variants [MS06]. Therefore, centralized control typically restricts its own decision space, for example, in terms of fixed cycle times in urban traffic, thus trading optimality against computational complexity. Furthermore, reaction to sudden dynamic changes in traffic patterns may be delayed or limited due to the communication to the central authority, its remote decision process, and the reconfiguration of the overall network that may be necessary. A centralized optimization approach fails if the central server or the means of communication between the server and the nodes are not available.

Centralized restrictions

In contrast, decentralized node control logic as proposed in this thesis has to be centrally determined and deployed once; afterwards, data or instructions from a central authority are no longer required. Instead, nodes rely on input data available locally (e.g. queue lengths, traffic density estimations) and decide autonomously which entities to prioritize, yielding response times, scalability, and robustness that typical centralized systems cannot match. However, approaches of decentralized control attempting to (over-)compensate for the lack of global information with the absence of decision space restrictions, faster responses, and robustness to traffic perturbations so far exist only for special cases of transport networks: in the case of urban traffic, for example, the literature suggests means of decentralized node control performing similar to or better than the best centralized solutions or (near-)optimal control determined analytically for some special network topologies, e.g. for Manhattan-like grids with one-way traffic [Ger05] or intersection inflows from different directions always assumed to be mutually exclusive [Läm07]. However, decentralized node control that is applicable to arbitrary network topologies does not yet exist; providing a procedure for establishing such node control will be the goal of this thesis .

Local data

1.2 Goal

The most important goal of this thesis is to propose a means of determining local node (traffic light, router, workstation) control facilitating the effective and efficient *decentralized* optimization of an *abstract* transport network not based on restrictions regarding the network's type, topology, or traffic flow patterns. Without depending on the availability of a central authority and communication to a central server or other nodes, each node, e.g. a traffic light controller as displayed in Figure 1.2, to which such control is deployed, independently decides which entity is processed next, yielding a *self-organizing* transport network: the

Self-organization

nodes' decisions rely on information that is available locally, enabling the nodes to autonomously conduct entity prioritization as long as they are able to obtain these data, e.g. induction loops and cameras and image processing capabilities at an urban intersection measuring queue lengths and traffic densities. Local node control is set such that, despite no agent being responsible for the network as a whole, (global) network performance *emerges* from the behaviour of the nodes, similar to the assumption that a society's prosperity is facilitated by the "invisible hand of the market" emerging from selfish economic decisions of all individuals as in the theory of Adam Smith (1723–1790). Such a self-organizing system becomes able to dynamically *adapt* its behaviour (i.e. entity processing is conducted by the nodes and the resulting spatial and temporal entity distributions) to the current environmental conditions [Ash62].

Adaption

Figure 1.2 – A traffic light control unit, including the electric power supply, induction loop detector interfaces, the controller itself, a panel displaying the traffic light's current state, and a keypad to manually control the traffic light; photo courtesy of *en.wikipedia* user *Analogue Kid*; see `http://en.wikipedia.org/wiki/File:TrafficControlBox.JPG`.

According to Poo10, Chapter 7, improving the behaviour of an agent (here: a node) in a system can be conducted in three ways, also, of course, including combinations thereof:

- The agent can do *new things*; its range of potential behaviours is extended,
- the agent can do *things better*; the quality of its behaviour is improved, and
- the agent can do *things faster*; an existing behaviour requires less time or computational resources.

While not providing new abilities to the nodes of the network, the second and third ways do apply: decentralized network optimization is conducted by

determining node control logic that performs better in terms of e.g. expected entity waiting times and requiring fewer computational resources.

As already mentioned in the previous section, node control will not be determined analytically but by applying genetic programming to heuristically assemble means of node control from the building blocks (i.e. input data, mathmatical/logical functions, and program control operators) of existing means of centralized and decentralized node control, requiring a *simulation model* to validly evaluate the performance of means of node control produced during the process mimicking evolution in nature. Therefore, the development of an appropriate simulation model of an (abstract) transport network is the second important goal of this thesis .

Simulation model

Beyond concrete means of node control for a specific transport network, which can be determined by applying an evolutionary search based on genetic programming as proposed in this thesis , this thesis can also be understood as a case study of engineering a self-organizing system; this is a task for which a generally applicable methodology does not yet exist [Che09, pp. 21ff] apart from the basic principles that the literature recommends applying, e.g. analytically determining component behaviour or trial -and-error procedures, iteratively modifying the behaviour of existing systems serving as archetypes.

Case study

1.3 Organization of the Remaining Chapters

The remaining chapters of this thesis are organized as follows:

- Chapter 2 describes transport network fundamentals, including definitions and explanations of concepts already informally used in this chapter, e.g. nodes, links, entities, flows, and entity processing. Similarities and differences between different types of transport networks are discussed; finally, transport network simulation modelling is addressed.

- Chapter 3 defines self-organization and related concepts and focuses on self-organization as a means to control complex systems, particularly on engineering self-organizing systems. This chapter can be read independently of Chapter 2.

- Chapter 4 discusses existing means of centralized and decentralized transport network optimization, focusing on urban traffic as a transport network type exhibiting the most complex patterns of entity motion. Combining the goal of engineering self-organizing systems (Chapter 3) with transport networks as the application area (Chapter 2), the preparatory steps and necessary components of *self-organizing transport networks* are identified.

- Chapter 5 presents the genetic programming-based approach to evolutionarily determining decentralized, self-organizing node control without restrictions limiting the applicability to a specific transport network type, topology, or traffic flow patterns.
- Chapter 6 describes the transport network simulation model subsequently used to empirically evaluate the performance of the proposed means of decentralized node control.
- Chapter 7 concludes with a summary of the most import results and constraints, followed by a description of open questions and potential extensions left for further research.

1.4 Related Publications

Parts of this thesis can be understood as a generalization of the author's diploma thesis [Göb06a], covering simulation modelling of telecommunication systems as special transport networks and domain-specific means of optimization (bandwidth trading, recovery capacity planning).

Furthermore, the following publications describe the intermediary results of this doctoral thesis: Intermediary results

- Göb08a – Motivation of decentralized transport network node control and description of self-organization basics as covered in more detail in Chapters 1 and 3 in this thesis .
- Göb09a – Extension of Göb08a, also including a discussion of fundamental properties of transport networks and their optimization, corresponding to parts of Chapters 2 and 4.
- Göb11b – Presentation of this thesis' main results, namely decentralized transport network optimization based on genetic programming as described in Chapter 5 and empirically evaluated in Chapter 6.

Chapter 2

Transport Network Fundamentals

The environment is not best conceived solely as a problem domain to be negotiated. It is equally, and crucially, a resource to be factored in the solutions.

— Andy Clark
Being There – Putting Brain, Body, and World Together Again, 1998

As preparation for the transport network optimization to be addressed in the following chapters, this chapter provides detailed definitions of the components comprising transport networks in Section 2.1 and discusses the complexity of predicting and controlling transport networks in Section 2.2. This is supplemented by a description of different approaches to transport network simulation modelling in Section 2.3 and the desirable properties of self-organizing transport networks in Section 2.4.

2.1 Basic Definitions

Chapter 1 named urban traffic, IP networks, and production systems as important examples of *transport networks*. However, discussing transport networks in the following chapters requires a unified ontology to avoid misunderstandings or lengthy statements in which alternatives (e.g. "the traffic light, workstation, or router") must be quoted repeatedly. The definitions and terms proposed below – see also the summary in Table 2.1 on p. 16 – use neutral terms not referring to a specific application domain wherever possible (e.g. node, link, entity). If a

Ontology

general expression does not exist, language will be borrowed from urban traffic: as will be argued below, urban traffic potentially exhibits a more complex behaviour than IP networks or production systems; thus, urban traffic will be the focus of modelling and optimization as conducted in Chapters 4 to 6. Nevertheless, the optimization approach to be developed during this thesis will be applicable to the other types of networks as well. The notion of the properties and behaviour of the different types of transport networks as assumed here is based on Kat06 and Läm07 for urban traffic, Göb06b and Göb09b for wired and wireless IP networks, and Kar05 for production systems.

Transport network types

2.1.1 Transport Network

A transport network – in the literature also referred to as a *switched queueing system* [e.g. Hel07a] or *traffic flow model* [e.g. Bih92] – consists of multiple nodes connected by links. The purpose of the transport network is to forward entities from their origin node to their destination node traversing a *route* specified for the entity. A route is formed by a sequence of nodes, in which the successor of each node is adjacent to the node, i.e. connected by a link. Entities sharing the same origin and destination do not necessarily use the same route.

Route

Examples include urban traffic (traffic light-controlled intersections, roads, vehicles), wired/wireless IP networks (router, links, IP packets), and production systems (workstations, conveyor belts, items).

2.1.2 Entities

An object that the transport network is supposed to transfer from one node to the next until eventually completing its route (see above) by reaching its destination is referred to as an *entity*. Entities' motion may be autonomous (vehicles) or dependent on external facilities (IP packets, items). Entities exhibit a property that determines how quickly they can be processed by the nodes, i.e. after which delay the next entity can be processed; this property, for the purpose of this thesis , is referred to as *length*. The length is not necessarily identical to the entity's physical length, such as in the case of vehicles in urban traffic; it may be measured differently, e.g. an IP packet's data length in bytes or an item's processing complexity corresponding to the processing steps necessary inside a workstation.

Length

Entity *motion* may be constant (e.g. items on a conveyor belt) or subject to autonomous (near-)uniform acceleration up to an entity's or link's maximum velocity (e.g. vehicles); autonomous motion is assumed to avoid collisions with previous vehicles by appropriately reducing velocity if necessary, which includes stopping if need be. Entities may be required to keep a safe distance between each other (vehicles, items). Faster entities passing slower ones may be possible (vehicles) or not (IP packets, items).

Patterns of motion

2.1.3 Links

Links represent connections from one node to the next. Entities traverse links to proceed from one node to another. The maximum distance of a link may be bounded (e.g. Wi-Fi transmission range) . For the purpose of this thesis , links are assumed to be directed; however, in most cases, links exist in both directions. A link may have more than one *lane* (e.g. traffic lane, wireless channels), permitting multiple entities to traverse the link not only sequentially, but also in parallel. Links may impose a maximum velocity limiting entity motion despite entities' theoretical ability to reach a higher maximum velocity (see above). Lane

Depending on the network type, a link may *block* so that no further entities can enter the link; thus, entities have to stay at a node ("spill-back"), preventing the node from processing further entities (see below), e.g. if a link's physical space is completely occupied by vehicles (and safe distances) in urban traffic networks. Links incident to a node are referred to as the node's *incoming* links when directed towards the node and otherwise as *outgoing* links. From the processing capacity of incident nodes (see below), the maximum velocity on a link, and entities' movement constraints (e.g. safe distances), a link's maximum *capacity* can be determined, which is measured, for example, in entities (e.g. vehicles) or entities' length (e.g. bytes) per unit of time. If the traffic offered matches or exceeds the capacity of a link or node, the link or node is referred to as *saturated* or *over-saturated*. Blocking Saturation

2.1.4 Nodes

Apart from acting as sources and sinks of entities (see above), the purpose of the *nodes* of a network is to process entities. *Processing* means temporarily allocating the node's resources, e.g. the space on an intersection or a workstation's internal facilities, to one or more specific entities. Entities' processing durations may differ, depending on their *length* (see above), implicitly defining a node's processing *capacity* by providing an upper bound to the number or overall length of entities that can be processed sequentially per unit of time. Observe that this capacity need not be constant; e.g. vehicles' velocities and safe distances determine how quickly another vehicle can be processed by an intersection or data transmission errors effectively decrease a router's usable bandwidth by requiring IP packets being retransmitted. Processing

Upon arriving at a node, entities are assigned to a FIFO (first-in, first-out) *queue* figuratively also referred to as a *lane* (compare links, see above). In such a queue, the entities have to wait until they are processed by the node. After processing, the entity is removed from the network if it has reached its destination node; otherwise, the entity proceeds on the outgoing link to the next node on its route unless the link is blocked. The latter implies that the entity temporarily Queue Lane

stays at the node, preventing further entities from the relevant queue from being processed until the entity is able to depart.

Typically, nodes dispose of multiple queues; the reason for providing more than one queue is either gathering entities which are supposed to receive similar treatment or independently conducting parallel processing (see below). "Similar treatment" may, for example, mean vehicles pursuing the same spatial trajectory since incoming and outgoing links are identical in urban traffic or a batch of specific items receiving identical processing steps in a production system, not incurring setup time unless a workstation switches to a different item type. Entities to be treated similarly by a node are referred to as belonging to the same *flow*. To each flow – consider e.g. a flow of vehicles reaching an urban traffic four-way intersection from the south (i.e. bottom) desiring to turn left; see Figure 2.2 (left) – one or more queues may be assigned, in which entities wait for processing independently of the entities associated to other queues. This includes multiple flows potentially sharing a queue. If more than one queue is assigned to a flow, the node or the entities may choose which queue to use, e.g. vehicles in urban traffic typically tending towards a lane where the fewest vehicles are queued. Figure 2.1 for example shows three alternative means of allocating flows to queues in urban traffic. Different queues, each associated with one or more flows, may exist physically, e.g. spatially separated lanes at an urban intersection, or virtually, representing a logical structure into which the entities, which are stored in the same physical queue (IP packet buffer), are inserted.

Flow

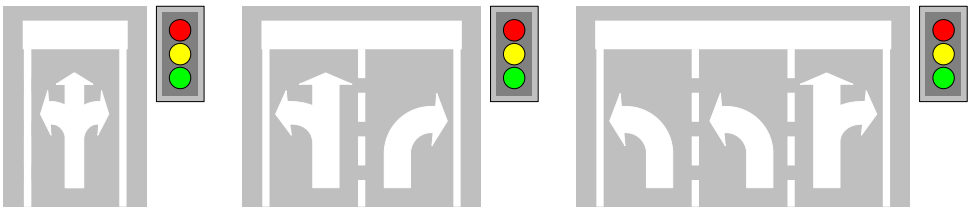

Figure 2.1 – Flows at urban traffic intersections: examples of a single queue shared by all flows from one direction (left) and flows being distributed between two (middle) or among three queues (right), the latter including more than one queue assigned to the flow of vehicles turning left

The number of entities per unit of time expected in a flow is referred to as *flow rate* or intensity. In contrast, the *feasible* or *potential flow rate* (potential flow) describes the number of entities that can be served per unit of time during the next interval with regard to the current state of the node; a positive potential flow rate requires a "supply" of queued entities or imminent arrivals and an unblocked outgoing link. A node may be able to process entities from different queues in parallel, even if such entities belong to different flows. However, depending on the parametrization of the network, processing of some (or all) flows may be mutually exclusive as their processing requires the same resources, e.g. crossing each other in urban traffic.

Exclusive processing

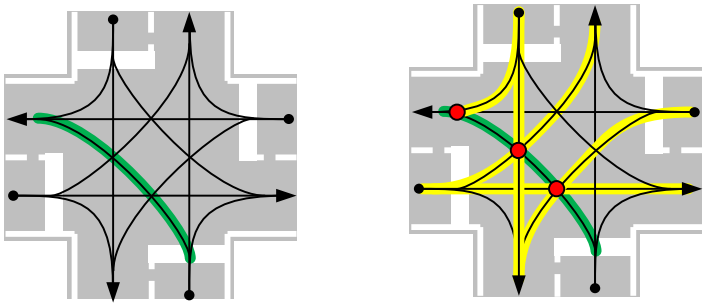

Figure 2.2 – Flows at urban traffic intersections: protected turning from south to west (left) constraints processing other flows simultaneously (right)

In the example of Figure 2.2 (right), the highlighted urban traffic flows cannot be served when vehicles from the south are permitted to turn left[1]. Which precise flows can be processed in parallel depends on the configuration of the network and the node, e.g. on intersection layouts as exemplarily depicted in Figure 2.3.

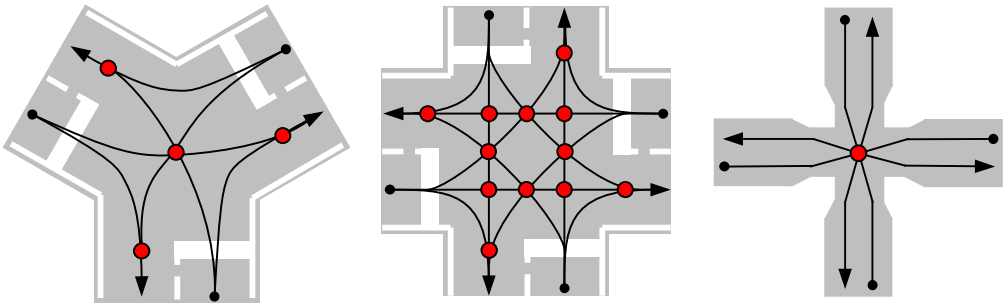

Figure 2.3 – Flows at urban traffic intersections: a three-way (left), four-way (middle), and narrow four-way (right) intersection, in which different flows are mutually exclusive

Control

Assuming that entity motion is either passive (IP packets, items) or autonomous (vehicles), the decision space to optimize the transport network, e.g. to minimize entity waiting times or maximize throughput, is limited to adjusting the nodes' behaviour as follows:

- *Accepting entities.* Nodes may be able to discard entities on arrival or when they are queued for processing, e.g. if a buffer is full in IP networks or if a manufacturing workstation detects a faulty item. Urban traffic intersections cannot discard vehicles.

[1] This thesis assumes right-hand traffic.

- *Processing prioritization.* After processing one or more entities, nodes decide which entity to process next. As the queuing disciplines of all lanes are assumed to be FIFO, it suffices to determine the queue or queues (see above) to serve next. For simplicity, this thesis assumes that all queues assigned to a flow have to be served simultaneously, i.e. either all such queues are served or none[2].

Conditional processing

This text refers to a flow as disabled or its status set to "red" (or, in short, the flow set to "red") if processing is not permitted. Conversely, a flow is considered enabled or set to "green" at a certain instant in time if processing is permitted on the flow, sometimes also referred to as *protected* processing to distinguish it from *unprotected processing*; depending on the network type, flows may also be set to "yellow"[3] (alternatively referred to as "orange" or "amber") if processing is possible in case certain conditions are fulfilled, e.g. the absence of opposing traffic allowing so-called unprotected left turns in urban traffic. In case multiple flows share a single lane (see Figure 2.1), such flows are typically set to "green" or at least to "yellow" simultaneously; otherwise, an entity belonging to a flow set to "red" at the head of a lane prevents the processing of entities queued behind it despite the fact that these entities' flows are enabled.

Phase

A period during which the status of all flows is set to either "green", "yellow", or "red" does not change is called a *phase*. In case the same sequence and duration of all phases is repeated indefinitely , the processing strategy is referred to as operating on *fixed cycles*. The *cycle length* is the period required to execute a single repetition of such a fixed-phase sequence; each flow should receive "green" or "yellow" at least once per cycle. Typically, the cycle lengths of all nodes are synchronized in such strategies, i.e. identical to or multiples of a fixed base cycle. The distribution of the cycle length among different flows is referred as *cycle split*.

Figure 2.4 shows the *two-phase model* of fixed-cycle traffic light control, which can be applied to three- or four-way intersections: the node alternates between two phases. In phase one, the flows from N and S (i.e. top/bottom[4]) are enabled with left turns unprotected, while the flows from W and E (i.e. left/right) are disabled, and vice versa in phase two. Multiple traffic lights in a two-phase arterial are

[2]Note that, for example, serving only one out of two lanes proceeding straight ahead at an urban traffic intersection not only poses a safety risk, as vehicle drivers may attempt to accordingly switch lanes; it also means wasting capacity since, in typical intersection layouts, no other flow benefits from serving only one of two parallel lanes, e.g. side road traffic cannot be processed anyway. Nevertheless, all optimization considerations as conducted in this thesis , particularly priority index-based node control (see Figure 4.5 on p. 65) can be analogously applied to lanes instead of flows in case this is appropriate in a specific application domain.

[3]This should not be confused with a short "yellow" phase indicating the imminent end of a "green" period, which is not explicitly modelled here.

[4]The four points of the compass (N, W, S, and E) are henceforth figuratively used to refer to flow or link directions at a node or a network. In graphical representations, top, left, bottom, and right are associated with these directions.

typically configured such that, in at least one direction, high-density sequences of vehicles (henceforth referred to as *platoons*), in which the distance from one vehicle to the next is small, are able to traverse multiple intersections without stopping, forming a so-called *green wave*. In the general case, i.e. an irregular topology, green waves can be established in one direction on an arterial only [Coo08].

Green wave

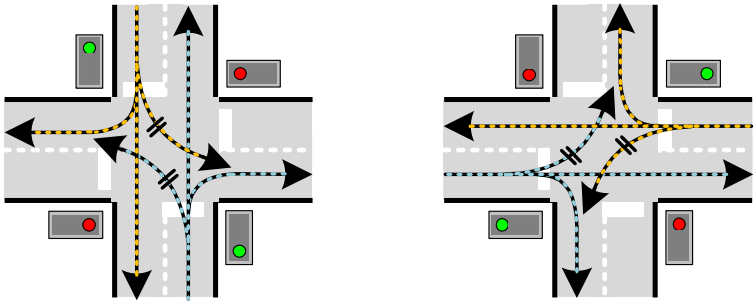

Figure 2.4 – The two-phase model of fixed-cycle traffic light control: phase one (left) and phase two (right)

Processing entities may imply setup or recovery periods during which no entities are processed on a flow. Switching phases, i.e. changing which flows are set to "green"/"yellow" and which to "red" may imply additional setup or safety periods; e.g. in urban traffic all flows are set to "red" for a short interval.

Safety periods

Finally, entity processing may be subject to network-specific constraints, e.g. a *wireless* telecommunication network not strictly matching the abstract view of a transport network node autonomously deciding which entity to process next as described above: the network requires two nodes (i.e. sender and receiver) at the same time taking part in "processing", while processing at other nodes within transmission range of either sender or receiver is *remotely blocked* (see Figure 2.5) since such nodes are temporarily unable to transmit or receive successfully [see Göb11a, p. 332]. Additionally, as already mentioned, packets may be lost due do data transmission errors, e.g. due to wireless interference [see e.g. Göb09b, pp. 2429ff], making retransmissions necessary.

Remote blocking

2.1.5 Summary

Table 2.1 compares IP networks, production systems, and urban traffic with respect to important elements of the ontology described above as well as typical constraints of the entity motion and the nodes' decision space.

The *optimization* of node control as to be conducted during this thesis will address entity prioritization, which is mandatory in any transport network, while

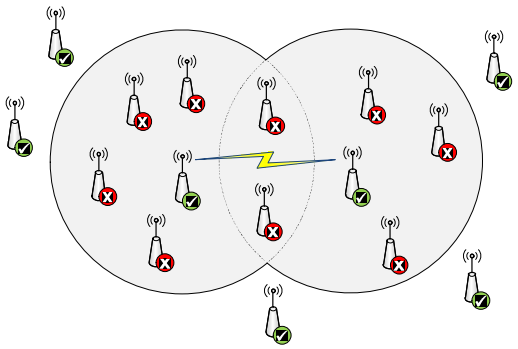

Figure 2.5 – A wireless transmission remotely blocking other nodes inside the sender's and receiver's transmission ranges

Table 2.1 – Different types of transport networks: fundamental terms and important constraints

		IP network	**Production system**	**Urban traffic**
Ontology	Node	Router	Workstation	Traffic light
	Link	Wired/wireless link	Conveyor belt	Road
	Entity	IP packet	Item	Vehicle
Constraints	Entity motion	(Near-)Infinite acceleration	Finite acceleration	Finite acceleration
		Uniform motion	Uniform motion	Non-uniform motion (e.g. different velocities, safe distances), conditional motion (e.g. unprotected turning)
		No overtaking	No overtaking	Overtaking
		Packet loss (transmission errors, wireless interference)	No loss	No loss
	Setup times	Potentially (wireless only, e.g. energy constraints)	Potentially	Yes (safety periods)
	Node decision space	Resource allocation (prioritization)	Resource allocation (prioritization, potentially in parallel)	Resource allocation (prioritization, potentially in parallel)
		Able to discard packets	Able to discard items	No vehicle discarding
		Remote blocking (wireless only)	Local decision	Local decision

the decision to accept or discard entities is not available in general[5]. As claimed above, urban traffic networks exhibit the most complex behaviour; optimization potentially has to take parallel processing, setup times, protected versus unprotected turning, and irregular entity motion in terms of overtaking and non-uniform velocities into account, the latter particulary yielding node and link capacity deviation as they are functions of velocity and velocity-dependent safe distances, which only partially applies to the other transport network types.

2.2 Complexity

With an ontology to name and describe the components of the system referred to as transport network, the question becomes how to *control* such a system and particularly how to predict and modify (i.e. optimize) the nodes' and, therefore, the system's behaviour as desired. Control

In classical or Newtonian physics, understanding a system's behaviour is based on the following assumptions [Hey90, pp. 434ff]:

- *Reductionism.* Decomposing a system into its components (nodes) and analyzing their properties and behaviour permits the inferring of an understanding of the system as whole. Consequently, potential errors in the description of the behaviour of the systems can be explained by overlooking or misinterpreting the behaviour and properties of the components.

- *Determinism.* The dynamic development of the system, i.e. the trajectory it takes through the space of potential states over time, can be uniquely predicted (future) and traced back (past) since its development is unambiguously defined by deterministic laws. Should such predictions or traces turn out inaccurate, errors are due solely to state information that is missing or incorrectly measured.

- *Comprehensibility.* A fundamental, true understanding of a system exists and can be identified by extensive observations. Means of observation are assumed that themselves do not influence the state of the system. Wrong conclusions are due to incomplete observations, yielding an understanding that is not yet sufficiently detailed.

- *Rationality.* Humans interact with a system such that their individual utility is maximized; their behaviour, thus, is deterministically predictable, too.

Modern physics acknowledges systems exist in which these assumptions do not hold; e.g. a true understanding of matter has to consider behaviour that

[5] However, the approach to node control developed in this thesis can analogously be extended to (also) decide about accepting entities if supported by the transport network; see Section 7.2.4.

is partially wave-like and partially particle-like, which is mutually exclusive, or uncertainty principles on atomic and subatomic scales as described by quantum mechanics postulating the absence of determinism [Hey90, pp. 439ff].

Complexity

So-called *complex* systems are another class of systems that do not necessarily fulfill these assumptions [Ger07, pp. 11f]. For complexity (from Latin *complexus*, translating to encircling, embracing, or grasping), no generally applicable definition exists, as any notion of complexity depends on the system under investigation [BY97, pp. 292f]; measurements of a system's complexity proposed in the literature typically aggregate its number of components, the degree to which feedback loops and potential interactions between two or more components exist, and the complexity of the components themselves, which is determined recursively [Ger02]. However, a means of determining a critical bound that the complexity of a system has to exceed in order to be referred to as *complex* is missing: any notion of complexity is subjective and depends on the observer [Nor11, p. 2f].

Alternatively, non-constructive definitions of complexity rely on, for example, the length and structure of a textual description of the system's behaviour [BY97, pp. 199ff] or explicitly require the violation of the assumptions stated above as sufficient property of a complex system, particularly if the behaviour of the system as whole cannot be reduced to its components [Jos00, p. 70]. Controlling complex systems is intrinsically difficult, as any interaction (feedback, design, modification) with the system may yield unexpected results; a complex system may behave significantly different when the same interaction is repeated when a system is in an (almost) identical state [Ger07, p. 19f].

Complex Adaptive Systems

Characteristic properties

John H. Holland (1929–*) argues that urban traffic in New York – as an example of a transport network – belongs to a subcategory of complex systems that he refers to as *complex adaptive systems* [Hol95, pp. 41ff]. Such systems' complexity is driven by the four following characteristic properties:

- *Aggregation of local interactions* [Hol95, pp. 10ff]. Contradicting reductionism, the complexity of a network *cannot* be expressed as a sum of the complexity of its parts, not even by approximation [Hol98, p. 184]: the system potentially exhibits highly sophisticated behaviour even if each single component's (i.e. node's) actions are determined by simple, stereotyped rules. The behaviour of the transport network as whole in terms of, for example, expected travel times, throughput, or queue lengths at bottlenecks typically cannot be directly deduced from the local node behaviour, unless specific conditions apply, e.g. uniform arrivals, fixed-time processing, and no link or node saturated; see Läm07, pp. 50ff. Conversely, feedback loops from the aggregate level to the individual parts dynamically further increase the complexity of the system. Such feedback loops combined with non-linear

behaviour and implicit communication (see below) permit at least limited means of *adaption*, i.e. dynamically adjusting each node's and, therefore, the system's behaviour to the current environmental conditions [Hol98, p. 184]. Adaptivity

- *Non-linear flow behaviour* [Hol95, pp. 15ff]. The performance of the transport network depends on how efficient vehicles, IP packets, items, or other entities "flow" through the network. Due to feedback loops, the impact of flow densities and node behaviours on the performance of the network may be non-linear; e.g. a small local change, such as increasing the ratio of a node's capacity dedicated to entities from a certain incoming link at the expense of the other links, may yield significant changes to the network's performance that are difficult to infer from similar system configurations, comparable to non-linear effects in weather forecasting: applying linear approximation techniques, e.g. linear regression and real convolution [Nor11, pp. 25ff], does not permit deterministic predictions of the behaviour of the system regardless of how precisely its current state is measured [Hol98, p. 184], referred to as the *law of unintended consequences* by Nor11, p. 7f, to emphasize the fundamental difficulty of forecasting the overall system's behaviour. An obvious example of a non-linear effect in transport network control is a slight decrease in the service times of saturated flows, potentially arbitrarily increasing the entities' expected waiting time as arbitrarily long queues build up. However, non-linear behaviour can also be more subtle, e.g. shifting the "green" phase offsets between different nodes while not modifying their cycle splits may cause a change between smooth network operation and severe congestion, the latter e.g. resulting from processing capacity that cannot be effectively used as the outgoing link is blocked since the relevant queue at the next node is cleared too late. Law of unintended consequences

- *Indirect communication based on flows and tags* [Hol95, pp. 23ff]. Despite not necessarily communicating directly, the network's nodes at least process an indirect means of communication based on *tags* [Hol95, pp. 12f]. Tags may be set explicitly, e.g. attached to the flows' entities and read or modified if need be (e.g. an IP packet's header data)[6]; tags also appear implicitly, e.g. a message "encoded" in the lengths of entity inter-arrival periods. Such tags permit *indirect communication*, which may be difficult to identify from outside the system, particularly if indirect communication is propagated between non-adjacent pairs of nodes, potentially facilitating the anticipation of future state changes otherwise not yet visible to a node [Hol98, p. 185]. Tags Indirect communication

- *Diversity* [Hol95, pp. 27ff]. The nodes of the network exhibit different properties; they may, for example, differ in processing durations, number of

[6] In other complex adaptive systems, yet not necessarily in transport networks, tags may also be deployed in the environment, e.g. pheromones used by foraging ants (see Section 3.1.2).

entities processed simultaneously, and setup times. Furthermore, the traffic (i.e. entities) offered may be subject to perturbations. Therefore, local strategies describing which entities a node should accept and/or process next require the ability to adapt to the specific local state of the node; thus, optimal network control potentially requires *different* cycle configurations between which each network node alternates.

Optimization

The combination of these properties renders traditional approaches of optimizing such "highly integrated and interconnected systems", i.e. determining an efficient node policy for processing capacity allocation based on analytical considerations or by applying heuristic *operations research* techniques, unlikely, if not impossible [Hol95, pp. 93ff]. The comprehensibility of the behaviour of such systems has been achieved only partially and will be subject to future research [Hol98, p. 185]. The solution space of potential node policy configurations is huge, and the fitness function (mapping node configurations to the resulting network performance) has a positive yet limited auto-correlation, potentially containing points of discontinuity; between the extreme cases of *ordered systems* in which marginal local changes cause small changes to the system's overall behaviour and *chaotic systems* in which the effect of minor local changes can hardly be predicted, complex adaptive systems are "systems at the edge of chaos", dynamically maintaining a compromise between stability and flexibility [Kau95, pp. 86ff].

Transport network optimization problem

Typical transport network optimization tasks minimize entity waiting or residence times or maximize throughput or resource utilization[7]. Side conditions may apply to the optimization of such performance indicators, e.g. bounds to queue lengths or maximum waiting times or target times when specific entities have to arrive at their destinations. Targets may also be specific to the type of the network, e.g. minimizing fuel or emissions of hazardous substances in urban traffic.

Queueing networks

Observe that the theory of *queueing networks* as introduced by John D. C. Little (1928–*) [Lit61] and discussed in textbooks such as Bol06, Chapter 7, or Gel98, Chapter 9, provides estimations of transport network performance mea-

[7] The literature partially requires that an *optimization* problem is based on a static domain, in which one solution is picked from a search space maximizing or minimizing a performance measure that can be obtained deterministically; dynamic network control under uncertainty, therefore, may be referred to as an *adaption* task, not as optimization [see e.g. Ger07, pp. 88f]. In a transport network, cases of such informed optimization may occur; consider a manufacturing system in which raw material input, processing times, and all other factors influencing performance may be available in advance for a planning period. However, in a typical urban network, performance may be subject to unforeseeable perturbations (see Section 4.1, particularly Table 4.1 on p. 46). For linguistic simplicity, *optimization* is nevertheless used in this thesis rather than *adaption* to refer to general measurements attempting to improve the performance of a network, regardless of the transport network type.

sures, such as expected throughput, queue lengths, waiting durations, or response times, assuming that entity arrivals are described by *Markov processes* and the choice of which entity to process next is based on an item's priority or FIFO, i.e. processing the item that has waited for the longest time; the focus of queueing theory typically is determining equilibrium states that, in a real system, given a concrete network topology including current node states and entity positions or arrival times, may never be reached [Hol98, pp. 184f]. Conversely, trajectories of transport network behaviour permanently not assuming equilibrium states due to continuous perturbations can even be viewed as a characteristic property of such networks [Hel08, p. 3].

Beyond analyzing idealized equilibrium states, queuing theory therefore provides no recommendations as to how the prioritization at a node should be set or how to distribute the processing of entities from different flows at specific instants in time to facilitate the efficient operation of adjacent nodes or the network in total [Hel08, pp. 3ff]. Such recommendations in practise require the precise state of the network, e.g. traffic densities, queue lengths, node processing capacities, and concrete realizations of future arrivals as predicable as possible. Traditional approaches of *centralized optimization* attempt to gather traffic densities and all other data considered relevant. Based on these data, they determine and propagate (approximations of) optimal node control policies matching the network's state as closely as feasible. However, even in the special case of a transport network in which entities are conserved, i.e. all entities periodically traversing the same nodes such that no entities are created or lost, determining the optimal entity to process next such that the network throughput is maximized given the processing durations for all entities at all nodes is an *NP-complete problem* as proven by Pap99[8]. Accordingly, an algorithm whose run-time requirement (only) increases polynomially with the size of the problem (number of nodes, links, entities) on a deterministic machine has not yet been found and is unlikely to appear: as a consequence, any other NP-complete problem (e.g. the travelling salesman or the Boolean satisfiability problem) could be reduced to entity prioritization in polynomial time and, thus, be deterministically solved in polynomial time as well [see e.g. Kes12, pp. 147ff].

Centralized optimization

NP-complete problem

To keep computation complexity within reasonable bounds, the solution space of centralized transport network optimization was reduced by introducing restrictions such as fixed cycle times; see Section 4.1 for details. Such restrictions are an important reason to apply *decentralized optimization* approaches, in which nodes

Decentralized optimization

[8] The proof even establishes a stronger statement, namely entity prioritization being an *EXP-space-complete problem*, i.e. the problem can solved by a deterministic machine in arbitrary time using no more than $O(2^{p(n)})$ space where $p(n)$ is a polynomial function of n, where n represents the length of a textual description of the network (nodes, links, entities with routes...), *and* all other problems that can be solved by such a machine – which includes all NP-complete problems – can be polynomially reduced to entity prioritization; see Pap99, pp. 294ff.

locally decide to which entities to allocate their processing capacity, aiming to compensate for the lack of knowledge about the global state of the network with flexible phases and immediate response; see Section 4.2.

The reason for this preliminarily outlining of transport network optimization before this area is re-addressed in Chapter 4 is the previous provision of the fundamentals of *self-organization* in Chapter 3, on which decentralized transport network control as proposed in this thesis will be based since the existing means of transport network optimization are considered insufficient. Thus, for the moment, assuming that a means of optimization exists to (presumably) determine efficient means of assigning node processing capacity to entities, the next section proceeds to the *simulation* of transport networks as the basis of empirically evaluating the performance of a network in which a specific means of node control is applied.

2.3 Simulation

Investigations of waiting times, throughput, or other performance indicators of a transport network by actually deploying and testing, for example, a proposed means of node control in a real-world environment may be a time-consuming and expensive task, particularly as such *experiments* with *real* systems incur *real* losses or other disadvantages should the network perform suboptimally.

This encourages deciders to rely on *simulation* studies in "sandbox" environments instead: a proposed transport network is mapped into an adequate simulation *model*, whose dynamic behaviour is analyzed by conducting experiments [Pag05, p. 9]. Models typically are abstractions and idealizations of real (or hypothetical) systems; the time-consuming and expensive task of interfering with real-world systems or even replicating them is not necessary and, thus, avoided. To qualify as "adequate", models need only be sufficiently exact and detailed to permit drawing *valid* conclusions about the real system. Therefore, adequate general-purpose models of a given system do not exist, as the feasibility of simplifications and the level of detail required depends on the nature of the insights and conclusions desired [Pag05, p. 8].

Model

Validity

The literature proposes a variety of transport network simulation models; an important classification is whether entities are modelled explicitly, referred to as *microscopic* modelling, or aggregated into flows, yielding *macroscopic* modelling. Figure 2.6 displays a visual comparison between microscopic and macroscopic modelling. A comprehensive overview of microscopic and macroscopic vehicular traffic models can, for example, be found in May90, Chapters 2–8, or Bun09, Chapters 7–8. Furthermore, in May90, Chapter 12, and Bun09, Chapter 9, generalizations applicable to other network types including telecommunication systems are discussed.

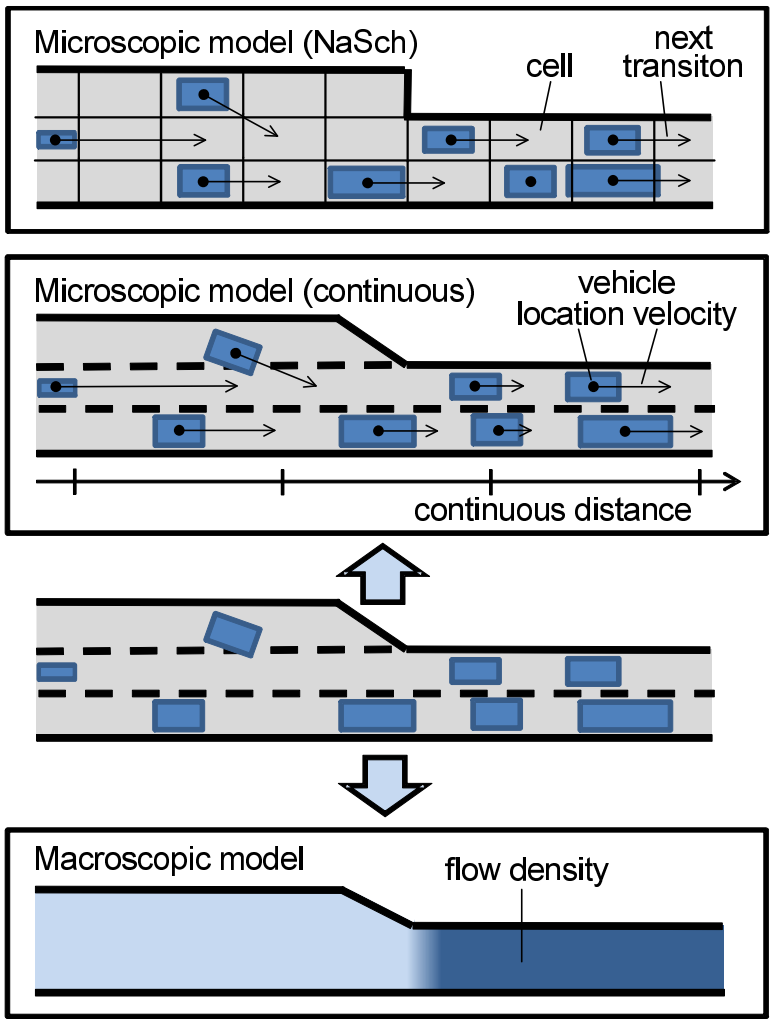

Figure 2.6 – Macroscopic and microscopic vehicular traffic modelling [extension of Bun09, p. 157]; for microscopic modelling, the Nagel-Schreckenberg (NaSch) model [Nag92] is also outlined (top)

Macroscopic Traffic Models

Macroscopic traffic models abstract from individual entities by assuming that the traffic as a whole behaves similarly to the physics of flows of liquids or gases in a system of pipes or current in an electrical network, permitting estimations of traffic densities on the network links based on traffic demand, from which, in turn, all data required for analysis is derived; e.g. in application domain of vehicular highway traffic, Tra00, Chapter 3, empirically proposes the dependence of an expected average velocity u [mph]

$$u = u_f \left[1 - \left(\frac{k}{k_j} \right)^{1.8} \right]^5$$

on free flow speed u_f [mph], current traffic density k [vehicles/lane/ mile], and maximum (congestion) density k_j [vehicles/lane/ mile], which permits the estimation of travel times and the extrapolation of the spatial dispersion of "waves" of different traffic densities over time [May90, pp. 205ff]. The main advantage of such macroscopic models is computational efficiency: the behaviour of a network is described in terms of differential equations in which time is an independent variable; the equations can be solved analytically or at least numerically, the latter yielding *continuous simulation* [Pag05, p. 11] as a typical means of predicting the behaviour of the network over time; the complexity does not increase with traffic densities [May90, pp. 55ff]. Though a variety of modifications and extensions have been proposed, e.g. describing the physics of car-following and the propagation of shock waves caused by a vehicle suddenly braking (see Hel97, Chapters 17–20, for a comprehensive overview), the main disadvantage of macroscopic models is their limited resolution, as they provide only aggregated traffic data such as average velocities and travel times. Furthermore, macroscopic models are valid only where the underlying physical assumptions are adequate, which in vehicular traffic simulation is typically true for highway traffic, but not necessarily for urban areas, where vehicle motion significantly depends on discrete dynamics, e.g. on traffic light phases [Bun09, p. 176f].

Continuous simulation

Microscopic Traffic Models

These constraints give rise to a more detailed *microscopic* modelling approach, explicitly including all individual entities (vehicles, IP packets, items) with all the properties necessary to determine their interactions with the nodes and other entities, e.g. length or complexity , position, velocity, and route, in principle permitting the behaviour of the network to be simulated and empirically evaluated arbitrarily precisely, assuming the near-continuous updating of all properties at intervals that are sufficiently small and requiring the inclusion of conditionals based on discrete state of the network [Bun09, p. 179f], the latter e.g. required to include the statuses of traffic light-controlled intersections. The obvious expense incurred by microscopic modelling is computational complexity; run-time typically increases proportionally to the number of entities whose behaviour and interactions have to be handled individually by applying discrete event [Pag05, pp. 23ff] or time-driven "fixed Δt" [Pag05, p. 11] simulation. Therefore, most microscopic modelling approaches – see Hel97, Chapters 12–13, for an overview – deliberately seek imprecision and thus a reduction of computational complexity by discretely coarsening the resolution of time and/or space, e.g. the *Nagel-Schreckenberg* (NaSch) *model* [Nag92, see also Figure 2.6 (top)] proposing that highway vehicle motion is approximated by the progress of tokens that advance from one discrete cell to another in a cellular automaton at discrete instants. The literature provides various extensions to the NaSch model, including urban traffic

Discrete event simulation

Nagel-Schreckenberg model

with signalized intersections at which turning is either protected or unprotected [e.g. Ros11].

Table 2.2 – Macroscopic, microscopic, and mesoscopic transport network modelling

	Macroscopic	**Microscopic**	**Mesoscopic**
Resolution	Entity flows/densities	Individual entities traced (near-) continuously including position, velocity, route...	Individual entities not necessarily traced continuously
Traffic dynamics	Extrapolation of flows/densities	Evaluation of entity interaction	Limited entity interaction, entity progress partially based on estimations or macroscopic data
Computational complexity	Low: proportional to simulation duration	High: proportional to simulation duration and number of entities	Medium or high: proportional to simulation duration and number of entities
Vehicle-level precision	Low: flow aggregation	High: explicit inclusion	Medium: explicit inclusion, yet less accurate
Application	Flow-level, e.g. vehicular trafffic: highway traffic forecasting and planning (lanes, speed limits...)	Entity-level, e.g. vehicular trafffic: traffic light programming, vehicle tracing...	Compromise: Both flow- and vehicle-level applications, to a lesser extent

Mesoscopic Traffic Models

Table 2.2 summarizes the key properties of macroscopic and microscopic transport network modelling. The table also outlines *mesoscopic* modelling as, for example, conducted in Tan04 and refined in Möl07 and Wit07, which represents a compromise between the macroscopic and microscopic traffic views: entities are modelled explicitly, yet their properties and behaviour are partially determined based on probabilistic estimations including macroscopic data such as (average) flow densities. The run-time requirements of such models are still proportional to the number of entities, yet the computational complexity typically is lower than in microscopic models, which is due to simplifications in the behaviour of the entities, e.g. Tan04 macroscopically determines entity velocities based on traffic densities without respect to the precise position of the vehicles, i.e. expressly disregarding whether or not collisions occur.

Another example of mesoscopic modelling is abstract *queuing networks*; see, for example, Bun09, Chapter 9, and May90, Chapter 12. Such networks' un-

Queuing networks

derlying assumption is that entity progress is largely determined by queueing at the nodes that choose the entities they process next, while motion on links can be validly approximated in terms of relatively simple rules. For example, unless congestion prohibits an entity from entering a link, the entity's arrival at the next node can be sufficiently closely determined by assuming uniform motion on telecommunication links or conveyor belts, while urban traffic can be approximated by near-uniform acceleration up to the speed limit (as far as potential preceding vehicles permit) or a lower velocity such that the vehicle's arrival does not occur earlier than the arrival of the previous vehicle plus a safety interval. In particular, despite not requiring the microscopic spatial position of the entities while traversing a link (i.e. unless queued or processed at a node) to be determined, microscopic data available or measurable at the nodes to base local decisions on, such as queue lengths, can be estimated relatively precisely [Nag03, pp. 5ff]. Therefore, such a mesoscopic queuing network is chosen for the simulation modelling of transport networks for the purpose of this thesis in Section 6.1 since it provides a reasonable tradeoff between validity and precision with respect to the data required on the one hand and computational performance on the other.

2.4 Vision

Simulation modelling of a transport network – as described in the previous section – serves the purpose of providing a "sandbox" environment to investigate the network's performance that this thesis seeks to *optimize*, e.g. empirically evaluating entity waiting times or throughput.

The vision of this thesis is a network in which nodes, independently of each other, apply local decision rules that yield a *self-organizing transport network* without the need for a central authority. As Joh01, Chapter 2, puts it, the gridlock is solved (or whatever the precise optimization target demands; see above) by making the grid itself smart; decentralized, local mechanisms in terms of the rules implemented by the nodes can be viewed as a natural approach to transport network optimization since complexity *arises* locally, e.g. queues building up at one node successively affecting adjacent nodes. In contrast, centralized optimization becomes more and more difficult as the complexity of the system (in terms of e.g. size, heterogeneity, and interactions) increases; thus, robustness and scalability are questionable, apart from centralized optimization implying dependence on a central controller.

Natural approach

As preparation, the next chapter summarizes the fundamental concept of self-organization with a focus on engineering self-organizing systems; then, Chapter 4 discusses the existing means of transport network optimization which provide the building blocks for the development of local rules to base self-organizing network behaviour on.

Chapter 3

Self-Organization

In all things which have a plurality of parts, and which are not a total aggregate but a whole of some sort distinct from the parts, there is some cause.

— Aristotle
Metaphysics, Book 8, 1045a,
approx. 350 BC

Seeking robust and simple mechanisms of decentralized control of complex systems, i.e. permitting such systems' software agents to *organize themselves*, has become a widely-used approach in recent decades in various areas such as the deployment and steering of aerospace vehicles, routing and path finding, storage logistics and production planning, distributed security systems, and traffic management; see, for example, Che09 and MS06 for overviews.

This chapter summarizes the basic concepts of self-organization and emergence in Section 3.1, before proceeding to the fundamental problems of engineering self-organizing systems in Section 3.2, and addressing the strengths and limitations of self-organizing systems in Section 3.3. Finally, Section 3.4 outlines the self-organizing transport networks to be developed.

3.1 Basics

The term *self-organization* is frequently attributed to the work of Ilya Prigogine (1917–2003) [Pri84] in the field of thermodynamics, though it has been used in the theory of corporate and social organizations since at least 1947 by William R. Ashby (1903–1973) [quoted in Ash62]. However, the concept of decentralized system coordination, i.e. a system maintaining *order* based only on its own forces,

thus exhibiting organization from "inside", has already existed for several centuries, consider the work of Adam Smith (1723–1790, "invisible hand of the market"), Charles Darwin (1809–1882, "spontaneous order in nature"), or studies of social insects such as Gra59, for example. Self-organization, as defined by Ash62, means coordinating components of complex systems without requiring global control; even direct interaction between the systems' entities is not strictly necessary: Section 2.2 mentioned *tags* as indirect means of communicating through flows or the environment sufficient to synchronize the system's components.

Absence of global control

The modern notion of self-organization, however, is indeed based on Prigogine's understanding of self-organization [Pri84, Chapters 5–6]: in thermodynamics, non-equilibrium systems organize themselves if they are able to increase or at least maintain order without external control. Depending on the provision of energy, such a system avoids its otherwise mandatory convergence towards disorder since more energy (or matter) is dissipated than received, referred to as *entropy increase*. The requirements of self-organization include a circular feedback relationship in which the system's components causally influence both each other and at least a subset of the system's components is subject to interaction with the outside world.

Maintain order

Observe that these two approaches to describe self-organization do not contradict each other; see Wol05 for a detailed comparison: the first constructively identifies self-organization based on the local/non-global behaviour of abstract system components, such as ants, people, and other biological organisms, facilitating coordination in the absence of centralized control. This allows self-organization being defined as a general organizational phenomenon without referring to thermodynamics or other specific application domains. In contrast, the second approach does not describe behavioural details of how the thermodynamics system achieves self-organization. The system is viewed as a black box, and the definition postulates the properties of the system components and the systems's energy balance, i.e. order maintained based on energy influx, but not on central control [MS06, Section 2]. Recent research goes further; e.g. Kau93, pp. 255ff, interpreted self-organization as a result of a fundamental force assumed to facilitate order, i.e. counteracting thermodynamically induced disorder.

Definition of self-organization

Adaptivity

Through a unified definition that is "compatible" with both views by slightly modifying Wol05, p. 7, self-organization can be understood as a dynamical process in which systems autonomously acquire and maintain order themselves despite external influence and perturbations. Therefore, self-organization is a means of achieving adaptivity, i.e. dynamically modifying the system's behaviour such that it best suits the current environmental conditions.

3.1.1 Self-Organization and Emergence

A concept closely related to self-organization is *emergence* [see Wol05]: a system exhibiting emergence requires at least two hierarchical levels, referred to as macroscopic (macro) and microscopic (micro) levels. Emergence is defined as behaviour at the macro-level that coherently and dynamically arises from the interactions between the system's components at the micro-level to which it causes feedback in return [MS06]. Although the components' interactions at the micro-level can typically be described in terms of *local rules* obeyed by the components of the system in which conditions or *triggers* are defined that, once fulfilled, cause a macroscopic emergent behaviour that is *novel*; i.e. it cannot be directly reduced to the microscopic components' behaviour, typically due to the non-linear effects that their behaviour causes [Joh01, pp. 8f], providing a fundamental unpredictability of the behaviour of complex systems whose parts interact sufficiently strongly, as already observed by Jules Henri Poincaré (1854–1912) [Boy68, Chapter 24]. Which kind of behaviour to consider *novel* and *unpredictable* of course depends on the knowledge and cognitive abilities of the observer; thus, judgement as to whether or not to refer to system behaviour resulting from macro/micro-interactions as emergent is an ontological question [Ger07, p. 31].

Emergence

Local rules

Novel behaviour

However, emergence requires a suitable degree of freedom, i.e. *self-organized criticality* "at the edge of chaos" [Kau93, pp. 209ff]: conditions that are "too ordered" prevent non-linear effects on the micro-level that potentially cause "interesting behaviour" on the macro-level, while conditions that are "too disordered" imply chaotic behaviour; self-organizing systems are characterized by dynamically adopting a compromise between adjusting to environmental *perturbations* and conserving useful structures based on the reinforcement of macro/micro-feedback loops. Environmental perturbations may even play a crucial role in sustaining a self-organizing system since they force the system to continuously adjust to the new conditions, permitting the system to, for example, leave a local optimum that it may have assumed [Kau95, pp. 164ff].

Perturbations

Observe that the literature does not necessarily distinguish between self-organization and emergence at all; e.g. Hel99 and Hey01 consider it one phenomenon by implying that self-organization is necessarily based on mutual macro/micro-feedback loops, which are referred to as emergence. In contrast, MS06 and Wol05 emphasize both of the following are conceivable:

- Emergence without self-organization, i.e. a macro/micro-interdependency in which order is not necessarily maintained; e.g. the volume of gas is an emergent from the number of molecules.
- Self-organization without emergence, i.e. order is maintained without macro/micro-interdependency; e.g. a multi-agent system in which the role of being in command of all system-wide decisions alternates among all agents.

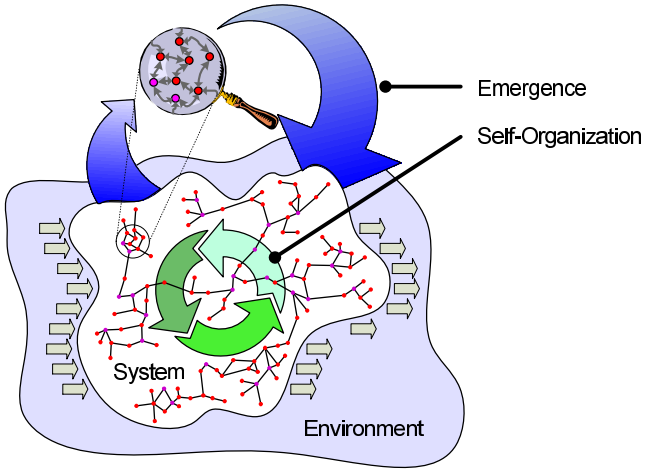

Figure 3.1 – Self-organization and emergence [Göb08a, p. 4]

Self-organization based on emergence

In this view, self-organization *based on* emergence as depicted in Figure 3.1 is interpreted as a special case: maintenance of order relies on macro/micro-feedback loops that dynamically regulate the system; this provides a means of system control that immediately locally adjusts to unforseen changes and perturbations, reacting much quicker than centralized control could redetermine and impose appropriate system behaviour [Ger07, p. 33].

3.1.2 Examples

With the goal of creating a self-organizing transport network in mind, a useful first step is to review examples of self-organizing systems, permitting the identification of local mechanisms facilitating self-organization. In particular, self-organization based on emergence (see above) is of interest here, as local rules have to be established that define the behaviour of the network's nodes, i.e. whether or not to accept an entity and which entity to process next.

Such examples can be found in various scientific areas, including biology, physics, and other natural sciences, social and life sciences, and economics. A comprehensive overview is provided by Tig07. In biological systems, the most well-known examples include [based on Cam01, Part 2]:

- Bird flocking and fish schooling: swarms following simple movement rules (e.g. alignment of position and direction, cohesion maintenance, collision avoidance, occasional exploration/foraging) such that exposure to predators or energy consumption is minimized.

- Social insect foraging: ants, bees, and termites use direct communication or information deployed in the environment to communicate about sources of food, e.g. ants marking paths using *pheromones*. Ants following a path

increase the path's pheromone concentration, while pheromone evaporation ensures that paths tend to be replaced by others if they are unnecessarily long or once the food source is gone.

- Bacterial pattern formation: application of alignment rules to adapt to temperature, moisture, nutrient concentration, and other environmental conditions.
- Sea-shell patterns: the dynamic process of sea shell growth can be algorithmically recreated based on simple rules [see Mei09].
- Evolution itself: the development of different species sustained by energy (sunlight, the planet's resources); feedback loops guarantee that populations tend towards equilibria in which species neither die out nor dominate all other species.

Swarm intelligence

The first three examples can be referred to as *swarm intelligence* [Res97, pp. 139ff] as a special case of self-organization based on emergence: swarm intelligence focuses on the macroscopic collective behaviour typically emerging from multiple kinds of interactions among animals or artificial agents. Also note that rules applied may switch periodically between different phases (see below), e.g. ants either randomly exploring seeking new food sources or exploiting sources to which pheromone trails exist [Bon99].

Patterns of self-organization can also be found in social systems defined by primates' and humans' behaviour [Hem05]: autonomous decisions and behaviour are typically based on "local" information obtained from peers. Complex patterns of decision-making, task-division, dominance interactions, or trend adoption can be described as emergent from relatively simple rules applied by the individuals. The rule of selfish behaviour in a market economy is supposed to maximize the common welfare. Distributed patterns of gossip exchange and trust management use local rules to reliably propagate information throughout a network or to judge the trustworthiness of an entity [Ram04]. Learning can be interpreted as social self-organization preserving a society's knowledge and skills over time [Vyg78].

Moreover, self-organization is not limited to the animate domains: examples from inanimate areas include pattern formation on various scales (molecules, e.g. crystals, as well as stars and galaxies), gas dynamics, laser light coherence, and chemical reactions yielding dissipative structures, e.g. the oscillating *Briggs-Rauscher reaction*. For details, the reader is again referred to Cam01 and Tig07.

A further example from physics is provided by the *Rayleigh-Bénard convection* [Pri84, pp. 142ff], outlined in Figure 3.2 (middle): the rules of fluid dynamics cause hexagonally shaped *Bénard cells* to emerge to efficiently exchange heat once a significant temperature difference exists between the hotter bottom layer of hot water in a pot and the upper counterpart.

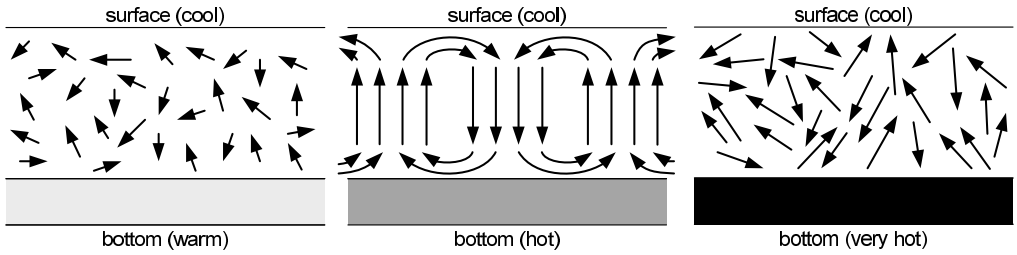

Figure 3.2 – Rayleigh-Bénard convection: phase transitions caused by adjusting the temperature of the bottom plate

Though a specific set of microscopic rules is consistently applied by the components of a self-organizing system (i.e. thermal fluid dynamics), the behaviour of the system as a whole may be subject to relatively sharp changes although the environmental conditions have been adjusted only marginally; such a behavioural change is referred to as *phase transition* [Pri84, pp. 144ff]. For example, Bénard cells, see Figure 3.2 (middle), start appearing at a specific bottom plate temperature that depends on the surface temperature, pressure, container height, and thermal properties of the fluid; this temperature is characterized by an "organized" transfer of warmer, less dense molecules and, therefore, of thermal energy from the bottom to the surface based on the Bénard cells becoming more energy-efficient than uncoordinated, chaotic flows, as shown in Figure 3.2 (left), due to molecule collisions becoming increasingly likely. Similarly, once fluid dynamics prevent the Bénard cells from exchanging a sufficient amount of energy between the bottom and the top as the temperature difference is increased, heat transfer again becomes chaotic *even though* collisions do occur more frequently; see Figure 3.2 (right). Note, therefore, that applying a stable set of microscopic rules may cause different types of macroscopic behaviour (i.e. cells emerging or not); such different *phases* permit the system to adjust to different environmental conditions (i.e. bottom plate temperature levels) such that the target of the self-organizing system (here, efficient energy transfer) is fulfilled. A *self-organizing transport network* such as that to be designed during this thesis will be subject to such phase transitions as well when dynamically adjusting to different environmental conditions like different traffic densities.

Phase transition

3.1.3 Entropy

Stating that self-organization is supposed to increase the "order" of a system requires a notion of *order*. An analogy to thermodynamics can be made, where order and *entropy* are related concepts. Entropy can be interpreted as a measure of the *un*availability of the energy in a system to do work [MS06]. The second law of thermodynamics claims that the entropy or, commonly, "disorder" of an isolated system never decreases, e.g. our universe – if it is assumed to be an isolated

Order

system – inevitably tending towards a maximum entropy equilibrium state in which energy (i.e. temperature) is evenly distributed and processes depending on energy exchange are no longer possible, which is referred to as the *heat death of the universe* [see Haw76].

Note, however, that any notion of "disorder" depends on the viewpoint of the observer [Ger03, p. 610]; consider the bounded environment of two connected boxes containing a gas; see Figure 3.3: from the phase space of all potential gas particle distributions, a configuration in which the spatial gas particle distribution is approximately even (right) is much more likely than an uneven configuration (left). Without energy influx, a high entropy state cannot be converted into a low entropy state[1]. While the high-entropy state can be referred to as more disordered than the low-entropy state when considering the viewpoint of the spatial particle distribution, both states do not significantly differ regarding the movement or velocity vectors, i.e. entropy and "disorder" do not necessarily match.

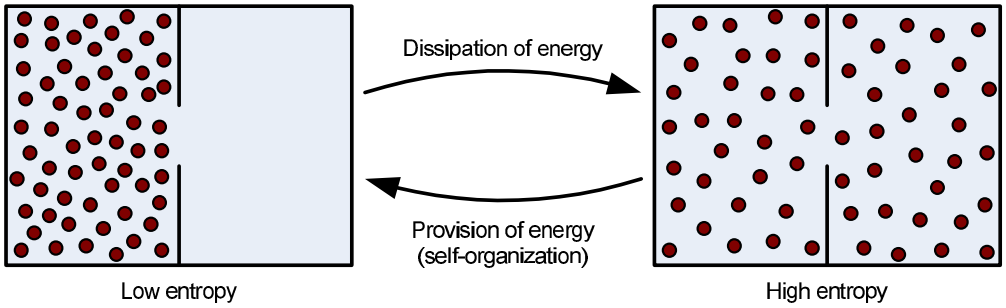

Figure 3.3 – Entropy in gas particle distribution [based on Göb08a, p. 4]

About the generalizations and extensions of the concept of entropy or (where adequate) "disorder" from thermodynamics to other areas, Dav03 provides a detailed overview. One of the first such extensions was introduced by Claude E. Shannon (1916–2001), who proposed applying the notion of entropy to information theory, representing statistical uncertainty in information contents [Sha48], e.g. defining the information entropy $H(X)$ of a discrete random variable X as

Generalizations

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_b p(x_i),$$

where x_i $(1 \leq i \leq n)$ are the possible states of X and $p(x_i)$ their probabilities.

[1] More precisely, this is not *impossible* but extremely unlikely to occur: internal energy (Brownian motion) in principle permits the coincidental acquisition of a state like that shown in Figure 3.3 (left); as this state is unstable (repulsion/pressure), the system quickly returns to a more even distribution like that shown in Figure 3.3 (right).

In a transport network N, entropy $S(N)$ as measurement of (dis-)order ascribed by the observer [Ger03, p. 610] can be expressed in terms of aggregated queue lengths, waiting times, or other criteria, see Table 4.4 on p. 63, e.g. as

$$S(N) = \sum_{i=1}^{n} W(\mathsf{NV\text{-}OQLc}_i),$$

where $\mathsf{NV\text{-}OQLc}_i$ is the number of entities queued at node i ($1 \leq i \leq n$) and W a weighting function that is strictly increasing and differentiable. Despite referring to order and *not* to uncertainty, i.e. the queue states of the network are assumed to be known, this definition satisfies important characteristic properties [see Weh78] of a measurement of entropy in analogy to thermodynamics; in particular, the second law of thermodynamics is fulfilled so that, without an energy influx, i.e. "work" conducted at the nodes, entropy never decreases and the entropy of a network composed of subnetworks can be derived from the entropies of its components; particularly, for two transport networks N, M combined into a larger network $N + M$, the *Shannon inequality* [Lan94, p. 169],

$$S(N + M) \leq S(N) + S(M),$$

holds; i.e. a reduction of the entropy of subnetworks implies a decrease in the entropy of the overall network.

To create a self-organizing transport network, it is necessary to facilitate a network behaviour achieving an entropy reduction to an extent, at least in the long run, compensating for the "natural" entropy increase resulting from entity arrivals at the node by appropriately choosing which entities to process, e.g. in terms of the suitable microscopic control of the system components (i.e. nodes) from which the desired macroscopic behaviour emerges. Self-organization necessarily requires entropy being bounded, i.e. prevented from arbitrarily increasing, which corresponds to an inevitable *self-disorganization*. For that purpose, the next section examines how to engineer self-organizing systems based on different feedback mechanisms permitting self-organization found in examples (see above) in nature and in social systems. Subsequently, the general advantages and problems of engineering self-organizing systems are reviewed.

Self-disorganization

3.2 Engineering Self-Organizing Systems

The more distributed and complex a system is, the less applicable traditional centralized management is [Has06, p. 63]; thus, self-organizing systems today are deployed in many different application areas and research fields; see, for example, Che09 and MS06, as already referenced for various examples from practice. Self-organizing control refrains from centrally explicitly assigning specific algorithmic tasks to the components of the systems, which in transport networks

can be conducted only to a limited (i.e. suboptimal) extent in practice, as will be described in Section 4.1; instead, the components of the system have to be set up such that their behaviour is autonomous and flexible but nevertheless includes feedback loops permitting coordination such that the components are able to jointly produce the desired results [Ger07, pp. 45f]. The introduction of autonomous, decentralized control to software systems, therefore, is necessarily based on identifying *feedback mechanisms* [MS06] facilitating component synchronization as a requirement of self-organization. The following classes of feedback mechanisms leading to self-organization can be distinguished from the examples of self-organization in nature and in social systems (see Section 3.1.2); the list was obtained by consolidating the categorizations found in Mam06, MS06, and Has06; all three also provide additional examples for each mechanism:

Feedback mechanisms

- *Direct interaction.* The components of the system – synonymously referred to as *agents* to emphasize their independent local sensing of their environment and their ability to act autonomously – are allowed to communicate with each other, permitting, for example, orders or position data to be exchanged. Such communication may be triggered by either the source or the destination of a "message", e.g. mobile sensors actively transmitting their positions to other sensors within range [Mam05], while birds observe the position of others while aligning with the flock [Cam01]. Communication flow is either one-to-one or one-to-many (diffusion).
  Among all means of self-organization, this mechanism is best-suited for archiving *pseudo-global control*: applying appropriate broadcasting protocols allows efficient propagation and confirmation of instructions e.g. originating from one node temporarily elected as a leader [Zam04]. Agent behaviour typically is a combination of selfish (i.e. pursuing local targets, e.g. personal prosperity in capitalism) and altruistic behaviour (i.e. accepting mutual agreements and majority decisions even if they are contradictory to local targets).

  Pseudo-global control

- *Indirect interaction.* The components of the system interact indirectly with each other as they are able to create and read *tags* assigned to flows (see Section 2.2) or deployed in the environment, but not to communicate directly with each other. Self-organization relies on the ability of agents to observe such tags and act appropriately. As there is no guarantee when a tag will be recognized (and potentially acted upon), global control cannot be enforced as efficiently as in directly interacting regimes.
  Tags occurring in terms of additional tokens whose only purpose is communicating information are referred to as stigmergy, e.g. ants' pheromone trails (see Section 3.1). Alternatively, tags are implicitly embedded in the environment; in most cases, such tags are intrinsically tied to the task that

  Tags

  Stigmergy

the agent performs, such as structure- or pattern-building on an incremental basis (e.g. morphogenesis) or the order in which items are sorted and processed by an artificial agent in a production system, implying a recommended order to the next agent as well [Mam06, pp. 444ff].

- *Self-reflection and learning.* Agents able to *learn* have the ability to review their own behaviour and dynamically modify it if this is considered advantageous. Beyond the basic means of adaption provided by self-organization based on direct or indirect interaction (see above), in which behaviour is typically determined by selecting the rule applicable in the current situation out of a set of rules, the agent is able to modify the set of rules itself or at least the means of deciding which rule to apply when specific environmental conditions hold. Such a rule modification can, for example, be triggered by reinforcements, in which a feedback in terms of a reward or punishment is assigned to each agent, providing a measurement for his recent performance. After e.g. randomly exploring alternative behavioural means in different states, the agent's future actions can be based on past reinforcements received from previous choices of behaviour when environmental conditions were (perceived as) similar [Wol03].

Reinforcement learning

- *Aggregation and explicit cooperation.* Agents may be dynamically composed and decomposed: an architecture or meta-model that is sufficiently flexible to generically combine and separate agents relies on an appropriate means of (dis-)aggregating agent behaviour, knowledge, and interaction capabilities as needed by the environment.

Engineering a self-organizing transport network requires the choice of one or more of these feedback mechanisms to base node behaviour on; for the purpose of this thesis , *indirect interaction* was chosen for the following reasons: in contrast to *direct interaction*, no communication among the nodes is needed, as demanded by the goal of this thesis (see Chapters 1.2), thus not requiring a suitable communication infrastructure and basing all decisions on locally available data, i.e. not applying pseudo-global control (see above). Unlike *self-reflection and learning*, the proposed approach based on indirect interaction does not rely on the nodes reconfiguring themselves and, hence, improving the rules implemented by the components at run-time: this thesis aims at establishing behaviour in terms of sufficiently complex rule sets to uniformly apply to *all* nodes deployed in a network (see Section 5.1) without the need to "train" the behaviour of each node separately, yielding an "untrained" node joining a network without suitable behaviour until its learning period is completed. Therefore, in transport network optimization as targeted by this thesis , techniques based on self-reflection and learning may potentially provide a means of further improving node behaviour as defined by microscopic rules subject to indirect interaction but no replacement

Pursuing indirect interaction

for (initially) determining the rules to be implemented by the nodes. Finally, *aggregation and explicit cooperation* were not pursued further since they also contradict the task of establishing local (i.e. a single node's) decisions, which do rely on communication among multiple or all nodes. However, a potential exception from not allowing explicit communication are nodes sufficiently close to each other such they can be treated as one node; see Figure 3.4, permitting direct interaction (i.e. synchronization) in a subsystem that is otherwise prone to inefficient behaviour due to a small queueing space[2].

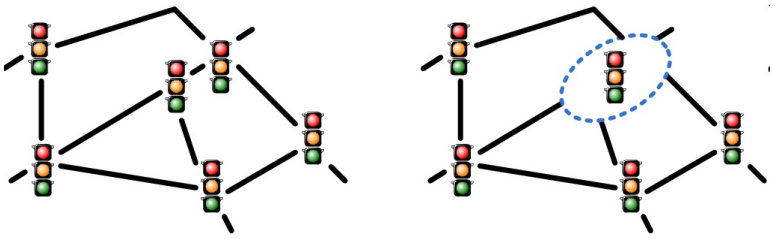

Figure 3.4 – Merging two urban intersections: before (left) and after (right) merging

Furthermore, the indirect interaction-based approach to facilitate transport network optimization proposed by this thesis has to rely on modification of the environment, not on tags assigned to the entities: while it may be possible to use stigmergy in IP networks by adding meta-data to packets and potentially even to items in production systems, e.g. based on radio-frequency identification (RFID) tags, urban traffic networks cannot necessarily "attach" data to vehicles. However, in all networks, the nodes are able to send implicit signals to each other in terms of how many entities are processed and forwarded at an instant in time. For example, a node may interpret a high-density sequence of vehicles, i.e. a *platoon*, approaching on an incoming link as a signal to not stop these vehicles if possible, facilitating the emergence of a *green wave*.

No tag assignment

Once the nature of the self-organizing mechanisms to apply has been determined or at least narrowed, the question arises as to how to engineer – i.e. design, develop, implement, and deploy – a self-organizing software system. Consolidating Elm09, Hol10, and Zam04, the following two different cases can distinguished.

Directly Engineering Self-Organization

If microscopic rules yielding macroscopic behaviour that is at least similar to what is desired are known, self-organization can be *directly engineered* [Zam04, pp. 304f]: implementation of e.g. bio-inspired behaviour in the system's agents is straightforward; see, for example, Göb08b describing how rules describing the motion of flocking birds were applied to mobile sensor deployment; see Appendix A.

[2] As Scenario 2 in Section 6.2 will exemplarily show, such a synchronization can nevertheless still be achieved without merging nodes located close to each other.

Unlike traditional approaches of designing distributed algorithms, synchronization is not necessary: shared memories or global state information apart from local knowledge do not exist; no binding tasks or contracts between two or more agents can be negotiated. Instead, self-organization is achieved by design, relying on the assumption that the local rules as adopted by the system guarantee convergence towards the desired macroscopic behaviour despite potential environmental perturbations.

Self-organization by design

Software development is done with either a "design" or an "evolutionary" approach [Hol10, p. 11], permitting both traditional methods of either *top-down* means, successively decomposing a predefined structure of a system until reaching sufficiently simple tasks that can be executed by components whose behaviour is determined by applying (parts of) the microscopic rules facilitating the system's self-organization, or *bottom-up* techniques, assembling increasingly complex entities out of preliminarily simple agents able to apply single microscopic rules.

Reverse-Engineering Self-Organizing Systems

However, in the general case, no a-priori insight about appropriate microscopic rules to facilitate self-organization exists. Such rules, therefore, have to be determined by *reverse-engineering* microscopic rules from the desired macroscopic features [Zam04, pp. 305f].

As, typically, no knowledge is available as to how the emergence of global system properties can be predicted from sums or other aggregations of local rules, a direct application of both traditional approaches of software development is impossible: *top-down* decompositions fail since global behaviour cannot be converted into the local rules it is based on, while *bottom-up* component assembly requires information regarding how atomic microscopic behaviour (e.g. processing an entity) should be combined.

Evolutionary trial-and-error procedures

This typically leaves the iterative refinement of evolutionary trial-and-error procedures subject to a sufficiently flexible software architecture [Che09, p. 11] as the only viable option: a design defining the behaviour of system components as guessed, heuristically determined, or proven (near-)optimal under special conditions is deployed in a real system or a simulation model so that data collection permits, at least, stochastic evaluation whether desired and undesired emergents do occur, yielding performance measures that provide the basis of the next cycle of iterative rule refinements until the system's ability to self-organize as desired is proven probabilistically [Elm09, p. 42]; Figure 3.5 summarizes this procedure.

A special case of reverse-engineering self-organizing systems applies if omniscient entities already exhibiting the behaviour desired based on global knowledge are available; the subject of reverse-engineering is reduced from the macroscopic to the microscopic level; i.e. the omniscient entity is treated as black box whose behaviour has to be reproduced by other entities, e.g. by applying internal finite

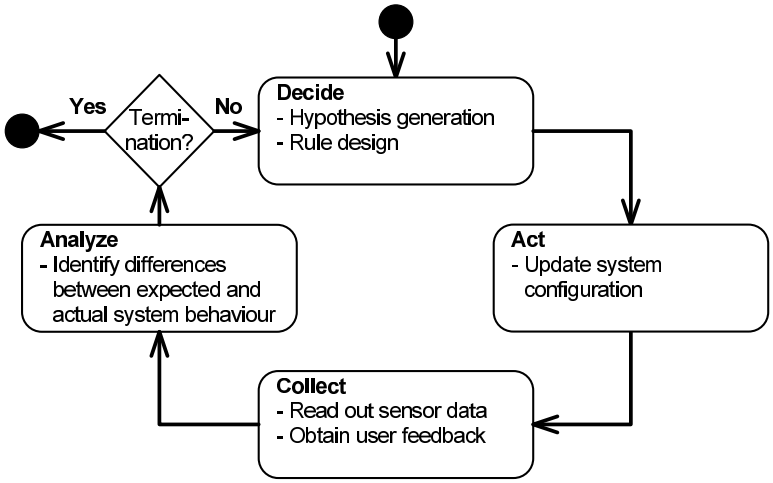

Figure 3.5 – A trial-and-error procedure to iteratively develop self-organizing systems [modified from Che09, p. 14]

state machines attempting to mimic the behaviour of an omniscient entity only based on local information [Elm09, p. 44].

Without suitable rules available to directly engineer self-organization into transport networks, the approach proposed in this thesis will have to rely on the computationally expensive trial-and-error search for reverse-engineering appropriate microscopic rules; see Section 5.1. The trial-and-error search for microscopic rules can be conducted either manually, based on analytical or heuristic assumptions, as e.g. conducted in Ger07, pp. 61ff, or automatically, by e.g. using evolutionary search techniques; see Elm09, p. 43. The latter requires an appropriate description of the search space of potential microscopic rules to be applied by the nodes.

Typically, due to the size of this search space and the existence of (many) local optima exhibiting similar performance, such an evolutionary search cannot be expected to always converge towards the same local rule set [Zam04, p. 306]. Furthermore, rules found by evolutionary search, in most cases, are relatively simple since typical search techniques are designed such that basic rules are more likely to be identified than complex behavioural instructions, i.e. the latter requiring a significant performance increase to "justify" an increase in complexity; see, for example, the case studies of designing efficient peer-to-peer (P2P) systems based on ant programming [Bab02] or spatially arranging service robots [Vas04]. The question of how to describe the search space of transport network control such that it can be effectively traversed by an evolutionary search will be addressed in Section 4.3.

Convergence

3.3 Strengths and Limitations

The previous section mentioned that suboptimal solutions are likely to occur when *reverse-engineering* self-organizing systems; i.e. convergence towards a (theoretical) global optimum is weak [Zam04, p. 306]. Fortunately, this does not render the reverse-engineering of self-organization inappropriate in typical transport network application contexts, as a solution that performs reasonably well may suffice, not requiring a strict optimum at all; conversely, exploring the search space by proposing different microscopic rules may be useful, such as choosing a solution best-suited with respect to side conditions, e.g. emission reduction in urban traffic.

Limited predictability

However, a typical disadvantage of self-organizing systems is that, even on the microscopic level, the predictability of a single "solution" (rule configuration) is limited. In particular, the accurate and efficient operation of the system itself – and thus the alleged ability of the system to *self-organize* – cannot be proven, but only made plausible by exposing the proposed system to different conditions and environmental perturbations [Che09, p. 15].

Bifurcation

Moreover, even if a system is trusted to self-organize, predicability is still limited, as minor smooth changes in environmental conditions may be responsible for the system behaving significantly differently, referred to as *bifurcation* [Kau93, p. 180f]: for instance, assume two large symmetric platoons of vehicles approaching an idle urban traffic intersection from two different directions at approximately the same time. If no other vehicles are present, minimizing the overall waiting time implies completely serving one of the platoons, followed by the other; further switching is not efficient due to the extra safety periods incurred (see Section 4.3). Selecting the platoon to serve first subsequently affects adjacent nodes, as either a large platoon or no traffic at all is offered for a period required to serve the platoon chosen first: small perturbations, e.g. the head of a platoon arriving a second earlier at an intersection than the head of the other, are potentially converted to large-scale changes in the network's state and behaviour. Observe that such an impact of a slight arrival time modification is impossible in, for example, fixed-time traffic light programming (see Section 2.1.4), where the consequences of a platoon arriving one second later are bounded by slightly fewer vehicles being served during the current "green" phase (if arriving while the flow is set to "green"). A self-organizing transport network, however, does not necessarily apply such regular switching patterns. As argued in Section 4.2, not being restricted to fixed cycles is important to match or even outperform typical centralized optimization approaches; nonetheless, the absence of regular switching patterns also implies understanding and predicting the behaviour of the network is more difficult, e.g. vehicle drivers cannot get accustomed to a periodic scheme of traffic light phases, leaving the network more prone to accidents if e.g. a phase is shorter than anticipated by a vehicle driver.

Challenges also include the limited applicability of standard *top-down* or *bottom-up* software development approaches, at least if no predefined microscopic rules are available; see Section 3.2: interfaces successively established at design time require sufficient flexibility not to impose restrictions to the microscopic rules to be determined [MS06, p. 51]. Software development itself may combine top-down and bottom-up aspects through a "meet in the middle" heuristic that attempts to identify useful behavioural patterns between the microscopic and the macroscopic level, which can be shown to be advantageous on the macroscopic level. At the same time, microscopic rules are available to create such patterns or at least not to prevent them from emerging [Zam04, p. 306], e.g. *green waves* in urban traffic based on creating platoons. However, a generally applicable approach to design (i.e. reverse-engineer) self-organizing systems does not yet exist, still requiring comprehensive research with a new theoretical background; see, for example, the research road map as proposed by Che09, pp. 21ff: in contrast to traditional control loop patterns of understanding the dynamics of a systems, the absence or negligibility of non-linear, complex emergent behaviour can no longer be assumed; the system's behaviour typically cannot be directly reduced to its microscopic components, see Che09, pp. 12ff.

Software development

However, once a self-organizing software system has been built, favourable properties are as follows, combining Che09, pp. 3f, and MS06, pp. 45f:

Favourable properties

- *Scalability* due to full local control, not relying on a central authority; thus, the complexity of a single agent's task does not increase with the size of the system.
- *Flexibility* of system behaviour that may be subject to one or multiple targets to which different levels of commitment may be assigned, e.g. mandatory, desirable, or optional fulfillment.
- *Adaptivity* by instantaneously and resiliently adjusting to internal and external changes as rapidly as possible.
- *Robustness*, permitting service to persist based on adaptation despite, for example, partial failures ("graceful degradation"), cases of imperfect information, or even the deliberate malicious behaviour of some system components (nodes).

Beyond the application-level of system operation, another advantage is *information hiding*: every node operates only on local data. Microscopic rules refer to the current state of the node; storing or transmitting data, e.g. describing previous decisions, is not necessary [Zam04, p. 404]. Indisputably, the typical requirement to build a simulation model to evaluate the performance of (presumed) self-organizing microscopic rules [MS06, p. 50] may be a time- and resource-demanding task. In return, however, various benefits and potentials of

Information hiding

Potentials of model building

model building as such are incurred; among others, they include an improved understanding of the system and the ability to conduct experiments and, thus, investigate the behaviour of the system apart from evaluating the performance of microscopic agent rules, e.g. exploring scenarios in which the structure of the system or environmental influences are parameterized differently [see e.g. Pag05, pp. 18f].

3.4 Outlook

Although the precise *microscopic* behaviour of a self-organizing system cannot be reliably predicted, self-organization may be the only feasible approach to effectively (scalably, flexibly, adaptively, and robustly) acquire efficient behaviour in a complex system at the *macroscopic* level, which, however, in principle, does not imply explicitly controlling microscopic node behaviour [MS06, p. 49]; for a transport network, this e.g. may mean minimizing overall entity waiting times, yet being unable to forecast which flow will be set to "green" at a specific instant or how long entities will have to wait on average at a certain node.

The implementation of *self-organizing transport networks* already outlined in Section 2.4, in which nodes autonomously reach local decisions about whether or not to accept entities and which entities to process next – this thesis will focus on the latter, as not accepting (discarding) entities is not feasible in all networks – requires an appropriate software development approach, into which, in particular, a suitable means of determining microscopic node rules has to be embedded. Software architecture should not constrain these rules, which otherwise are not necessarily optimal, permitting these rules to be flexibly chosen such that efficient overall system behaviour *emerges* from the node interactions; the system becomes able to *self-organize* such that desired properties (e.g. minimal waiting times) are maintained or – if subject to environmental perturbations – dynamically reacquired as soon as possible. Before pursuing such an approach in Chapter 5, existing approaches of transport network optimization are reviewed in Chapter 4, addressing their shortcomings.

Chapter 4

Existing Approaches to Transport Network Optimization

Once we accept our limits, we go beyond them.

— Albert Einstein
verbally, approx. 1940

Before undertaking the comprehensive task of developing a new means of decentralized transport network optimization, existing approaches have to be reviewed: conducting this task is reasonable only if it is plausibly justified, e.g. by resolving or at least alleviating restrictions and disadvantages of existing optimization procedures.

As stressed in Section 2.1, the focus of this thesis from now on is urban traffic, which exhibits the most complex behaviour of all transport networks: properties such as safety or setup periods, parallel processing, conditional processing as imposed by unprotected turning, and non-uniform motion due to finite acceleration are not or not necessarily shared by networks in other domains. Unless it is explicitly stated otherwise, the optimization goal is to minimize waiting times. To achieve this target, this thesis focuses on entity prioritization, i.e. determining the entities to process next: note that accepting or discarding entities is not necessarily feasible in all transport networks, including urban traffic. Therefore, "node control" henceforth refers to entity prioritization unless it is explicitly stated otherwise.

Section 4.1 briefly discusses existing approaches to centralized urban traffic optimization, while Section 4.2 investigates the state of the art in decentralized optimization. Both centralized and decentralized approaches provide the "ingredients" of *self-organizing* transport network optimization, as preliminarily described in Section 4.3, heuristically assuming an optimization approach applicable to urban traffic, as the network exhibiting the most complex behaviour patterns of all transport networks also suffices for the optimization of other network types.

Regarding the existing optimization approaches to transport network types other than urban traffic, the reader is referred to e.g. Sha03, Chapters 4–11, for a comprehensive overview of production system optimization, while SR05 focuses on decentralized production system optimization approaches. An in-depth presentation of various telecommunication system optimization problems and approaches can, for example, be found in Res06, Chapters 17, 21, and 26–27, covering different aspects of data transmission performance optimization. This is supplemented by the literature referring to the optimization of abstract queuing networks, e.g. Bol06, Chapters 7–10, as an overview and Hel07a discussing decentralized optimization; the latter is included in the review of decentralized transport network optimization approaches in Section 4.2.

4.1 Centralized Optimization

Attempts to coordinate traffic lights in a network can be traced back to the 1960s, particularly to the work of John D. C. Little (1928–*): the increase in individual transport traffic required more and more intersections to be signalized to efficiently synchronize vehicle flows. Traffic light programming was conducted *offline*, separating optimization as the first step from the second step of static traffic light operation in which the optimization was implemented without permitting dynamic adjustments [Poh10, p. 2]. Typical approaches, as e.g. described in Lit66, are based on the so-called two-phase model (see Figure 2.4 on p. 15); after determining a global cycle length, the ratio of how this period is distributed between the flows from N/S and from E/W at each node, i.e. the *cycle split*, is defined. The cycle split is typically set proportional to traffic counts or estimations of the relevant flows at each node. Protected turning or intersections with more than four incident links were not considered. Once the global cycle length as well as all nodes' cycle splits are established, the only remaining degree of freedom to optimize the network is setting up *offsets* describing when each intersection starts its cycle relative to a reference intersection such that the expected waiting time of the overall traffic is minimized; traffic is assumed to form continuous and uniform flows, abstracting from individual vehicles. Under the condition that the network is not over-saturated (i.e. every node's processing capacity allocated to each flow in a cycle is greater than or equal to the traffic offered), this can be

Offline optimization

formulated as a *mixed-integer linear programming problem*. Though this way of reducing the decision space of an arbitrary means of node control to determine global cycle lengths and local cycle splits and offsets, a mixed-integer linear programming problem is still NP-complete [see e.g. Kes12, pp. 157ff], see Section 2.2, which therefore cannot be deterministically solved in polynomial run-time.

NP-complete

Aggregating global traffic demands and determining the programming of all traffic lights at once by solving a mixed-integer linear optimization problem is a typical *centralized* approach to transport network optimization; its characterizing property is global communication to propagate data such as traffic densities to a centralized authority and the resulting phase allocation in the reverse direction. The central server may be a "real" central server separated from all traffic lights or simply one or more nodes acting as a "virtual" centralized server. Further observe that the real-world performance of traffic light programming as obtained from such an approach is not necessarily optimal, as realizations of vehicle densities and inter-arrival times during a (short) interval deviate from the assumed average continuous/uniform flows that the optimization is based on; for this reason, such a means of traffic light control is also referred to as (only) *near*-optimal to differentiate from a fictitious *optimal* intersection control based on precise traffic data known in advance [see e.g. Pan05, p. 2]. The centralized approach of Lit66, more specifically, is an example of *fixed-time control*, in which a fixed sequence of phases (or a set of different sequences, based on traffic density scenarios typical for e.g. different hours of the day) is applied without responding to potential changes in traffic by adjusting these phases. Various improvements have been implemented in traffic light coordination proposed by Lit66, including a global cycle period no longer potentially suboptimally set in advance but determined during the optimization procedure in which cycle splits and offsets are defined; see Gar75.

Near-optimality

Furthermore, technological advances have permitted the dynamic determination or at least estimation of the current spatial vehicle distribution and directly incorporating this data into optimization, henceforth conducted *online* [Poh10, p. 2]: acknowledging that traffic is subject to dynamic perturbations and changes on various time scales , see Table 4.1, traffic light optimization and operation are no longer strictly separated. For example, most installations of the commercial urban traffic control system SCOOT (Split Cycle Offset Optimisation Technique), see Bre12a, Bre12b, and further details below, use either video surveillance with image processing or induction loops – the latter typically deployed in pairs close to the start and the end of each link – to estimate vehicle positions and queue lengths; see Figure 4.1, on which dynamically repeated optimization is based. Recent technology provides more precise means of traffic observation, e.g. microwave- or sonic-based vehicle detectors [Mim00] or "smart" vehicles automatically communicating their position to nearby intersections [Sun06].

Online optimization

Table 4.1 – Urban traffic perturbations and changes by time scale [extended from Kes09, p. 3]

Time scale	Perturbations and changes	Scope
0.1 s	Drive train, ESP	Single vehicle
1 s	Reaction time and safe distance	Multiple vehicles
10 s	Accelerating and braking	
1 min	Delays caused by buses , vans, and emergency vehicles	Multiple links
	Traffic light cycles	
10 min	Stop-and-go waves, periods of ferries and bascule bridges	
1 hour	Transitions between rush hours and off-peak periods	Network
	Public events such as football games and concerts	
	Changes in weather conditions and daylight	
1 day	Patterns of weekday and Sunday traffic	
1 month	Road work and diversions	
	Seasons and holiday periods	
1 year	Infrastructure building measures	
10 years	City development and demographic changes	
	Economic growth and technological advancements	

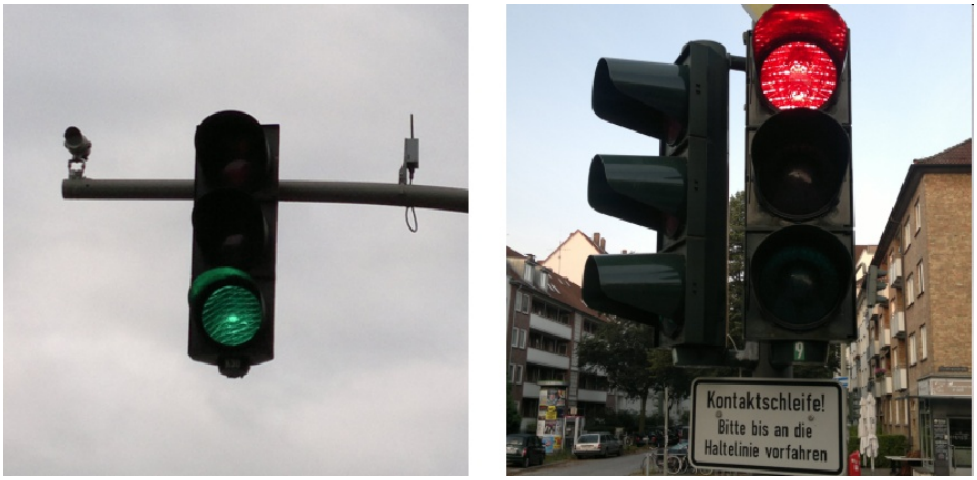

Figure 4.1 – Adaptive traffic lights: video surveillance (left) and a sign advising vehicles on a side road to advance to the stop line as, otherwise, the induction loop will not detect the vehicle, so the "green" phase of the side road will be skipped in favour of the arterial street (right)

Dynamic traffic light control

Regarding how online traffic light programming dynamically adjusts traffic light phases based on the recognition of traffic conditions changing over time, the categories of *responsive*, *actuated*, and *adaptive* traffic light control [modified from Kat06, pp. 227f] can be distinguished:

- Repeating phase allocation at regular intervals based on traffic measurements instead of conducting traffic light optimization only once yields *responsive control*, thus no longer implementing a fixed set of predefined signal plans.
- *Actuated control* can immediately adjust certain traffic light programming parameters from a well-defined decision space in real-time depending on current traffic demand, e.g. extending a single phase at the expense of side road traffic without waiting for phase reallocation at the start of the next optimization interval.
- Furthermore, *adaptive control* flexibly determines traffic light programming as best-suited given current conditions, particularly when it is not restricted by cycle lengths or periodically repeated signal plans[1]; see also the general definitions of adaptivity in Sections 2.2 and 3.1. A special case of adaptive control is *anticipative control* [Ger07, p. 46], in which traffic light programming adapts to future changes that have not yet occurred, but are predicted to occur soon.

As already mentioned, the run-time complexity of centralized near-optimal optimization approaches, such as that of Lit66 based on mixed-integer linear programming, grows faster than polynomially (i.e. exponentially), so such approaches cannot be applied dynamically in practice [Pap03, p. 2049], yielding a need to improve computational performance. Progress in this area is limited; see, for example, Imp82 for an equivalent formulation of Lit66 providing a performance increase of a significant factor assuming typical MILP solvers are used or Kat06, p. 2f, for an overview of approaches proposing *branch & bound* and other exhaustive search techniques guaranteeing that the (near-)optimal solution is identified; however, their algorithmic complexity still grows exponentially with the number of nodes and links.

Recent Approaches

Modern approaches of centralized traffic light optimization, therefore, heuristically trade improved run-time performance for the potential to obtain suboptimal solutions by relying on simplifying assumptions, e.g. exploiting properties of a

[1] Observe that the literature and especially advertising may define these categories differently; e.g. SCOOT (see below) is advertised as *adaptive* in Bre12a despite being *actuated* only according to the definition of Kat06, pp. 227f.

Heuristics

specific network topology or not exhaustively exploring all feasible signal plans by applying various *heuristic* search algorithms. Determining phases is e.g. conduced by using genetic algorithms (GA) as a means of parameter adjustment, consider e.g. Foy92 as one of the first approaches, while Geh04 or Gor11 are more recent examples; see Poh10, pp. 18ff, for a review of determining traffic light phases based on genetic algorithms. Other heuristic search techniques the literature proposes to base traffic light configurations on include neural networks [Che12, Nak95], learning classifier systems applying fuzzy logic rules [Bul04, Lee95], reinforcement learning [Mik95, Wie04], and hybrid combinations thereof, e.g. a neural network in which the weights of neurons are adjusted by a genetic algorithm [Roy04].

As an exhaustive overview of contemporary centralized approaches to urban traffic control is beyond the scope of this thesis , the reader is referred to Poh10, Chapter 2, and Woo93: modern centralized means of online urban traffic control

Non-adaptive

are responsive or actuated but not adaptive, as global synchronization depends on restrictions such as fixed cycle times. Despite heuristically no longer exploring the complete search space, basing phase allocation on data dynamically measured in addition to (average) traffic flow, rates such as current queue lengths, yields performance similar to near-optimal means of traffic light control like Lit66 despite

Progress

exhibiting a much lower computational complexity; moreover, their superior runtime behaviour permits further improvements and innovations:

- Proposing protected turning phases where possible and appropriate, e.g. TRANSYT-7F (Traffic Network Study Tool, version 7F); see Luk84.
- Relaxing the requirement to base traffic light programming on periodically repeating fixed cycles, such as
  - skipping a switching cycle so that the main arterial remains set to "green" for the full cycle, typically applied to intersections of arterials and minor side streets, e.g. SCOOT; see Bre12a and Figure 4.1 or
  - dynamic "rolling horizon" optimization in which cycles are still enforced globally, yet their length may be continuously adjusted, e.g. OPAC (Optimization Policies for Adaptive Control); see Che87.
- Inclusion of routing, i.e. joint optimization of traffic light programming and route choice; see e.g. Cey04.
- Optimization with respect to side conditions to be fulfilled apart from minimizing waiting times such as
  - emission reduction, e.g. TRANSYT-7F; see Luk84,
  - minimizing pedestrian waiting times, e.g. SCOOT; see Bre12a,
  - public transport prioritization, e.g. TUG (Traffic-responsive Urban Control); see Dia03, or

  - not permitting the longest waiting times to exceed predefined bounds if feasible; see e.g. Läm07, Chapter 6.

- Distributed algorithms to determine traffic light phases, thus at least partially transferring computation complexity to the local nodes, although still requiring a central authority; see e.g. Hel10.

- Meta-strategies to modify existing means of traffic light control under special conditions, e.g. explicitly determining transition phases for smoothly switching from one signal configuration to another as proposed by Poh10; meta-strategies may also dynamically adjust traffic light programming if the failure of a traffic light has been detected [Let07] or if obstructions e.g. caused by accidents require diversions [Läm12].

Taking SCOOT as an example of one of the most successful commercial systems of dynamic urban traffic control, installed in more than 200 cities and areas in 14 countries [Bre12a], the following restrictions and challenges have not yet been satisfactorily resolved [based on Bre12a, Bre12b]: Restrictions

- Traffic light programming relies on communication to a central authority: in case the central server is down or communication is faulty, affected traffic lights resort to suboptimal fixed emergency signal plans.

- Due to the complexity of the optimization procedure, adaptive real-time optimization is not possible: signal plans are responsively adjusted in intervals of 15–60 minutes (length depending on parametrization and network size), based on the data of the previous cycle. Thus, the reaction to perturbations or e.g. a partial failure of the network's traffic lights is delayed. To facilitate quicker reaction times, SCOOT permits subnetworks to be optimized separately subject to "synchronization contracts" in terms of predefined signal plans deployed at boundary nodes, thus improving run-time performance, but potentially producing inferior results. This allows spatially limited hotspot areas to be re-optimized every five minutes, which, however, is still too long to near-immediately respond to specific local conditions, e.g. emptying a queue that suddenly builds up in a single lane while all other flows at the intersection are idle.

- Traffic light programming is constrained by cycle times: although traffic lights are able to autonomously and dynamically modify the cycle split within certain bounds (i.e. extend a period for a few seconds at the expense of another phase), the cycle length is fixed. Though situations exist where enforcing a globally fixed cycle period is advantageous, such as the propagation of green waves on arterial roads throughout a sequence of intersections where processing capacity is similar (e.g. identical maximum

velocity and number of lanes), fixed cycle times constrain traffic light programming, particularly if the processing capacity is not uniform or traffic patterns are perturbed. Apart from the ability of extended phases on demand, SCOOT relieves this restriction by supporting selected intersections using a multiple of the (default) global cycle time and permitting cycles to be skipped, referred to as *actuated control* according to Kat06, pp. 227f; see above. Nevertheless, signal plans remain constrained in terms of cycles and synchronization contracts: particularly if parts of a network are overloaded, the performance of SCOOT suffers from wasting intersection capacity in terms of unnecessary safety periods that could be avoided by using longer cycles [Pap03, p. 2049].

Centralized traffic control as conducted by SCOOT and similar approaches is responsive and actuated according to the definition stated above, yet it is *not* adaptive; despite significant progress towards relaxing constraints such as fixed cycle times, traffic light programming remains bound to global restrictions and, thus, not necessarily optimal. This motivates the removal of global constraints by attempting *decentralized* optimization approaches as discussed in the next section.

4.2 Decentralized Optimization

Theoretically inferior

Decentralized optimization approaches – in which nodes autonomously decide which entities to process next, relying only on locally available information, not communicating with other nodes or a central authority – are theoretically inferior to centralized optimization approaches, as less information is available to base node control on: any means of decentralized node control can also be adopted by a centralized system by ignoring its ability to communicate globally and, for each node's control decision, disregarding data not locally available at the node, but not necessarily vice versa. However, due to the lower computational complexity of deciding about the control of *one* node, not a network, autonomous local decisions are better suited for real-time control [see e.g. Por96]. This especially permits *adaptive* intersection control to immediately respond to current traffic conditions, no longer necessarily constraining the solution space as follows based on Hel08, pp. 10f:

- Typical centralized, non-adaptive approaches globally use fixed cycle times or multiples of fixed cycle times,
- apply predefined phases at nodes representing the bounds of subnetworks that are optimized separately,
- use unnecessarily long periods to serve significantly more than the average number of vehicles to avoid having vehicles waiting more than a cycle despite inflow variations, and

- conduct green wave synchronization on an arterial in one direction, while opposite, crossing, and merging traffic often is not only disregarded, but potentially even obstructed.

Such optimization procedures, therefore, are "blind" to deviations from the assumed traffic patterns, e.g. a platoon proceeding either faster or slower than originally assumed (e.g. due to deviating from maximum speed allowed) will miss its green wave phase. Moreover, systematically perturbed traffic conditions e.g. caused by football fans leaving a stadium after a match require that traffic agents manually regulate the traffic [Ger07, p. 62].

Conversely, due to the ability to flexibly adjust traffic light programming, adaptive control may outperform near-optimal means of fixed phase allocation [see Ger12, p. 399f]: dynamic, local adaption to the precise conditions at a node, which are typically not known in advance, permits better results than those obtained by basing node control on average flows. For such means of (potentially) superior decentralized adaptive system control, Res97 proposes the term *supraoptimality*[2]. Supraoptimality
Figure 4.2 shows an example of an intersection configuration that would benefit from adaptive traffic light control: since unusually many vehicles attempt to turn left, an adaptive traffic light not bound to periodically repeated plans and fixed cycles would be able to immediately permit protected turning, thus resolving the congestion at the link and also enabling the vehicles not desiring to turn left (e.g. the busses at the rear) to proceed.

Figure 4.2 – From optimal to supraoptimal intersection control

Note that, for the purpose of this thesis , an optimization approach is considered local and decentralized only if no explicit communication with other nodes is

[2]This aims at emphasizing the efficiency of decentralized adaptive control; however, strictly speaking, "supraoptimality" does not exist, as "optimal" already is a superlative: in fact, only a step from *near-optimality* (see above) towards *optimality* is made.

required; in particular, permitting the nodes to exchange information with other nodes inside a limited environment as e.g. in Hel10 or Lee95 still yields the potential to "emulate" the availability of a central server and global communication by deploying suitable distributed protocols of leader election and information propagation [see e.g. Lyn96, Chapters 14–15], resulting in *pseudo-global control* based on direct interaction, as referred to in Section 3.2: the set of all means to control a node, i.e. determining entity prioritization, subject to communication with nodes up to n ($n \geq 1$) links away does not differ from the set of all means in which global node-to-node communication is enabled[3].

Pseudo-global control

Applicability

Decentralized urban traffic optimization approaches as found in the literature are not universally applicable to any *transport network configuration* (this term henceforth refers to a scenario of a specific network type, node and link topology, and traffic offered): optimal or near-optimal results at least approximately matching the performance of centralized optimization solutions exist only for exceptional cases, particularly those in which special properties of the topology are known in advance, which may be exploited.

Transport network configuration

For example, for the following urban traffic network configurations, near-optimal decentralized optimization strategies exist:

Existing decentralized control

- A single isolated intersection[4], e.g. Läm07, Chapter 4. Minimizing waiting times is reduced to minimizing switching periods subject to first serving the queues for which the difference in the contribution to all entities' waiting time between serving and (preliminarily) not serving the queue is maximized. The special case of arrival rates that are small in comparison to processing rates will be exemplarily discussed in Section 4.3.

- Two intersections in proximity, e.g. Yi06; in comparison to considering each intersection as a single intersection (see above), two intersections additionally require that the link ("connection") between the intersections be monitored: intersection control has to avoid admitting too few vehicles to the other node, implying unnecessary waiting time if e.g. processing at the node had been possible immediately, as well as the admission of too many vehicles, potentially blocking the connection so that side road traffic that does not need to be forwarded to the other node is unnecessarily delayed.

[3] Observe that information propagation in an underlying telecommunication network is much faster (i.e. an order of milliseconds, even if based on multi-hop communication) than entity processing or node switching (i.e. an order of seconds).

[4] In this configuration, centralized and decentralized optimization do coincide. In addition, note that in networks in which the nodes are sufficiently far away from each other to e.g. exclude links from blocking, the network can be assumed to consist of *multiple* single, isolated nodes.

- A single arterial, e.g. Coo08, in which traffic density in one direction is significantly higher than in the other; a typical example is a rush hour during which opposing traffic is negligible. A near-optimal solution is to permit the creation of green waves based on a two-phase model (see Section 2.1.4) in rush hour direction: side road traffic is served until a critical number of vehicles is queued at the head of the arterial. Once processing of those vehicles begins, they form a platoon that is not interrupted at the successive nodes.

- Manhattan-like grids, e.g. Ger05, with dominant or exclusive rush hour traffic in one of the directions on both "vertical" and "horizontal" links (e.g. traffic from N to S and from W to E significantly exceeding opposing traffic, while turning is not relevant): based on the single arterial case, symmetry permits the creation of green waves in both rush hour directions; a special case are symmetrical blocks of identical length in which a suitable choice of offsets and phase lengths permits green waves in all four directions; see e.g. Ger12, pp. 389f. Furthermore, Ger07 discusses a generalization to more general topologies consisting of only two competing flows per node that need be considered.

Restrictions

Tables 4.2 and 4.3 summarize examples of decentralized (and centralized) optimization approaches for urban traffic and abstract transport networks. Despite also covering the most wide-spread used commercial tools [see Pap03], the tables focus on scientific proposals of node control published during the last decade. The table states whether or not the following restrictions apply (all proposed means of intersection control are subject to at least one type of constraint):

Restriction types

- Specific topologies that the optimization approach is restricted to, i.e. exploiting spatial patterns of node locations and link layouts to base optimization on. Typical cases include single arterials, symmetric Manhattan-like grids, or arbitrary layouts in which nodes are sufficiently far away from each other to provide enough queuing space to assume that link blocking does not occur.

- Assuming special patterns of traffic, e.g. flows not significantly deviating from being uniform/constant without arrival distributions and traffic densities changing over time or assuming saturated traffic equal to or exceeding the network's capacity, so optimization focuses on such critical conditions; thus, performance is suboptimal otherwise.

Table 4.2 – Constraints of existing approaches for urban traffic optimization 1

	Topology limitations	Traffic patterns	Global communication	Other requirements or constraints
A. Commercial tools				
BALANCE [Gev12]	–	–	•	• Fixed cycles[1]
LA-ATCS [Mar03]	–	–	•	• Fixed cycles[1]
MOTION [Sie12]	–	–	•	• Fixed cycles[1]
OPAC [Che87, Mar03]	–	–	•	• Fixed cycles[1], exponential complexity[2]
PRODYN [Hen83]	–	–	•	• Exponential complexity[2]
RHODES [Mar03]	–	–	•	• Fixed cycles
SCATS [Dut10]	–	–	•	• Fixed cycles[1]
SCOOT [Bre12a]	–	–	•	• Fixed cycles[3]
TRANSYT-7F [Luk84, Mar03]	–	–	•	• Fixed cycles
UTOPIA [Swa12]	–	–	•	–
B. Scientific concepts				
Baz05	• Manhattan	–	•	–
Bih92	• Manhattan	–	–	• One-way flows
Bro01	• Manhattan	–	–	• One-way flows
Bul04, Cao99	• 1–4 intersection(s)[4]	–	–	–
Cey04	–	• Routes fixed[5]	•	• Fixed cycles
Che04	–	–	•	• Fixed cycles[1]
Che12	• One intersection[4]	–	•	–
Coo08	• Single arterial	–	–	–
Dav83	–	• Saturated traffic	–	–
Del95	• Manhattan	–	•	• Fixed cycles
Fer10	–	–	•	• Vehicle tracking
Foy92	–	–	•	• Fixed cycles
Gar75, Imp82, Lit66	–	• Continuous/uniform	•	• Fixed cycles, exponential complexity[2]
Ger05	• Manhattan	–	•	• One-way flows
Ger07	• Two flows per node[6]	–	–	–

. . . continued in **Table 4.3**

[1] The cycle length to be obeyed by all nodes may be dynamically adjusted – [2] Application not possible in typical networks under (near-)real-time conditions [Pap03, p. 2049] – [3] Cycles may be skipped and their lengths may be multiples of each other – [4] Applicable to larger networks when assuming sufficient queueing space to exclude link blocking – [5] Patterns defined by an origin-destination matrix that determines optimal routes – [6] See Table 4.3

Table 4.3 – Constraints of existing approaches for urban traffic optimization 2

	Topology limitations	Traffic patterns	Global communication	Other requirements or constraints
Gor11	–	–	•	• Fixed cycles
Gra07	• One intersection[4]	–	–	• Vehicle tracking
Hel07a, Hel07b, Läm06	• One intersection[4]	• Continuous/uniform	n/a	–
Hel08	–	–	–	• All flows exclusive
Hel09	• One intersection[7]	–	–	• Two-phase model[8]
Hel10	–	–	• Subnetwork only	• All flows exclusive
Hoa02	–	–	•	• Two-phase model[8]
Läm07, Chap. 4	• One intersection[4]	–	n/a	• All flows exclusive
Läm07, Chap. 6	–	• Continuous/uniform	–	• All flows exclusive
Läm09	–	–	•	–
Lee95	–	–	• Subnetwork only	–
Let07	–	–	•	• Vehicle tracking
Mik95	–	–	•	• Fixed cycles
Nak95	–	–	•	• Fixed cycles
Pap84	–	• Saturated traffic	•	–
Poh10	–	• Continuous/uniform	•	• Fixed cycles
Por96	–	–	–	• Two-phase model[8]
Por97, Por98	–	–	•	• Fixed cycles
Ros11, Ger12	• Manhattan	–	–	• One-way flows
Roy04	• One intersection[4]	–	–	• Fixed cycles
Sen97	• One intersection[4]	–	–	–
She11	–	–	•	• Fixed cycles
TUG [Dia03]	–	–	•	• Fixed cycles
Wie04	• One intersection[4]	–	–	• Fixed cycles
Wu10	• One intersection[4]	–	–	–
Yag94	• One intersection[4]	–	–	–
Yi06	• 1–2 intersection(s)	• Over-saturated traffic	–	–
Yi08	–	–	•	• Fixed cycles[1]
Zho10	• One intersection[4]	–	–	–

[1–5] See Table 4.2 – [6] Proposes a set of methods best suited for different traffic conditions, see Ger07, pp. 77ff, lacking a means to determine which method to apply given certain network configuration – [7] Outlook (Section 6) proposes an extension to Manhattan-like grids – [8] See Figure 2.4 on p. 15

- Global communication is used during the run-time, i.e. node control is based on instructions or data that is not (completely) available locally, typically determined by a cental server.
- Other limitations of node control apart from global communication rendering the approach not adaptive, e.g. phases based on periodically repeating cycles with a fixed global cycle length or sequence of phases.

However, beyond specific network configurations solved near-optimally or even supraoptimally (see above), all centralized and decentralized means of dynamic traffic light control not restricted to a specific topology found in the literature produce potentially suboptimal results, as they artificially restrict their search space by

Limited search space

- *either* proposing complex means of node control containing, for example, arithmetic operations, loops, and conditionals, potentially recursively nested, e.g. Ger05, Ger07, Ger12, Hel08, Läm07, or Ros11, yielding a means that is sufficiently powerful (i.e. *Turing complete*) to describe arbitrary adaptive node behaviour without, however, at least heuristically exploring the search space containing node control programs based on similar programming logic and operators; for example, Ger07, p. 69, proposes switching between two flows (i.e. the decision space is reduced to keep the flow currently set to "green" or switching to the other) that can only be served alternatively based on the pattern

```
repeat every second
  if ((minimum phase duration passed)
    and (no imminent platoon arrival on the flow to be disabled)
    and (sufficiently many vehicles queued or approaching on the other flow))
    switch phases
```

in which search space exploration is restricted to investigating different parameter settings (minimum phase length, minimum platoon size, critical number of vehicles waiting on the other flow), yet not permitting structural modifications to this pattern, i.e. heuristically assuming that this pattern suffices to represent optimal node control,

Search space exploration

- *or* conducting a full or at least a heuristic exploration of a search space based e.g. on genetic algorithms [Gor11], neural networks [Nak95], classifier systems/fuzzy logic [Cao99], or reinforcement learning [Mik95]; for further references, see above. However, in such cases, the search space addressed is reduced, e.g. containing only relatively simple means of (typically non-adaptive) node control; e.g. a heuristic search is used to determine the cycle lengths and offsets of predefined phase programming.

This gives rise to the assumption that network performance may be improved by conducting an adaptive approach of complex node control *and* at least heuristic (i.e. non-exhaustive) search space exploration, which – based on the preparations of the next section – will be proposed in Chapter 5.

4.3 Towards Self-Organization

A first step towards *self-organizing transport networks* in which complex node control and search space exploration are combined is to identify desirable patterns of node behaviour (rules) to be adopted by the nodes; see for example green waves already mentioned as a "meeting in the middle" approach to overcome the inapplicability of pure top-down or bottom-up approaches of designing self-organizing systems in Section 3.3. Optimization (or adaption; see Section 2.2) based on identifying such rules rather than imposing fixed traffic light phases or similarly fixed means of node control permits *adaptive* node behaviour to cope efficiently with perturbations[5], which have to be expected in transport networks; see Section 4.1. Examples of suitable patterns observed in existing centralized and decentralized optimization approaches include:

Patterns

- Maximizing node utilization and, thus, throughput, as node capacity would be wasted otherwise: serving the largest potential flow; see e.g. Läm07, Chapter 4.
- Avoiding long waiting times: preferentially serving entities waiting for a long period, in particular guarantee that waiting time will not exceed a bound if possible; see e.g. Läm07, Chapter 6.
- Low traffic as a trivial case: ideally, each vehicle encounters a "green" phase when arriving at an intersection. In case arrivals in different flows do not occur too close to each other, the entities are served in arriving order, yielding FIFO processing; see e.g. Ger07, pp. 75ff. Deviating from FIFO processing by delaying switching is recommended in case another arrival in a "green" flow will occur soon, i.e. in less than the period required for switching and processing one vehicle such that switching to another flow and back would be less efficient in terms of overall waiting times; see e.g. Ger12, p. 391.
- (Over-)saturated traffic as a trivial case: minimizing switching penalties by alternately serving each flow as long as possible, i.e. unless no more entities are waiting or unless the outgoing link is blocked; see e.g. Dav83.

[5] As an analogy, Ger07, p. 89, refers to the difference between a teacher either telling a student exactly what to do (which works as long as the student's problem sufficiently closely matches the solution proposed by the teacher) or enabling the student to decide on the necessary steps himself (such that a broader class of problems can be solved at the "expense" of requiring more complex reasoning from the student).

Selfish behaviour

These patterns can be referred to as *selfish* since nodes attempt to serve entities as quickly as possible or at least to preserve their ability to serve entities (i.e. not wasting capacity) at all. However, solely pursuing selfish interests may produce suboptimal global results; consider Robert M. Axelrod's (1943–*) *prisoners dilemma* [Axe84, pp. 109ff] as an instructive example: in the case of transport networks, selfish rules have to be supplemented with *altruistic* behaviour not necessarily optimizing local entity processing but facilitating the performance of the overall network [Hel08, p. 28]:

Altruistic behaviour

- Backward planning: a node should prevent incoming links from blocking; otherwise, the previous nodes may be temporarily unable to process further entities, e.g. by preferentially serving queues close to blocking and by retaining platoons of approaching entities ("green waves") prone to quickly build up queues unless served immediately, even if e.g. traffic from side roads where arrival rates are smaller has to wait longer; see, for example, Pap84.
- Forward planning: the entity output of a node should be balanced such that neither too few nor too many entities are transferred to every outgoing link, thus neither starving the next node by denying sufficient entity supply so that its processing capacity is used only partially nor blocking the outgoing link; see, for example, Läm12.

Phase transitions

Despite such patterns, the key question remains how to combine or merge them into a single means of node control, requiring precise conditions or priorities to determine which pattern to facilitate at a specific instant and (if appropriate) under which conditions to switch from one pattern to another, yielding *phase transitions*; see Section 3.1.2: observe that at least some of the patterns are mutually exclusive, e.g. maximizing utilization tending not to switch the flow served at a moment versus switching to a different flow on a different incoming link that is close to blocking unless it is served soon.

Pattern Evaluation

Entropy

A special case of choosing a behavioural pattern applies if a means exists to determine the impact of different behaviours of a component, i.e. a node, on the performance of all other relevant components, i.e. entities and other nodes. Such an impact on the performance of other components may be positive or negative, e.g. switching causing the waiting time of some entities to not be further increased (since set to "green"), while others will have to wait longer than without switching (since set to "red"). By aggregating such measurements, a specific node's decision's impact on the entropy of the system can be derived, measured, for example, in average global waiting times. Depending on the sign, Ger07, p. 41, refers to such a measurement of the change of the global performance caused by a local decision

as *synergy* or *friction*, which in his approach to network control forms the basis of deciding about local node behaviour. Synergy

However, for the case of a transport network, synergy (friction) can be established only imprecisely by analytical or numerical means for different reasons: uncertainty prevents e.g. precise instants of future entity arrivals from being included in the evaluation. More severely, determining synergy and friction has to consider a network's feedback loops: the decision of a node (the tagged node) modifies the behavioural alternatives available to other nodes (e.g. since a link is blocked), i.e. their potential decisions attempting to maximize synergy or minimize friction have to be included in the tagged node's evaluation of switching alternatives: according to the Shannon inequality (see Section 3.1.3), reducing the entropy of a subnetwork (i.e. waiting times of entities at specific nodes) is not necessarily advantageous if another subnetwork's entropy (i.e. other entities' waiting times) is increased. In Ger07, p. 61, it was possible to approximately determine synergy and friction by assuming a relatively simple topology, namely a Manhattan-like grid of one-way flows (see Tables 4.2–4.3), while lacking a means of determining synergy (friction) in a more general network. Finally, a hypothetical measurement of synergy (friction) would also depend on time-scale, e.g. a short-term advantage potentially turning negative in the long run [Ger07, p. 42].

Single Pattern Approaches

Before again addressing the question of choosing the behavioural pattern for a node without a measurement of the synergy (friction) being available, this thesis should at least make it plausible that combining multiple patterns and thus potentially triggering phase transitions is necessary; this particulary requires showing that a single pattern does not suffice for urban traffic (or transport network) optimization in general. Without being able to exhaustively discuss all such patterns, an attempt to minimize the longest waiting periods by serving FIFO can be ruled out, as such node control is obviously inefficient if setup or safety times exist and the network is (nearly) saturated, potentially incurring a setup period after every single entity served such that the network's capacity is significantly reduced. Necessity of multiple patterns FIFO service

This motivates the avoidance of setup times as long as possible by serving the largest potential flows, i.e. preferring flows where most entities can be processed per unit of time. However, such an approach fails, as it may cause some flows to never be served, e.g. if the average arrivals on one flow exceed the potential flow rate on another flow, where an arbitrarily long queue builds up; see the transport network configuration of Figure 4.3[6]. Greedy service

As an alternative, starting with the largest potential flow or the longest queue, flows may be alternately served until their queue is cleared, thus delaying switch- Alternating service

[6] An optimal solution to this transport network configuration is *optimal isolated intersection control* (OIIC); see p. 67.

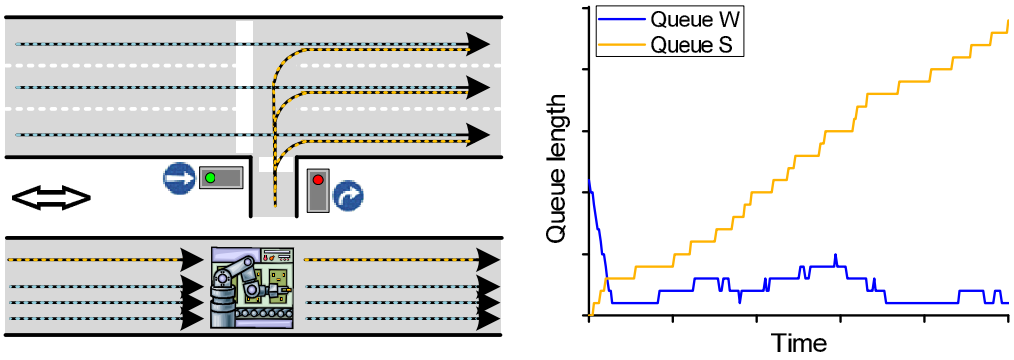

Figure 4.3 – Failure of simple decentralized optimization rules: flow starving; network configuration (left) and queue length development (right, conjecture; see Läm07, pp. 87ff, for details)

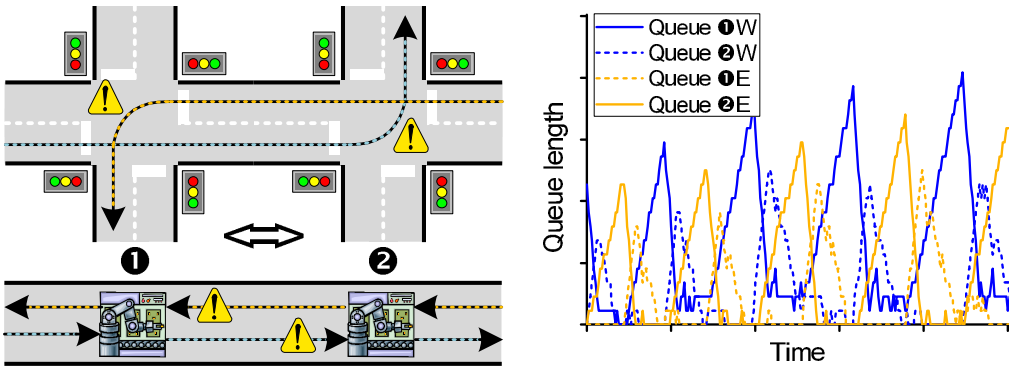

Figure 4.4 – Failure of simple decentralized optimization rules: missing synchronization; network configuration [based on Göb11b, p. 141] (left) and queue length development (right, conjecture; see Läm07, pp. 153ff): assume two flows referred to as W→N and E→S crossing each other twice at nodes ❶ and ❷; further suppose buffer space between nodes ❶ and ❷ is limited and an initial "outside" queue at node ❶ from W exists; finally, assume that the processing rate is smaller where each flow is tagged with an exclamation mark; e.g. turning requires a lower velocity than proceeding straight ahead. Serving first the flow from W at both nodes finally clears the queue from W at node ❶ (in short, ❶W); when queue ❶W is cleared, the "inside" queue ❷W is not yet due to the delay of entities traversing the link between the nodes and processing being slower at node ❷. Consequently, node ❶ "wastes" capacity until node ❷ clears queue ❷W and thus switches to serving queue ❷E, finally supplying entities from E to node ❶. Symmetrically, queue ❶E takes longer to be cleared than queue ❷E, forcing node ❷ to not fully use its processing capacity until entities arrive on queue ❶W after node ❶ has switched back to serving the flow from W and so forth; thus, further assuming (near-)saturated flows implies arbitrarily long queues building up.

ing as long as possible, yet also guaranteeing that all flows receive service though their arrival rates may be small. As an example of this approach failing based on Läm07, pp. 153ff, consider two interweaved flows W→N (short notion for "from W to N" from now on) and E→S that can only be processed mutually exclusively at two nodes, which are configured such that the processing time requirements at the second node passed by each flow are higher than at the first node; see Figure 4.4, whose caption describes how the nodes force each other to waste capacity by alternately serving each flow until the queue is cleared, which does not supply enough vehicles to the other node, resulting in what is referred to as *dynamic instability* in Kum90 since, instead of resolving perturbed traffic conditions, queue lengths arbitrarily increase[7]. In contrast to patterns to facilitate, such as flow maximization, this observation provides an *anti-pattern* of node behaviour which should be avoided: serving the same flow for a long period may "starve" adjacent nodes that run out of entities to process, thus strictly requiring that flow maximization be combined with altruistic patterns such as forward planning; see above.

Dynamic instability

Anti-pattern

As described in Sections 3.1–3.2, self-organization is typically achieved by a set of microscopic decision rules used independently by each component. However, the development of such rules if they are not known in advance is non-trivial; designing a self-organizing system can be interpreted as reverse-engineering such rules from the desired macroscopic behaviour of the system. This behaviour (e.g. efficient transportation) is an emergent property of such rules, for which research thus far does not offer an agreed-upon and universally applicable means of obtaining, leaving patterns between microscopic and macroscopic level as a typical approach to derive microscopic rules from [Zam04, pp. 306f]. Note that transport network configurations exist whose optimization is possible based on a single selfish or altruistic pattern facilitated by appropriate microscopic rules, e.g. "green wave" arterial node control facilitating platoons established at the first node and not stopped by the following nodes assuming other flows than in the rush hour direction are negligible; however, in the general case, such single patterns do not necessarily suffice; see the examples described above, requiring multiple patterns being combined (or anti-patterns avoided). This renders the approach of Zam04, pp. 306f, infeasible for the optimization of general transport networks since it is unclear how to combine the patterns; alternatives include choosing patterns to facilitate based on priorities or by defining conditions describing where or when to apply a specific pattern or how to determine a compromise between patterns that are contradictory, e.g. the target of saving setup times by delaying switching needs be balanced against preventing a link from blocking by immediate switching.

[7] A near-optimal solution to this transport network configuration is described during the evaluation of Scenario 2 in Section 6.2.2.

Therefore, the microscopic rules (e.g. described in terms of *triggers* that, once fulfilled, cause a specific node behaviour; see Section 3.1) that this thesis aims to develop for the purpose of transport network optimization *cannot* be directly deduced from existing rules facilitating the patterns described above; furthermore, a macroscopically optimal behaviour not constrained by cycle times or other restrictions is not known, so that no basis of top-down rule development or mimicking the behaviour of omniscient entities exists. Among the approaches of engineering self-organizing systems described in Section 3.2, this leaves bottom-up rule evolution: microscopic rules have to be newly engineered from the input side core components, i.e. local input data such as queue lengths, algorithmic and logical means of processing this input data, e.g. sums and conditionals, and behavioural instructions, e.g. enabling or disabling specific flows, requiring evolutionary rule development due to the potential occurrence of undesired emergents.

Determining rules

Generic Node Control

The requirements of such an approach include the identification of appropriate input data and a sufficiently general means of node control (i.e. determining the flows to serve next) from this input data without imposing restrictions on how this data is processed and on which flows are served and for what duration.

Regarding the input data, this thesis heuristically relies on the data other approaches consider to be available locally, measured e.g. using video surveillance with image processing or induction loops. From Baz05, Coo08, Ger05, Hel08, Läm07, and other approaches proposing at least partial decentralized solutions to the transport network optimization problem, a set of numerical values and conditionals can be obtained, upon which entity prioritization at a given instant is based. These values and conditionals will be referred to as *prioritization criteria*. Such criteria apply either to a specific flow on a node (i.e. a queue for vehicles arriving at a node on a certain link, waiting for processing and departure on one or more links), to an intersection node as a whole (e.g. maximum queue length, total estimated arrival rate), or even to the overall network (e.g. a switching penalty, during which all lanes are "red" for safety reasons).

Prioritization criteria

Table 4.4 provides an overview of all prioritization criteria, containing the level on which the criterion is defined, i.e. whether referring to a specific flow, a node, or the network as a whole, the type of result, an identifier, and a short description. The identifiers include the criterion's level (**F**low, **N**ode, **Ne**twork), variability (**V**ariable or **C**onstant), and an abbreviation. A more precise definition of every criterion can be found in Appendix B.

Regarding a sufficiently flexible software interface not imposing restrictions to local node data processing and adaptive flow prioritization (see Section 3.2), this thesis applies the approach of Läm07, p. 23, referred to as *priority index* (PI)-based flow selection: priority indices are values dynamically assigned to

Priority index

Table 4.4 – Entity prioritization criteria collected from Baz05, Coo08, Ger05, Hel08, Läm07, and others [extension of Göb11b, p. 138]

Level	Result type	Identifier	Description	Further details
Flow	Float	FV-AR5	Arrival rate estimate (last 50)	P. 149
		FV-ARO	Arrival rate estimate (overall)	P. 149
		FV-EAS	Entities arriving soon	P. 149
		FV-GS	Green duration	P. 149
		FV-GYS	Green or yellow duration	p. 149
		FC-LLI	Link length (incoming)	P. 149
		FC-LLO	Link length (outgoing)	P. 149
		FV-LUI	Link utilization (incoming)	P. 150
		FV-LUO	Link utilization (outgoing)	P. 150
		FC-LVI	Link max. velocity (incoming)	P. 150
		FC-LVO	Link max. velocity (outgoing)	P. 150
		FV-NGS	No green since	P. 150
		FV-NGYS	No green or yellow since	P. 150
		FV-PAD	Platoon arrival duration	P. 150
		FV-PF	Potential flow (absolute)	P. 150
		FV-PFR	Potential flow (relative)	P. 150
		FV-QLc	Queue length (count)	P. 150
		FV-QLp	Queue length (physical)	P. 150
		FV-WTA	Waiting time (average)	P. 150
		FV-WTL	Waiting time (longest)	P. 150
	Condition	FV-iG	Is green	P. 150
		FV-iGY	Is green or yellow	P. 150
		FV-iID	Is idle	P. 151
		FV-iCO	Is congested (outgoing link blocked)	P. 151
Node	Float	NV-AR5	Arrival rate estimate (last 50)	P. 151
		NV-ARO	Arrival rate estimate (overall)	P. 151
		NV-PD	Duration of current phase	P. 151
		NV-MQLc	Maximum queue length (count)	P. 151
		NV-MQLp	Maximum queue length (physical)	P. 151
		NV-OQLc	Overall queue length (count)	P. 151
		NV-OQLp	Overall queue length (physical)	P. 151
		NC-SLI	Shortest link length (incoming)	P. 151
		NC-SLO	Shortest link length (outgoing)	P. 151
		NV-WTA	Waiting time (average)	P. 151
		NV-WTL	Waiting time (longest)	P. 151
	Condition	NV-iEM	Is empty	P. 151
		NV-iID	Is idle	P. 151
		NV-iBL	Is remotely blocked	P. 151
Network	Float	NeC-MEA	Maximum entity acceleration	P. 152
		NeC-ST	Setup time	P. 152
	Condition	NeC-iPP	Passing is possible	P. 152
		NeC-iRi	Processing requires nearby nodes idle	P. 152
		NeC-iSP	Has switching penalty	P. 152

every flow by evaluating the *priority index function*, which depends on dynamic measurements of queue lengths, average flow rates, and other prioritization criteria; a node serves the flow for which the highest priority index has been determined unless specific network *stability* side conditions are violated since e.g. a flow has not received service for a maximum waiting period (see below). No restrictions are imposed as to which criteria in Table 4.4 are used and how they are combined logically and algebraically to assign a PI value to every flow. Assuming the availability of sufficiently powerful operators – details to follow in Chapter 5 – any node control logic can be expressed as a priority function, i.e. no loss of generality incurred [Läm07, p. 23], e.g. a pattern such as

Stabilization

```
repeat every second
   if (c) switch phases
```

as proposed by Ger07, p. 69 (see Section 4.2, p. 56), basing the choice between two flows on a condition c – despite c being relatively complex – being fulfilled translating to e.g. the PI function

```
if flow enabled
   PI = c ? -1 : 1
else
   PI = 0
```

where _?_:_ represents an in-line case switch similar to the *Java* operator of the same name.

Figure 4.5 proposes an extension of the priority index mechanism using a set of flows instead of single flows, taking into account intersections serving multiple flows simultaneously, e.g. opposing traffic from north and south proceeding straight ahead at the same time. Every Δ seconds, flow priorities are re-evaluated, switching the set of flows to "green" which exhibits the highest accumulated priority[8] if (and only if) this set's priority exceeds the currently "green" set's priority by a certain positive amount or factor, thus preventing oscillating behaviour in which processing capacity is wasted due to the setup time incurred by switching too frequently, namely as soon as the priority index of another flow marginally exceeds the priority of the flow currently being served.

PI-based flow choice

Additionally, further stabilization requirements such as minimum phase lengths and maximum waiting periods may be applied, which, however, requires caution since imposing constraints on a node's switching behaviour does not necessarily improve performance; furthermore, such side conditions require prioritization themselves as e.g. guaranteeing both a minimum phase length (e.g. keeping

[8] Instead of *accumulating* (summing up) PI values, more complex means of combining flow priorities into priorities of sets of flows are possible, e.g. Wu10 allocating phases to groups of flows based on *fuzzy logic* membership functions that, in turn, operate on flow priority (queue length) values; however, as shown by empirical proof in Chapter 6, this simple means of aggregating flow priorities suffices for the purpose of this thesis .

a flow set to "green") and a maximum waiting period (e.g. switching to a different flow) may be conflicting targets.

Moreover, a *tie-breaking* rule should be defined for the case of equal priorities, e.g. out of two flows to be mutually exclusively switched from "red" to "green", picking the one not served for a longer period, implicity yielding a "round robin" procedure of equally prioritized flows served alternately. After determining which flows are set to "green", deciding which additional flows to switch to "yellow" may be based on the same procedure, i.e. out of potentially multiple flows whose conditional processing simultaneously to the flows already set to "green" is supported by the network, the flows to which a higher PI value has been assigned are preferred. Tie-breaking

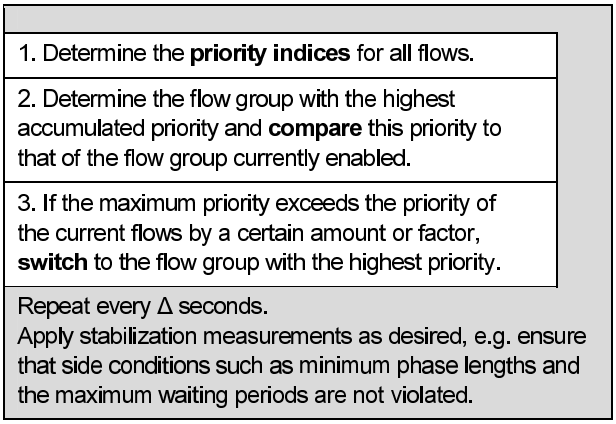

Figure 4.5 – Flow choice based on priority indices; extension of Göb11b, p. 138, which is a generalization of Läm07, p. 23

According to the ontology of Hol95, pp. 62ff, PI function-based node control can be interpreted as *schema*: combining a PI function with stabilization constraints such as means of preventing oscillations, minimum phase lengths, and tie-breaking rules provides a framework for effectively defining (and refining) the behaviour of the transport network nodes from which network-wide self-organization is desired to emerge; the set of all possible PI functions represents the search space traversed manually or by applying heuristic search methods to determine node control; see Section 3.2. Thus, the transport network optimization problem is reduced to determining PI functions, i.e. to how numerically and logically combine the prioritization criteria stated above into a PI value. The PI function is the *defining position* [Hol95, p. 64] of the PI-based node control schema. Schema

Determining Priority Index Functions

For special cases, optimal priority index functions can be identified analytically: consider, for example, an isolated intersection serving two flows a and b that are mutually exclusive. Assume that the queue lengths are q_a and q_b. Further assume that f_a and f_b entities can continuously be served per unit of time when the flows are set to "green" (including the processing capacity of potentially multiple lanes) and that a safety period s is required between serving either of the flows. Disregarding arrivals during the next service period (or at least assuming them negligibly small in comparison to f_a and f_b), it is optimal to first clear one queue and afterwards the other [Läm07, p. 69f] to minimize safety periods: serving a first, the overall accumulated waiting time of all entities $W_{a,b}$, is

$$W_{a,b} = \underbrace{\frac{q_a^2}{2f_a}}_{\text{A}} + \underbrace{\frac{q_a q_b}{f_a}}_{\text{B}} + \underbrace{s q_b}_{\text{C}} + \underbrace{\frac{q_b^2}{2f_b}}_{\text{D}},$$

see Figure 4.6 for a visualization of A, B, C, and D. Similarly, serving b first yields an overall accumulated waiting time $W_{b,a}$ of

$$W_{b,a} = \underbrace{\frac{q_b^2}{2f_b}}_{\text{D}} + \underbrace{\frac{q_a q_b}{f_b}}_{\text{B'}} + \underbrace{s q_a}_{\text{C'}} + \underbrace{\frac{q_a^2}{2f_a}}_{\text{A}}.$$

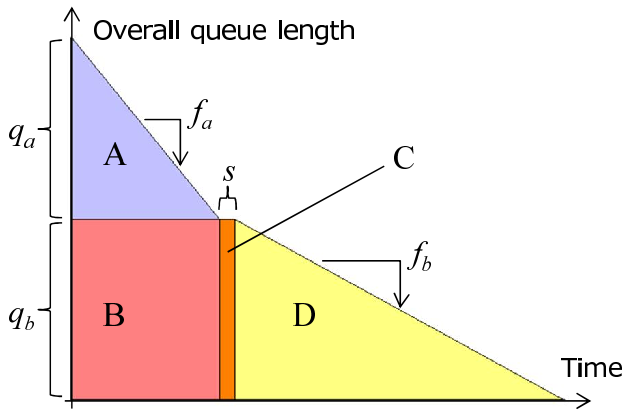

Figure 4.6 – Aggregated waiting times at an isolated intersection [extended from Läm07, p. 71]

As A and D appear in both $W_{a,b}$ and $W_{b,a}$, it suffices to compare B and C to B' and C': flow a should be served first if and only if

$$\frac{q_a}{q_a/f_a + s} > \frac{q_b}{q_b/f_b + s};$$

i.e. the expression

$$\frac{q}{q/f + s}$$

can be used as PI function to adaptively determine the flow to serve first until it is cleared under the assumptions stated above. More advanced approaches do not rely on the assumption of future arrivals being negligible, which requires including the waiting times of vehicles served during successive phases of alternatingly clearing queues.

This is typically done by decreasing the weight of waiting times of vehicles processed later due to the increasing uncertainty about the instants at which arrivals will occur; such time-based discounting allows the determination of which flow to serve first despite a theoretically infinite planning period based on *Bellman's equation*; see e.g. Kat06, pp. 4ff, for details: in particular, the waiting times of the vehicles that have already arrived at the intersection when the decision of which flow to serve is due (not discounted, as waiting has begun by then) contributes most to the overall expected waiting times associated with the decision to serve either of the flows first. This justifies referring to alternately clearing queues of isolated intersections based on this priority index as (near-)*optimal isolated intersection control* (OIIC). This holds if the flow rates during the next period are unknown; e.g. assume significant perturbations may occur which cannot be predicted in advance. Also, observe that the considerations leading to the PI equation stated above can analogously be extended to more than two competing flows at an isolated intersection [Läm07, p. 79ff].

Optimal isolated intersection control

In general, however, other OIIC assumptions apart from the fact that arrivals during serving flows need not be negligible may not be satisfied : the service of discrete entities need not be near-continuous, and limited space between adjacent nodes may be too small to consider the nodes isolated, at least by approximation. Combined with examples of the failure of single-criterion node control approaches (see above), this gives rise to the (heuristic) conjecture that more complex priority index functions are needed to efficiently control general transport networks: particularly observe that Rot84 proved the non-existence of optimal PI functions for arbitrary network topologies using queue lengths and processing rates as their only input, thus highly likely requiring more complex PI functions based on further input data (i.e. additional prioritization criteria); see also the means of node control proposed e.g. in Ger07, Hel08, Chapter 6 of Läm07, and Zho10.

For example, Hel08, p. 14, proposes a default priority index similar[9] to

$$\frac{f'q/(f'-a)}{s+ws+q/(f'-a)},$$

subject to variable queue length q, arrival rate a, effectively feasible processing rate f' (reflecting the node's ability to process depending on the supply of sufficient entities and the outgoing link not blocking), switching safety period s, and

[9] The proposal of Hel08, p. 14, employs a submodel of forecasting future traffic patterns, see Hel08, pp. 6ff, while the equation listed here is simplified such that future arrivals are estimated based on the (average) arrival rate a; i.e. arrivals are assumed uniformly distributed.

a switching penalty weight w, which is defined as $w = 0$ if the flow is already set to "green"; otherwise, $0 < w < 1$ holds, describing the average extent to which entities on the link, of which only a fraction have already arrived at the node, are delayed by the switching safety period. This PI equation is embedded in a stabilization strategy guaranteeing that queues do not exceed critical lengths [Hel08, p. 15], yielding a means of PI-based node control that despite being designed for flows which are assumed to be mutually exclusive, see Table 4.3, can be viewed as a generalization of OIIC since it adaptively

- explicitly incorporates future arrivals,
- considers the effective processing ability of *non*-isolated nodes (e.g. link blocking), and
- facilitates self-organization in terms of e.g. green waves emerging from local node control [Hel08, p. 24].

Further taking into account its ability to outperform a variety of other adaptive traffic light control approaches [Hel08, pp. 26ff], this scheme is heuristically referred to as (near-)*optimal network control* (ONC) for the purpose of this thesis ; details about the stabilization strategy into which this PI function is embedded are deferred to p. 87 in Section 5.3 as the notation of so-called *PI programs* as a generalization of PI functions has to be introduced first.

Optimal network control

Attempting to determine PI functions less specialized than OIIC (and even ONC), Chapter 5 focuses on evolutionary means of assembling such functions; thus, their structure, i.e. the precise input data used and how or under which conditions they are combined, will be subject to an evolutionary search. Determining the PI function is, therefore, a bottom-up search, according to Section 3.2, not further directly implementing the patterns of node behaviour as stated initially in this section. As the behaviour of PI function-based node control cannot be directly evaluated based on measurements such as synergy and friction, simulation experiments are necessary to determine the performance of a specific PI function.

Despite not explicitly setting up a self-organizing nodes' and, thus, the network's behaviour, due to evolutionary pressure self-organization can be achieved without explicitly programming green waves and other suitable patterns; according to Ger07, p. 88, such patterns are not "forced" but "*induced*": observe that all patterns mentioned can be facilitated by serving flows in which specific prioritization criteria such as those in Table 4.4 exceed or fall below a specific bound (details to follow in Table 5.1 on p. 74); thus, the proposed evolutionary search also implicitly decides which patterns to use under which conditions.

Patterns induced

Chapter 5

Self-Organizing Transport Networks

The computer programmer is a creator of universes for which he alone is responsible. Universes of virtually unlimited complexity can be created in the form of computer programs.

— Joseph Weizenbaum
Computer Power and Human Reason, 1976

After providing the fundamentals of transport networks in Chapter 2, self-organization in Chapter 3, and transport network optimization in Chapter 4, this chapter finally combines these building blocks into *self-organizing transport networks*, fulfilling this thesis' goal of determining adaptive decentralized node control that does not impose restrictions such as fixed cycle times and periodically repeating phases and that is applicable regardless of the network's type, topology, or traffic flow patterns.

To establish microscopic rules to be followed by the nodes such that macroscopic efficiency emerges from the nodes' behaviour, genetic programming (GP) is chosen (Section 5.1) and implemented (Section 5.2) because it is a flexible and robust search technique that – if combined with PI-based flow choice – does not constrain the node control search space, i.e. in principle permitting arbitrarily complex means of entity prioritization. The decision of whether to accept or discard entities is *not* addressed; see also the introductory remarks in Chapter 4. Note, however, that Section 7.2 describes how the GP-based approach can anal-

ogously be extended such that node control as genetically assembled also makes decisions about accepting or rejecting entities.

GP control and priority index programs as generalization of PI functions are described in Section 5.3. In addition, various approaches to improve the convergence (i.e. the run-time performance) of the GP evolution were necessary, as described in Section 5.4, before summarizing this chapter in Section 5.5.

5.1 Optimization Approach Choice

This thesis' vision is to reduce gridlock by making the grid smart (Section 2.4) by applying local (node) rules to adaptively decide which entity to process next such that the network is able to self-organize based on indirect interaction (Section 3.2); thus, an overall efficient performance emerges from these local rules.

Regarding the development of rule-based entity behaviour, Poo10, Section 1.4, decides between three approaches:

- *Design time reasoning.* The designer directly implements the rules in the agent (node).
- *Offline computation.* Rules are determined after the agent (node) has been designed, *before* it is used in the environment it is designed for.
- *Online computation.* Rules are continuously (re-)determined after the agent (node) has been designed, *while* it is used in the environment it is designed for.

Since node rules to solve the transport network optimization program (see Section 2.4) in general are not available yet and most likely cannot be determined analytically (see Section 4.3), design time reasoning is ruled out. Furthermore, to avoid the disadvantages of exposing a transport network to the preliminary suboptimal behaviour of "learning" nodes in online computation, offline computation is left as means of determining node rules[1].

Applying offline computation

Offline computation [Poo10, Section 1.4.3] typically uses training scenarios in terms of simulation models to evaluate proposed agent behaviour. Such behaviour is typically obtained from heuristic search techniques, of which e.g. Poo10, Section 4–9, provides a comprehensive overview. As offline computation is neither based on logically proven optimality, which ideally forms the basis of design time reasoning, nor able to modify the behaviour during usage in the environment if

[1] Note that *offline computation* as a means of determining rules should not be confused with traditional *offline urban traffic control* (see Section 4.1) since the subject of what is computed *offline* is different: the offline computation of behavioural rules determines fixed rules to base adaptive, dynamic node behaviour on, while offline urban traffic control directly establishes non-adaptive behaviour.

need be, a potential drawback of offline computation is being prone to overfitting [Poo10, Section 7.5]: nodes performing well in training scenarios need not be efficient in a real-world environment since the training scenarios may contain regularities or consist of special cases not matching the real conditions. Therefore, evaluation such as that to be conducted in Chapter 6 should test scenarios differing in e.g. topology, traffic patterns, or densities from those used for determining node rules.

Furthermore, an important design decision is to develop a *single* set of rules to apply to *all* nodes, yielding a *homogenous system* [Ger07, p. 52]: an attempt to design specialized rules for each node is again prone to overfitting, e.g. yielding a node behaviour optimized to its local training conditions, but it will no longer be efficient if local conditions change; self-organization requires *universal behaviour*, similar to that of flocking birds or water molecules forming a Rayleigh-Bénard convection (see Section 3.1.2), which do not differ from each other in their biological or physical behaviour, i.e. they use the same underlying rule set. Demanding that the rules to be developed are homogenously applicable to *all* nodes, regardless of whether traffic density is high or low, whether queuing space (e.g. link length) is large or small, whether arrivals are evenly distributed or forming platoons, whether few or many incident links exist, and so forth, significantly increases the probability that the resulting rules will still perform well if the network is extended by dynamically adding further nodes (for which no rules would exist if developed on a per-node basis) or traffic patterns are modified: such universal robustness relies on rules empirically guaranteed to efficiently control *all* nodes, not only a single node or a subset exhibiting special conditions.

Homogenous system

Universal behaviour

In particular, *self-organizing transport networks*, as the goal of this thesis , do not rely on constraints to be fulfilled by the network, such as

- being applicable to specific topologies only, e.g. optimization based on fixed lattice lengths or other constraints on the topology,
- exploiting patterns of traffic known in advance,
- communicating to a real or virtual central authority (thus, the input data that node control is based on is completely obtained locally e.g. by cameras and induction loops),
- limiting which entities are processed at which instant of time e.g. by imposing fixed cycles,

see the structure of Tables 4.2 and 4.3 on p. 54f. Furthermore, no specific optimization target should be factored into determining node control: for the purpose of this thesis , optimization attempts to minimize entity waiting times; nevertheless, alternative targets such as emission reduction, are to be supported without redesigning the optimization approach.

The following three subsections describe the attempts undertaken to facilitate the design or evolution of self-organizing transport networks; the first (energy transmission in crystalline grids) refers to the category of direct engineering, while the others (genetic algorithms, genetic programming) represent examples of reverse-engineering self-organizing systems (see Section 3.2).

5.1.1 Energy Transmission in Crystalline Grids

As stated in Section 3.2, *directly* engineering self-organization based on existing means of appropriate node control should be preferred to *reverse-engineering* an initially unknown scheme of local node control due to only refining rules already available instead of creating new rules from scratch. The literature has proposed the adoption of different means of node control from various areas of science and engineering to traffic light control; however, such attempts do not necessarily meet the targets named above, e.g. Let07 suggests centralized traffic light control analogously to real-time multitasking central processing unit (CPU) scheduling.

Regarding decentralized node control, applications of local rules similar to those facilitating energy transmissions among molecules forming crystalline grids, see Col99, Section 8, and Wic99, initially seemed to be a model worth adopting; see Figure 5.1: above a critical energy threshold, energy strokes may be released spontaneously. "Swinging" molecules dynamically align with each other based on local rules such that energy is conveyed in different directions, but only in one direction at a time, i.e. strokes from different directions are mutually exclusive.

Swinging molecules

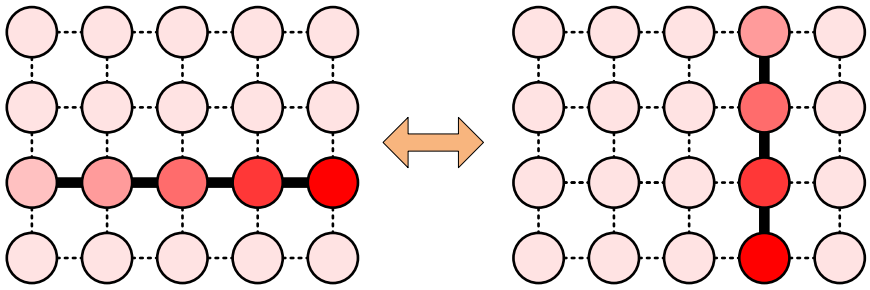

Figure 5.1 – Energy strokes in crystalline grids: exclusive flows [Col99, Section 8]; alignment represented using continuous, thick lines

Mimicking energy transmissions in crystalline molecule patterns might be an appropriate approach for traffic management in Manhattan-like grids of one-way streets as elaborated in Hel99; this particularly holds if the ratio of vehicles turning is small. However, this thesis was unable to propose a generalization of this approach to a less regular topology and multiple flows processed in parallel by each node; thus, a transport network based on "swinging crystals" was not pursued further, as imposing layout constraints has to be avoided; see Figure 5.2 for an

example of two flows (energy strokes) that have to be merged, requiring a decision as to which flow is to be delayed.

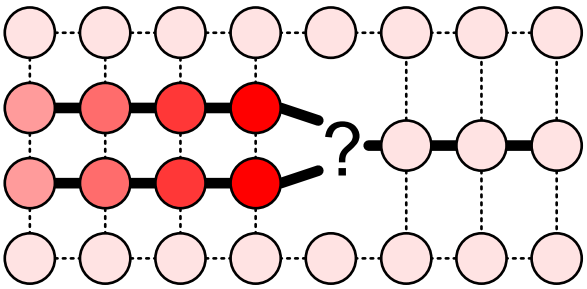

Figure 5.2 – Energy strokes in crystalline grids: merging conflict

Similarly, other existing self-organizing local rule sets, e.g. routing algorithms inspired by ant pheromones, see Section 3.1, apart from applying stigmergy do not cover entity prioritization at nodes or other critical areas. However, without rules available that can be directly adopted for transport network optimization, it is necessary to determine new ones. This can be conducted manually, e.g. by applying a combination of analytical considerations and trial-and-error exploration as, for example, in Ger07 or Hel08; however, this thesis seeks to cover a larger part of the search space of potential means of node control by applying heuristic search approaches, namely genetic algorithms (Section 5.1.2) and genetic programming (Section 5.1.3).

Search space

5.1.2 Genetic Algorithms (GAs)

Local rules to deploy to the network nodes need not necessarily be adopted from a single concrete phenomenon in nature or social systems; rules can also be determined manually, e.g. by combining rules and solutions empirically shown or mathematically proven to be (near-)optimal given special topology or traffic conditions.

Potential building blocks of rules were discussed in Section 4.3, where a combination of selfish and cooperative behaviour was proposed, summarized in Table 5.1. The table also states which prioritization criteria from Table 4.4 on p. 63 can be used as triggers for these rules in case their values exceed (↑) or fall below (↓) critical bounds or either an upper or lower bound being hit requires the rule to be activated (↕). Note that all rules – including altruistic behaviour – are covered by at least one criterion, i.e. the criteria suffice as triggers to potentially apply all patterns.

Furthermore, the rule statements in Table 5.1 are not network-specific, permitting application to different types of transport networks (urban traffic, IP networks, production systems), satisfying this thesis' goal to apply a unified ap-

Table 5.1 – Basic rules of selfish and altruistic node behaviour

Selfish behaviour	**Altruistic behaviour**
• Maximize throughput (prefer largest flow) FV-PF↑, FV-PFR↑ • Avoid long waiting times (prefer entities waiting for a long period, guarantee waiting time not to exceed a bound) FV-WTA↑, FV-WTL↑ • Low traffic as a trivial case (prefer the only entity waiting) FV-QLc↑, FV-QLp↑ • Saturation as a trivial case (prefer vehicles not blocking the node as departing is possible) FV-LUO↓	• Backward planning, e.g. preventing an incoming link from blocking (avoid blocking incoming links, retain platoons of approaching entities even if local traffic has to wait longer than necessary) FV-LUI↑, FV-PAD↓, FV-QLc↑ • Forward planning, e.g. output balancing (neither feed too many nor too few entities to an outgoing link, thus neither starving nor blocking the next node) FV-LUO↕

proach to optimization of all types of transport networks. In fact, they are *too* general since node behaviour is underspecified in two ways:

- The *trigger* is not well-defined; i.e. it is not clear under which condition a rule is applied, e.g. lacking a specification of the bound from which an entity's waiting time is considered *too* long so that the entities to be processed are chosen by waiting time until further notice.

- A *decision* is necessary regarding which rule to apply if the triggers of more than one (or none) fire, yielding phase transitions between environmental conditions in which different rules are used for entity prioritization.

The ability to not necessarily choose (only) a single rule to apply at a given instant, but also being able to combine rules, was a reason to apply priority indices as described in Section 4.3: in priority index-based flow choice (see Figure 4.5 on p. 65), different triggers can be combined into a PI function subject to individual weights; by relying on the prioritization criteria from Table 4.4 on p. 63 instead of explicitly applying rules from Table 5.1, the PI functions use the measurements that are directly available at the nodes, not restricting how they are aggregated, e.g. a "backward planning" rule increasing a flow's priority with incoming link utilization or decreasing it with the duration until the next platoon arrival.

For example, a *simple PI function* has the form [analogously to Cao99] Simple PI function

$$\mathsf{PI} = \underbrace{w_1\mathsf{FV\text{-}AR5} + w_2\mathsf{FV\text{-}ARO} + \ldots}_{\text{Criteria returning floats}} + \underbrace{v_1[\mathsf{FV\text{-}iG}] + v_2[\mathsf{FV\text{-}iGY}] + \ldots}_{\text{Criteria returning Boolean values}},$$

where $w_i, v_i \in \mathbb{R}$, and for any condition c, $[\mathsf{c}]$ yields 1 if c is true and 0 otherwise.

Even though the search space made up of all PI functions that can be expressed in this form represents only a subset of all possible PI functions, systematic search techniques such as *branch & bound* and *variable elimination* [see Poo10, Section 4.3-4.7] cannot be applied: the space is infinite and no means of effective partitioning is available, i.e. it is impossible to recursively split the space into disjoint subsets such that – based on a search tree successively determining bounds to the performance of PI functions whose variables (criteria weights) are restricted to a subset – evaluating all subspaces is less complex than searching the space as a whole.

This leaves *heuristic search techniques* as the only means to approximate a solution (i.e. determine a near-optimal PI function; *solution* henceforth is used synonymously with members of the optimization program search space): sacrificing the guarantee that the best solution is found, these approaches only partially cover the search space, typically returning *local optima* [see Poo10, Section 4.8]. Such heuristic search techniques are particularly useful when multiple (many) solutions that perform well are likely to exist: for the transport network optimization problem, it is not necessary to prove that an overall best PI function has been found; instead, a PI function performing *reasonably well* (e.g. average waiting times similar to centralized optimization) suffices; see also Section 3.3. Heuristics

From such heuristic search techniques, as presented in the literature, local search [see e.g. Poo10, Section 4.8.1-4.8.4] and population-based methods [Poo10, Section 4.9] can be distinguished, differing in that either a single element or a set of elements from the search space is considered at once, proposing a variety of means for how to obtain the next element or set of elements in a next "step" or "generation" that likely represents or contains a better solution (i.e. PI function) than was available previously. Not exhaustively traversing the search space implies the inability to prove that a (near-)optimum solution has been found; such heuristic search techniques terminate e.g. once a certain number of steps or generations has been completed in total or consecutively without obtaining further improvements.

Evolutionary Search

Among such heuristic search techniques, *genetic algorithms* (GAs) as proposed by Hol98 were chosen as a means to determine simple PI functions: based on an initial "population" of solutions, e.g. generated at random, a non-deterministic search process adopted from biological evolution is applied, enabling GAs to iteratively Genetic algorithms

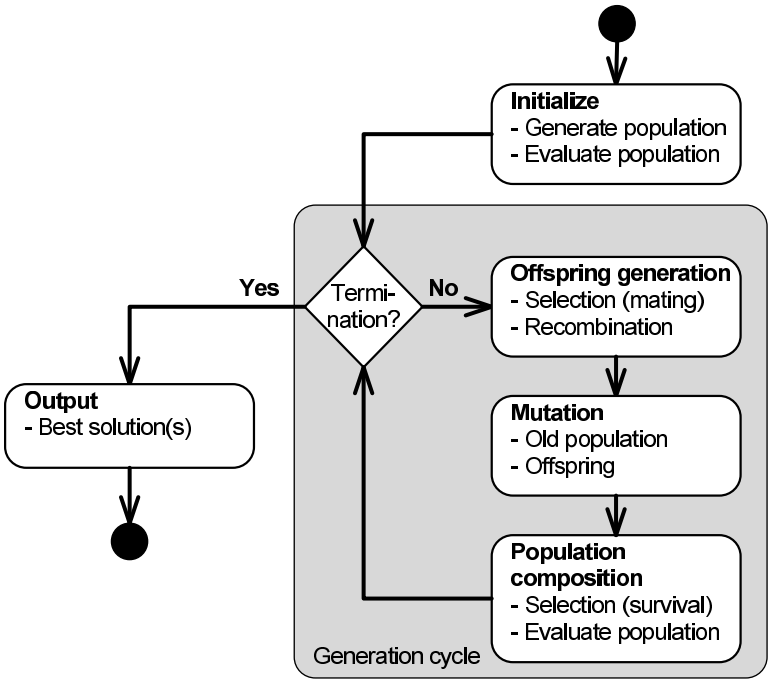

Figure 5.3 – GA: evolution; from Göb11b, p. 138, which is based on Koz92, p. 76

improve the "population" of PI functions by combining properties of two (or more) existing PI functions and mutating them from generation to generation.

A summary of GA search can be found in Figure 5.3. The reasons for this choice are as follows [see also Poo10, Section 4.9]:

- Since GAs are a population-based method, a GA is less likely to converge towards a local optimum solution that is significantly worse than a local search approach.

Fitness function

- As the input to base the evolutionary process on, a GA relies on a performance measure referred to as *fitness function* being available for each member of the population: the "fitter" a member of the population is, e.g. the shorter the expected waiting time of an entity at a node, the greater its chance of being selected for mating or survival will be. Additionally, the "fittest" individual (or even more than one) may always be selected for survival to retain the preliminarily best solution. A GA requires no other problem-specific data apart from such a fitness measurement; alternative optimization targets can be pursued by simply replacing it with a different fitness function. Conversely, search techniques requiring further information apart from assigning fitness values to solutions may be less appropriate for the purpose of determining PI functions; for example, consider gradient-based search techniques [Poo10, Section 7.3.2] using ratios comparing the change in network performance to the change in one or more components

of the PI function, i.e. adjustments to one or more weights w_i, v_i, which are not available or at least computationally expensive to obtain by, for example, conducting multiple experiments subject to marginal modifications of PI function weights.

- Setting up a GA to generate PI functions involves some design decisions: apart from encoding the properties of a population member into a data structure referred to as a *chromosome*, the so-called *genetic operators* have to be specified, describing how to select, recombine, and mutate PI functions. The selection probability of a solution can, for example, be assigned proportional to its fitness, referred to as *roulette wheel* selection [Mic96, pp. 34f]; while genetic encoding and particularly how to combine two or more solutions into a new one may be difficult to define in some domains (e.g. how to merge two *travelling salesman* routes into one, as discussed in Mic96, pp. 209ff), this can be done straightforwardly for simple PI functions: such a function is described by a chromosome made up of the sequence of all weights w_i, v_i; recombination means adopting some values from each "parent" or a sum or weighted mean of both, and mutation may be conducted by randomly modifying weights; see Figure 5.4.

Chromosome

Genetic operators

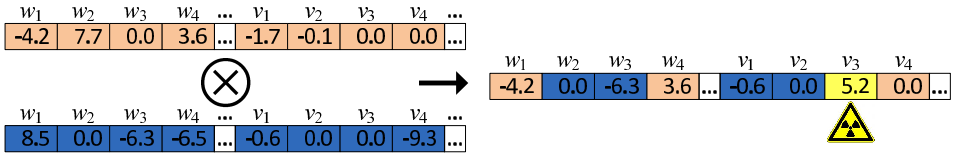

Figure 5.4 – GA: recombination and mutation; recombination for example applies *uniform crossover* [Mic96, p. 90f], adopting each chromosome entry from either of the parents with equal probability unless mutation yields a random adjustment

Finally, setting up a GA also requires parameterizing the evolutionary process cycle itself, i.e. determining the size of the population, the ratio of members of the next generation obtained from recombination and members selected and transferred without modification, and the mutation rate; for instance, a chosen mutation probability to randomly modify PI functions that is too small may mean that properties which are not part of the initial *gene pool* are unlikely to appear, yielding suboptimal results, versus evolution potentially suffering from useful combinations of weights that are inevitably destroyed if mutations occur too frequently. However, GAs are considered relatively *robust* in comparison to other search techniques, particularly

Robustness

  - obtaining good results in terms of solution quality and computation duration does in most cases not depend on relatively narrow ranges of parameter choices, specific chromosome encoding, and genetic operator design [see e.g. Hol98, pp. 121ff], and

– as a multi-dimensional search based on a population of solutions, a GA is able to determine structurally different results, typically closely approximating the global optimum's fitness despite being a heuristic optimization method [see e.g. Mic96, pp. 16f].

JGAP

A preliminary implementation of a GA to determine PI functions was developed based on the GA infrastructure of the JGAP (Java Genetic Algorithms Package; see Mef12) library, which provides *Java* classes for solution population members (abstract) and default genetic operators (selection, recombination, mutation), thus facilitating efficient GA development since the remaining mandatory task of the JGAP user is only to implement a "chromosome" representation of the population members and a fitness function that is able to evaluate such population members. The library can be flexibly customized, for example by applying user-defined genetic operators, and supports various extensions to the GA scheme as outlined above, e.g. "island evolution" based on multiple fitness functions. For details about the transport network simulation model based on the discrete event simulation framework DESMO-J, which is used to evaluate the fitness of a PI function see Section 6.1.

The default parameter and genetic operator configuration of JGAP was used, see the API documentation [Mef12], with one exception: mutation was refined such that, whenever a PI function weight w_i, v_i undergoes a mutation, it is set to a uniformly distributed random floating number (which is the default) in 50 % of all cases and to 0 otherwise so that the number of criteria used in a PI function tends to be limited; without this adjustment, it is highly likely that all criteria will be used subject to some positive or negative weights, producing a "clutter" in which meaningful criteria are difficult to identify.

Application

Single isolated intersection

For simple scenarios, this approach was successful. For example, for a single isolated intersection of two flows whose processing is assumed to be mutually exclusive, the processing rates identical, and arrivals during processing negligible (satisfying the assumptions of OIIC; see Section 4.3, p. 67, see also Scenario 1 in Section 6.2.1), the PI function

$$\mathsf{PI} = 30\mathsf{FV\text{-}Qlc} + (-99319)[\mathsf{FV\text{-}iID}]$$

Alternating service

was evolved[2], with the numbers rounded to integers. This is an optimum solution, as one flow is served until the queue is empty and, thus, idle so that the "penalty" of the second summand applies to both the empty queue (idle since there are no

[2]Important parameter values: 25 generations of size 100, a new generation is composed from selection (roulette wheel) and crossover (uniform) without directly transferring solutions. To prevent oscillating behaviour, switching required the aggregated priority of the set of flows currently set to "green" exceeded by the priority of a different set by at least 20 %. As further

further vehicles) and the other (idle since it is set to "red"). Now the other flow will be cleared, effectively minimizing switching safety periods, which suffices to minimize the average waiting periods: due to the symmetry of this scenario (same arrival and processing rates, no link blocking possible), no reasons exist to prefer one of the flows apart from delaying switching as long as possible to minimize average waiting times [see e.g. Läm07, Chapter 4].

However, even a minor generalization such as dropping the assumption of identical processing rates requires a PI function that can no longer be expressed as a *simple PI function* (see above), as OIIC, discussed in Section 4.3, is equivalent to e.g.

$$\mathsf{PI} = \frac{\mathsf{FV\text{-}Qlc}}{\mathsf{FV\text{-}Qlc}/\mathsf{FV\text{-}PF} + \mathsf{NeC\text{-}ST}} + (-99319)[\mathsf{FV\text{-}iID}],$$

note that the summand $(-99319)[\mathsf{FV\text{-}iID}]$ has been kept to ensure that the flow is served until it is cleared despite the PI potentially being re-evaluated earlier.

These considerations motivate the application of a search technique that is able to flexibly generate more general PI functions in which the algebraic and logical operators used to determine PI values are subject to evolutionary search themselves. A search technique fulfilling this requirement is genetic programming (GP), which can be interpreted as a generalization of genetic algorithms.

5.1.3 Genetic Programming (GP)

Genetic programming (GP) as initially proposed in the work of Koz92, Chapters 5–6, is a population-based heuristic search technique mimicking evolution in nature to determine (near-)optimal solutions to optimization problems. A compact description also covering recent extensions can, for example, be found in Pol08, Chapters 2–3, to which the reader is referred for details beyond the following short outline.

Genetic programming can be interpreted as a generalization of genetic algorithms: GP uses the same "evolutionary" process cycle of selecting, recombining, and mutating search space elements, see Figure 5.3 on p. 76, thus exhibiting the same characteristic properties discussed in the previous section, including robust behaviour regardless of genetic operator parametrization and universal applicability not relying on domain-specific structures and data apart from the availability of an abstract chromosome representation of all solutions and a fitness measurement; see e.g. Koz92, pp. 569ff, for a study demonstrating robust GP behaviour despite inaccurate fitness estimations, changing environmental conditions, or even erroneous modifications to significant parts of the population.

Generalization of GA

stabilization requirements, the minimum phase length is 5 seconds. "Round robin" tie-breaking is not necessary since, in the two-flows model, multiple "red" flows competing for processing with the flow currently set to "green" do not exist. See Section 4.3 for details about parameterizing PI-based flow selection.

However, while GAs operate on a static chromosome data structure in which all relevant information about a search space member is encoded (e.g. weights in simple PI functions; see above), the data structure of a GP chromosome itself is subject to evolution .

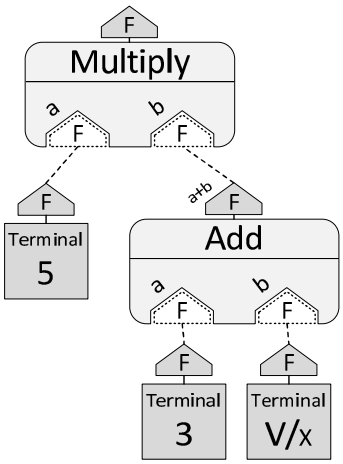

Figure 5.5 – GP: example of a function in a tree-based representation [Koz92, pp. 9ff]

Genetic programs

A dynamic chromosome permits the "evolution" of search space members, so-called *genetic programs*, of arbitrary complexity: genetic programs typically are represented using a tree-like structure consisting of building blocks or *nodes*[3] [Koz92, pp. 91ff] that can be flexibly combined, with the only restriction that arguments and result data types of adjacent genetic nodes have to match. Due to their meaning in a GP application context, genetic nodes are also called functions/operators or terminals, depending on whether or not arguments are required as input: as an example, Figure 5.5 shows how the term $5 \cdot (3 + x)$ could be represented by genetic nodes: the expression is composed of functions such as Multiply (top), representing a node accepting two floating point numbers (F) as arguments and returning the product as a floating point number as the result. The overall result of the GP tree [see Koz92, pp. 9ff] is the return value of the top-level function (here: Multiply), whose arguments are determined by functions one level below, which themselves may be based on input from other functions on a lower level and so forth. Note that the closure of genetic programs [Koz92, pp. 81f] assumes the availability of terminal nodes not requiring any argument, e.g. numerical or Boolean constants or values read from variables providing external input to the genetic program; this thesis uses a notation V/... to distinguish input variables from constant terminals.

Closure

[3] To avoid confusion with *network nodes*, these are also referred to as *genetic nodes* where needed.

GP Operators

The *GP operators* are applied analogously to the GA evolution cycle depicted in Figure 5.3 on p. 76; they are defined as follows:

- Evolutionary pressure towards a population becoming fitter from generation to generation requires the probability of being *selected* [Koz92, pp. 100f] for mating (i.e. recombination with another program) or surviving (i.e. old programs and new offsprings being transferred from the current population to the next) increasing with the program's fitness. A straightforward approach is assigning selection probabilities proportional to the fitness of the programs (*roulette wheel* selection, Koz92, p. 22; see also Section 5.1.2). However, a roulette wheel selection may exhibit undesired properties: a single, relatively fit program, for example, initially generated at random may quickly dominate the gene pool, as other programs are hardly selected so that evolution may be stuck in a local optimum since all other solutions quickly vanish (i.e. the evolutionary pressure is too large), while e.g. close to termination when the fitness values of all programs typically differ by only small amounts, fitter programs are not sufficiently preferred to others (i.e. the evolutionary pressure is too small). Therefore, the literature provides other schemes for assigning selection probabilities, e.g. *rank-based* selection [Pol08, p. 91f] where the selection probability is proportional to the number of inferior programs in the population (i.e. programs that are less fit) or *tournament selection* [Pol08, p. 14f] in which the fittest program of a randomly determined subset of the population is chosen: observe, that in both rank-based and tournament selection (the latter will be used for the purpose of this thesis), absolute fitness differences do not matter; for determining a rank or the winner of a subset "tournament", it is not important whether the next best programs are marginally or significantly less fit.

  Evolutionary pressure

  Tournament selection

- *Mutation* [Koz92, pp. 105ff] is responsible for making random changes to the population: each genetic node has a small chance of being replaced by other nodes with the same input and result parameter types, including every terminal potentially being replaced by other terminals whose return type matches.

- In contrast to GAs, the *recombination* [Koz92, pp. 101ff] of hierarchical programs of potentially different structures can no longer be based on an 1:1 mapping of the parents' chromosomes, where recombination can be conducted by, for example, adopting a property such as the weight to assign to a criterion in a simple PI function from either the "mother" or the "father". Figure 5.6 shows examples of GP recombination operators, particularly subprogram transfer ❶, aggregation ❷, projection ❸, and permutation ❹.

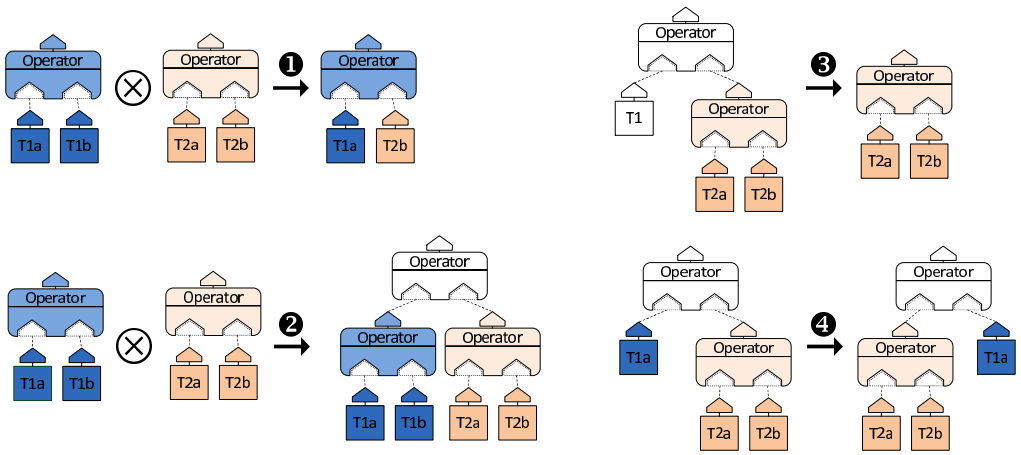

Figure 5.6 – GP: recombination operators, namely subprogram transfer ❶, aggregation ❷, projection ❸, permutation ❹ [from Göb11b, p. 140]

Self-Organization Based on Genetic Programming

Genetic programming has been chosen as the final approach to determine PI functions in this thesis for the following reasons: besides the advantages "inherited" from genetic algorithms (see introductory remarks), GP fulfills the requirement of being sufficiently flexible and powerful to generate PI functions whose complexity matches the PI functions proposed in the literature for special cases of transport network optimization such as OIIC, assuming that suitable building blocks (genetic nodes) are made available to the evolutionary process (e.g. Add, Multiply, Divide in the case of OIIC). Furthermore, a GP approach to determining PI functions can be extended seamlessly by providing more building blocks in case additional prioritization criteria as measured by the genetic nodes become available or further algebraic operators are considered worth being included.

Self-organizing transport networks

As biological evolution can be interpreted as self-organization itself (see Section 3.1.2), this yields a twofold interpretation of *self-organizing transport networks*: a network's ability to *self-organize* such that efficient performance is achieved by applying a *self-organizing* evolutionary optimization approach; genetic programming with its fitness evaluation provides the positive or negative reinforcement required to create (here: reverse-engineer) self-organizing systems; see Section 3.2.

Nevertheless, observe that this thesis does not claim that GP is the only or the even best-suited search technique to determine PI functions for transport network optimization: according to the empirical proof conducted in Chapter 6, it only claims that it works reasonably well, i.e. producing results comparable to other (e.g. centralized) strategies or better in acceptable time, at least if the standard GP approach proposed by Koz92 is adjusted as discussed in Sections 5.3–5.4.

Approximately a decade ago, the literature has shown GP and GAs are equivalent: in particular, every GP design of genetic nodes and GP operators can be equivalently mapped to a GA design [Fer01]. Thus, a GP approach is *not* able to determine PI functions that a (suitably extended) GA could not have produced as well; the search spaces are identical. Thus, preferring GP over GAs does not necessarily mean that a larger search space is covered or convergence is quicker – the preference for GP over GA is reduced to a question of choosing a notation; however, for the purpose of determining PI functions, the GP notation is more intuitive and comprehensible and thus easier to modify and extend than the *gene expressions* proposed by Fer01 to equivalently map a hierarchical program structure to a linear GA chromosome.

Gene expressions

5.2 GP Implementation

The implementation of GP-based node control evolution, like the GA, is based on the JGAP library [Mef12]; for details about the DESMO-J-based transport network simulation model, the reader is again referred to Section 6.1. Average waiting times still serve as the preliminary fitness measurement.

Node control evolution as in Section 5.1.2 means obtaining PI functions based on the prioritization criteria in Table 4.4 on p. 63. To avoid capacity losses due to rapid oscillation between different flows so that unnecessary setup times are incurred, flow choice, as summarized in Figure 4.5 on p. 65, is stabilized as follows (see also Section 4.3): the current set of flows set to "green" is replaced by a different set if this set's flow priorities exceed the current set of flows set to "green" by a default of at least 20 % but no earlier than after a minimum phase length of 5 seconds, thus using the same values as in the preliminary GA experiments (see Section 5.1.2). If two or more flows set to "red" are equally prioritized, *round robin* tie-breaking prefers the flow set to "red" for a longer period if their processing is mutually exclusive.

Node control evolution

Despite the fact that the minimum phase length of 5 seconds can already be justified due to safety considerations (i.e. shorter periods may not be anticipated by the drivers of the vehicles and thus pose a safety risk; see Section 3.3), "green" durations shorter than 5 seconds are unlikely to be efficient due to the high ratio of safety periods per unit of time. However, the 20 % PI value switching barrier is an arbitrary choice. Observe, though, that no loss of generality is incurred since arithmetic operators are provided (see below) to accordingly transform PI function values such that the switching margin is mapped to a difference of 20 % or greater.

No loss of generality

The set of genetic nodes made available to node control evolution includes the operators typically used in GP-based mathematical functions [see Mef12], namely numerical functions (Plus, Minus, Multiply, Divide, Power, Root, Exp, Log, Abs),

logical operators (And, Or, Xor, Not, Greater, Smaller), and statements to define piecewise functions, namely the case switches If_else and _?_:_; the latter is the same as in *Java*, providing a Boolean differentiation between two cases that can be nested into other arithmetic or logical expressions. This is supplemented by the prioritization criteria from Table 4.4 on p. 63 and numerical and Boolean constants as terminals. However, additional means of controlling how PI values as determined at a node are required, as will be discussed in the next section.

5.3 GP Control and Priority Programs

The standard GP approach to determining the fitness of a program is shown in Figure 5.7: the full program – e.g. serving the purpose of creating a Mona Lisa forgery by drawing lines and filled polygons on a canvas [courtesy of Mef12] – is executed once, followed by evaluating the fitness by comparison to the "real" Mona Lisa. Other examples, e.g. programming of artificial ants [Koz92, pp. 54ff], repeat program execution for a predefined period or until a stopping condition is fulfilled.

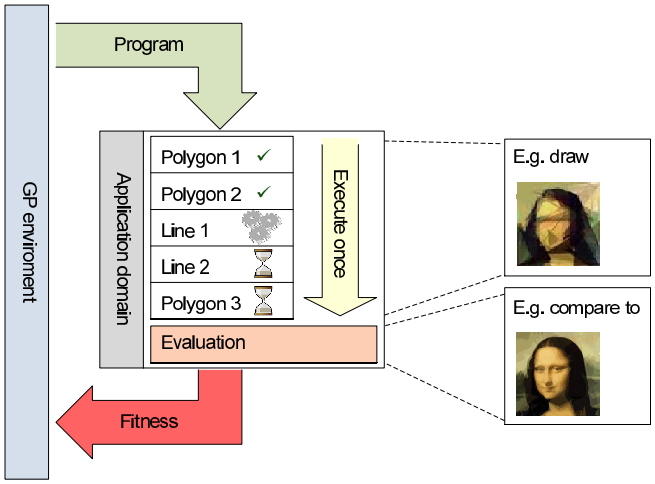

Figure 5.7 – GP: fitness evaluation of Mona Lisa forgeries; from Göb11b, p. 140, which is based on Mef12

For two reasons, this paradigm is not appropriate for determining node priority index (PI) functions:

First, the evaluation of a PI function is driven by a simulation experiment calling the function whenever necessary, not by single or multiple executions of the function.

Second, PI functions as proposed by e.g. Ger05, Hel08, Chapter 6 of Läm07, and Zho10 make use of control structures such as

- loops, e.g. a While operator permitting repeatedly using the same means of obtaining a PI value until a certain condition is no longer fulfilled, and

- partial conditionals, e.g. an If operator without else, leaving the current PI calculation to *either* the conditional expression after the If statement *or* to the next "line"; thus, the line used during the *next* PI calculation depends on how far program execution did proceed: in contrast to piecewise functions as defined by If_else or Switch nodes, where a choice is made (one out of a set; the evaluation of one specific subprogram is guaranteed), program control may *or* may not skip one or multiple lines, depending on the condition,

to switch behaviour based on environmental conditions, compare e.g. ants alternating between exploring and harvesting; see the examples in Section 3.1.

PI Programs

Providing genetic nodes representing loops and partial conditionals permits a transition from PI functions to *PI programs*: a PI function does not maintain an *internal state*; determining a PI function's value means evaluating the function (or the appropriate sub-function if defined piecewise) *without* influencing the next evaluation. Applying a PI program, however, is a state-dependent means of determining PI values: evaluation resumes from the previous state of program execution; see Figure 5.8. Which of potentially multiple statements, i.e. subprograms, determining a PI value is evaluated, depends on previous evaluation in which e.g. a While loop surrounding a statement may have either been left or not.

Internal state

In a textual program representation, subprogram statements determining and "returning" a PI value whose computation does not depend on (further) nested control structures have the form return PI = ...; note, however, that this return operator is not to be confused with `return` statements in imperative programming languages such as *Java*: both transfer the result to the environment, i.e. to a node requesting the priority values of one or multiple flows from the PI program . However, at the next instant that the PI program is called, the PI program does not necessarily start over like a method in Java would; instead, program execution is state-dependently resumed (see above).

Observe that such PI programs are strictly more powerful than PI functions: every PI function also is a PI program, despite not making use of its ability to include loops and partial conditionals. On the contrary, not every PI program can be represented by an equivalent PI function. As an example, consider a program maintaining a primitive state in terms of a "trap":

Strictly more powerful

```
repeat {
  return PI = ...
  if (c) {                                    [1]
    while (true) {                            [2]
      return PI = ...
    }
  }
}
```

Program execution cannot leave the trap [2], which is entered once the condition c is fulfilled [1]. An equivalent PI function deciding which return statement to apply would be required to determine whether condition c has ever been fulfilled in the past. Assuming is it not possible to deduce past states of condition c from the (current) values of the prioritization criteria, e.g. c = NV-OQLc > 100 ("Have overall queue lengths ever exceeded 100 entities?"), yields PI functions unable to reproduce the behaviour of this PI program; thus, the example program cannot be expressed as an equivalent PI function.

That fact that PI programs are strictly more powerful than PI functions, of course, is no proof that PI programs exist which provide more efficient node control than the best PI function for a certain transport network configuration (type, network topology, traffic offered). On the contrary, for certain network configurations, PI functions do suffice; see the simple PI function examples as discussed in Section 5.1.2. The reason to use PI programs rather than PI functions *despite* the larger search space needs be traversed, which can be expected to increase the computational complexity of the evolutionary process, is heuristical: as decentralized transport network approaches such as Ger05, Hel08, or Läm07, which are applicable only to specialized network configurations, require such program control statements, it is unlikely that, in a more generalized network, efficient decentralized control is feasible without them. For example, the approach of Hel08 is approximately represented[4] by this PI program:

Larger search space

```
repeat {
  if (NV-MQLc > q) {                                              [1]
    return PI = FV-QLc                                            [2]
    while (NV-PD < m or (NV-PD < p and not(NV-iID)))
      return PI = FV-iG ? 1 : 0                                   [3]
  } else {
    return PI = FV-PF * FV-QLc / (FV-PF - FV-AR5) /
      (NeC-ST * (FV-iG ? 2 : 1) + FV-QLc / (FV-PF - FV-AR5))      [4]
    while (NV-PD < m)
      return PI = FV-iG ? 1 : 0                                   [5]
  }
},
```

Its reasoning is as follows: a default PI value taking into account queue lengths, arrival and processing rates, and switching penalties [4], the weight of the latter differing depending on whether or not the flow is served at the moment, is applied unless the longest queue exceeds a critical queue length q [1]; in this case, the longest queue is preferred [2]. Expressions [3] and [5] guarantee that phases do not change during a minimum phase duration m by prioritizing flows already set

[4] See Section 4.3: the sub-model of forecasting arrivals, see Hel08, pp. 6ff, has been simplified by assuming that future arrivals are uniform, subject to an arrival rate identical to the recent past, i.e. anticipative control reduced to adaptive control (see Section 4.1). Conversely, the PI program has been analogously extended such that it is applicable to multiple flows served in parallel, not only a single flow at once.

to "green", yet serving the longest queue (despite the default priority potentially recommending a different flow) no longer than a full maximum stabilization period p and no more if the node is idle (i.e. the queue cleared or outgoing link blocked). As one of the most efficient means of decentralized entity prioritization available in the literature [see Hel08, pp. 19ff] not restricted to specific network topologies or traffic patterns, Section 4.3 referred to this scheme as (near-)*optimal network control* (ONC).

Optimal network control

Note that PI programs determined by GP evolution are by no means required to actually use the control structures of loops or partial conditionals such as ONC; in fact, if efficient PI programs not containing loops or partial conditionals exist, they are more likely to appear than complex programs based on these control structures; see the results of evolving single intersection control (Scenario 1) in Section 6.2.1, especially since PI programs receive a fitness penalty proportional to their length; see Section 5.4.2.

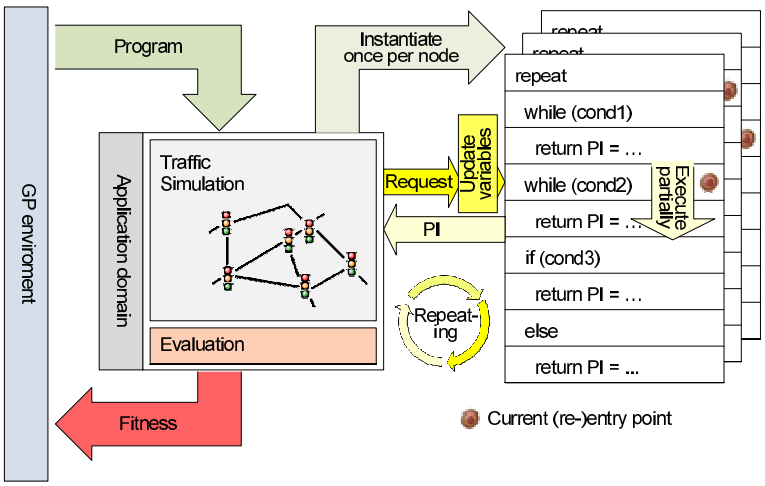

Figure 5.8 – GP: fitness evaluation of PI programs [from Göb11b, p. 140]

PI Program Execution

The GP implementation reflects the inverted control flow as outlined by Figure 5.8, subject to loops and partial conditionals: in particular, a special function referred to as Node Programming Logic (see Figure 5.9 for an example) is proposed, to which subprograms determining priorities can be attached. A Node Programming Logic operator keeps a reference to the current (re-)entry point, i.e. the "line" (subprogram) to use at the moment to determine the priority of a flow. Note that multiple nodes in a network require multiple program instances with a different entry point token each to reflect nodes potentially being in different states. Upon a request to determine the priority index of one or more flows, all

Entry

variables (prioritization criteria) to which the program tree refers are updated; this is followed by updating the entry point reference if need be, thus emulating program execution. Now that the return statement to currently calculate PI values has been determined, all flow priorities requested are computed. After the next request for PI values, the updating of the variable values and entry references is repeated before returning priority indices.

Nesting

The return type of the Node Programming Logic operator is a PI floating point value, which permits recursively nesting partial conditionals and loops using additional Node Programming Logic functions. After all lines have been used for determining priority indices, the program execution resumes at the first line.

Termination

Note that the specific Node Programming Logic operator which is used as the top-level node of a program has to guarantee that at least one subprogram is *not* constrained by a loop or partial conditional to ensure that at least one unconditional means of determining a priority is always available, preventing a PI request from not terminating due to infinitely restarting program execution as all While and If conditions evaluate to false.

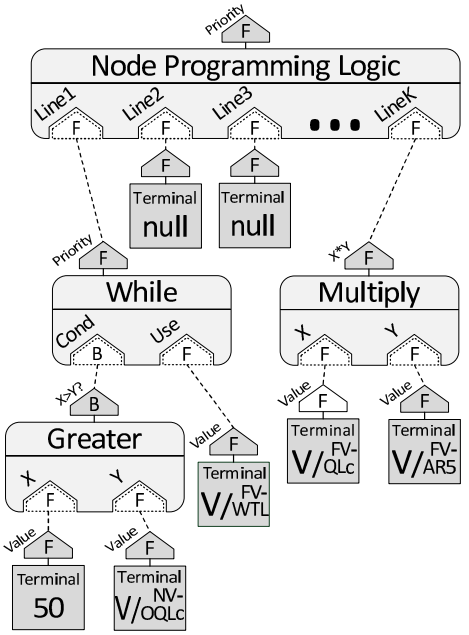

Figure 5.9 – GP: example of a priority index function for the transport network optimization problem

PI program example

Summarizing, the example program in Figure 5.9 will assign the highest priority to the flow containing the entity with the longest waiting time, which yields a FIFO (first in, first out) service, see the Use branch of the While loop, as long as congestion is moderate, i.e. less than 50 entities queued in total. Otherwise, the flow priority is the product of queue length and overall arrival rate estimate: as congestion increases, the function tends to prefer flows exhibiting higher traffic densities and to serve multiple entities before switching to other flows.

5.4 Run-Time Performance Improvements

In principle, the GP approach described in the previous section suffices to determine any PI program based on the prioritization criteria shown in Table 4.4, subject to the algebraic, logical, and program control operators explained above: any (near-)optimal node control program that can be combined from these building blocks will appear earlier or later. The crux here is the "later" part: addressing the concern expressed in the adage claiming that, giving a monkey a typewriter and letting it type long enough, eventually, the *Encyclopedia Britannica* will be produced, run-time performance has to ensure that a (near-)optimal node control program can be determined within a *reasonable time* as, otherwise, genetically engineering and deploying node PI programs is not possible in practice: the goal set for this thesis was to be able to establish PI programs in a duration whose order is hours or a few days, not weeks or months, on a single standard PC for network configurations similar to those that will be investigated in Chapter 6.

Infinite search

Initial experiments conducted after completion of the transport network simulation environment (see Section 6.1) indicated that this target was missed by a factor of more than 10, such that different measures were necessary to improve run-time performance by facilitating quicker evolution of suitable node control programs, which are described in the following subsections.

5.4.1 Explicitly Reducing the Search Space

As the evaluation of a single program, requiring one or more discrete event simulation runs, is relatively expensive in comparison to all other operations, particularly the genetic operators to generate offspring programs by selecting, recombining, and mutating programs, run-time performance improvements necessarily have to reduce the number of experiments or their duration.

PI Not Dependent on Flow Criteria

A typical example of a PI program that need not be evaluated at all is a program not containing a single criterion referring to a flow, see Table 4.4 on p. 63, e.g.

```
repeat
   return PI = NV-WTA
```

yielding the same priority index for every flow so that there is no means of dynamically deciding which flows to set to "green" except the (fixed) rule of resolving ties; see Section 5.2.

Trivial Conditions and Unnecessary Complexity

Apart from programs not containing at least one flow criterion, programs are not evaluated if they are unnecessarily complex since they can be expressed

equally using fewer genetic nodes, e.g. due to containing branches that are never executed, such as conditions that are always or never fulfilled:

```
repeat
  if (FV-iG and not(FV-iG)) ...
```

Such programs are removed from the search space as equivalent representations not containing such trivial conditions do exist; discarding such programs, i.e. at least partially implementing the *minimum description length principle*, as proposed by Iba94, by permitting only *normalized* programs, which are free from trivial conditions. This reduces the probability of repeatedly evaluating equivalent programs so that computation time can be allocated to evaluating other programs[5]. Table 5.2 summarizes examples of fragments of unnecessarily complex programs.

Minimum description length principle

Table 5.2 – Examples of fragments of unnecessarily complex programs

Fragment	Reason
if (true)	Always executed (if-clause can be removed)
if (false)	Never executed (block can be removed)
while (true)	Infinite loop (potentially following blocks can be removed)
while (false)	Never executed (block can be removed)
X and not(X)	Contradiction
X or not(X)	Tautology
X and false	Contradiction
X or true	Tautology
not(not(X))	Cancellation (can be replaced by X)
V > V	Contradiction
V * 0	Cancellation (can be replaced by a constant)
V – V	Cancellation (can be replaced by a constant)
V / V	Cancellation (can be replaced by a constant)
log(exp(V))	Cancellation (can be replaced by V)

X is an expression returning a logical value, V is an expression returning a numerical value.

If a program cannot be proven to be unnecessarily complex, it still cannot be guaranteed to return meaningful results: due to the variables' ranges, conditions may always return the same results. Consider the following example:

```
repeat ...
  if (FV-QLc < -3) ...
  while (18.2 < FV-QLc and FV-QLc < 18.5) ...
```

As the queue length (entity count) cannot assume values that are negative or between 18.2 and 18.5, these conditions are never fulfilled. To heuristically exclude tautological or unsatisfiable conditions, for each variable, 10 different

[5] Or, more precisely, at least the probability that the programs evaluated are not equivalent is increased, as programs are recognized as unnecessarily complex only if they contain certain patterns of which some are displayed in Table 5.2; therefore, the text refers only to *partially* implementing the minimum description length principle [Iba94].

values are tested. These test values are chosen such that most of the variables' feasible range – typical and extreme cases – are covered, e.g. queue length counts of 0, 1, 2, 5, 20, 60, 120, 480, 2000, 10,000 entities. Recursively, the results of all logical (sub-)expressions are determined assuming that all variables take every test value. Should a logical (sub-)condition always return the same truth value, the condition is assumed to be constant. Note that this heuristic approach falsely recognizes expressions such as

```
repeat ...
    if (FV-QLc > 13000) ...
    while (13 < FV-QLc and FV-QLc < 19) ...
```

as constant, i.e. the generation of rules to specifically apply if and only if between 13 und 19 entities or more than 13,000 entities are queued is no longer possible, assuming rules to refer to such small intervals or extreme values are not needed. Due to the combinatorial explosion of means to assign values to multiple variables, this test is restricted to (sub-)conditions consisting of no more than four variables for performance reasons. Expressions containing more variables are assumed to be not constant, yet their (sub-)conditions are still evaluated recursively. For example, the condition in

Combinatorial explosion

```
repeat ...
    if (FV-QLc < 13000 or FV-GS * FV-WTL > 0 or NV-iID or exp(FV-GS) > 0) ...
```

cannot be judged as whole. Nevertheless, the condition is recognized as constant as, for example, 13000 < FV-QLc is constant with respect to the feasible range of the variable FV-QLc.

Genetic Node Type Restrictions

Finally, the program search space was explicitly reduced by requiring conditions evaluated by genetic nodes (While/If without else) to refer only to network and node criterion data, not to flow criteria: therefore, the search space is heuristically restricted to programs whose execution at a node is synchronized for all flows so that the same subprogram is responsible for determining PI values for every flow at a node at a given instant. Figure 5.10 (left) provides an example: the While loop is not left, while less than 12 entities are queued at the node. Such a condition would not be allowed to include flow criteria, e.g. referring to the flow queue length, which is avoided by recursively the demanding genetic sub-nodes of program control conditions using only node (or network) data. Observe that this does *not* exclude priority index functions whose piecewise definition uses flow criteria as described by genetic If_else nodes; see Figure 5.10 (right) for a simple example[6].

Flow synchronization

[6] Note that, for simplicity, the description of partially evaluating PI programs in Section 5.3 already assumed node synchronization; thus, no need to store program entry points on a per-flow basis.

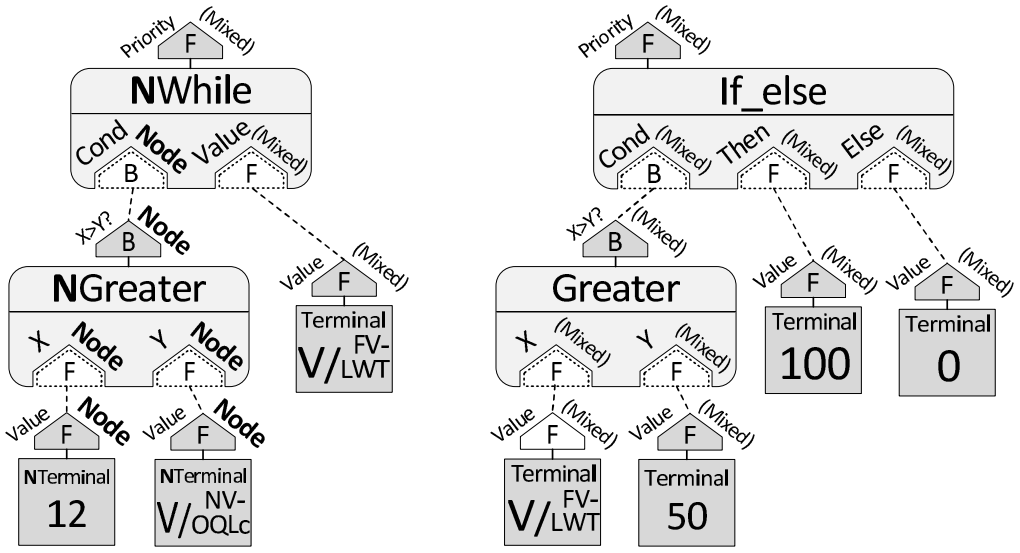

Figure 5.10 – GP: genetic node type restrictions; While (left) versus If_else (right)

5.4.2 Implicitly Reducing the Search Space

Furthermore, means that do not *prevent* but *discourage* solutions with undesired properties were used: artificially degrading the fitness of a PI program reduces its probability of being selected and, thus, being used for offspring creation. Although such programs are not excluded, GP evolution is guided away from them [Koz92, pp. 94ff]. Potentially, two penalties are assigned to a PI program's fitness (i.e. the average entity waiting time at a node, see Section 5.2):

- Fitness penalties proportional to program length [see e.g. Zha95]: to prevent PI grograms from becoming arbitrarily complex (and thus arbitrarily long), a penalty proportional to the length of the program has been assigned that defaults to each character in the program representation as character string counting as a waiting time increase of a hundredth of a second. To be considered fitter than a less complex program, it no longer suffices that a longer program yields a marginally shorter waiting time: an increase in program length has to be *justified* with a waiting time improvement proportional to the extent of this increase; see Section 3.2. This can be interpreted as a stronger version of the minimum description length principle [Iba94] mentioned in Section 5.4.1, ruling out not only more complex program descriptions yielding *identical* behaviour but also longer programs performing only *marginally better*[7].

Justifying complexity

[7] Observe, though, that discarding unnecessary complex programs as in Section 5.4.1 should still be conducted despite applying the stronger minimum description length principle in terms

- Fitness penalties in case where PI program uses fewer than two flow criteria: while PI programs containing no flow criteria are excluded (see above), a default fitness penalty of 10 % is applied to programs that include the input of only a single flow criterion: the underlying heuristic assumption is that existing decentralized node control strategies as proposed by the literature (see Section 4.2) always use more criteria; thus, a single-criterion approach is unlikely to be sufficient: even for simple network configurations like that of OIIC, i.e. single isolated intersections in which the processing rates are much larger than the arrival rates (see Section 4.3), multiple flow criteria are necessary (queue length, potential flow), which justifies the inclusion of an artificial incentive to expedite solution development towards using multiple flow criteria.

For these two cases, a means of implicit discouragement ("penalty") was preferred to explicit exclusion (see 5.4.1) for two reasons: first, to still permit the presence of such PI programs in the *gene pool*, particularly retaining their availably for selection and recombination, though – due to penalties – to a lesser extent than without implicit search space reduction: despite being too complex in comparison to their waiting time results or (yet) containing too few flow criteria, such programs may still contain useful building blocks to combine better programs from: observe that e.g. in the case of a non-normalized program in Section 5.4.1, this is not necessary as the equivalent normalized program is still included in the search space. Second, particularly for the case of confining program length, explicit exclusion would have required the difficult decision of which maximum program length to arbitrarily choose. This gives rise to undesired boundary effects, e.g. a minor extension of relatively long programs performing well being "suddenly" rejected instead of receiving a marginally larger length penalty.

5.4.3 Higher-Level Nodes

Encapsulating subprograms

Furthermore, the convergence of GP-based PI program evolution was improved by providing "higher-level" nodes: such genetic nodes *encapsulate subprograms* that can be heuristically assumed to be useful in more complex programs: the overhead of the evolutionary recreation of such components is removed from the process determining PI programs. A typical example is providing a higher-level node to determine the optimal priority of a single isolated intersection (OIIC) where arrival rates are small in comparison to processing (see Figure 5.11) or the default priority of ONC.

Another means of improving GP convergence is removing the need of "co-evolution" by allowing a sub-tree to be referenced several times: if the results

of complexity penalties: the latter still requires the program to be evaluated; however, detecting unnecessary complex programs saves further computation time since it does not require experiment conduction.

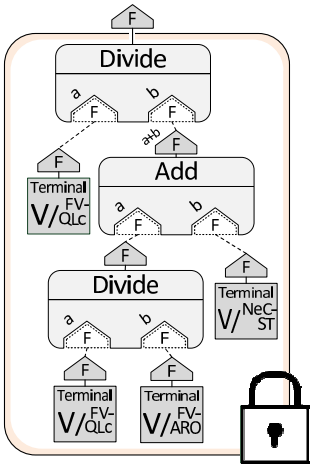

Figure 5.11 – Near-optimal priority index for isolated intersections; see OIIC as described in Section 4.3

determined by a certain sub-tree are used more than once, the current GP approach would be required to create this pattern repeatedly; see the first and last sub-tree of Operator3 operator in Figure 5.12 (left). Allowing multiple references facilitates smaller programs without the need for different branches to undergo the same evolution; see Figure 5.12 (middle).

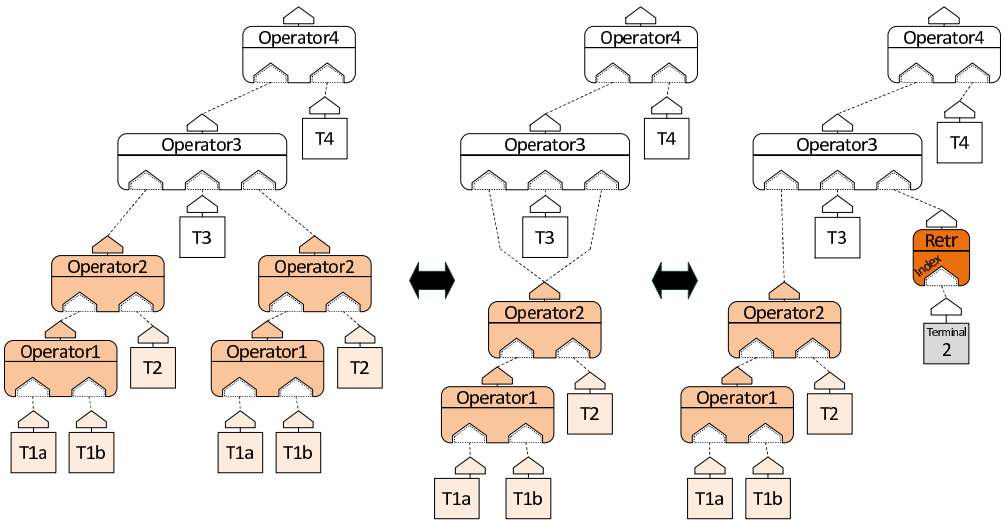

Figure 5.12 – GP: multiple references to a sub-tree [based on Göb11b, p. 143]

To avoid loops in the program tree, which is prone to infinite recursion, this thesis proposes an indirect means of realizing multiple references to a sub-tree: program trees are evaluated depth-first; multiple sub-tree references are implemented by augmenting evaluation such that every genetic node writes its current result not only to the parent node but also to a *retrieve list* assigned to the

top-level Node Programming Logic operator [see Koz92, pp. 473ff]: apart from hierarchically transferring results from the bottom to the top, culminating in a single result as determined by the top-level node, the retrieve list holds the last intermediate results of all nodes in evaluation order.

Retrieve operator

A special genetic operation referred to as *retrieve*, encapsulated in a Retr node, is able to access the last (or second/third to last and so forth, depending on the index provided as a parameter) entry on the retrieve list, see Figure 5.12 (right), in which the last sub-tree of Operator3 retrieves the second to last result so that the program logic described by the right tree is equivalent to the trees in the left and middle. Infinite recursion is prevented, as the *retrieve* operator is able to access only the results of genetic nodes whose evaluation has already been completed; thus, cycles in which, for example, operator A requires the result of an operator B and B, in turn, is based on the result of A, are impossible, as either A can access the result of B in depth-first order or vice versa, but not both. Figure 5.13 for example shows the nodes whose results the retrieve operator of Figure 5.12 (right) is able to access. To limit memory requirements, the number of past results stored in the retrieve list is finite; note that in general, program trees are executed repeatedly, providing, in principle, infinite supply of sub-tree results: the retrieve list stores only a fixed number of results, automatically deleting the least recent entry from a "full" list upon adding a new result.

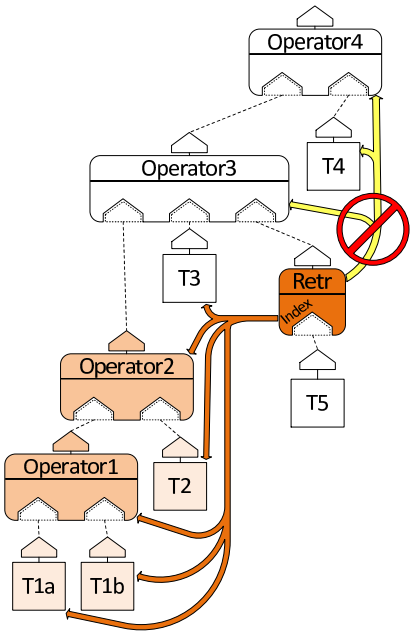

Figure 5.13 – GP: nodes accessible to the retrieve operator in Figure 5.12 (right)

5.4.4 Experiment Control

Cache

Finally, run-time performance is improved by partially avoiding duplicate experiments: fitness values as determined by experiments are stored in a *cache*, mapping program (character string) representations to fitness values. The attempt to add further data if the cache is full leads to the least recently accessed entries being deleted, yielding a so-called *least recently used* (LRU) cache [Sle85, p. 202]; the underlying assumption of efficient caching is that, due to the generation-based development of the gene pool from which new offspring programs are combined, a reproduction of recent programs is more likely than recreating programs that have appeared in previous generations.

Deadlock recognition

Moreover, experiment execution is terminated early if *deadlocks* are recognized: if an entity cannot leave a node because the outgoing link is blocked unless the adjacent node processes at least one entity, the node is referred to as *waiting for* this adjacent node. If the graph formed by this *waiting for* relationship contains cycles, a deadlock has occurred that cannot be resolved unless discarding entities is permitted, which is not true for e.g. urban traffic. In this case, the experiment is terminated immediately, applying a severe fitness penalty by assuming average waiting times that are twice as long as measured this far, heuristically assuming that efficient node control is to be able to avoid deadlocks.

5.5 Summary

Due to the run-time improvements of GP node code control program evaluation described in the previous section, it becomes feasible to practically pursue the means of decentralized transport network optimization proposed in this chapter, namely to conduct genetic programming evolution to generate microscopic rules in terms of PI programs, which requires measuring the "fitness", i.e. waiting time performance, by running simulation experiments. Note that the simulation model used for this purpose *does* require centralized control to set up node PI programs and to measure overall performance data in terms of average waiting times or other global results; the same holds for the real-world deployment of the best means of node control as identified after enough evolutionary cycles. Therefore, this *offline approach* to determining and finally setting up node control *itself* is not decentralized; in fact, it is not even strictly evolutionary in the sense of *Darwinism*, as centralized deployment and retaining the "fittest" programs during evolution can be interpreted as a *Lamarckian* means of actively transferring information similar to social learning in a culture [see e.g. Mat94, p. 16]. However, node control itself as determined and deployed once is decentralized; it will *not* use external or centralized instructions or non-local data.

Not strictly evolutionary

The next chapter will compare this optimization approach to other (e.g. centralized) methods for six different urban traffic scenarios before Chapter 7 provides a conclusion and outlines directions for further research.

Chapter 6

Evaluation

The great tragedy of Science –
the slaying of a beautiful hypothesis by an ugly fact.

— Thomas H. Huxley
Biogenesis and Abiogenesis
(Collected Essays, vol. 8), 1870

In this chapter, the simulation model to evaluate the performance of transport networks in terms of average waiting times or other targets is presented, particularly describing its mesoscopic logic of entity motion in Section 6.1. This permits comparing GP-based transport network optimization, as proposed in Chapter 5, to existing (near-)optimal methods of centralized or decentralized node control in Section 6.2, with the most important findings summarized in Section 6.3.

6.1 Simulation Model

A discrete event simulation model for mesoscopic transport network simulation was built, which is sufficiently parameterizable to represent different kinds of transport networks, such as urban traffic and IP packet routing, including differences in entity motion and the logic of entity processing, which are compared in Table 2.1 on p. 16. This model is based on the open-source simulation framework DESMO-J (Discrete-Event Simulation and Modelling in Java), developed at the University of Hamburg (Germany), which provides a comprehensive set of *Java* classes facilitating discrete event modelling and simulation experiment conduction; see Pag05, Chapter 10, and the web page at `http://www.desmoj.de`.

DESMO-J

6.1.1 Modelling Approach

Discrete-event simulation

A *discrete-event simulation* approach assumes entity state changes occur only at well-defined instants in time [see Pag05, p. 23ff]. Avoiding the application of computationally expensive, numerical techniques to near-continuously approximate entity states, discrete-event modelling is best-suited for models in which either the absence of continuous state changes can be validly assumed (i.e. discrete state changes happen near-instantaneously) or means of aggregating such continuous state changes and resulting behaviour over positive periods of time exist, e.g. a law of motion describing the spatial progress of a uniformly accelerating vehicle and its successors – the latter accelerating themselves, yet bounded such that a safe distance to the previous vehicle is not violated – until the vehicle reaches a maximum speed; then, its velocity is no longer continuously increasing.

Most transport network simulation environments, like *MovSim* (Multi-model open-source vehicular-traffic Simulator; see Tre10 and `http://www.movsim.org`) and *SuMo* (Simulation of Urban Mobility; see Per12 and `http://sumo.sourceforge.net`) as examples of open-source road traffic simulators or telecommunication network simulators like *OMNeT++* (Objective Modular Network Testbed in C++; see `http://www.omnetpp.org`) and *ns3* (Network Simulator3; see `http://www.nsnam.org`), see Wei09, p. 1, conduct microscopic traffic modelling based on discrete event simulation.

However, microscopic modelling is not necessary if macroscopic traffic models can sufficiently accurately estimate the performance of a network, e.g. interpreting vehicles on highways as near-continuous flows (see Section 2.3) or telecommunication networks assumed to operate on streams of data without requiring microscopic traffic details in terms of individual packets to validly determine behaviour of nodes and the overall network; see e.g. Cro04 and Göb06b. In such models, the nodes' processing capacities are expressed in terms of maximum flow rates; nodes serve as forks and joins constraining access to the next link only if the link is congested. However, for the general transport network simulator required for the purpose of this thesis , macroscopic traffic modelling is not appropriate, as entities have to be considered explicitly to determine arrivals at the nodes, queue lengths, feasibility of unprotected turning in urban traffic, and so forth, which cannot be derived from macroscopic traffic flow rates.

General purpose simulation framework

Furthermore, the decision to develop an own simulation model based on a general purpose simulation framework – see the animation screenshot in Figure 6.1 – rather than using one of the microscopic, discrete event simulation environments mentioned above, has two reasons, namely

- providing a general transport network simulation environment not restricted to or at least emphasizing one of the transport network types, permitting, for example, to explore the feasibility of an optimization approach to different network types and

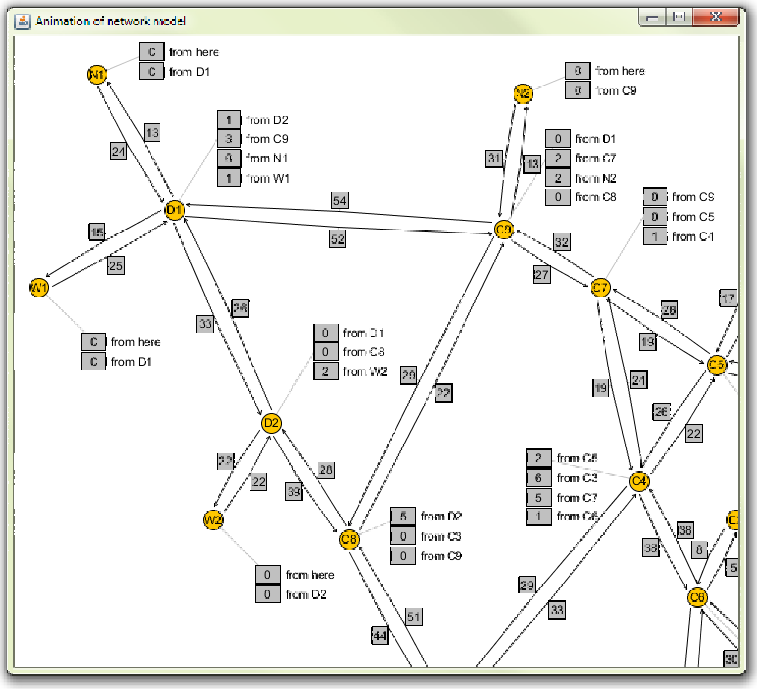

Figure 6.1 – A transport network simulation model animation screenshot

- obtaining a performance gain from applying a *mesoscopic* simulation model based on a queuing network (see Section 2.3), which does *not* include full microscopic entity properties and behaviour, particularly Mesoscopic model
  - urban traffic and production systems not near-continuously tracing entity positions until a node or the tail of a node queue is reached[1] and
  - IP network modelling abstracting from routing and data transmission control at the protocol-level, assuming that appropriate routing tables already exist and that data transmission does not require initiations and confirmations based on explicit "ready to send/receive" or "acknowledgement" signals,

see the entity motion model as described in the next subsection.

6.1.2 Entity Motion

As vehicle motion patterns are more complex than those encountered in IP networks or production systems due to non-instantaneous acceleration, conditional motion, and overtaking, see Table 2.1 on p. 16, the mesoscopic transport network

[1] In the case of IP networks, these two are identical since the entities (packets) do not have a physical dimension.

model used to evaluate the performance of GP node control is derived from the urban traffic queuing model proposed by Nag03, pp. 252ff[2], since it is able to sufficiently precisely determine urban traffic entity arrivals at the nodes, despite mesoscopically *not* requiring the computational complexity of continuously determining each entity's microscopic position and velocity; at the same time, it is abstract enough to apply to other network types.

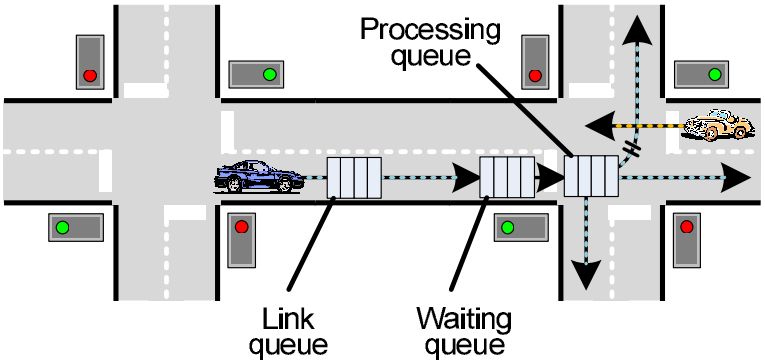

Figure 6.2 – Entity motion model: structure [based on Nag03]

Motion logic

The structure of the motion model is summarized in Figure 6.2: the core concept of the entity motion logic is keeping track of the earliest possible arrival of each entity at the next node by assuming the maximum velocity or acceleration up to maximum velocity e.g. on entering a link after a standstill. This guarantees that an entity cannot reach the next node before it has traversed the full link distance. This earliest possible arrival time, however, is realized by the entity under free flow conditions only. If a preceding entity that cannot be passed exists, particularly if entities are queued in front of the next node, the entity's arrival will be delayed if need be until the preceding entity has left the queue, plus a delay reflecting the necessary safe distance and a further random delay incorporating the different vehicles' and human drivers' nonhomogeneous accelerations and reaction times. Thus, the model abstracts from other microscopic entity interactions than queueing in front of the nodes and the entities' ability or inability to pass each other (the latter depending on the parametrization of the model), thus appropriately describing entity progress in urban networks (but not, for example, in highway traffic) since the behaviour of the traffic light-controlled intersections dominates vehicle motion.

Abstraction

Table 6.1 on p. 101 provides the important details of the mathematical logic of the model as applied to IP networks and urban traffic: assuming an entity has just been processed by a node, the table shows how to determine the earliest

[2] As exception to the statement that urban traffic represents the most complex mode of entity motion, see Table 2.1 on p. 16, packet processing in wireless IP networks may be constrained by remote blocking and wireless interference, which are explicitly included into the simulation model; see below.

Table 6.1 – Entity motion model: mathematical logic [extended from Göb09a, Tables 1–2]

Property	**IP networks (IP packets)**	**Urban traffic (vehicles)**	**Unified view**
Data length ℓ [MBit]	$\ell > 0$	$\ell = 0$	$\ell \geq 0$
Physical length l_p [m]	$l_p = 0$	$l_p > 0$	$l_p \geq 0$
Physical length of distance to next entity l_d [m]	$l_d = 0$	$l_d > 0$	$l_d \geq 0$
Physical space occupied [m]			
• o_l, when traversing a link of length d_l	$o_l = d_l$ (assuming $\frac{\ell \cdot v}{b_l} \gg d_l$)	$o_l = l_p + l_d$	(appropriate case switch)
• o_n, when waiting at a node	$o_n = 0$	$o_n = l_p + l_d$	(appropriate case switch)
Link blocking	if $\sum^L o_l \geq d_l \cdot \#_L$, (max. one packet per channel)	if $\sum^L o_l + \sum^N o_n \geq d_l \cdot \#_L$ (multiple vehicles per lane)	if $\sum^L o_l + \sum^N o_n \geq d_l \cdot \#_L$
Processing and forwarding of entities at a node			
• **Earliest possible arrival at next node** after $t = t_c + t_p + t_t$ [s], yet no earlier than $\frac{\ell}{b_l} + \frac{l_p + l_d}{v} + s_t$ after the previous entity			
• **Earliest departure attempt of the next entity** after $t = t_c + t_p + s_t$ [s]. In wireless IP networks, power available per unit of time and remote blocking may constrain packet sending, e.g. assuming the power requirement of receiving equal to transmitting on a link whose length is 10 % of the maximum distance, yields an idle period of $\max(t_p, \ell\hat{e}(0.01 + (d_l/d_m)^2)/p)$ until the next packet is processed [Göb09b], potentially further delayed if the link is remotely blocked.			
with Position clearing time t_c [s]	$t_c = 0$	$t_c = \frac{l}{v}$ if not queued $t_c = d_v^a(q, l_p + l_d)$ if queued	$t_c = \frac{l}{v}$ if not queued $t_c = d_v^a(q, l_p + l_d)$ if queued
Processing time t_p [s]	$t_p = \ell/\min(b_n, b_{l_{\text{in}}})$ $+\ell/\min(b_n, b_{l_{\text{out}}})$	$t_p = 0$	$t_p = \ell/\min(b_n, b_{l_{\text{in}}})$ $+\ell/\min(b_n, b_{l_{\text{out}}})$
Link traversing time t_t [s]	$t_t = \frac{d_l}{v}$	$t_t = \frac{d_l}{v}$ if not queued $t_t = d_v^a(q, d_l)$ if queued	$t_t = \frac{d_l}{v}$ if not queued $t_t = d_v^a(q, d_l)$ if queued
... where the function $d_v^a(\tilde{s}, s)$ represents the duration of traversing a distance of s after already uniformly accelerating by a over a distance of $\tilde{s}$ up to the maximum speed v:	$d_v^a(\tilde{s}, s) = \begin{cases} \sqrt{\frac{2(\tilde{s}+s)}{a}} - \sqrt{\frac{2\tilde{s}}{a}} & \text{if } \tilde{s} + s \leq \frac{v^2}{2a} \\ \frac{\tilde{s}+s}{v} + \frac{v}{2a} - \sqrt{\frac{2\tilde{s}}{a}} & \text{if } \tilde{s} < \frac{v^2}{2a} \text{ and } \tilde{s} + s > \frac{v^2}{2a} \\ \frac{s}{v} & \text{if } \tilde{s} \geq \frac{v^2}{2a} \end{cases}$		

Additional symbols: a – entity acceleration [ms^{-2}]; b_l – link bandwidth [MBit s^{-1}]; b_n – node bandwidth [MBit s^{-1}]; d_l – link distance [m]; d_m – maximum link distance [m]; $\hat{e}$ – node energy needed per unit of data to transmit over the max. link distance d_m [J MBit^{-1}]; $\#_L$ – number of parallel lanes/channels on a link; $\sum^L$ – all entities on a link; $\sum^N$ – all entities waiting at a node; p – node power [W] ; q – physical distance from the node to the tailback (i.e. sum of lengths of preceding entities enqueued and safe distances) [m]; s_d – safe distance at the maximum speed [m], $s_d = v \cdot s_t$; s_t – safety interval between two entities successively passing the same location [s]; v – desired speed, e.g. minimum of max. link speed and desired entity speed [ms^{-1}]

possible arrival of this entity at the next node and when a potential succeeding entity may depart from the node, subject to the time it took the entity to clear its position, to be processed, and to traverse the next link. However, the departure of the next entity is delayed

- if processing is impossible, e.g. if the flow has been set to "red" (or "yellow" and conditional processing, such as an unprotected turn in urban traffic, is temporarily impossible),
- if wireless energy restrictions or transmission constraints apply, the latter resulting from the sender or receiver being *remotely blocked* due to other transmissions inside their transmission ranges [see e.g. Göb11a, p. 332], or
- if the outgoing link is blocked or the packet is lost due to data errors; thus, the next entity cannot be processed before the previous entity successfully repeats its departure or transmission attempt.

Validation of the transport network model as *mesoscopic* traffic model can, by design, only address the (partial) microscopic properties and behaviour modelled explicitly on the one hand and overall macroscopic traffic data on the other hand. During the course of this thesis , *operational validation* [see Pag05, p. 16] was limited to special transport network configurations; e.g. in the case of urban traffic, microscopic node performance (throughput, waiting times) and overall travel times in a network consisting of a single arterial subject to fixed-time green wave programming (see evaluation of Scenario 4 in Section 6.2.4) for different traffic densities closely matches the results of *MovSim*; the same holds for global throughput and overall travel times as macroscopic performance measures. For wireless IP packet transmission, the macroscopic behaviour (throughput, bandwidth usage) of the simulation model described in Göb09b, Sections 2–3[3], could be reproduced. The restriction to such special cases leaves the task of exhaustive operational validation to further research; however, observe that for the purpose of this thesis , absolute validation is not strictly required – a model to at least *plausibly* compare different means of node control and identify the most efficient means suffices.

Operational validation

6.1.3 Simulator Design

As a basic modelling decision, the *event-oriented world view* of discrete event simulation [Pag05, pp. 108ff] was used to describe the behaviour of the transport network, i.e. system dynamics arise from sequentially executing *events* which describe the changes the components of the system undergo at specific instants in time, e.g. the instant at which an entity is created at its origin node, a node

Event-oriented world view

[3] Observe that *node motion* as described in Göb09b, Section 4, however, is *not* supported.

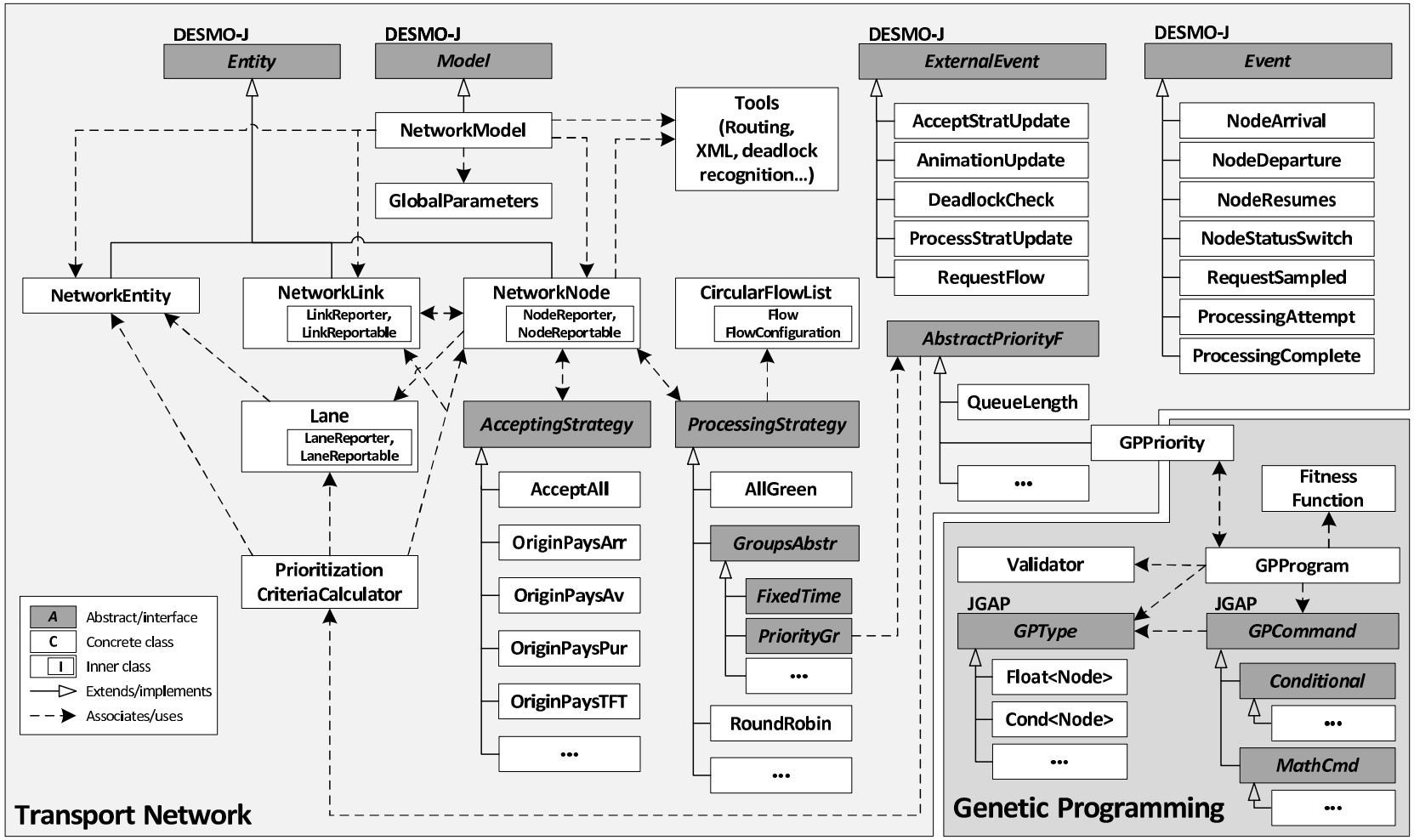

Figure 6.3 – Simulator classes

finishes processing an entity, a phase switch occurs, or the animation window is updated (see Appendix C, pp. 156ff). Observe that despite complex patterns of network dynamics *emerging* from executing such events, the behaviour of each single entity and node is relatively simple although depending on various conditions to evaluate. For example, on arrival at a node, an entity that

Arrival event

- has not yet reached its destination,
- has not been lost due to wireless interference (if appropriate), and
- that is not discarded by the node, e.g. due to insufficient buffer space

is processed immediately if all necessary preconditions are fulfilled, namely

- the relevant lane (queue) is empty,
- the relevant flow is set to "green" or "yellow",
- conditional processing is possible if the flow is set to "yellow", e.g. unprotected turning in urban traffic requiring the absence of opposing traffic,
- no entity already being processed on the lane,
- neither the node nor the outgoing link is blocked, and
- no other node inside the sender's and receiver's transmission range already processing an entity (i.e. remotely blocked) and energy available for wireless transmission at the node suffices to start processing immediately (if appropriate).

However, if at least one of these conditions is violated, the entity is queued for processing; see also the example of conditions required for processing a subsequent entity after an entity departure already stated in Section 6.1.2. The complexity of network behaviour (see Section 2.2) is driven by the relationships between entities and nodes influencing each other indirectly, as described in these conditions, while explicit synchronization requirements are elementary: the only time-consuming activities are nodes processing entities and entities traversing links. Therefore, using the *process-oriented world view* [Pag05, pp. 98ff] instead of event modelling, i.e. describing the behaviour of the system by encapsulating all time-consuming activities conducted by each entity during their existence into a so-called *life-cycle*, improves conceptual clarity only to a limited extent: the description of entity and node behaviour would be less fragmented (distributed over 3–5 process types instead of 12 event types), yet the conditions as described above, which represent the most important limitations of conceptual clarity, do persist. Therefore, event-based modelling was chosen for the purpose of developing a transport network simulation model based on DESMO-J: process modelling would have provided minor improvements in transparency e.g. by providing a more compact

Process-oriented world view

representation of the entities' behaviours and thus also improved code readability and maintenance, yet at the price of a significantly worse run-time performance. Note that the ability of processes to (conceptually) act in parallel is based on dedicated *Java threads* assigned to each process [Pag05, p. 274] due to *Java*'s lack of *Modula*-like *coroutines* as a light-weight means of suspending and resuming code execution. Particulary, a process-oriented transport network would have limited the number of entities active simultaneously to approximately 5,000–10,000, depending on the parametrization of the *Java virtual machine* in use, while the event-based model is able to handle 100,000 or even 1,000,000 entities in parallel; these are observations from own experiments.

Figure 6.3 on p. 103 shows a *Unified Modelling Language* (UML) class diagram [see e.g. Pag05, pp. 65ff] of the transport network simulator based on DESMO-J developed during the course of this thesis . Observe that only the most important classes and extension/association relationships are included. Furthermore, in some cases, class names are abbreviated; see Appendix C, which also contains details about most of the classes. UML

Node Control

Based on modelling the components of the network and their behaviour, this thesis focuses on designing efficient means of node control: therefore, strategies describing whether or not to accept entities arriving at a node (based on `AbstractEntityAcceptingStrategy`, short version AcceptingStrategy in Figure 6.3) and how to prioritize entities to be processed (derived from `AbstractEntityProcessingStrategy`, in short ProcessingStrategy) are encapsulated into their own classes, facilitating substitution; based on an *Observer* design pattern [Gam94, pp. 293ff], these strategies are notified on the arrival and departure of an entity, on an outgoing link being blocked, on dropping an entity due to insufficient buffer space (telecommunication networks only), and at regular intervals every Δ seconds. In line with the event-oriented world view, the reception of such notifications permits the strategies to update their internal states and act accordingly if need be, e.g. by triggering phase switchs. A special case of processing strategies are subclasses of `PriorityGroupsAbstract` (in short, PriorityGr) representing strategies using a priority index function or program, which, in turn, has to be derived from `AbstractPriorityFunction` (in short, AbstractPriorityF). One such subclass of `AbstractPriorityFunction` is `GPProgramPriorityFunction` (in short, GPPriority), which is able to read out PI values as computed by a GP-engineered PI program to be evaluated; see Section 5.3, particularly Figure 5.8 on p. 87. Therefore, `GPProgramPriorityFunction` represents the interface between transport network simulation and GP-based node control evolution. For more details about most transport network simulator classes the reader is again referred to Appendix C. Accepting Processing Interface to GP

Application

Beyond the core entity motion model based on a queuing network, the simulation of wireless IP networks is addressed by supporting *remote blocking* (see above and Section 2.1.4), as well as packet transmission losses due to data errors or wireless interference. The latter are covered stochastically, yielding a probability of a packet being lost to apply each time a packet reaches a node; see Göb09b, p. 2431, for further details about how wireless interference is modelled. Furthermore, specifically targeting urban traffic, protected and unprotected turning is supported; flows to give way to are automatically detected if turning is unprotected. Phases of different flows may also overlap, see Figure 6.4, providing partial phase switches: flows enabled both before and after a phase switch do not need a safety period, despite other flows being set from "red" to "green" or vice versa.

Phase overlapping

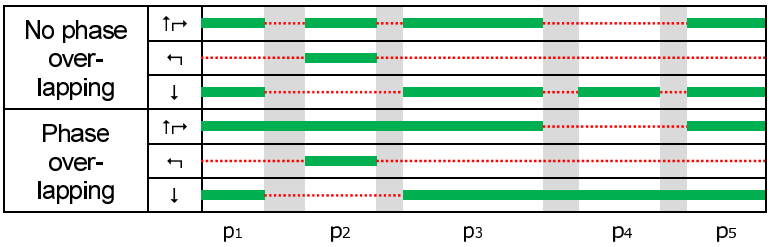

Figure 6.4 – Phase overlapping disabled (top) and enabled (bottom)

Further features include a description of a network configuration (network type, topology, traffic scenario) using XML files and a basic online animation (see Figure 6.1 on p. 99); the latter is based on the Piccolo2D graphics framework, which efficiently supports drawing zoomable 2D animations; see Bed04 and the web page at `http://www.piccolo2d.org`[4].

Simulation experiment results, as automatically collected by DESMO-J and displayed in the *experiment report* [Pag05, p. 283], contain the following data (selection):

Report

- Network: total number of entities, ratio of entities that have reached their destination, entities lost by reason (discarded by a node, no route found, interference), and average waiting and travel times.
- Nodes: total number of entities originating/relayed/terminating, ratio of entities that have reached their destination, originating/relayed/terminating entities lost by reason (see above), throughput, usage of bandwidth/energy, number of phase switches, Jain's fairness index[5].

[4] Since version 2.3.0 from April 2011, DESMO-J has its own 2D animation component, see web pages `http://www.desmoj.de` and `http://www.wi-bw.tfh-wildau.de/~cmueller/SimulationAnimation`, which, however, was not used in this thesis since it started earlier.

[5] Named after Raj Jain (1951–*), *Jain's fairness index* [Jai84, pp. 5ff] is a measurement of how equally or unequally entities from different classes (here: entities sharing the same origin or destination) are treated in terms of processing resource allocation.

- Lanes: total number of entities, total "green" time, throughput, average waiting times.
- Links: total and average number of entities, ratio of time blocked.

Completing the implementation of the transport network simulation model described above, as well as the GP-based means of PI program evolution described in the previous chapter, permits proceeding to empirically evaluate the performance of GP-based node control.

6.2 Experiments

To empirically answer the question of whether the proposed scheme of decentralized transport network optimization based on genetic programming is competitive to existing centralized or decentralized approaches of node control, six scenarios representing different transport network configurations are investigated: network type is urban traffic; topologies and traffic pattern are as follows:

Scenarios

- Scenario 1 (S1) consists of a single isolated intersection with incoming traffic from two directions.
- In Scenario 2 (S2), two intersections are located only 100 meters apart. Symmetrical traffic is restricted two flows whose service is mutually exclusive at both intersections; see Figure 6.5 on p. 111.
- Scenario 3 (S3) is an irregular network consisting of 11 intersections plus 11 entry/exit nodes from southern Hanover (Germany), subject to various flows from almost any entry to almost any exit; see Poh10, Section 3.3 and Figure 6.7 on p. 115.
- Scenario 4 (S4) contains six successive intersections of a single arterial during a rush hour; see Figure 6.8 on p. 115.
- Scenario 5 (S5) assumes that the centralized traffic light control at one intersection of S4 is out of order, so that this intersection defaults to suboptimal fixed-time programming, permitting analyzing the impact of such a faulty node to the overall network performance.
- Scenario 6 (S6) modifies the traffic pattern of S4 by assuming a temporary heavy additional load from one of the side roads onto the arterial.

Finally, investigating the universal application of the GP approach, a final experiment investigates whether a *single* set of node control rules suitable for S2, S3, S4, *and* S6 can be evolved.

Universal control

Parametrization

Unless explicitly stated otherwise, parameters are as follows: the target is the minimization of entities' overall waiting times; the simulation duration is 5,000 seconds. Experiment results as displayed in Tables 6.2–6.8 represent batch means of 10 runs[6]. Calculation confidence interval uses a confidence level of 95 % and assumes that waiting time samples from the batch runs are approximately normally distributed; observe that each sample as determined in a single run is itself a mean of many vehicle waiting time samples.

Vehicular traffic

The parameters defining traffic itself are adopted from Kes09, pp. 7ff: sampled from uniform distributions, vehicles' lengths are between 2 m and 3 m, acceleration varies between 1.0 ms^{-1} and 1.5 ms^{-1} up to a desired speed between 80 % and 120 % of the speed limit (default 50 kph), unless hindered by preceding vehicles or traffic lights. The safety interval is 1.5 seconds, i.e. the minimum distance a vehicle keeps to its predecessor is the space traversed at the current speed in 1.5 seconds, but no less than 0.5 m, e.g. when queued. Inter-arrival times of vehicles at the boundaries of the network are exponentially distributed. The

PI-based node control

intersection safety period between switching a flow to "red" and a competing flow to "green" is 3 seconds. GP node control based on PI programs assumes a minimum phase length of 5 seconds, during which a flow remains set to "green" despite PI values may potentially recommend switching earlier. Stabilization further requires the PI value of a new set of flows to be set to "green" exceeding the priority of the current set by at least 20 % to prevent node control from oscillating in short cycles; moreover, a longer "red" period serves as a *round robin* tie-breaker among equally prioritized flows; see Section 5.2.

GP evolution

As already mentioned, GP node control evolution is based on the infrastructure of JGAP (Java Genetic Algorithms Package, see Mef12 and Section 5.2); it uses all genetic nodes and the extensions to GP control described in Chapter 5, except that higher-level nodes (Section 5.4.3) are not used: therefore, the experiments probe the GP's ability to evolve efficient node control "from scratch", i.e. assuming no previous knowledge about suitable higher-level nodes; observe that e.g. in the case of an unknown transport network type or configuration, such knowledge cannot be presupposed. GP evolution lasts 25 generations, the population size is 100. GP fitness is the average waiting time in milliseconds (the smaller, the fitter) subject to the penalties described in Section 5.4.2. The best program is always transferred from one generation to the next; a total of 10 % of a generation is filled with survivors selected from the previous generation. The remainder of a new generation is recombined as offspring from the previous gen-

[6] To save computation time, preliminary evaluation of genetically engineered programs uses only single runs; however, the evaluation of the best program of each generation is repeated 10 times, thus achieving more precise estimates of their performance. The heuristical expense is that efficient programs may be not identified and, therefore, evolutionarily disregarded if their (single) performance sample was biased.

eration. The genetic operators (selection, recombination, mutation) in use are described in Section 5.1.3 on p. 81; tournament selection is based on subpopulations of size 5, the mutation probability of each node being replaced by a random node whose input and result parameter types do match is 5 %. Should no such alternative node be available, a node with different input parameters is randomly chosen, in this case picking appropriate sub-nodes at random . The copy of the best program transferred to the new generation is *not* mutated; thus, the performance of the best program available is monotonically increasing during the evolutionary process, in which the best program cannot be lost.

Note that Appendix D shows the "fittest" programs found for S1 to S6 and the combined scenario.

6.2.1 S1: Single Isolated Intersection

Scenario 1 is a single isolated four-way intersection. Traffic is offered from N and W on single lanes. All vehicles proceed straight ahead, i.e. the two flows can only be processed mutually exclusively. All other conditions, including arrival rates and processing rates (the latter depending on speed limits, acceleration, and safe distances), are identical.

As this basic scenario of urban traffic control is symmetrical, no reasons exist to prefer serving one of the flows to the other, apart from delaying switching as long as possible: observe that outgoing link blocking is not possible, as further intersections do not exist. Thus, a near-optimal strategy is serving each flow alternatingly ("round robin") until the queue is empty; see Section 4.3 and Läm07, Chapter 4: switching earlier is not optimal, as intersection capacity would be given away due to the additional switching penalty incurred; switching later wastes capacity since no vehicle is served[7]. The same holds for OIIC[8] and ONC (see Section 4.3), whose presumptions, namely flows continuous and arrivals marginal (OIIC) or uniformly distributed (ONC), are not strictly fulfilled.

Alternating service

S1 was evaluated applying three different load levels ("low", "medium", "high" load), approximately covering 33 %, 66 %, and 100 % of the saturation capacity of the intersection. The results as shown in Table 6.2 also include a hypothetical "free flow" node control assuming all flows set to "green" all the time, proving a bound to performance measures like average waiting time and throughput. ONC requires parameter values being determined; see Section 5.3, p. 87. This was done by fully exploring all combinations of a critical queue length $\mathsf{q} \in [5;100]$ (granularity of 5), minimum phase duration $\mathsf{m} \in [3\,\mathrm{s};15\,\mathrm{s}]$ (granularity of 1 s), and full maximum stabilization period $\mathsf{p} \in [30\,\mathrm{s};150\,\mathrm{s}]$ (granularity of 10 s) such that

[7] This is true unless the next arrival of a vehicle not yet queued is marginally close at hand so that it might be optimal to serve this additional vehicle and switch immediately afterwards; thus, round robin is only a *near*-optimal strategy.

[8] Note that OIIC is identical to alternating round robin service in case of (only) two flows.

the average waiting time in all three load levels was minimized, yielding a critical queue length of 19, a minimum phase duration of 8 s, and a maximum stabilization period of 100 s. For genetic programming (GP), the performance of the best node control programs available after 5, 10, and 25 generations is included in the table. Similarly to the parametrization of ONC, the GP scheme was required to apply the *same program* to all three load levels, thus emphasizing general applicability without inadequately specializing to one of the load levels (overfitting).

General applicability

Table 6.2 – Experiment results: S1

Scenario	Node control		Avg. wait per node [s]	Avg. throughput	Comp. duration [(d) hh:mm]
S1/low load	Free flow		0.1	3110	< 00:01
	OIIC		2.1	3105	< 00:01
	ONC		3.1	2934	02:30
	Round robin		2.0	3105	< 00:01
	GP	Gen. 5	2.6	3101	04:11
		Gen. 10	2.4	3103	19:58
		Gen. 25	2.1	3104	2 d 12:27
S1/med. load	Free flow		0.1	6268	< 00:01
	OIIC		5.5	6245	< 00:01
	ONC		5.9	6208	02:30
	Round robin		5.5	6247	< 00:01
	GP	Gen. 5	6.7	6235	04:11
		Gen. 10	5.6	6247	19:58
		Gen. 25	5.4	6247	2 d 12:27
S1/high load	Free flow		0.2	8997	< 00:01
	OIIC		39.2	8674	< 00:01
	ONC		39.2	8670	02:30
	Round robin		39.3	8669	< 00:01
	GP	Gen. 5	37.6	8703	04:11
		Gen. 10	36.6	8710	19:58
		Gen. 25	36.4	8704	2 d 12:27

Avg. wait time confidence interval half widths are smaller than 0.1 s for low/med. load and smaller than 0.4 s for high load. Avg. throughput confidence interval half widths are smaller than 10.

The throughput of round robin and OIIC (as expected not significantly differing from each other) is only slightly worse than under free flow conditions, especially for low and medium load, see the 95 % confidence intervals as displayed under the table; of course, they are subject to longer waiting times, though . ONC is inferior to round robin and OIIC for low and medium load (recall the same parameter values are applied in each load level), yet performance is approximately even under near-saturated conditions. ONC's computation duration is due to parametrization; see above.

GP is able to evolve a means of node control performing similarly to round robin and OIIC after 25 generations for low and medium load; for high load it even achieves slightly better results as the PI function (see Table D.1 in Appendix D) includes vehicles arriving during the next seconds, thus potentially delaying switching despite the queue being empty due to imminent arrivals, which is particularly useful under high load as avoiding safety periods conserves processing capacity. Under low and medium load this is no significant advantage as processing capacity is less scarce in comparison to processing demand.

6.2.2 S2: Two Intersections in Close Proximity

The next scenario (see Figure 6.5) investigates two intersections for which traffic light synchronization is necessary: only two flows exist, namely W→N and E→S, competing for processing capacity at both intersections (since right-hand traffic is assumed). Flows again are symmetrical in terms of arrival rates and velocities. Since the length of the connecting link between the intersections is limited (100 m), node control is prone to *dynamic instabilities*; see Section 4.3: traffic lights have to be set such that states in which one of the nodes wastes capacity due to its inability to serve either of the flows occur as infrequently as possible. Such undesired states arise if either the "outside" queues from W/E are empty or the link towards the other intersection is fully congested (thus, no further incoming vehicles from W at the western intersection or from E at the eastern intersection can be served), while simultaneously at the "inside" queues, no waiting vehicles bound for N/S already served by the opposing node exist. Traffic offered is close to saturation as otherwise, traffic light synchronization is not necessary to achieve an acceptable waiting time or throughput performance.

Dynamic instabilities

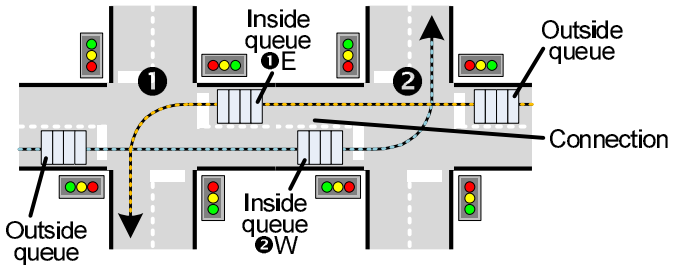

Figure 6.5 – Network topology: S2 (two intersections in close proximity; see also Figure 4.4 (left, top) on p. 60)

A near-optimal centralized solution for S2 is "fill & clear" [see Kum90], exploiting the symmetry of the traffic flows offered. First, the flows from "outside", i.e. from W at the western intersection and from E at the eastern intersection, receive "green" until the connection between the intersections is filled or until both "outside" queues are empty. After the connection has been filled, the vehi-

Symmetrical service

Table 6.3 – Experiment results: S2

Scenario	Node control		Avg. wait per node [s]	Avg. throughput	Comp. duration [(d) hh:mm]
S2	Free flow		0.2	5230	< 00:01
	Fixed time		10.6	5063	00:10
	Queue lengths		957.9	410	< 00:01
	OIIC		11.7	4877	< 00:01
	ONC		6.1	5195	01:41
	Fill & clear		9.3	5155	00:12
	GP	Gen. 5	14.8	4676	00:47
		Gen. 10	6.2	5191	03:54
		Gen. 25	6.2	5193	06:11

Avg. wait time confidence interval half widths are smaller than 0.2 s (exception: smaller than 25 s in case of queue lengths node control). Avg. throughput confidence interval half widths are smaller than 10 (exception: smaller than 30 in the case of queue lengths node control).

cles on the connection receive "green" at both intersections until the connection (including the "inside" queues) is cleared. By a series of experiments, the optimal maximum number of vehicles to "fill" the connection was determined, which is 13 for this scenario: allowing more vehicles onto the link permits more vehicles served per cycle (and thus saving safety periods since fewer cycles are necessary), which, however, is more than compensated by the longer duration of clearing the connection as the vehicles have to slow down. Filling the connection with a maximum of (only) 13 vehicles permits green wave-like passing of the connection and the second node without deceleration. Assuming sufficient supply of incoming vehicles, neither of the intersections ever wastes capacity apart from symmetry deficits caused by random noise, e.g. one intersection clearing its vehicle queue on the connection faster than the other.

The results shown in Table 6.3 indicate fill & clear is more efficient in terms of average waiting times and throughput than the best fixed time node configuration in which with a granularity of 0.5 s, all phase lengths between 5 s and 60 s have been probed; optimal was a phase duration of 20.0 s. This fixed-time scheme in turn still performs better than OIIC or particularly always serving the longest queue, using a bound of 20 % by which another queue's length must exceed the length of the queue currently served in order to switch since these approaches do not synchronize node phases. ONC[9] and GP control, however, significantly outperform fill & clear, the latter at least from generation 10 onwards. Despite the fact that fill & clear is theoretically optimal when flows are assumed continuous and uniform, ONC and GP are not only able to implicitly conduct phase

[9]The parameters in this and the following experiments are determined the same way as in the S1 experiment.

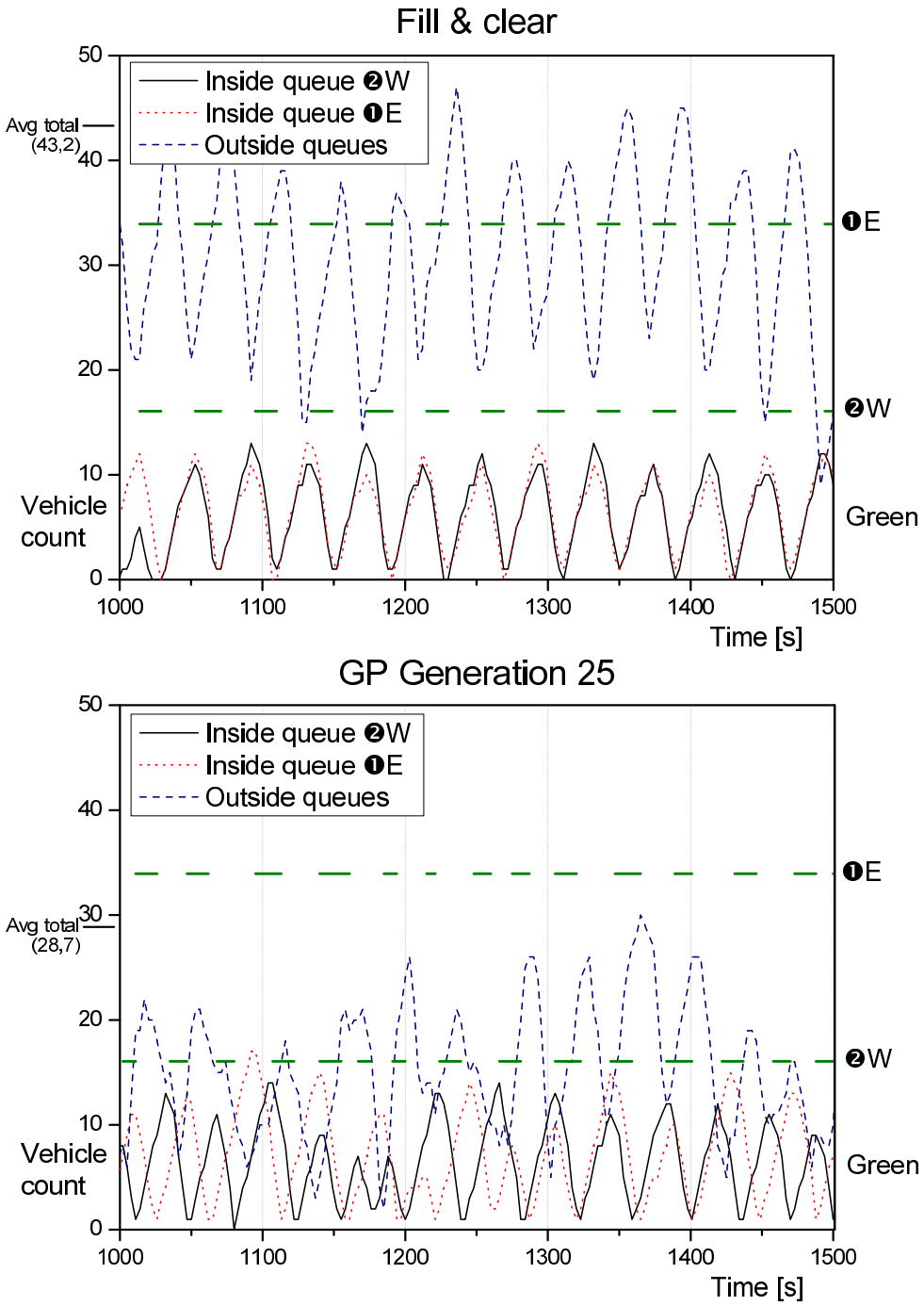

Figure 6.6 – Vehicles on the connection W→E including queue ❷W, on the connection E→W including queue ❶E, and vehicles waiting "outside" (left scale) and "inside green" phases (right scale) in a S2 single sample experiment run subject to arrivals at the same instants, exemplarily showing the period between seconds 1000 and 1500: fill & clear (top) and GP as of Generation 25 (bottom)

synchronization, they can also asymmetrically adjust phases due to minor perturbations caused, for example, by one or more vehicles that are exceptionally slow or by unusually many/few arrivals at one boundary during an interval, thus deviating from strict synchronization when appropriate; see Figure 6.6 for a plot of a 500 seconds period of fill & clear versus GP (generation 25) node control. The overall number of entities queued is much lower for GP (28.7) than for fill & clear (42.3): modifying the strict synchronization scheme as conducted by fill & clear, GP node control admits more vehicles onto the connection, when possible.

Supraoptimality

Thus, S2 is an example of *supraoptimality*; see Section 4.2: local adaption permits a performance superior to a strategy proven optimal when assuming flows are continuous and uniform.

6.2.3 S3: Irregular Topology

Next, a larger topology subject to more irregular and complex patterns of traffic was investigated, namely a minor simplification of a southern suburban area of Hanover (Germany) as described in Poh10, Section 3.3; see Figure 6.7. Dimensions are approximately 6×5 kilometers, including 11 intersections plus another 11 entry/exit nodes at the boundaries. The largest flows are northeast to northwest and south to (splitting up) northwest and northeast; minor flows occur between any pair of boundary nodes. Nodes are at least 500 m away from each other so that node synchronization is less critical than in S2. Similarly to S1, three different traffic intensities were applied, again requiring a single GP PI program (and ONC parametrization) to serve all load levels, which this time include over-saturating the network ("overload"). Observe that even free flow node control yields positive expected waiting times under such conditions; see the results in Table 6.4. Despite the fact all flows are assumed set to "green" at every intersection, vehicles may have to wait, as the traffic demand resulting from merging multiple flows may exceed the link's capacity so that the link is temporarily blocked.

Since no centralized/globally (near-)optimal node control is available, for comparison purposes Table 6.4 includes ONC and the heuristic always serving the longest queue; parametrization of the latter requires setting a bound by which another queue's length must exceed the length of the queue currently served in order to switch, which is set to 22 %, yielding the best performance out of probing all bounds between 4 % and 150 % in steps of 2 %. This heuristic performs significantly better than OIIC as serving longest queues – in the absence of S2-like synchronization requirements, including admitting *neither* too few *nor* too many vehicles to an outgoing link – most likely prevents the links from blocking, since nodes are unlikely to be incapacitated due to outgoing links not able to accept further entities. The justification of serving the longest queues is heuristically assuming that the nodes are productive most of the time implies setup times have to be minimized. This approach considerably outperforms OIIC. ONC, in

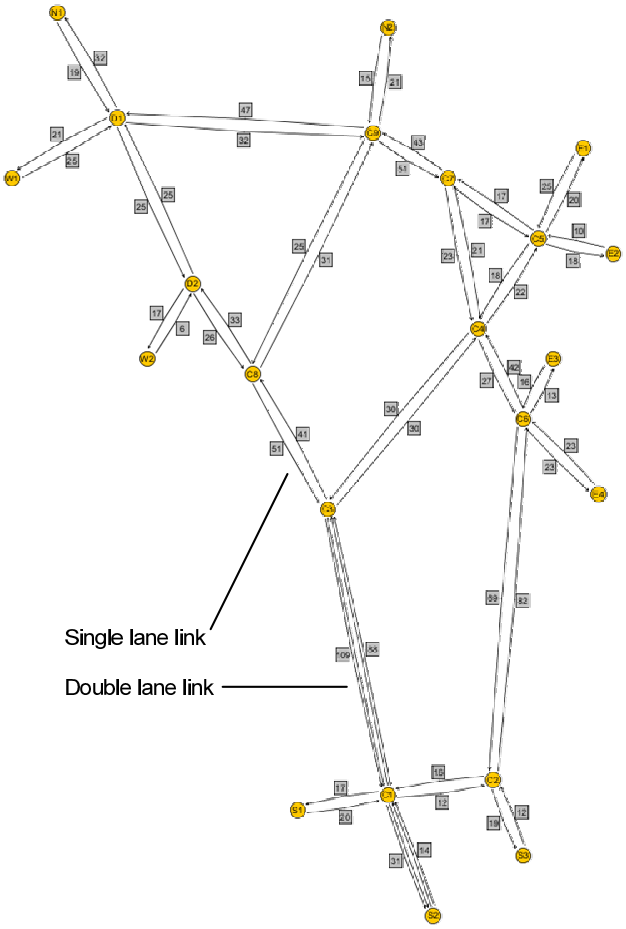

Figure 6.7 – Network topology: S3
(see Göb11b, p. 141, based on Poh10, Section 3.3)

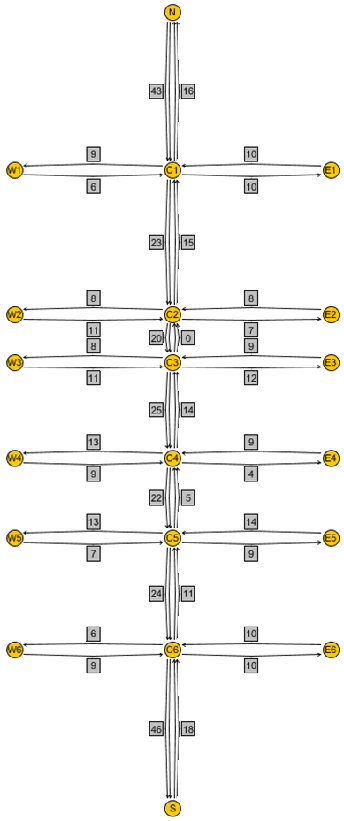

Figure 6.8 – Network topology: S4 (an arterial)

turn, yields smaller waiting times, except in the case of the lowest load level, in which serving the longest queues suffices to prevent links from blocking; recall link lengths are longer than in S2. However, node control as evolved by genetic programming, incorporating more detailed node data (flow queue length, overall/maximum queue lengths, potential flow rates, duration of current phase, status idle or not), is significantly more efficient than any other means of node control for high load and overload traffic densities, yet not for medium load.

6.2.4 S4: Arterial

Providing a topology consisting of more than two nodes with at the same time centralized near-optimal global node control available for comparison purposes, S4 considers a single arterial (5 kilometers, 6 intersections); see Figure 6.8: assuming

Table 6.4 – Experiment results: S3

Scenario	Node control		Avg. wait per node [s]	Avg. throughput	Comp. duration [(d) hh:mm]
S3/med load	Free flow		0.1	16803	< 00:01
	OIIC		10.7	16011	< 00:01
	Queue lengths		3.6	16728	03:21
	ONC		4.5	16718	12:45
	GP	Gen. 5	5.3	16689	09:22
		Gen. 10	5.5	16673	23:17
		Gen. 25	4.2	16720	4 d 21:32
S3/high load	Free flow		7.5	21479	< 00:01
	OIIC		83.3	16960	< 00:01
	Queue lengths		35.1	19853	03:21
	ONC		17.3	21162	12:45
	GP	Gen. 5	14.4	21358	09:22
		Gen. 10	17.1	21444	23:17
		Gen. 25	13.9	21389	4 d 21:32
S3/overload	Free flow		63.8	24043	< 00:01
	OIIC		204.5	17187	< 00:01
	Queue lengths		151.4	18934	03:21
	ONC		115.1	20741	12:45
	GP	Gen. 5	88.2	21298	09:22
		Gen. 10	76.9	22403	23:17
		Gen. 25	77.9	22493	4 d 21:32

Avg. wait time confidence interval half widths are smaller than 0.2 s for low load and smaller than 0.5 s for med./high load. Avg. throughput confidence interval half widths are smaller than 40.

Green wave

rush hour traffic from N to S, which is five times larger than the traffic in the opposing direction or crossing the arterial at the side roads yields a *green wave* N→S near-optimal [see Coo08], permitting the vehicles to form a platoon at the most northern node and traverse the remainder of the network without stopping. Further assuming that there is no turning (i.e. all vehicles traverse the system by staying on the N↔S arterial or one of the W↔E side roads) and identical traffic densities on all side roads for symmetry reasons implies that only two sets of flows that can be set to "green" simultaneously exist (either both arterial flows or both side road flows, as no other non-exclusive combination of flows is feasible) and that the best fixed-cycle node control uses the same cycle length and cycle split at all intersections, subject to suitable offsets to permit the emergence of the desired green waves. By again systematically conducting experiments, optimal fixed-cycle control phase durations were identified as 35 seconds (arterial) and 4 seconds (side road), plus – mimicking the behaviour of SCOOT [Bre12a] – an actuated extension of up to 5 seconds if queues are not cleared (added to the next

phase if not used), which yields a fixed cycle of 55 seconds including two safety periods of 3 seconds each.

Table 6.5 – Experiment results: S4

Scenario	Node control		Avg. wait per node [s]	Avg. travel time [s] N→S	Avg. travel time [s] S→N	Avg. travel time [s] Side	Comp. duration [(d) hh:mm]
S4	Free flow		0.1	441	423	158	< 00:01
	Queue lengths		3.6	489	453	165	02:00
	OIIC		3.7	483	464	168	< 00:01
	ONC		3.5	486	453	169	07:59
	Green Wave N→S		4.2	464	435	183	04:49
	GP	Gen. 5	3.1	482	451	164	07:12
		Gen. 10	3.1	477	445	167	1 d 15:56
		Gen. 25	2.9	479	446	165	3 d 20:23

Avg. wait time confidence interval half widths are smaller than 0.2 s.
Avg. travel time confidence interval half widths are smaller than 3 s.

In terms of overall expected waiting times, Table 6.5 shows that green wave programming is slightly less efficient than OIIC, ONC, or the node control heuristic based on queue lengths; none of them are restricted by cycle constraints. Observe, though, that the green wave approach achieves best travel time results for N→S as the rush hour traffic direction. As of generation 25, node control evolved by GP performs significantly better in terms of overall waiting times, which is the target of evolution; see Section 6.2.

However, the rush hour direction traffic efficiency of green wave programming is not reached; observe, though, that by, for example, appropriately modifying the fitness function such that the weight of rush hour vehicles is emphasized, GP could be directed towards preferring rush hour traffic.

6.2.5 S5: Arterial with a Faulty Intersection

Scenario 5 modifies S4 by assuming traffic is perturbed: a faulty traffic light at the third out of the six intersections (counting from the north, i.e. in rush hour direction) is assumed to apply a suboptimal fixed-time control, periodically switching between the arterial flows set to "green" for 90 seconds and the side road flows for 30 seconds. This modification of the third intersection also applies to "free flow" node control.

Table 6.6 shows the robustness of green wave programming and GP node control: although overall results are inferior to S4, unmodified green wave node control as of S4 still yields better rush hour traffic travel times, while in terms overall expected waiting times, GP node control as evolved in S4 is still significantly more efficient than green wave programming, OIIC, ONC as of S4, or a version of the queue lengths heuristic whose bound when to switch was optimally adjusted

Robustness

Table 6.6 – Experiment results: S5

Scenario	Node control		Avg. wait per node [s]	Avg. travel time N→S	Avg. travel time S→N	Avg. travel time Side	Comp. duration [(d) hh:mm]
S5	Free flow		1.6	457	431	165	< 00:01
	Queue lengths		4.5	499	459	174	02:20
	OIIC		4.5	497	465	174	< 00:01
	ONC (S4)		4.4	498	459	174	s. Tab. 6.5
	Green Wave N→S (S4)		5.0	481	442	187	14:49
	GP (S4)	Gen. 5	4.5	497	471	173	s. Tab. 6.5
		Gen. 10	4.0	490	451	174	s. Tab. 6.5
		Gen. 25	4.0	492	452	171	s. Tab. 6.5
	GP (new)	Gen. 5	4.5	497	470	174	03:26
		Gen. 10	4.0	496	454	174	11:15
		Gen. 25	3.8	490	450	172	1 d 18:15

Avg. wait time confidence interval half widths are smaller than 0.2 s.
Avg. travel time confidence interval half widths are smaller than 4 s.

to this scenario. The robust applicability of GP node control determined during optimizing S4 is further underlined by a new GP execution to evolve a PI program specifically adapted to S5 only yielding insignificant additional improvements; see Table 6.6 (bottom).

6.2.6 S6: Arterial with Traffic Perturbed

As a second modification to S4, in this scenario, the flows themselves are dynamically adjusted: the scenario assumes the end of a football match after 5000 seconds, causing side-road traffic at intersection 4 (again counting from the north) thrice as large as the side road traffic at the other intersections, yielding this intersection over-saturated, i.e. queues building up. Furthermore, the departing sport fans are assumed turning into the arterial (evenly distributed into N and S direction), thus affecting the other intersections as well. After an additional 5000 seconds, sports traffic ceases so that traffic conditions – particularly the queues at S4 – are able to recover. Simulated duration accordingly is extended to 15,000 seconds.

As Table 6.7 shows, the waiting time performance of GP as of S4 after 25 generations is comparable to OIIC, ONC as of S4, or the queue lengths heuristic adjusted to conditions of S6, again demonstrating the robust applicability to traffic patterns to which GP evolution was *not* exposed; however, this does not hold for the PI program version as of generation 10. A new GP run slightly improves the results, yielding the best average node waiting time ($10.6\,\mathrm{s} \pm 0.5\,\mathrm{s}$), though further experiments would be necessary to empirically prove outperforming queue lengths control ($11.3\,\mathrm{s} \pm 0.6\,\mathrm{s}$). The result is also supported by Figure 6.9: as the different means of node control potentially yield queues building up at different

Table 6.7 – Experiment results: S6

Scenario	Node control		Avg. wait per node [s]	Avg. travel time [s]				Comp. duration [(d) hh:mm]
				N→S	S→N	Side	Sport	
S6	Free flow		7.6	702	402	157	437	< 00:01
	Queue lengths		11.3	718	466	165	661	02:35
	OIIC		11.9	651	472	167	854	< 00:01
	ONC (S4)		11.5	722	472	168	680	s. Tab. 6.5
	GP (S4)	Gen. 5	17.8	810	587	166	702	s. Tab. 6.5
		Gen. 10	818.8	2931	397	1738	262	s. Tab. 6.5
		Gen. 25	11.5	760	461	164	681	s. Tab. 6.5
	GP (new)	Gen. 5	11.7	769	476	167	551	04:10
		Gen. 10	11.0	644	454	169	860	1 d 12:27
		Gen. 25	10.6	635	454	165	851	5 d 11:23

Avg. wait time confidence interval half widths are smaller than 0.6 s (exception: smaller than 40 s in the case of GP S4 Gen. 10 node control). Avg. travel time confidence interval half widths are smaller than 6 s (exception: smaller than 75 s in the case of GP S4 Gen. 10 node control).

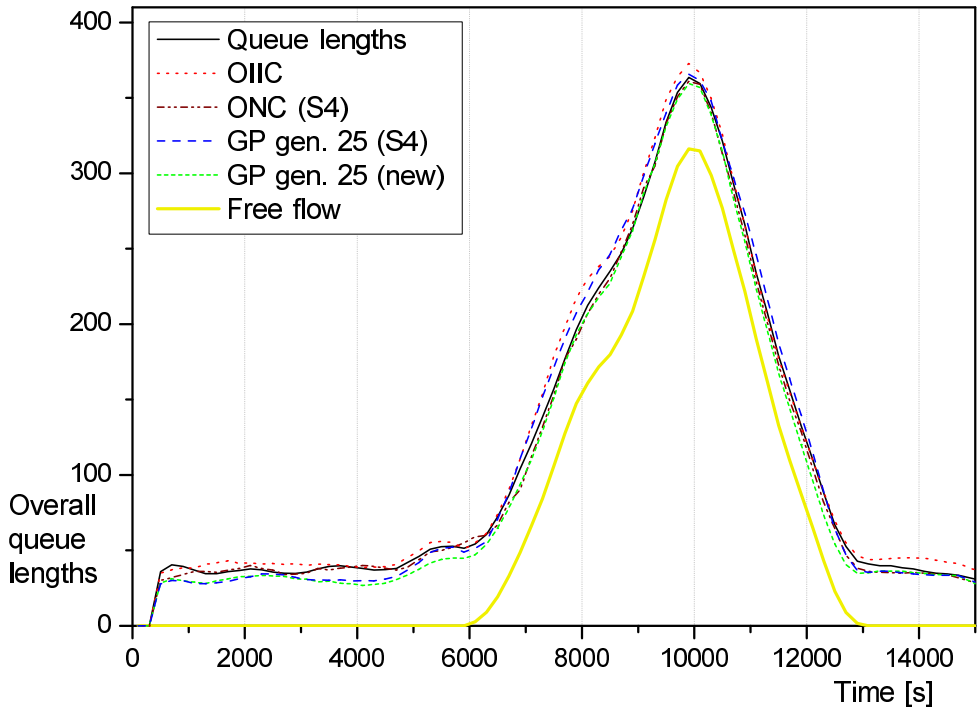

Figure 6.9 – Overall queue lengths in S6 (average of 10 runs) for each means of node control

nodes (e.g. rush hour traffic queued at intersections 2 and 3 before reaching intersection 4 where the sport event traffic meets the arterial, depending on whether intersection 4 control tends to prefer rush hour or side road traffic), a comparison of their ability to resolve the spill-back resulting from temporary over-saturation has to be based on displaying aggregated overall queue lengths throughout the network for each means of node control (average of 10 runs): overall queue lengths of GP node control as of S4, generation 25, most of the time are slightly lower than when applying the other strategies (except free flow control), to which a newly generated GP node control (generation 25) adapted to this scenario provides only a minor further advantage, similar to Scenario 5.

6.2.7 S2/S3/S4/S6 Combined

The previous experiments have shown the robustness of GP node control, which was exposed to different traffic flow densities (S1/S3), partially deviating node control (S5), and perturbed traffic flow patterns (S6). However, in each such experiment, the same *spatial topology* was used, not yet refuting the presumption that GP node control might be prone to overfitting with respect to a specific spatial topology. However, universal applicability requires the ability to evolve a *single* means of entity prioritization performing well in different scenarios in which not only traffic flow densities or patterns, but also the topology itself, is significantly different. Therefore, this last scenario investigates the ability of GP evolution to provide a PI program efficiently controlling S2, S3/high load, S4, *and* S6 at once. Note that such an (approximation to a more) universal urban traffic operation cannot be expected to perform better than the original GP PI programs, as determined in the previous sections, which were specially adapted to each single scenario. However, should the attempt to determine a node control program generally applicable to all four scenarios yield a performance *significantly* worse, it is not plausible that GP node control as determined once is able to "universally" adopt to unforseen environmental conditions (e.g. modifications to the topology or traffic flow patterns), thus not deserving to be referred to as *self-organizing*. However, if the performance is only slightly worse (and thus still better than the results of the non-GP approaches), it is likely – though of course not proven – that traffic management obtained from GP is also efficient in other scenarios not used during evolution, which, for example, combine some of the topological properties and traffic patterns of S2, S3/high load, S4, and S6.

Controlling different topologies

According to the expected increase in PI program complexity, GP evolution was increased to 50 generations; the longer experiment duration, see Table 6.8, is due to four scenarios (rather than one) have to be evaluated to determine a program's fitness. Fortunately, the combined scenario results confirm the expectations to self-organizing network control described above, as the S2, S4, and S6 waiting time performance of the best single ("universal") PI node control program

Table 6.8 – Experiment results: S2/S3/S4/S6 combined

Node control		Avg. wait per node [s] S2	S3-HL	S4	S6	Comp. duration [(d) hh:mm]
Non-GP (old best)		6.1	17.3	3.5	11.5	n/a
GP (old best)		6.2	13.9	2.9	10.6	n/a
GP (combined)	Gen. 5	6.6	17.6	4.0	12.7	11:05
	Gen. 10	6.6	17.5	4.0	12.7	3 d 16:03
	Gen. 20	6.7	17.9	3.2	11.9	8 d 07:11
	Gen. 30	6.5	13.8	3.2	10.9	15 d 20:58
	Gen. 50	6.3	13.0	3.2	10.9	21 d 10:02

Avg. wait time confidence interval half widths for GP (combined) node control are smaller than 0.2 s for S2/S4 and smaller than 0.4 s for S3-HL/S6. For non-GP and old GP control see previous tables.

is only slightly (or even insignificantly) worse than the results of the GP node control specialized to a specific scenario and still significantly better than next-best non-GP means of node control. S3/high load, for which combined control performs even better, is an exception: note that the "specialized" S3 control was a compromise to best serve three different load levels, thus not necessarily optimal when seeking optimization of S3/high load only.

6.3 Conclusion

Efficient decentralized control

Comparing the performance results of GP node control to the (near-)optimal solutions of S1 to S6 yields a GP performance similar or even better with the only exception of the case of the smallest load in S3, despite *not* applying centralized control, e.g. explicit traffic light synchronization in S2: GP node control benefits from the absence of restrictions imposed on node control, such as fixed-cycle programming, which particularly permits exploiting the marginal optimization potential provided by traffic stochastically slightly deviating from expected densities or arrival times, so that e.g. in S2 slightly asymmetrically clearing the link between the intersections may be advantageous in terms of overall waiting durations.

Reproducibility

A further aspect to be expected from a "robust" means of urban intersection control is *reproducibility*: repetitions of GP evolution typically cannot be expected to produce *identical* PI programs; see Section 3.3. However, rerunning a means of determining node rules referred to as self-organizing can be expected to produce node controls *performing similarly*, as otherwise it is likely that applying different scenarios may also yield inferior PI programs in at least some cases. Fortunately,

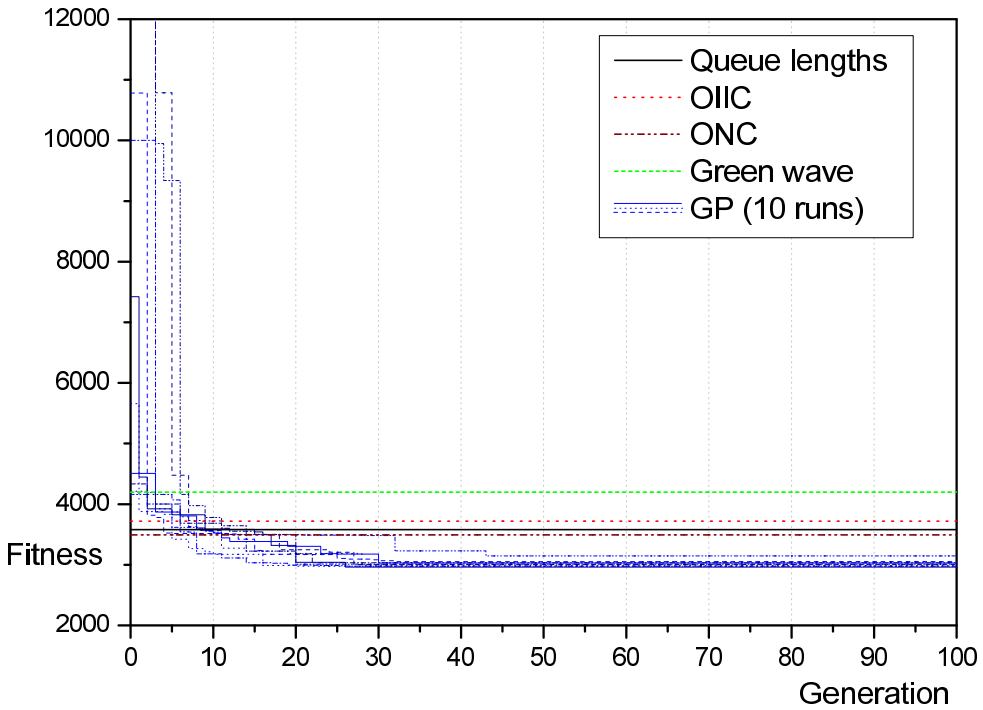

Figure 6.10 – Convergence in S4, i.e. development of the fitness of the best member of each generation in 10 independent GP runs plus the fitness of green wave, OIIC, and queue length node control for comparison

Figure 6.10, which depicts the fitness development of 10 GP runs for Scenario 4, dispels such doubts: although fitness (i.e. the average waiting time in milliseconds subject to penalties as of Section 5.4.2) does not converge towards the same value – range is [2965;3147] in generation 50 – all 10 runs finally outperform all other means of node control including ONC; observe that to the latter, no length or other penalties are applied.

Gradual Traffic Modifications

Evaluation so far explored the ability of GP node control to dynamically adapt to different traffic densities, patterns, and topologies. However, such conditions were *significantly* different, not probing the behaviour of GP node control if traffic conditions were modified gradually. A final experiment therefore investigates how the best node control as evolved for S2/S3/S4/S6 combined (see Section 6.2.7) performs when the traffic is slowly adjusted, covering the full range of feasible traffic densities from zero to saturation. S4 is again used as an application example. Figure 6.11 shows the network's behaviour in terms of average entity waiting times per node, throughput, switching rate, average queue length, and average idle lanes. The figure compares GP node control (top) to SCOOT-like green wave programming (bottom) in which, analogously to Section 6.2.4, cycles, period ex-

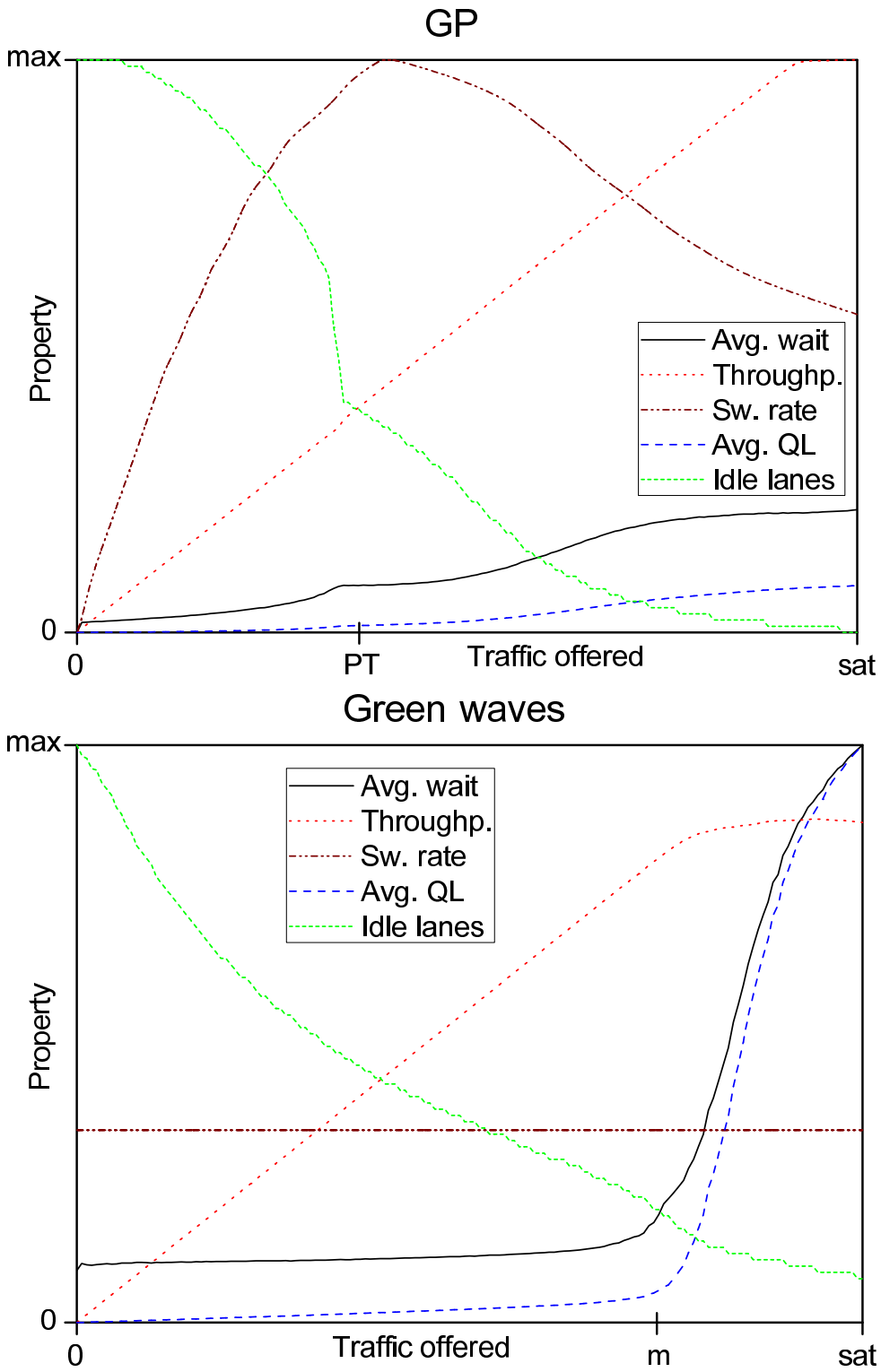

Figure 6.11 – Characteristic properties in S4, namely average entity waiting times per node, throughput, switching rate, average queue length, and average idle lanes as traffic increases from zero to saturation: best GP of S2/S3/S4/S6 combined (top) and green wave programming (bottom), using the same linear scale

tensions, and offsets are set so that waiting time is minimized for an assumed maximum traffic density m equal to approximately three quarters of the saturation flow. The scale is linear and identical for both diagrams; absolute numbers are omitted, as they are different for each characteristic property.

Observe how GP node control adjusts its switching rate (and, conversely , average phase lengths) to traffic density: for low traffic densities, GP node control serves any vehicle as soon as feasible, yielding small average waiting times and a switching rate increasing approximately proportionally to the traffic offered, i.e. every single vehicle receiving a "green" period as soon as possible, followed by switching to the next flow where a vehicle is waiting. However, at a traffic density PT of approximately one-third of the saturation flow, a *phase transition* (see Section 3.1.2) occurs, as serving single vehicles is no longer feasible: the switching rate, which is bounded by a minimum phase length and safety periods, can no longer be increased; thus, further throughput increases require decreasing switching rates (i.e. increasing phase lengths), thus serving more vehicles per phase and reducing processing capacity losses incurred by safety periods. Note how this approach differs from green wave control, the latter applying a constant switching rate regardless of the traffic density, resulting in an almost equal expected waiting time for traffic densities smaller than m. However, if traffic density is larger so that a single phase does no longer suffice to clear the queues, waiting times and queue lengths quickly increase.

Phase transition

Therefore, despite SCOOT-like dynamic phase extensions, green wave control is (near-)optimal only for traffic conditions for which cycle lengths and offsets were optimized, yielding unnecessary waiting times if the traffic density is smaller (the cycles during which vehicles have to wait for "green" are too long) or larger (waiting for multiple cycle periods). GP node control, however, is *adaptive* and *self-organizing*, as it is able to more efficiently cope with traffic density perturbations, which have to be expected permanently; see Table 4.1 on p. 46: apart from a traffic density around m, average waiting times and queue lengths are smaller *and* due to the transition from serving single vehicles (low density, switching frequently) to serving large platoons of vehicles (high density, switching less frequently) perturbations have a smaller influence to the performance of the network; despite the fact that such patterns have not been explicitly included into PI-based node control (see Section 5.1.2), they nevertheless have *emerged* from GP node control evolution.

Practical Considerations

It also should be mentioned that the best GP PI programs determined for each scenario and for S2/S3/S4/S6 combined (as displayed in Appendix D) are relatively simple in comparison to e.g. ONC (see Section 5.3, p. 87), which is typical for *reverse-engineering* self-organizing behaviour based on microscopic rules [see

Zam04, p. 306]: nodes representing If conditionals are limited to two levels (i.e. no sub-conditionals of sub-conditionals exist) and nested loops do not occur. Thus, more comprehensive programs were not able to compensate for the increase in complexity as penalized by the target function with a sufficient decrease in average expected waiting time. Regarding the broad applicability – e.g. patterns of topologies, traffic perturbations, potential malicious node behaviour or combinations thereof not investigated here – this indicates that GP control node programming is applicable to more complex scenarios because GP evolution has *complexity reserves* to determine more specialized PI programs.

Complexity reserves

Furthermore, an implementation of the proposed GP-based, self-organizing node control in urban intersections is straightforward and relatively cheap: sensor technology required to locally obtain the input data shown in Table 4.4 on p. 63 is already present at many traffic lights in inner cities, and the PI programs are sufficiently simple (see above) not to exceed the computational capabilities of typical traffic light controllers [see Ger07, pp. 89f]. Finally, vehicle prioritization based on genetically engineered PI programs need not be applied to all intersections in an urban area at once, permitting incremental introduction; see experiment S5: self-organizing node control is able to efficiently adjust to the behaviour of a node exhibiting a deviating behavioural pattern. This particularly holds if such conditions are included in GP node control evolution, which, however, is no necessity as shown in Scenario 5.

Chapter 7

Conclusion

Science is always wrong. It never solves a problem without creating ten more.

— George B. Shaw
verbally, 1930

This final chapter provides a summary of the most important results of this thesis in Section 7.1 before outlining further research in Section 7.2.

7.1 Summary

Summarizing the key results of this thesis, three areas have to be highlighted:

- *Transport network simulation modelling.* The development of a simulation model based on queuing networks [Nag03, pp. 252ff], in which entity motion and node behaviour can be generically parameterized to represent urban traffic, IP networks, production systems, and potentially other types of networks is already useful on its own, permitting, for example, to adopt optimization approaches from one network type to the others and comparing the networks' behaviours.

- *Genetic programming-based transport network optimization.* This thesis has proposed a GP-based optimization approach to engineer node control relying on locally available information only. Node control is described in terms of priority index programs as schema, which are evolved from their core components obtained from the literature, namely prioritization criteria available at the node-level as shown in Table 4.4 on p. 63, mathmatical/logical functions, and program control operators, subject to suitable means of stabiliza-

tion. Adjustments were necessary to the standard paradigm of GP evolution [Koz92, Chapter 5] to cover the inversion of control in fitness evaluation (simulation calling the genetic program to be evaluated, not vice versa) and state-dependently only executing the program fragments to be currently evaluated. Different measurements were undertaken to improve run-time performance, including avoiding the need for a co-evolution of identical program structures by using a *retrieve* operator to access intermediate results already determined in depth-first evaluation, thus excluding infinite recursions. These adjustments were implemented by extending JGAP [Mef12].

- *Efficient decentralized transport network node control.* Most decentralized optimization approaches available in the literature are limited by *either* permitting complex program control operators (loops, conditionals, potentially recursively nested) without at least heuristically traversing the search space made up of node control programs using similar operators *or* conducting heuristic search space exploration, but addressing a reduced search space containing only relatively simple means of node control, e.g. parametrization of existing node control. In contrast, the proposed GP-based means of evolving decentralized node control rules combines (potentially) complex node behaviour with heuristic search space exploration.

  This means was evaluated for different scenarios in the domain of urban traffic. Advantages in comparison to centralized control include scalability (no central authority required), flexibility (e.g. of the optimization target), adaptivity (resiliently coping with perturbations), and robustness ("graceful degradation" in case of partial failures). The approach is not dependent on the availability of each and every prioritization criterion (see above) – in case only a subset or different criteria can be measured, the GP program evolution will determine programs performing as efficiently as possible under these conditions.

  Properties

  Evaluation of experiment results empirically indicates that a performance similar to centrally (near-optimally) controlled systems can be achieved: the lack of global data to be considered during optimization is adaptively compensated by immediate response, absence of restrictions like fixed cycle times, and the inclusion of detailed local data shown in Table 4.4, which in centralized approaches like SCOOT is not possible, due to the complexity of the global transport network optimization problem – at least not to the same extent, consider e.g. limited adaptivity facilitated by cycle skipping or phase extensions. For some scenarios, this "compensation" in fact was an over-compensation, referred to as *supraoptimality* [Res97] of decentralized control. Furthermore, the computational complexity of PI programs to be executed by the nodes – see Appendix D for examples – is relatively low, permitting deployment, for example, in traffic light controllers without

  Performance

  Supraoptimality

requiring additional hardware, assuming only an infrastructure to measure the input data required exists, e.g. video surveillance with image processing or induction loops as used by SCOOT.

These achievements are subject to constraints and limitations, of which the most important include:

- *Limitation to local input data.* An optimization approach that strictly excludes explicit node-to-node communication may mean unnecessarily *ignoring* global data: in practice, global information or even readily applicable node control instructions, as issued by a central server, may be available sufficiently reliably and quickly, especially in small networks. Applying optimization based on GP evolution, as proposed, would involve deliberately discarding this data. GP programming, however, can straightforwardly be extended to include prioritization criteria measured on the global level, e.g. the number of entities in an area or warnings indicating that adjacent nodes on outgoing links are blocked. Global data
  Furthermore, in case global information exchange does not only provide traffic data, but phases that can directly be adopted by the nodes, local control determined by GP evolution may at least still play the role of a backup strategy, which is applied in the case of failure of the central authority or means of communication; thus, it is no longer necessary to resort to fixed-time control, as in SCOOT [Bre12a], as the next best option. Backup

- *Limitation to local node decisions.* Transport network optimization relied on the decision space of a node to determine entities to be prioritized in processing, which, however, can analogously be extended to also decide which entities to accept or discard (if possible); see next section. Nevertheless, potential optimization decision spaces beyond local node control were not addressed, which particularly means excluding entity routing and network design: IP packet routing may be influenced by the nodes by appropriately adjusting routing tables [see e.g. Res06, pp. 119ff/213ff] and even vehicles in urban traffic may at least to some extent follow the advice that, for example, is displayed by a traffic management system, requiring augmenting local node control with an non-local means of determining routes (i.e. centralized or distributed, explicit communication) unless routes are fixed or determined dynamically by autonomous entities anyway. Routing
  Beyond lane allocation as one of the basic forms of network design, which may be included into an urban traffic node's local decision space, see next section, recommendations as to how the network should be designed or redesigned cannot be obtained from the proposed GP node control; see e.g. Hel07b, pp. 10, as a part of the optimization process recommending the Network design

introduction of underpasses and roundabouts to modify urban intersections so that crucial flows are no longer mutually exclusive.

- *Limited evaluation.* Empirical evaluation was limited to urban traffic. Detailed experiments with networks from other application domains are left for future research, permitting comparisons between different network types, e.g. investigating how the presence or absence of setup times influences resulting PI programs. Generalizing the "combined" experiment of Section 6.2.7, the question to which extent more universal node control programs, which are applicable to different networks types, can be evolved.
  Similarly, evaluation focused on minimizing waiting times as the optimization target, not considering side conditions like reducing longest waiting times, fuel consumption, or emissions of air pollutant and noise: Ger07, pp. 77ff, has shown that waiting time minimization may also significantly reduce emissions of air pollutants like carbon dioxide (CO_2) or mono-nitrogen oxides (NO_x). However, Coe12 reports these targets may also be conflicting, e.g. if a new means of node control, despite reducing waiting times, leads to additional cycles of vehicles accelerating and braking before passing an intersection ("stop & go traffic"). However, GP-based node control evolution is sufficiently flexible to pursue other targets that only need be encoded into the fitness function, which can be interpreted as a "black box" on whose structure evolution does not depend. By applying an appropriate fitness measurement, *multi-objective optimization* may be conducted, e.g. by determining a scalar fitness as a weighted combination of rewards and penalties depending on different aspects of transport network performance; alternatively, GP selection may be based on independent fitness functions applied to different subpopulations. This yields "islands" in which programs best-suited for each fitness function are evolved during "high tide" periods; their behaviour (i.e. subprograms) may be combined in "low tide" intervals, potentially producing overall better programs than evolved by pursuing all targets at once by encoding them into a single fitness function [Pol08, pp. 75].

Other domains

Multi-objective optimization

- *Self-organizing network behaviour.* Achieving overall waiting time-efficiency has been empirically proven to be subject to advantages like robustness and scalability; however, disadvantages include a computationally expensive trial-and-error-based search phase of determining node rules and the potentially *supraoptimal* network behaviour principally lacking the predictability of node control based, for example, on fixed cycles, typically *not* evolving regular, periodically repeating phases to which vehicle drivers in urban traffic can get accustomed to.

- *Disadvantages of GP as a means to determine node rules.* The GP-based evolutionary search approach in theory is flexible enough to yield "every" possible node control (but not to modify the structure of traffic lights or other types of nodes themselves; see Figure 7.1). However, in practice the set of programs that can be generated is constrained for two reasons:
  - The genetic nodes as building blocks on which GP operates to determine PI functions, particularly operations and data input in terms of terminals, need to be sufficient: a hypothetical optimal node program cannot be created if its "ingredients" are missing. While the program logic itself (including loops/conditionals, potentially nested recursively) is likely sufficiently flexible, this is not necessarily true for the set of mathematical operations in use and particularly not for the available node criteria, as shown in Table 4.4.
  - Run-time performance provides an implicit bound as GP execution does not guarantee determining optimal PI programs. Instead, as in any other heuristic approach, a local optimum is returned, which is obtained from evolving a population of fixed size over a certain number of generations.

These constraints provide directions for future research to be discussed in the next section.

Figure 7.1 – Traffic light Tree (Canary Wharf, London, UK; see Iva12): fortunately *not* overgrowth of genetically engineered traffics lights, but an art project of French sculptor Pierre Vivant (1952–*); photo courtesy of *Flickr* user *Metro Centric*; see http://www.flickr.com/people/16782093@N03

7.2 Extensions and Further Research

Beyond the scope of this thesis, various areas touched in this work may be further refined and extended; the following compilation is sorted from the "inside" model and technical level to the "outside" application areas.

7.2.1 Model Validity and Experiment Execution

Microscopic validation

As mentioned in Section 6.1, *operational validation* of the transport network model was restricted to review node behaviour and macroscopic traffic data; a more detailed investigation particularly could focus on the microscopic properties, which the mesoscopic entity motion logic only determines by approximation, e.g. analyzing whether the estimated delay until a vehicle is able to leave an urban intersection by entering a previously blocked link as the queue on this link starts being resolved sufficiently closely matches microscopic traffic models. Significant deviations are unlikely, as otherwise an impact on macroscopic performance measures, like throughput, could have been expected, but their absence has not been strictly proven: multiple microscopic deviations may theoretically cancel out each other so that they are not noticeable in terms of macroscopic performance measures. The requirement of model validity may also imply the mesoscopic movement logic being refined in future applications, e.g. targeting emission minimization in urban traffic potentially requiring a microscopic resolution of entity motion to sufficiently exactly determine how frequently a vehicle had to stop and start up again on a congested link.

Usability

Experiment execution supports batch repetitions of the same scenario subject to different random number generator seeds, i.e. different realizations of all random variables, particularly entity inter-arrival times. The usability of experiment execution could be improved, replacing the current means of parametrization of experiment duration, network topology, and accepting and processing strategies, which is based on command-line switches, with a graphical user interface. Furthermore, batch experiments are executed sequentially; neither batch execution of the network model itself, nor the GP evolution environment supports conducting more than one experiment in parallel. However, the duration of GP evolution could be reduced by distributing experiment tasks among multiple cores of a single processor or among multiple computers in an IP network based as, for example, conducted in Geh04 based on *Java*'s *remote procedure calls* feature.

Finally, PI programs obtained from GP-based node control evolution lack a persistence feature: PI programs as genetically assembled are represented as dynamic objects (i.e. a Node Programming Logic object as the root node of a genetic program tree referencing other nodes), which can be used for multiple simulation experiments at run-time; however, they are lost once the *Java virtual machine* is shut down, at the moment requiring the user to manually re-enter a PI program if additional experiments with the same program are desired.

7.2.2 GP Evolution

Furthermore, GP evolution itself may be refined: The GP operators (i.e. selection, recombination, and mutation; see Section 5.1.3) applied are standard operators already included in JGAP: they were able to generate node control programs performing sufficiently well.

Note that the surface of the fitness function mapping GP node control functions to fitness values is *rugged*, according to the definition of Kau93, pp. 33ff: The likelihood of a one-mutant, i.e. a single mutation of an existing means of node control, being fitter than the original exponentially declines with the fitness of the original: replacing a variable (prioritization criterion) or function in a relatively fit node control with another genetic node typically yields a fitness significantly worse than the original node control, as *epistatic interactions* are highly likely [Kau93, pp. 41ff], i.e. the overall fitness cannot be reduced to the contributions of single genetic nodes. Instead, efficient node control requires *specific combinations* of genetic nodes.

Nevertheless, the GP operators used currently are "blind" from an application-level point of view, by no means e.g. facilitating such useful node combinations being preserved as recombination and mutation operators act completely randomly, i.e. using uniform distributions to sample the subprogram or node to which they are applied. Therefore, future research could investigate whether these operators might be replaced by GP operators specifically designed for transport network node control, thus potentially obtaining better programs or at least improving convergence. For instance, JGAP's default *mutation* operator is limited to replace a function with other functions with the same input and result parameter types (or different input types and randomly chosen subprograms should no such function be available; see Section 6.2); this, for example, excludes a Node Programming Logic operator being replaced by another Node Programming Logic operator with different number of lines (i.e. subprograms), thus extending a program with an additional line (subject to retaining all subprograms without change) or deleting lines is no "natural" (i.e. atomic) mutation. Instead, multiple steps of other operators are required, e.g. repeated selection and recombination such that duplications of all subprograms of a Node Programming Logic operator from one program are transferred to a different program. To facilitate convergence, an operation that is simple and straightforward from an application point of view, like adding or deleting a line from a program, should be directly applicable by GP evolution as well, thus more effectively (i.e. less likely missing a means of generating superior offspring) and efficiently (i.e. faster, due to fewer steps required) exploring the neighbourhood of similar programs.

Specific GP operators

Furthermore, GP operators may be subject to evolution themselves, e.g. if no fitness improvement is detected for a couple of generations, the mutation probability may be temporarily increased such that local optima, in which typically a low

to moderate mutation probability only produces inferior programs, are avoided: programs more different from the currently best program may appear, which may prevent evolution from being stuck in a local optimum, as the gene pool is no longer restricted to the fittest program and its similar offsprings. Beyond such basic means of parameter adjustment, the structure of GP itself may be modified, e.g. also by genetic programming, which may be used for genetically evolving the logic a recombination operator uses to identify subprograms that should be preserved [see Koz92, Chapter 28].

7.2.3 Node Control

Furthermore, the range and capabilities of the local rules applied by nodes to decide in which order discrete entities are processed may be refined and extended as follows:

Input Data

Additional or more precise criteria for entity prioritization may be included into evolution of rules. Observe that a "bound" as to what to consider a *local criterion* has not been strictly established in this thesis : observing a queue length or (an estimation of) the utilization of incident links requires that links are monitored at least partially, e.g. using induction loops or video surveillance with image processing in urban traffic. Provided the availability of suitable means of monitoring, like cameras sufficiently high above the intersections, the "locality" of a node might already commence beyond adjacent nodes or even include global data (see Section 7.1).

Since an analytical solution to the node control problem is not available, there is no judgement whether providing a node with such additional information will improve network performance; thus, empirical evaluation is required.

Separated Rule Sets

Another approach could involve specialized *rule sets* for groups of nodes: as the extreme case of per-node-rule sets is prone to overfitting (see Section 5.1), an identification of, for example, two, three, or four groups of nodes operating on *different* rules may represent a compromise balancing optimization based on permitting specialization on the one hand and achieving robust universal control by avoiding over-adaption on the other hand. For example, an urban traffic network might differentiate between junction types, such as main arterial/main arterial, main arterial/side road, and side road/side road. The behaviour of each group of nodes is determined independently, which might provide more flexibility to adjust the node's behaviour in that it best suits the requirements of such nodes, relieving evolution from the alternative of generating case switches to differentiate between

such groups. Overfitting is still unlikely assuming that each group still contains sufficiently many nodes that – apart from sharing the property of representing, for example, an intersection of a main arterial and side road – node configurations as different as conceivable are included in every group, e.g. nodes with three, four, and five outgoing links; nodes whose queuing space is small and large; nodes whose distance to next nodes is short and long; and nodes adjacent to nodes from the same group and from other groups.

Additional Flows

Furthermore, node modelling may not yet be *sufficiently detailed*: in urban traffic, for instance, pedestrians are not yet regarded in optimization at all; therefore, crossing streets may be subject to significant delays, assuming the proposed means of GP node control is applied: alternating phases between different flows (queues) may result in unacceptably long pedestrian waiting periods between passing different parts of an intersection. However, a natural solution is to explicitly incorporate pedestrians as entities into the model, splitting an intersection node into sub-nodes, as outlined by Figure 7.2.

Pedestrians

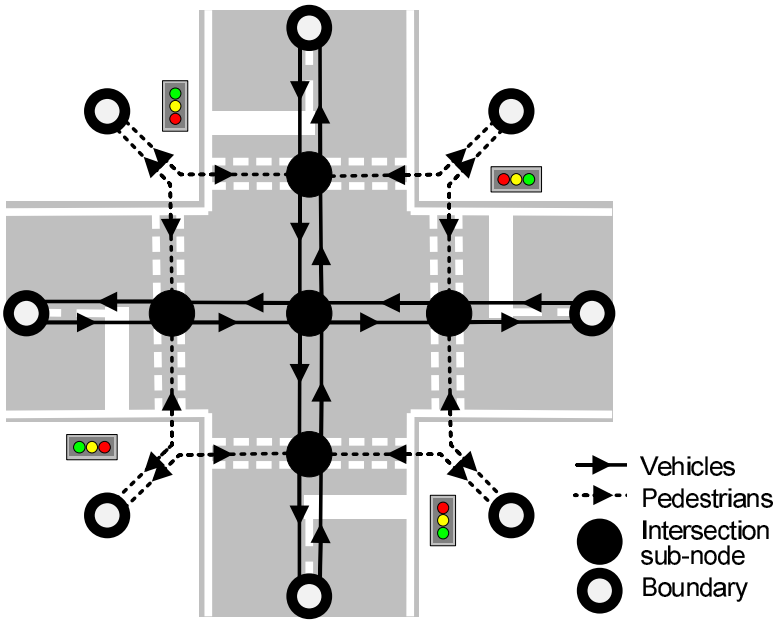

Figure 7.2 – Inclusion of pedestrian flows into traffic light optimization

Apart from including pedestrian waiting times in the fitness function, such an approach should be combined with some of the considerations described above, particularly developing different per-node-rule sets for main intersection nodes and pedestrian nodes, on the one hand, and the current state of the main intersection node should be made available to the adjacent pedestrian nodes, on the other

hand; the latter relieves genetic programming from coordinating the sub-nodes by e.g. implicitly developing a way that permits pedestrian nodes to deduce the current state of the relevant main intersection node.

Dynamically Adjusting Processing Capabilities

Furthermore, node behaviour may include modifying its own (so far) static entity processing structure; for the case of urban traffic, this may mean dynamic lane allocation, as exemplarily depicted by Figure 7.3.

Figure 7.3 – Dynamic lane allocation: examples of how flows may be allocated to lanes [from Göb08a, p. 9] (left) and a traffic sign displaying the current lane allocation (right)

PI-based lane allocation

Similar to phase allocation, queue (lane) allocation can also be conducted dynamically based on priority index functions; in this case, however, the interpretation of PI values is no longer only binary (i.e. a flow either served or not, depending how the flow's PI value compares to the other flows), but requires a choice of *how many* queues to allocate, yielding the additional step of mapping priorities to a number of queues.

For example, assume a distribution of three lanes or six "halves" of lanes among three urban traffic flows is necessary; enabling the assignment of "halves" of lanes serves to potentially generate lanes shared between two flows. Further assume flow priorities are 1.0 (turn left), 2.9 (straight ahead), and 2.1 (turn right). Analogously to a simple means of allocating seats for representative assemblies in party list voting systems [see Tan10, Chapter 4], an elementary solution could be a lane assignment proportional to the priorities of the flows, resulting in 0.5/1.5/1 lanes in the described example, which matches the upper half of Figure 7.3 (left), subject to suitable rounding procedures and guaranteeing at least half a lane (i.e. a lane shared with a different flow) is allocated to each flow. Moreover, as typically overhead costs are incurred from modifying the processing structure of a node,

e.g. closing a section of a link for a short period to avoid the safety risks of reallocating lanes "on the fly", lane allocation should not be conducted too frequently, e.g. no more than once an hour. As lane allocation, therefore, is not well-suited as a fast-response means of handling sudden perturbations, prioritization criteria reflecting the node state at a particular instant (e.g. queue lengths) should be disabled or replaced by average values during a full lane allocation period.

Different Classes of Entities

Integration of *prioritized entities* like, for example, busses and other means of public transport and ambulance vehicles in urban traffic, is straightforward: to such entities, a higher weight could be assigned when measuring prioritization criteria like queue lengths or waiting durations (i.e. counting a bus as ten normal vehicles and an ambulance as 100 vehicles; in the latter case also assuming a waiting time five times longer than actually waited). The presence of such a vehicle could also be made available directly to an intersection, e.g. a Boolean criterion to determine whether or not a prioritized entity is waiting on a certain lane, enabling genetic programming to adjust to such a situation without implicitly recognizing it from other prioritization criteria, like a sudden significant increase in the relevant lane's queue length. To evolve rules facilitating such entities' prioritization, the fitness function in use should significantly penalize delaying prioritized entities.

Anticipative Control

Furthermore, evolutionarily determined node behaviour is adaptive, yet not *anticipative* (see Section 4.1): PI programs allow for adjusting processing capacity allocation to current, yet not to future conditions. Entity prioritization may benefit from a node's ability to predict the development of prioritization criteria during a coming interval, e.g. not further feeding an outgoing link likely to block soon. An example of anticipative control is the nodes' internal model of forecasting vehicle arrivals in Hel08, pp. 6ff, from which ONC was obtained by replacing predictions with assuming future traffic density is equal to the traffic density of the recent past. This thesis did not explicitly conduct anticipative control; however, some prioritization criteria refer to the future, permitting at least a basic means of anticipatively determining future values of others, e.g. a queue length estimation based current queue lengths plus entities arriving soon (FV-EAS), from which potential flow (FV-PF) has to be subtracted if the relevant lane is set to "green".

Non-Local Constraints

Finally, traffic administration may also require that *non-local constraints* are included in node behaviour. A typical example in urban traffic is referred to as

Gating *gating* [see Bre12b]; Figure 7.4 provides an example, assuming the critical node on the right is over-saturated. Its utilization, therefore, is higher than the utilization of the node on the left, compare the widths of the arrows which are assumed to be proportional to the flow rates: to prevent a queue from building up on the segment between the two traffic lights, the node on the left acts as a gate, which deliberately serves fewer vehicles than feasible, thus effectively moving the queue of vehicles traversing the intersections from west to east to the left node. Typical reasons for gating include keeping traffic out of downtown areas or faster access to critical nodes for flows treated preferentially, e.g. public transport reaching the gate from the top.

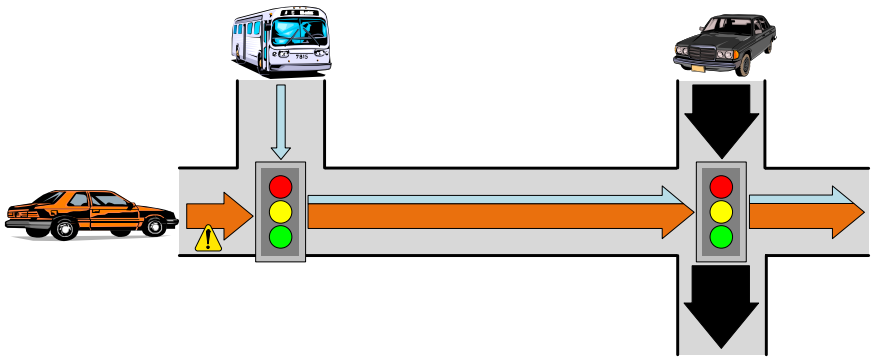

Figure 7.4 – Gating: the node on the left is used as gate

Including gating-like non-local constrains into GP-based traffic light optimization does not differ from centralized systems like SCOOT: based on the ability of non-locally communicating the necessary data between the relevant nodes, either a "soft" approach to *recommending* gating, e.g. discounting priority indices of flows to be held by an appropriate penalty, or a "hard" means of *enforcing* gating is applied. The latter typically involves disabling flows once the non-local constraint applies, e.g. as long as the queue length at the node on the right of Figure 7.4 exceeds a critical bound, the W→E flow is disabled at the node on the left, i.e. the priority index-based choice of which flows to serve is applied to the remaining flows only.

7.2.4 Application Areas

Section 7.1 has already stated that the evaluation conducted in Chapter 6 has focused on the special case of urban traffic, in which travel times were minimized, which leaves an investigation of the performance of GP node control in other transport networks like production systems and IP networks, despite being supported by the simulation model, for further research. The same holds for alternative optimization targets, e.g. emission reduction; see above.

Performance Depending on Dynamically Discarding Entities

In contrast to urban traffic intersections, IP network nodes can discard entities (packets): a "best effort" network need not guarantee all packets are delivered to their destination nodes; quality of service requirements typically only include the packet loss probability not exceeding certain bounds [see Göb06b]. The decision whether or not to discard a packet – while typically trivial in urban traffic (always accept) and production systems (e.g. always accept unless faulty) – may become crucial for the performance of a near-saturated IP network: switching penalties are negligible or non-existent, as the duration of processing a packet does not depend on the route or flow to which the previously processed packet was assigned. However, the performance of the network may benefit from packets being dropped by one of the first nodes on their routes if by this means avoiding spending processing capacity on them in the event that successive nodes drop the packets anyway, e.g. due to their queue (buffer) being full. Some models of determining whether or not to accept packets have been investigated in parallel to this thesis , e.g. proposing the introduction of a virtual currency where the origin or destination node has to be able to pay the other nodes to accept and forward the packet; see Appendix A and references therein.

Analogously to flow prioritization, a GP node control approach to determine whether or not to accept an entity based on genetic program evolution could be conducted; as the main difference to the generation of numerical processing priority indices, program expressions determining whether or not to accept a packet determine Boolean results. Exploring the decision space of employing arbitrary strategies for both accepting and processing entities by genetic programming, therefore, results in a *genotype* consisting of *two chromosomes*, one responsible for accepting entities and one for processing entities, yielding two differently typed programs that are evolved simultaneously [Pol08, pp. 21f].

Two chromosomes

Wireless IP Networks

The applicability of local node-control to the optimization of wireless IP networks, e.g. *mobile ad-hoc networks* (MANET) consisting of mobile wireless nodes and other devices forming an IEEE 802.11s mesh [see e.g. Göb08c], however, is limited, even if accepting or discarding entities is included into the optimization (see above): in such wireless networks, the decision of which packet should be processed next by a sender and receiver is non-local; see also Figure 2.5 on p. 16. Strictly, such networks do not qualify as transport networks according to the ontology described in Section 2.1.4: data transmissions inside a node's transmission range are mutually exclusive; thus, nodes may be remotely blocked. Even if means of time-division or frequency-division multiplexing are applied, thereby providing multiple channels on which communication is possible in parallel [see e.g. Res06, pp. 840ff], only a fixed maximum number of transmissions can be

Mobile ad-hoc networks

conducted simultaneously: there is no guarantee that packets can be transmitted as recommended by priority index-based node control.

Therefore, node transmissions have to be synchronized; this can, for example, be conducted stochastically based on *request to send/clear to send protocols* (RTS/CTS), in which nodes that are remotely blocked repeat transmission attempts at increasing intervals , using an exponential backoff [see Göb09c]. Hence, local node behaviour is biased by the means of node synchronization in use since the node only partially controls its own ability to process packets, which also depends on other nodes' behaviour. Overcoming this mismatch between the scope of the decision space (the node's own behaviour) and the scope on which the node's behaviour depends (all nodes inside its transmission range) requires a distributed optimization protocol explicitly conducting node synchronization throughout a subnetwork [see Loo12, pp. 256], e.g. propagating priority indices throughout a cluster of nodes, and prioritizing transmissions that are most "urgent", e.g. preferring a transmission from a node whose buffer is (almost) full and thus potentially forced to reject packets in the near future; see the distributed and decentralized urban traffic optimization approach of Hel10. Alternatively, a completely different means (scheme) of self-organized node behaviour may be applied, which "naturally" more suitably represents wireless transmissions from a sender to a receiver, e.g. by mimicking how male fireflies communicate in terms of synchronized light flashes [Mir90].

Energy restrictions

Additionally, the *energy supply* of MANET nodes may be limited, potentially imposing a bound on how packets can be transmitted during a unit of time or during the life of a battery, such that the decision to receive and transmit a packet affects the node's ability to process packets in the future [see Loo12, pp.201ff]; packets queued do not only compete for being processed with other packets queued simultaneously, but potentially also with packets that do *not* yet have arrived at a node or with packets that even may not yet have been created; this conflict is, in principle, disregarded by the proposed means of GP-node control.

Summarizing, the appropriateness of local GP-based node control as proposed in this thesis is considered to be questionable for the purpose of MANET optimization. This is due to not representing the limited decision space constrained by remotely blocking nodes and limited battery power. Nevertheless, these constraints, at least, are included in the simulation environment, permitting measuring the performance of alternative means of node control *despite* local node decisions potentially being void due to RTS/CTS transmission failures or energy being depleted.

Node motion

However, further characteristic properties of MANETs or other wireless networks exist that are *neither* included into GP-based optimization of node control *nor* supported by the simulator; an important example is *node motion* [Loo12, pp. 313ff], which may be responsible for dynamically establishing and losing links.

Establishing decentralized (distributed) node control incorporating motion is left for further research, e.g. by emphasizing the priority of packets to be transmitted on a link that likely will be lost soon based on appropriate additional prioritization criteria that measure the length or transmission success probability of the length of a link. Some MANET protocols also support dynamically controlling the *packet size* [Loo12, pp. 492f], i.e. packets may be merged or separated at the nodes, which provides another means of influencing the performance of the network not yet addressed in GP-based node control: larger packets require less overhead and are less prone to the order of the packets being perturbed (jitter), yet their transmission blocks the resources of the nodes inside the transmission range for longer consecutive periods, and their loss means larger blocks of data need be retransmitted.

Packet size

Networks Posing Flow Allocation Problems

As a constraint on the application-level of GP-based node control, Section 7.1 did already mention that the proposed means of transport network does not determine or consider *routing*. This particularly renders the approach inadequate for application areas in which routing is an important (or even the only) degree of freedom to optimize a network; for example, the task of how to route electric power from producers to consumers through a grid whose links exhibit capacity limitations can be appropriately interpreted as an example of a *flow allocation problem*. Solutions to flow allocation problems in networks satisfying *Kirchhoff's circuit laws* particularly exploit the property that flows (e.g. energy) from a certain producer need not be transferred to a *specific* consumer [see e.g. Kat99]: in solutions proposed for power routing, such as Rio09, it suffices to transfer enough energy to every consumer subject to link capacity limitations, flow conservation (i.e. a flow reaching a node must also exit it, unless it is consumed), and flow non-negativity; the origin of the power provided to a consumer does not matter. Such problems can efficiently be solved by linear programming, i.e. they belong to *complexity class P*, which can be deterministically solved in a polynomial amount of computation time [Kes12, pp. 156f]. Observe that the nodes in such a network do not decide about entity (or flow) prioritization, i.e. they need not determine which flows to forward first, thus clearly separating the problems of flow allocation and transport network optimization, the latter being NP-complete (see Section 2.2).

Kirchhoff's circuit laws

Network Superimposition

Finally, this thesis has not considered that different transport networks may be *superimposed* on each other, i.e. components of *different* networks able to either communicate to each other or entities or nodes even simultaneously being members of *multiple* networks. For example, wirelessly communicating vehicles in urban traffic may form a *vehicular ad-hoc network* [Eic05], permitting "smart"

VANET

vehicles to exchange information with each other; for instance, collision warnings may be issued when overtaking or approaching a traffic jam [Loo12, pp. 333f]. Vehicles may also act as relays to, for example, provide Internet access to other vehicles that are out of the transmission range of Internet gateways offered by road-side fixed infrastructure units [Loo12, pp. 353ff]. Simulation modelling of such VANETs requires a software architecture combining road traffic and IP network simulators [see Weg08].

Optimization of a superimposed network may be based on features of the other if they are able to appropriately exchange information. For example, vehicles forming a VANET and urban traffic control networks may each benefit from the presence of the other: the VANET may propagate a vehicle's current velocity or its route pursued to the next intersection, thus supplying traffic light optimization with traffic data otherwise not available or imprecisely estimated e.g. using induction loops. This may, for instance, allow a traffic light to extend a "green" phase such that a platoon which has registered its imminent arrival is able to pass before switching to "red"; see the research outlined in Gra07. Conversely, "smart" vehicles potentially benefit from data aggregated by and received from the traffic lights, which may be used to reduce fuel consumption when approaching a traffic light not switching to "green" during the next period and to include current waiting times into routing decisions [see Tie10].

7.2.5 Beyond Node Control

Finally, Ger07, p. 89, who views traffic lights as *mediators*, emphasizes that even today, traffic lights increasingly are not or no longer necessary where traditional vehicles are already able to efficiently regulate prioritization without mediation; consider e.g. roundabouts as efficient means of flow control in areas exhibiting low or medium traffic densities only. "Smart" vehicles (see above) may further reduce the need for traffic lights as external mediators at the node-level. Therefore, Fer10 claims that traffic lights are a *transitory technology* not required anymore once all entities are sufficiently "intelligent" and equipped with appropriate means of sensing and communicating all necessary data such that the right of way can efficiently be negotiated under arbitrary traffic conditions: no stationary traffic lights are needed where each car carries its own "in-vehicle traffic light" [Fer10, p. 87]; thus, traffic lights may disappear from cityscapes one day.

Transitory technology

After arguing for abandoning network-wide centralized control in favour of control exhibited locally by the nodes, node-level control in principle can be questioned as well, giving rise to *further decentralization*: node control as proposed in this thesis may be *decentralized* from the multi-node network perspective, yet it can be interpreted as still *centralized* from the viewpoint of various flows, lanes, and ultimately vehicles, which have to be synchronized at a node. Centralization, originally serving the purpose of guaranteeing synchronization and simpli-

Further decentralization

fying network optimization by, for example, aggregating many vehicles into a few platoons [Lit66], is no longer necessary, given the flexibility and distributed computational power of "smart" entities. Consequently, the vision of Fer10 suggests that decentralization should be pursued further: the vehicles themselves should autonomously decide which vehicles have the right of way, without jointly considering vehicles, flows, or whole nodes in centralized structures, which become responsible for entity prioritization in a (sub-)network.

Therefore, Hoa02 proposes that car motion imitates ant swarming, particularly how ants continuously align to each other and compete for space which each ant likes to traverse first; this permanent means of conflict resolution (i.e. not restricted to intersections) may be an appropriate base of self-organizing rules applied *below* node-level. Nevertheless, the vehicle (entity) prioritization problem still has to be solved on this lower, further decentralized level. Negotiations among vehicles about which driver proceeds first still have to be based on a set of microscopic rules to which the behaviour of the vehicles has to conform. As on node-level, these rules should be determined such that *self-organized* efficient transport *emerges* from their application. Thus, if this vision comes true, these rules will be applied by single vehicles instead of single nodes, yet rule identification – this time, for example, *directly engineered* based on ant swarming as proposed by Hoa02 and again refined or even re-assembled by genetic programming or other heuristic search techniques – will remain crucial for the performance of transport networks, regardless of the level of decentralization.

Appendices

Appendix A

Various Applications of Self-Organization in Mobile Ad Hoc Networks

During establishing and working on this doctoral thesis, additional studies were conducted in optimizing telecommunication networks as special types of transport networks, particularly addressing mobile ad hoc networks (MANETs) in cooperation with Prof. A. E. Krzesinski. Although not directly related to the genetic programming optimization approach described in Chapter 5, a short introduction to this work is included in this appendix: it provides other approaches (or "mechanisms", as referred to in Section 3.2) of how optimization based on *self-organization* can be achieved in a transport network.

Decentralized Bandwidth Management

For a general connection-oriented telecommunication network, local rules are proposed to dynamically determine bandwidth prices and subsequently to shift bandwidth among the so-called bandwidth managers assigned to each route: without requiring centralized control, this means of self-organizing bandwidth reallocation globally yields a near-optimal quality of service despite random and time-dependent traffic fluctuations.
For further details, see Arv09, Göb06b, and Göb07b.

Resource Allocation in a MANET

If in a MANET a destination node is beyond the transmission range of an origin node, the network's nodes must cooperate to provide a multi-hop route. A simulator to investigate MANET behaviour, especially quality of service and bandwidth/energy (resource) usage, was built based on DESMO-J, in which data transmissions were macroscopically assumed continuous flows. The scheme of

self-organizing resource management as proposed by Cro04 in terms of local rules to determine whether or not a node will spend its resources on relaying other nodes' traffic was applied to this simulation model. As this scheme introduced a virtual currency, which was referred to as credits, paid to compensate relays for their resources temporarily being allocated to transit traffic, the scheme needed refinements such as nodes at the periphery not being excluded from transmitting data despite hardly earning credits since they are rarely used as relays. Further extending the approach of Cro04, explicit signalling of prices, radio interference, and autonomous motion were included in the model. When allowed to adjust locations such that throughput is maximized, node topology self-organized towards a compromise between being close together (thus increasing connectivity by establishing or retaining links) and further apart (reducing interference).
For further details, see Göb07a, Göb08c, Göb09b, and Göb09c.

Energy-Efficient Routing and Traffic Balancing in a MANET

The MANET model (see above) is extended by assuming the node's energy supply is finite. The proposed self-organizing credit-based approach to compensating relays is extended by local rules to balance traffic and to adjust a nodes' transmission range, thus contributing to a network's stability as critical nodes heavily used as relays are less prone to run out of power. Additionally, the effect of nodes belonging to two different authorities, e.g. internet service providers (ISPs) cooperating versus not cooperating (i.e. relaying packets sent between nodes that are associated to the other ISP) is investigated.
For further details, see Göb11a.

Area Coverage in a MANET

This work proposes to apply local rules inspired by spatial sorting in animal groups as described in Cou02 to deploy MANET nodes, which are assumed to act as sensors subject to limited sensing range. Despite the absence of global control, territorial deployment on a plane or around a hill is efficient, even if barriers are introduced which the nodes cannot cross while positioning themselves: the "swarm" of nodes flows around such barriers.
For further details, see Göb08b.

Appendix B

Entity Prioritization Criteria

This appendix defines all transport network entity prioritization criteria as summarized by Table 4.4 on p. 63. These criteria represent the input made available to the genetic programming approach to determine the rules the nodes apply to decide which entities to process next. The criteria are grouped by level (flow, node, and network). The table also includes the return type (F for float, B for Boolean) and unit of measurement as used in the transport network simulation environment.

Flow criteria

Criteria	**FV-AR5/ARO**	Result	**F**	Unit	**[s^{-1}]**

Estimates the number of entity arrivals per unit of time; this ratio is either determined from a recent period (FV-AR5, by default the period is the interval during which the last 50 entities arrived – observe that a fixed period as a basis would have implied larger fluctuations, e.g. an arrival rate estimated zero if there were no arrivals recently) or the overall running average (FV-ARO).

Criterion	**FV-EAS**	Result	**F**	Unit	**[1]**

Estimates the number of entities arriving soon; by default, this yields the number of entities on the link expected to join the queue during the next 20 seconds.

Criteria	**FV-GS/GYS**	Result	**F**	Unit	**[s]**

Returns how long the flow is already set to "green" (FV-GS) or "green" or "yellow" (FV-GYS). If the flow is set to "red" (or "yellow" when evaluating FV-GS), the result is zero.

Criteria	**FC-LLI/LLO**	Result	**F**	Unit	**[m]**

Returns the length of the incoming (FC-LLI) or outgoing (FC-LLO) link, i.e. the distance to the next node. If more than one outgoing link is associated with this flow, the shortest length is returned, assuming this is the most "critical" length potentially constraining entity processing since the queuing space is smaller than on the other outgoing links.

Criteria	**FV-LUI/LUO**	Result	**F**	Unit	**[1]**

Determines the utilization of the incoming (FV-LUI) or outgoing (FV-LUO) link. If more than one outgoing link is associated with this flow, the maximum utilization is returned, assuming this is the most "critical" utilization potentially constraining entity processing as the link may become congested.

Criteria	**FC-LVI/LVO**	Result	**F**	Unit	**[ms^{-1}]**

Returns the maximum velocity on the incoming (FC-LVI) or outgoing (FC-LVO) link. If more than one outgoing link is associated with this flow, the lowest maximum velocity is returned, assuming this is the most "critical" velocity potentially constraining entity processing as entities may not be able to clear the node sufficiently quickly.

Criteria	**FV-NGS/NGYS**	Result	**F**	Unit	**[s]**

Analogously to FV-GS/GYS, these criteria determine how long this flow has *not* received a "green" phase (FV-NGS) or either a "green" or "yellow" phase (FV-NGYS). If the flow is set to "green" (or "yellow" when evaluating FV-NGYS), the result is zero.

Criterion	**FV-PAD**	Result	**F**	Unit	**[s]**

Estimates the duration until the next platoon of entities arrives. By default, a sequence of four or more entities counts as a platoon if the distance between each entity and its successor is less than twice the minimum distance (e.g. "safe distance" in urban traffic). If such a platoon does not exist inside the range the node is able to monitor (e.g. a video camera/image processing in urban traffic), the result is 1.5 times the duration required to traverse this range at the maximum speed, i.e. assuming a platoon approaching the node at the maximum speed is 1.5 times this range away.

Criteria	**FV-PF/PFR**	Result	**F**	Units	**[s^{-1}]/[1]**

Based on current queue lengths and estimated arrivals during the next period (which defaults to 10 seconds), this criterion estimates the potential flow, defined as the average number of entities that can be processed per unit of time during this period, assuming the flow is set or remains set to "green". The result is either an absolute flow ratio (FV-PF) or the flow ratio divided by aggregated flow ratios of all other flows to which this flow is mutually exclusive (FV-PFR), e.g. in right-hand urban traffic, a left-turn is the most expensive flow in terms of other flows excluded from processing entities simultaneously, as visualized by Figure 2.2 on p. 13, where the flows that have to be set to "red" when enabling turning from south to west are highlighted in the subfigure on the right.

Criteria	**FV-QLc/QLp**	Result	**F**	Units	**[1]/[m]**

Returns the flow's queue length, either as entity count (FV-QLc) or aggregated entity physical length (FV-QLp). The latter is zero for entities not having a physical body (e.g. IP packets).

Criteria	**FV-WTA/WTL**	Result	**F**	Unit	**[s]**

Obtains the average (FV-WTA) or longest waiting time (FV-WTL) of all entities currently queued; the latter is the waiting time of the head of the queue. Returns zero if no entities are queued at the moment.

Criteria	**FV-iG/iGY**	Result	**B**	Unit	**[-]**

Obtains the flow's current phase: result is true if the flow is set "green" (FV-iG) or set to "green" or "yellow" (FV-iGY) and false otherwise.

Criterion **FV-iID**	Result **B**	Unit **[-]**

Determines the flow's current processing state, namely either being idle (true) or processing an entity (false).

Criterion **FV-iCO**	Result **B**	Unit **[-]**

Provides the flow's congestion state, i.e. yields true if an entity that has already been processed cannot leave the node as, for example, the outgoing link is blocked and false otherwise.

Node criteria

Criteria **NV-AR5/ARO**	Result **F**	Unit **[s^{-1}]**

Estimated arrivals per unit of time. Same as FV-AR5/ARO (see above), but aggregating all flows processed by the node.

Criterion **NV-PD**	Result **F**	Unit **[s]**

Returns the duration of the current phase up to now, i.e. the period since the last flow has been switched from "green" to "yellow" or "red" or vice versa.

Criteria **NV-MQLc/MQLp**	Result **F**	Units **[1]/[m]**

Determines the maximum queue length of all flows, see FV-QLc/QLp, measurement is either entity count (NV-MQLc) or physical length (NV-MQLp).

Criteria **NV-OQLc/OQLp**	Result **F**	Units **[1]/[m]**

Determines the overall queue length, aggregating the queues of all flows, see FV-QLc/QLp, measurement is again either entity count (NV-OQLc) or physical length (NV-OQLp).

Criteria **NC-SLI/SLO**	Result **F**	Unit **[m]**

Returns the shortest length of either all incoming (FC-LLI) or all outgoing (FC-LLO) links, see FV-LLI/LLO.

Criteria **NV-WTA/WTL**	Result **F**	Unit **[s]**

Obtains the average (NV-WTA) or longest waiting time (NV-WTL) of all entities queued at the node. Returns zero if no entities are queued at the moment.

Criterion **NV-iEM**	Result **B**	Unit **[-]**

Tests whether or not the node is empty: returns true if neither an entity is waiting nor being processed and false otherwise.

Criterion **NV-iID**	Result **B**	Unit **[-]**

Identifies whether or not the node is idle: returns true if no entity is being processed and false otherwise.

Criterion **NV-iBL**	Result **B**	Unit **[-]**

Determines whether or not the node is remotely blocked: returns true if a transmission in a wireless IP network is sent or received by another node within transmission range temporarily preventing this node from sending or receiving (processing) packets and false otherwise. If the type of the network is not a wireless network, the criterion always returns false.

Network criteria

Note that these criteria do not originate from literature research: they cover differences between network types like urban traffic, production systems, and IP networks. In algorithms to optimize a single network type, they are not necessary as they can be treated as *constants*.

Criterion **NeC-MEA**	Result **F**	Unit **[ms^{-2}]**

Estimates the maximum entity acceleration.

Criterion **NeC-ST**	Result **F**	Unit **[s]**

Determines the safety period or special setup time necessary when switching from one flow to another competing for the same resource (e.g. crossing each other in urban traffic).

Criterion **NeC-iPP**	Result **B**	Unit **[-]**

Provides whether or not passing is possible, e.g. faster vehicles being able to overtake slower vehicles (true) or not (false). If set to false, entities always leave links in the same order in which they entered the link.

Criterion **NeC-iRi**	Result **B**	Unit **[-]**

Determines whether or not entity processing requires other nodes within transmission range being idle (true, such a state referred to as *remotely blocked*, see NV-iBL) or not (false).

Criterion **NeC-iSP**	Result **B**	Unit **[-]**

Determines whether or not switching from one flow to another competing for the same resource (e.g. crossing each other in urban traffic) requires a safety period or special setup time, i.e. whether or not NeC-ST > 0 holds.

Appendix C

Transport Network Simulator Classes

This appendix provides a brief description of all the *Java* classes the transport network simulator developed for this thesis consists of. Order is first alphabetically by packages, then by class names. For classes included in the class diagram in Figure 6.3 on p. 103, the alias is given as well, as some class names have been shortened to improve the readability of the diagram.

Package network.control

This package contains provides means of model initialization and (batch) experiment execution.

Class	`Runner`
Alias	(not included in Figure 6.3)
Executable convenience class to start a simulation run; command line switches include the model (XML file), the entity accepting/processing strategies to use, simulation duration, number of batch runs, and whether or not to display animation. For a list of all command line switches, execute this class with parameter `--help`.	

Class	`RunnerDataCollector`
Alias	(not included in Figure 6.3)
A data collector to report the mean and confidence intervals of one of the models random variables determined from multiple batch runs. Confidence interval calculation assumes the random variables are approximately normally distributed with their means and variances unknown.	

Class	`RunnerDataCollectorGroup`
Alias	(not included in Figure 6.3)
Stores and maintains a set of data collectors (`RunnerDataCollector`) to be updated during a batch run.	

Class	StartupTools
Alias	Tools

Serves to initialize a NetworkModel by parsing node, link, and origin-destination flow data from an XML file and creating appropriate model components.

Package network.control.GP

This package contains the components required for genetic programming.

Class	FitnessFunction
Alias	FitnessFunction

Encapsulates the function to estimate the fitness of a genetically engineered node processing strategy, which defaults to the global average entity waiting time in milliseconds (the smaller, the better), subject to different penalties, e.g. a raise proportional to the length of the strategy representation as String.

Class	GPFunction
Alias	(not included in Figure 6.3)

Executable class; a special case of GPProgram in which the node processing strategy consists of a single priority index function.

Class	GPProgram
Alias	GPProgram

Executable class; contains all components necessary to determine node processing strategies by genetic programming: sets of potential nodes, i.e. program control (loops/conditionals), variables, functions, terminals; genetic operators (selection, recombination, mutation); parameters like population size, number of generations, references to the fitness function (FitnessFunction) and the priority index program validator to use (Validator); and a main() method to start genetically engineering node processing strategies. Recombination also includes dynamically invalidating programs by recognizing (sub-)programs that are not meaningful due to conditions that are highly unlikely or numerically or logically impossible to fulfill, e.g. negative queue lengths. Default scenario is S3 as defined in Section 6.2; to run other scenarios, provide an appropriate command line parameter as read by method getScenario(...).

Class	Validator
Alias	Validator

Statically analyzes node processing strategies in order to save computation time by e.g. automatically rejecting strategies containing conditions that are always or never fulfilled (such strategies should be simplified) or not containing at least a single flow criterion (cannot assign *different* priorities to a node's flows).

Package network.control.GP.nodes

This package holds the nodes from which genetic programs are assembled. Only selected classes are shown, particularly classes required for program control; excluded are classes containing numerical operators (e.g. plus, minus, multiply, divide, power, root, exp, log, abs), classes implementing logical operators (e.g. and, or, xor, not, greater, smaller), classes to access values of flow, node, and network variables (i.e. prioritization criteria), and classes representing types (i.e. Boolean or numerical values applicable to network, node, or flow operators).

Classes	`FlowConditionRetr`/`NodeConditionRetr`
Alias	(not included in Figure 6.3)

A node to access a Boolean value determined by a node before this node was evaluated in depth-first order, generically restricted to data referring to both flows and nodes (`FlowConditionRetr`) or nodes only (`NodeConditionRetr`).

Classes	`FlowFloatRetr`/`NodeFloatRetr`
Alias	(not included in Figure 6.3)

A node to access a numerical value determined by a node before this node was evaluated in depth-first order, generically restricted to data referring to both flows and nodes (`FlowConditionRetr`) or nodes only (`NodeFloatRetr`).

Class	`ProgrammingLogic`
Alias	(not included in Figure 6.3)

The top-level node to represent a priority index program, referred to as Node Programming Logic in Chapter 5, holding references to multiple branches `ProgrammingLogicBranch`. Each branch holds either a priority index function or a subprogram (`ProgrammingLogicSubPL`). Entry to each subprogram can be restricted by assigning conditionals ("execute only if...") or loops ("repeat while condition... holds"). Furthermore, a default branch to which no condition is assigned must be defined; this branch is used to determine a priority index if all other subprograms cannot be accessed. The `ProgrammingLogic` remembers how far the program has been executed and resumes where appropriate when called next to obtain a priority index.

Class	`ProgrammingLogicBranch`
Alias	(not included in Figure 6.3)

A "line" or holder of multiple "lines" in a genetic program; see `ProgrammingLogic`.

Class	`ProgrammingLogicSubPL`
Alias	(not included in Figure 6.3)

A subprogram as potentially held by the `ProgrammingLogic`, consisting of branches which contain priority index functions or – to permit recursive programs – further subprograms in terms of `ProgrammingLogicSubPL` objects. Note that no default branch is necessary: if no entry condition is fulfilled and all loops have terminated, the priority index is computed by the next branch as defined by the parent programming logic.

Class	SwitchGeneric
Alias	(not included in Figure 6.3)

A node representing an "exclusive-or" case switch: depending on a condition (first sub-node), a priority index is either determined as defined by the second sub-node or the third.

Package network.logic

This is the location of the core classes to represent a network model's structure and behaviour and to generate statistics output.

Class	EventAcceptingStrategyUpdate
Alias	AcceptStratUpdate

An event to periodically trigger re-evaluation of the node's strategies for accepting entities (e.g. "Evaluate condition... every two seconds; if true, change accepting as follows...") if desired.

Class	EventAnimationUpdate
Alias	AnimationUpdate

An event to update the animation window (queue lengths, node arrivals/departures).

Class	EventDeadlockCheck
Alias	DeadlockCheck

An event to recognize deadlocks in terms of circular relationships of nodes waiting for each other to process entities before they are able to process entities themselves.

Class	EventEntityNodeArrival
Alias	NodeArrival

An event in which an entity arrives at a node. The entity is assigned to one of the node's queues (lanes) and either queued (if necessary) or immediately processed, i.e. an EventEntityProcessingAttempt is scheduled.

Class	EventEntityNodeDeparture
Alias	NodeDeparture

An event in which an entity departs from a node. The entity enters the next link on its route, i.e. an EventEntityNodeArrival is scheduled. Furthermore, an EventNodeResumingService is scheduled for the instant in which processing of the next entity is possible; for urban traffic, this happens immediately, while in wireless IP networks, e.g. insufficient energy supply or the node being remotely blocked may result in a delay.

Class	EventEntityProcessingAttempt
Alias	ProcessingAttempt

An event in which an entity attempts being processed. If all conditions are met (e.g. no preceding entity blocking the node, the flow is set to "green" or "yellow", the latter requiring absence of opposing traffic in urban traffic), the entity will be processed, i.e. an EventEntityProcessingCompleted is scheduled.

Class	EventEntityProcessingCompleted
Alias	ProcessingCompleted

An event in which an entity attempts leaving a node: if at its destination, the entity is removed from the network. Otherwise, the entity departs (EventEntityNodeDeparture) if the next link on its route is not full; contrariwise, the entity preliminarily stays on the node, blocking further processing until it can enter the link desired.

Class	EventNodeArrivalOfRequestRouteFlow
Alias	RequestFlow

An event representing an entity is sent between a fixed pair of nodes.

Class	EventNodeArrivalOfRequestRouteSampled
Alias	(not included in Figure 6.3)

An event representing a node attempting to send an entity to a destination sampled at random.

Class	EventNodeResumingService
Alias	NodeResumes

An event in which one of the queues (lanes) of a node attempts processing an entity (if available).

Class	EventNodeStatusSwitch
Alias	NodeStatusSwitch

An event causing a phase ("green", "yellow", "red") change in at least one of the node's queues (lanes).

Class	EventProcessingStrategyUpdate
Alias	ProcessStratUpdate

An event to trigger re-evaluation of the nodes' strategies for processing entities (e.g. "Evaluate condition ... every two seconds; if true, change processing as follows:..."), if desired.

Class	EventQueueStatDump
Alias	(not included in Figure 6.3)

An event in which detailed queue (lane) data, e.g. its current phase and queue length, is written to an output file.

Class	GlobalParameters
Alias	GlobalParameters

Represents the network's configuration, e.g. entities' data length, physical length, maximum speed, acceleration, safety offset, and loss probabilities and nodes' bandwidth, buffer size, and safety period/setup time. The method useSet() permits loading exemplarily predefined data sets representing either urban traffic or IP networks.

Class	Lane
Alias	Lane

A NetworkNode's facility for queuing and processing entities belonging to one or multiple flows, targeting a specific set of adjacent nodes. A lane's phase is either "green", "yellow", or "red", representing whether entities can be processed unconditionally, conditionally, or not at the moment. Observe that more than one lane may exist at a node for the same outgoing direction, permitting parallel processing.

Class	NetworkEntity
Alias	NetworkEntity

This class represents entities to be moved from one NetworkNode to the next, e.g. vehicles or IP packets.

Class	NetworkLink
Alias	NetworkLink

A directed connection between two NetworkNodes on which NetworkEntity objects can be transferred.

Class	NetworkModel
Alias	NetworkModel

The base class to represent a network model. As subclass of desmoj.core.simulator.Model, this class is responsible for creating all model components and initializing their dynamic behaviour.

Class	NetworkNode
Alias	NetworkNode

A network node, connected to other NetworkNode objects by NetworkLinks. Nodes are able to create, forward, and receive traffic in terms of entities (NetworkEntity). Entities passing through the node may be discarded by its accepting strategy (see package network.logic.strategy.accepting). Entities not discarded are assigned to one of the node's Lanes. The order in which entities are processed is defined by the node's processing strategy (see package network.logic.strategy.processing).

Class	NetworkNodeStatusSwitchDescription
Alias	(not included in Figure 6.3)

This is a temporary entity to describe which lanes (Lane) are set to which phase ("green", "yellow", "red") in an EventNodeStatusSwitch.

Class	PrioritizationCriteriaCalculator
Alias	PrioritizationCriteriaCalculator

A helper class to calculate the values of all prioritization criteria on network/node/flow-level as given by Table 4.4.

Class	Tools
Alias	Tools

Various auxiliary methods, e.g. determining a route between an origin and a destination node.

Package network.logic.strategy.accepting

These classes serve to determine whether an entity reaching a `NetworkNode` is either accepted and queued for processing or discarded.

Class	`AbstractEntityAcceptingStrategy`
Alias	AcceptingStrategy

Abstract class to base strategies for accepting entities on. Note that despite a positive vote of such an "accepting strategy", an entity may still be discarded if a node's data buffer is full in an IP network.

Class	`AcceptAll`
Alias	AcceptAll

Accepting strategy: all entities are accepted. This is the default strategy.

Class	`OriginPays`
Alias	(not included in Figure 6.3)

Accepting strategy: an entity will be accepted if the origin is able to pay one credit to any relay in advance. To guarantee a certain amount of bandwidth to be available to all nodes, credits are implicitly redistributed by discounting over time; see e.g. Göb09b.

Class	`OriginPaysArrears`
Alias	OriginPaysArr

Accepting strategy: same as `OriginPays`, but payment for relay service is conducted after the transmission has been confirmed.

Class	`OriginPaysAverage`
Alias	OriginPaysAv

Accepting strategy: same as `OriginPays`, but applying congestion pricing; credits paid by the origin will reflect the average amount charged by the relays. Payment for relay service is conducted after transmission.

Class	`OriginPaysPacketPurse`
Alias	OriginPaysPur

Accepting strategy: same as `OriginPays`, but applying congestion pricing; the origin pays one credit per relay, yet credits not spent since real prices are smaller than the amount contained in the purse will be returned to the origin.

Class	`OriginPaysTitForTat`
Alias	OriginPaysTFT

Accepting strategy: nodes will mutually act as relays ("service") for each other in a "tit for tat" scheme: packets are forwarded as long as the difference of services provided and services received does not exceed a certain bound.

Class	`ReceiverPays`
Alias	(not included in Figure 6.3)

Accepting strategy: same as `OriginPays`, yet it is the destination's responsibility to pay for a transmission.

Package network.logic.strategy.processing

These classes may be used to decide which lanes (Lane) of a NetworkNode are allowed to process entities at a given instant and which not.

Class	AbstractEntityProcessingStrategy
Alias	ProcessingStrategy

Abstract class to base strategies for processing entities on. Based on various triggers, e.g. entity arrival, entity processed, outgoing link blocked, at a regular interval, the processing strategy may change a Lane's phase to either "green" (processing allowed), "yellow" (conditional processing allowed), or "red" (no processing). Note that despite being e.g. set to "green", a Lane does not necessarily process entities, as other restrictions may apply, e.g. a previous entity still occupying the node's processing facility as the outgoing link is blocked in urban traffic or insufficient energy in a wireless IP network.

Class	AllGreen
Alias	AllGreen

Processing strategy: assumes all lanes are set to "green" anytime; its main use is to provide lower bounds on how quickly entities can be processed for comparison purposes.

Class	GroupsFixedAbstract
Alias	GroupsAbstr

Processing strategy: abstract class for strategies using fixed lane groups to set to "green" alternately.

Class	GroupsFixedGreenWaveNtoSNoTurn
Alias	(not included in Figure 6.3)

Processing strategy, subclass of GroupsFixedAbstract: group 1 (N↔S) and group 2 (E↔W) receive "green" alternately. No turning. The length of a "green" phase is a fixed part plus a potential extension used in case entities are still queued at the end of a phase. An offset can be set to align the node's phase with a previous node, yielding green waves in direction N→S. Note that establishing a green wave N→S does not necessarily yield a green wave in the opposite direction.

Class	GroupsFixedTimeAbstract
Alias	FixedTime

Processing strategy, abstract subclass of GroupsFixedAbstract: to lane groups set up in advance, fixed phase lengths are assigned. Subclasses differ on how lane groups are determined.

Class	GroupsFixedTimeSepByOpposeAbstract
Alias	(not included in Figure 6.3)

Processing strategy, abstract subclass of GroupsFixedTimeAbstract: groups are set such that opposing lanes receive "green" simultaneously. Subclasses differ on whether turning left is allowed or not.

Class	GroupsFixedTimeSepByOpposeNoYellow
Alias	(not included in Figure 6.3)

Processing strategy, subclass of GroupsFixedTimeSepByOpposeAbstract: no left turning.

Class	GroupsFixedTimeSepByOpposeYellowLeftTurn
Alias	(not included in Figure 6.3)

Processing strategy, subclass of GroupsFixedTimeSepByOpposeAbstract: turning left is only permitted when opposing lanes do not process entities (set to "yellow").

Class	GroupsFixedTimeSepByOpposeYellowLeftTurnPlusProtectedLeft
Alias	(not included in Figure 6.3)

Processing strategy, subclass of GroupsFixedTimeSepByOpposeAbstract: same as GroupsFixedTimeSepByOpposeYellowLeftTurn, but including an additional phase for protected left turning from opposing directions.

Class	GroupsFixedTimeSepByOrigin
Alias	(not included in Figure 6.3)

Processing strategy, subclass of GroupsFixedTimeAbstract: groups are set such that only lanes from one direction receive "green" simultaneously.

Class	PriorityGroupsAbstract
Alias	PriorityGr

Processing strategy: abstract class for strategies using a priority index function to determine the lane group to set to "green": the group containing the lane with the highest priority is set to "green". If other groups exist such that every lane can operate independently of all lanes of a group with the highest priority, out of these groups the group containing the lane with the highest priority is set to "green" as well (and so on). Stabilization restrictions may be set. Priority functions are defined in package network.logic.strategy.processing.priorityFunction.

Class	PriorityGroupsUsingAnyGroups
Alias	(not included in Figure 6.3)

Processing strategy, subclass of PriorityGroupsAbstract: no restrictions on which lanes to combine to a priority group.

Class	PriorityGroupsUsingOriginGroups
Alias	(not included in Figure 6.3)

Processing strategy, subclass of PriorityGroupsAbstract: all lanes from one direction form a priority group. This is the default processing strategy, using ProcessingQueueLengthPhysicalFunction as priority index function.

Class	PriorityGroupsUsingSingleLanes
Alias	(not included in Figure 6.3)

Processing strategy, subclass of PriorityGroupsAbstract: every lane forms a priority group of its own, i.e. (in conjunction with PriorityGroupsAbstract) the lane with the highest priority is set to "green"; out of the remaining lanes that operate independently of this lane, the lane exhibiting the highest priority is set to "green" and so forth, until no further lanes exist that can be set to "green".

Class	PrioritySingleLane
Alias	(not included in Figure 6.3)
Processing strategy: a priority index function is used to determine the (single) lane to be set to "green", all other lanes are set to "red".	

Class	RoundRobinClearDirection
Alias	RoundRubin
Processing strategy: alternately, the lanes from each direction receive "green" until all queues are cleared.	

Class	RoundRobinClearSingleLane
Alias	(not included in Figure 6.3)
Processing strategy: alternately, each single lane receives "green" until its queue is cleared.	

Package network.logic.strategy.processing.priorityFunction

The package contains priority index functions and programs to be applied by node processing strategies based on class PriorityGroupsAbstract in package network.logic.strategy.processing.

Class	AbstractPriorityFunction
Alias	AbstractPriorityF
Abstract base class for all priority index functions and programs.	

Class	ConstantFuction
Alias	(not included in Figure 6.3)
A constant priority index function (only for debugging purposes).	

Class	FillClearFunction
Alias	(not included in Figure 6.3)
A priority index function attempting to alternately fill and clear one of its incident links.	

Class	GPFunctionPriorityFunction
Alias	(not included in Figure 6.3)
A priority index function as determined by genetic programming; see GPFunction in package network.control.GP.	

Class	GPProgramPriorityFunction
Alias	GPPriority
A priority index program as determined by genetic programming; see GPProgram in package network.control.GP. A priority index program is a program structure potentially containing multiple priority index functions as well as conditionals and loops to decide which function to apply.	

Class	GPProgramPriorityFunctionWithoutStabilization
Alias	(not included in Figure 6.3)
Same as GPProgramPriorityFunction, yet without stabilization restrictions, thus forcing genetic programming to evolve a means of avoiding oscillating behaviour to produce competitive results.	

Class	OIIC
Alias	(not included in Figure 6.3)
Near-optimal urban traffic priority index function for a single isolated intersection of (by approximation) constant flows; see Figure 5.11 on p. 94.	

Class	ONC
Alias	(not included in Figure 6.3)
Near-optimal network control based on Hel08; see Section 5.3, p. 87.	

Class	ProcessingQueueFIFOFunction
Alias	(not included in Figure 6.3)
Longest waiting time priority index function, effectively yielding FIFO (first in, first out) processing.	

Class	ProcessingQueueLengthCountFunction
Alias	QueueLength
Queue length priority index function (entity count).	

Class	ProcessingQueueLengthCountFunctionWithoutStabilization
Alias	(not included in Figure 6.3)
Queue length priority index function (entity count) without stabilization restrictions.	

Class	ProcessingQueueLengthPhysicalFunction
Alias	(not included in Figure 6.3)
Priority index function: queue length (physical length).	

Class	ProcessingQueueLengthPhysicalFunctionWithoutStabilization
Alias	(not included in Figure 6.3)
Priority index function: queue length (physical length) without stabilization restrictions.	

Package util.animation

This is the location of classes to schematically draw the current state of the network.

Class	AnimationElementBoxedNumber
Alias	(not included in Figure 6.3)
Draws a number in a box; used by AnimationElementLink and AnimationElementNode to show entity counts.	

Class	AnimationElementLine
Alias	(not included in Figure 6.3)
Draws a line with or without an arrowhead in the AnimationWindow.	

Class	AnimationElementLink
Alias	(not included in Figure 6.3)
Represents a NetworkLink in the animation.	

Class	AnimationElementNode
Alias	(not included in Figure 6.3)
Represents a NetworkNode in the animation.	

Class	AnimationElementText
Alias	(not included in Figure 6.3)
Inserts text into the AnimationWindow.	

Class	AnimationWindow
Alias	(not included in Figure 6.3)
The window containing the network animation. Supports arbitrarily scrolling and zooming the current scope as well as saving screenshots. Based on the Piccolo 2D drawing library (http://www.piccolo2d.org).	

Package util.misc

This package contains miscellaneous tools.

Class	CacheKeyValue
Alias	(not included in Figure 6.3)
A cache for key/value pairs of objects. If the cache is full, the least recently read pair of elements is discarded.	

Class	CacheKeyValueValue
Alias	(not included in Figure 6.3)
A cache for mapping keys to two value objects each. If the cache is full, the least recently read triplet (key/value1/value2) of elements is discarded.	

Class	CircularFlowList
Alias	CircularFlowList
A list representing the inflows and outflows at an urban intersection. Based on the angle of each flow, this data structure is able to infer from which inflow to which outflow entities can be processed independently of each other; thus, the class can determine all valid "green", "yellow", and "red" phase configurations for arbitrary intersection layouts.	

Class	CycleDetection
Alias	(not included in Figure 6.3)
An auxiliary class to detect cycles in graphs, used for detecting deadlocks of cycles of nodes waiting for each other.	

Class	OutputTable
Alias	(not included in Figure 6.3)

A window with a table of data to which rows can be appended; used for displaying GP optimization progress.

Class	XMLTools
Alias	Tools

Methods for reading XML files, based on the org.w3c.dom XML parser.

Appendix D

Best GP Node Control Programs

As a supplement to the GP-based transport network optimization experiments conducted in Chapter 6, this appendix lists the best programs found to determine priority indices (PI) to decide which entities (or precisely, entities from which flow/queue) to process next; see Table D.1 on the next page.

The target was to minimize the average waiting time an entity spends at a node, subject to a penalty proportional to program complexity: each character in the program representation as character string counts as equivalent to waiting time increase of a hundredth of a second.

For readability, some programs were equivalently simplified, e.g. sums or products of constants replaced with the result. Short notation includes + (Plus), − (Minus), * (Multiply), / (Divide), ^ (Power), > (Greater), and < (Smaller). Retr(n) retrieves the n-th last floating point number, counting from one, i.e. Retr(1) refers to the last result. Usage of operations if (If), if...else (If_else), while (While), and _?_:_ is same as in *Java*. Constants are rounded off to one decimal place. Where brackets are missing, the usual mathematical evaluation order applies.

Table D.1 – Best node control programs found for each scenario and the combined experiment

Scenario	Program

S1	repeat
	  if (NV-iID)
	  return PI = FV-PFR
	  else
	  return PI = (FV-QLp + FV-EAS)^NV-PD

S2	repeat {
	  return PI = FV-NGS * NV-MQLp * FV-NGYS
	  while (NV-iID)
	  return PI = Retr(1)
	}

S3	repeat {
	  if (NV-MQLp > (NV-OQLc – (NV-MQLp > 38.9 ? NV-OQLc : 38.9)))
	  return PI = FV-QLp
	  if (NV-iID)
	  return PI = FV-NGYS^FV-PF
	  else
	  return PI = FV-WTL
	}

S4	repeat
	  if (NV-iEM)
	  return PI = FV-QLc
	  else
	  return PI = NV-ARO * (FV-QLc^FV-QLc – 4.9) + FV-AR5

S5	repeat
	  if (NV-iEM)
	  return PI = FV-ARO
	  else
	  return PI = exp(FV-QLp) * exp(FV-PF) * FV-ARO * FV-ARO

S6	repeat
	  return PI = FV-AR5 + 103.2 + exp(FV-QLp) + FV-QLc * FV-QLp
	  + exp(FV-QLc)) * Retr(2) + FV-WTL

S2/S3/S4/S6 combined	repeat {
	  if (NV-iID) {
	  if (FV-iCO)
	  return PI = 1.8
	  else
	  return PI = FV-QLp
	  } else
	  return PI = FV-ARO
	  if (Retr(1) > NV-MQLc) {
	  if (NC-SLI < 131.4)
	  return PI = -51.6
	  else
	  return PI = -0.4
	  }
	}

Bibliography

[Arv09] Å. Arvidsson, J. Göbel, A. E. Krzesinski, and P. G. Taylor. "A Distributed Scheme for Value-Based Bandwidth Reconfiguration". In: R. Valadas and P. Salvador (eds.), *Traffic Management and Traffic Engineering for the Future Internet*, pp. 16–35. Springer, Berlin/Heidelberg, Germany, 2009.

[Ash62] W. R. Ashby. "Principles of the self-organizing system". In: H. v. Foerster and G. W. Zopf jr. (eds.), *Transactions of the University of Illinois Symposium*, pp. 255–278. Pergamon, London, UK, 1962.

[Axe84] R. M. Axelrod. *The Evolution of Cooperation.* Basic Books, New York, NY, USA, 1984.

[Bab02] O. Babaoglu, H. Meling, and A. Montresor. "Anthill – A Framework for the Development of Agent-Based Peer-to-Peer Systems". In: *Proceedings of the 22nd Conference on Distributed Computing Systems*, pp. 15–22. 2002.

[Bar12] S. Bardua and G. Kähler. *Die Stadt und das Auto – Wie der Verkehr Hamburg veränderte.* Museum of Work Hamburg/Dölling und Galitz, Hamburg, Germany, 2012.

[Baz05] A. Bazzan, D. de Oliveira, and V. Lesser. "Using Cooperative Mediation to Coordinate Traffic Lights – A Case Study". In: *Proceedings of Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 463–469. July 2005.

[Bed04] B. B. Bederson, J. Grosjean, and J. Meyer. "Toolkit Design for Interactive Structured Graphics". *IEEE Transactions on Software Engineering*, 30(8), pp. 535–546, 2004.

[Bih92] O. Biham, A. A. Middleton, and D. Levine. "Self-organization and a dynamical transition in traffic-flow models". *Physical Review A*, 46, pp. 6124–6127, 1992.

[Bol06] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi. *Queueing Networks and Markov Chains.* Wiley, Hoboken, NJ, USA, 2nd edn., 2006.

[Bon99] E. Bonabeau, M. Doringo, and G. Théraulaz. *Swarm Intelligence – From Natural to Artificial Systems.* Santa Fe Institue Studies on the Sciences of Complexity. Oxford University Press, New York, NY, USA, 1999.

[Boy68] B. C. Boyer (ed.). *A History of Mathematics – Henri Poincaré*. Wiley, Toronto, Canada, 1968.

[Bre12a] D. Bretherton, J. Crowling, and G. Hay. *SCOOT Advisory Leaflet 01 – The SCOOT Urban Traffic Control System*. UK Department for Transport, London, UK, 2012.

[Bre12b] D. Bretherton, J. Crowling, and G. Hay. *SCOOT Advisory Leaflet 02 – Congestion Management in SCOOT*. UK Department for Transport, London, UK, 2012.

[Bro01] E. Brockfeld, R. Barlovic, A. Schadschneider, and M. Schreckenberg. "Optimizing traffic lights in a cellular automaton model for city traffic". *Physical Review E*, 64(5), p. 056132, Oct 2001.

[Bul04] L. Bull, J. Sha'Aban, A. Tomlinson, J. Addison, and B. G. Heydecker. "Towards distributed adaptive control for road traffic junction signals using learning classifier systems". In: L. Bull (ed.), *Applications of learning classifier systems*, pp. 279–299. Springer, New York, NY, USA, May 2004.

[Bun09] H.-J. Bungartz, S. Zimmer, M. Buchholz, and D. Pflügler. *Modellbildung und Simualation – Eine Anwendungsorientierte Einführung*. Springer, Berlin/Heidelberg, Germany, 2009.

[BY97] Y. Bar-Yam. *Dynamics of Complex Systems*. Addison-Wesley, Reading, MA, USA, 1997.

[Cam01] S. Camazine, J.-L. Deneubourg, N. R. Franks, J. Sneyd, G. Theraulaz, and E. Bonabeau. *Self-Organization in Biological Systems*. Princeton University Press, Princeton, NJ, USA, 2001.

[Cao99] Y. J. Cao, N. Ireson, L. Bull, and R. Miles. "Design of a Traffic Junction Controller Using Classifier System and Fuzzy Logic". In: *6th Fuzzy Days – Computational Intelligence Theory and Applications International Conference*, pp. 342–353. Dortmund, Germany, May 1999.

[Cey04] H. Ceylan and M. G. H. Bell. "Traffic signal timing optimisation based on genetic algorithm approach, including drivers' routing". *Transportation Research Part B – Methodological*, 38(4), pp. 329–342, 2004.

[Che87] H. Chen, S. L. Cohen, N. H. Gartner, and C. C. Liu. "Simulation study of OPAC – A demand-responsive strategy for traffic signal control". In: N. H. Gartner and N. H. M. Wilson (eds.), *Proceedings of the Tenth International Symposium on Transportation and Traffic Theory*, pp. 233–249. Adelaide, Australia, 1987.

[Che04] S.-F. Cheng, M. A. Epelman, and R. L. Smith. *CoSign – A Fictitious Play Algorithm for Coordinated Traffic Signal Control*. Technical Report number 04-08, University of Michigan, Ann Arbor, MI, USA, 2004.

[Che09] B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee. "Software Engineering for Self-Adaptive Systems – A Research Roadmap". In: *Self-Adaptive Systems*, pp. 1–26. Springer, Berlin/Heidelberg, Germany, 2009.

[Che12] J. C. Chedjou. "Cellular Neural Networks Based Local Traffic Signals Control at a Junction/intersection". In: *Proceedings of the 1st Conference on Embedded*

Systems, Computational Intelligence and Telematics in Control, pp. 80–85. Würzburg, Germany, 2012.

[Coe12] B. de Coensela, A. Cana, B. Degraeuweb, I. de Vliegerb, and D. Botteldoorena. "Effects of traffic signal coordination on noise and air pollutant emissions". *Environmental Modelling & Software*, 35(2), pp. 74–83, Jul 2012.

[Col99] J. Collier and C. A. Hooker. "Complexly Organised Dynamical Systems". *Open Systems and Information Dynamics*, 6, pp. 241–302, 1999.

[Coo08] S.-B. Cools, C. Gershenson, and B. D'Hooghe. "Self-organizing traffic lights – A realistic simulation". In: M. Prokopenko (ed.), *Advances in Applied Self-organizing Systems*, pp. 41–50. Springer, London, UK, 2008.

[Cou02] I. D. Couzin, J. Krause, R. James, G. D. Ruxtony, and N. R. Franks. "Collective memory and spatial sorting in animal groups". *Journal of Theoretical Biology*, 218(1), pp. 1–11, 2002.

[Cro04] J. Crowcroft, R. Gibbens, F. Kelly, and S. Östring. "Modelling Incentives for Collaboration in Mobile Ad Hoc Networks". *Performance Evaluation*, 57(4), pp. 427–439, 2004.

[Dav83] E. J. Davison and U. Özgüner. "Decentralized control of traffic networks". *Transactions on Automatic Control*, 28, pp. 677–688, 1983.

[Dav03] M. Davison and J. S. Shiner. "Many Entropies, Many Disorders". *Open Systems & Information Dynamics*, 50(3), pp. 281–296, 2003.

[Del95] P. Dell'Olmo and P. B. Mirchandani. "REALBAND – An Approach for Real-Time Coordination of Traffic Flows on Networks". *Transportation Research Record*, 1494, 1995.

[Dia03] C. Diakaki, V. Dinopoulou, K. Aboudolas, M. Papageorgiou, E. Ben-Shabat, E. Seider, and A. Leibov. "Extensions and new applications of the traffic signal control strategy TUC". *Transportation Research Record*, 1856, pp. 202–211, 2003.

[Dut10] U. Dutta and D. S. McAvoy. "Comparative Performance Evaluation of SCATS and Pre-timed Control Systems". In: *Proceedings of the 2010 Annual Transportation Research Forum*. Arlington, VA, USA, 2010.

[Eic05] S. Eichler, B. Ostermaier, C. Schroth, and T. Kosch. "Simulation of Car-to-Car Messaging – Analyzing the Impact on Road Traffic". In: *Proceedings of the 13th Annual Meeting of the IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, p. 4. Atlanta, GA, USA, 2005.

[Elm09] W. Elmenreich, R. D'Souza, C. Bettstetter, and H. de Meer. "A Survey of Models and Design Methods for Self-organizing Networked Systems". In: T. Spyropoulos and K. A. Hummel (eds.), *Self-Organizing Systems – Proceedings of the 4th IFIP TC6 International Workshop*, pp. 37–49. Springer, Zurich, Switzerland, Dec 2009.

[Fer01] C. Ferreira. "Gene Expression Programming – A New Adaptive Algorithm for Solving Problems". *Complex Systems*, 13(2), pp. 87–129, 2001.

[Fer10] M. Ferreira, R. Fernandes, H. Conceição, W. Viriyasitavat, and O. K. Tonguz. "Self-Organized Traffic Control". In: T. Kosch and A. Weimerskirch (eds.), *Proceedings of the Seventh ACM International Workshop on Vehicular Internetworking*, pp. 85–90. Sep 2010.

[Foy92] M. D. Foy, R. F. Benekohal, and D. E. Goldberg. "Signal Timing Determination Using Genetic Algorithms". *Journal of the Transportation Research Board*, 1365, pp. 108–115, 1992.

[Gam94] E. Gamma, R. Helm, R. E. Johnson, and J. Vlissides (eds.). *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley Longman, Amsterdam, The Netherlands, 1994.

[Gar75] N. H. Gartner, J. D. C. Little, and H. Gabbay. "Optimization of traffic signal settings by mixed-integer linear programming". *Transportation Science*, 9(4), pp. 321–363, 1975.

[Geh04] B. Gehlsen. *Automatisierte Experimentplanung im Rahmen von Simulationsstudien – Konzeption und Realisierung eines simulationsbasierten Optimierungssystems*. Doctoral Thesis, University of Hamburg, Germany, 2004.

[Gel98] E. Gelenbe and G. Pujolle. *Introduction to Queueing Networks*. Wiley, Chichester, UK, 2nd edn., 1998.

[Ger02] C. Gershenson. "Complex philosophy". In: *Proceedings of the 1st Biennial Seminar on Philosophical, Methodological & Epistemological Implications of Complexity Theory*. Havana, Cuba, 2002.

[Ger03] C. Gershenson and F. Heylighen. "When Can we Call a System Self-organizing?" In: W. Banzhaf, T. Christaller, P. Dittrich, J. T. Kim, and J. Ziegler (eds.), *Proceedings of the 7th European Conference Advances in Artificial Life (ECAL)*, pp. 606–614. Dortmund, Germany, Sep 2003.

[Ger05] C. Gershenson. "Self-Organizing Traffic Lights". *Complex Systems*, 16(1), pp. 29–53, 2005.

[Ger07] C. Gershenson. *Design and Control of Self-organizing Systems*. PhD Thesis, Vrije Universiteit Brussel, Belgium, 2007.

[Ger12] C. Gershenson and D. A. Rosenblueth. "Adaptive self-organization vs static optimization – a qualitative comparison in traffic light coordination". *Kybernetes*, 41(3), pp. 386–403, 2012.

[Gev12] Gevas Software. *BALANCE – Intelligent Signal Control for Sustainable Road Traffic*. Offical Product Information, Gevas Software Systementwicklung und Verkehrsinformatik GmbH, Munich, Germany, http://www.gevas.eu/produkte/individualverkehr/verkehrssteuerung, 2012. Retrieved Oct 2012.

[Gor11] P. Gora. "A Genetic Algorithm Approach to Optimization of Vehicular Traffic in Cities by Means of Configuring Traffic Lights". In: D. Ryżko, H. Rybiński, P. Gawrysiak, and M. Kryszkiewicz (eds.), *Emerging Intelligent Technologies in Industry*, pp. 1–10. Springer, Berlin/Heidelberg, Germany, 2011.

[Gra59] P. Grassé. "La reconstruction du nid et les interactions inter-individuelles chez les belliconsitermes natalenis et cubitermes sp. La théorie de la stigmergie –

essai d'interprétation des termites constructeurs". *Insects Sociaux*, 6, pp. 41–83, 1959. As translated and quoted in MS06.

[Gra07] V. Gradinescu, C. Gorgorin, R. Diaconescu, V. Cristea, and L. Iftode. "Adaptive Traffic Lights using Car-to-Car Communication". In: *Proceedings of the 2007 IEEE Vehicular Technology Conference*, pp. 21–25. Dublin, Ireland, April 2007.

[Göb06a] J. Göbel. *Development of a Model for Telecommunication Network Simulation with Bandwidth Trading and Link Failure Recovery.* Diploma Thesis, University of Hamburg, Hamburg, Germany, 2006.

[Göb06b] J. Göbel, A. E. Krzesinski, and D. Stapelberg. "Responsive Network Engineering". In: *Proceedings of Stochastic Performance Models for Resource Allocation in Communication Systems*, pp. 31–34. Amsterdam, The Netherlands, Nov 2006.

[Göb07a] J. Göbel, A. E. Krzesinski, and M. Mandjes. "Analysis of an Ad Hoc Network with Autonomously Moving Nodes". In: *Proceedings Australasian Telecommunication Networks and Applications Conference (ATNAC) 2007*, pp. 41–46. Christchurch, New Zealand, Dec 2007.

[Göb07b] J. Göbel, A. E. Krzesinski, and D. Stapelberg. "A Distributed Scheme for Robust Network Engineering". In: *IEEE International Conference on Communications 2007 (ICC-2007)*, pp. 2070–2075. Glasgow, Scotland, Jun 2007.

[Göb08a] J. Göbel. "Self-Organising Transport Networks". In: T. Schulze (ed.), *Simulation and Visualization 2008 – Proceedings of the Postgradute Forum*, pp. 1–10. Magdeburg, Germany, Feb 2008.

[Göb08b] J. Göbel and A. E. Krzesinski. "A Model of Autonomous Motion in Ad Hoc Networks to Maximise Area Coverage". In: *Australasian Telecommunication Networks and Applications Conference 2008 (ATNAC 2008)*, pp. 258–263. Adelaide, Australia, Dec 2008.

[Göb08c] J. Göbel and A. E. Krzesinski. "Modelling Incentives and Protocols for Collaboration in Mobile Ad Hoc Networks". In: *ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems 2008 (MSWiM'08)*, pp. 78–85. Vancouver, Canada, Oct 2008.

[Göb09a] J. Göbel. "On Self-Organizing Transport Networks – an Outline ". In: F. B. I. Troch (ed.), *Proceedings of the 6th Vienna International Conference on Mathematical Modelling (MATHMOD)*, p. 82. Vienna, Austria, Feb 2009.

[Göb09b] J. Göbel, A. E. Krzesinski, and M. Mandjes. "Incentive-based control of ad hoc networks – a performance study". *Computer Networks*, 53(13), pp. 2427–2443, Sep 2009.

[Göb09c] J. Göbel, A. E. Krzesinski, and B. Page. "The Discrete Event Simulation Framework DESMO-J and its Application to the Java-based Simulation of Mobile Ad Hoc Networks". In: R. M. Aguilar, A. G. Bruzzone, and M. A. Piera (eds.), *Proceedings of the 21st European Modeling and Simulation Symposium (International Mediterranean and Latin American Modeling Multiconference), Vol. I*, pp. 95–104. Puerto de la Cruz, Spain, Sep 2009.

[Göb11a] J. Göbel and A. E. Krzesinski. "Improving Cooperation in Mobile Ad Hoc Networks". In: *Proceedings SATNAC 2011 Southern African Telecommunication Networks and Applications Conference*, pp. 331–336. East London, South Africa, Sep 2011.

[Göb11b] J. Göbel, A. E. Krzesinski, and B. Page. "Transport Network Optimization – Self-Organization by Genetic Programming". In: R. M. Aguilar, A. G. Bruzzone, and M. A. Piera (eds.), *Proceedings of the 23rd European Modeling and Simulation Symposium (International Mediterranean and Latin American Modeling Multiconference)*, pp. 137–143. Rome, Italy, Sep 2011.

[Has06] S. Hassas, G. D. Marzo-Serugendo, A. Karageorgos, and C. Castelfranchi. "On Self-Organising Mechanisms from Social, Business and Economic Domains". *Informatica*, 30(1), pp. 63–71, 2006.

[Haw76] S. W. Hawking. "Black holes and thermodynamics". *Physical Review D*, 13(2), pp. 191–197, Jan 1976.

[Hel97] D. Helbing. *Verkehrsdynamik – Neue physikalische Modellierungskonzepte*. Springer, Berlin/Heidelberg, Germany, 1997.

[Hel99] D. Helbing and T. Vicsek. "Optimal Self-Organization". *New Journal of Physics*, 1(13), pp. 1–17, 1999.

[Hel07a] D. Helbing, S. Lämmer, and R. Donner. "Anticipative control of switched queueing systems". *The European Physical Journal B*, 63(3), pp. 341–347, 2007.

[Hel07b] D. Helbing, J. Siegmeier, and S. Lämmer. "Self-organized network flows". *Networks and Heterogeneous Media*, 2(2), pp. 193–210, 2007.

[Hel08] D. Helbing and T. Vicsek. "Self-Control of Traffic Lights and Vehicle Flows in Urban Road Networks". *Journal of Statistical Mechanics – Theory and Experiment*, P04019, 2008.

[Hel09] D. Helbing and A. Mazloumian. "Operation regimes and slower-is-faster effect in the control of traffic intersections". *The European Physical Journal B*, 70(2), pp. 257–274, 2009.

[Hel10] D. Helbing and S. Lämmer. *Self-Stabilizing Decentralized Signal Control of Realistic, Saturated Network Traffic*. Santa Fe Institute Working Papaer 10-09-019, Santa Fe Institute, Santa Fe, NM, USA, http://www.santafe.edu/research/working-papers, 2010.

[Hem05] C. Hemelrijk (ed.). *Self-Organization and Evolution of Social Systems*. Cambridge University Press, New York, NY, USA, 2005.

[Hen83] J. J. Henry, J. L. Farges, and J. L. Tufal. "The PRODYN Real Time Traffic Algorithm". In: *Processdings of the 4th IFAC/IFIP/IFORS International Conference on Control in Transportation Systems*, pp. 307–311. Baden-Baden, Germany, 1983.

[Hey90] F. Heylighen. "Heyligen Classical and Nonclassical Representations in Physics – Physics 1". *Cybernetics and Systems – An International Journal*, 21(4), pp. 423–444, 1990.

[Hey01] F. Heylighen. "The Science of Self-organization and Adaptivity". In: L. D. Kiel (ed.), *Knowledge Management, Organizational Intelligence and Learning, and Complexity*. Eolss, Oxford, UK, 2001.

[Hoa02] R. Hoar, J. Penner, and C. Jacob. "Evolutionary Swarm Traffic – If Ant Roads had Traffic Lights". In: *Proceedings of the 2002 IEEE Conference on Evolutionary Computation*, pp. 1910–1915. May 2002.

[Hol95] J. H. Holland. *Hidden Order – How Adaption builds complexity*. Basic Books, New York, NY, USA, 1995.

[Hol98] J. H. Holland. *Adaption in Natural and Artificial Systems*. MIT Press, Cambrigde, MA, USA, 5th edn., 1998.

[Hol10] R. Holzer, P. Wüchner, and H. de Meer. "Modeling of Self-Organizing Systems – An Overview". *Electronic Communications of the European Association of Software Science and Technology (EASST)*, 27, pp. 1–12, 2010.

[Iba94] H. Iba, H. de Garis, and T. Sato. "Genetic programming using a minimum description length principle". In: K. E. Kinnear (ed.), *Advances in Genetic Programming*, pp. 265–284. MIT Press, Cambrigde, MA, USA, 1994.

[Imp82] G. Improta and A. Sforza. "Optimal Offsets for Traffic Signal Systems in Urban Networks". *Transportation Research Part B – Methodological*, 16(2), pp. 143–161, 1982.

[Iva12] A. Ivanov. *Mind Z Gap – London's Lesser Known*. School of Media Arts and Design, University of Westminster, London, UK, http://mindzgap.co.uk/issue3/stories/alexs_places.html, 2012. Retrieved Oct 2012.

[Jai84] R. Jain, D. Chiu, and W. Hawe. *A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Computer Systems*. DEC Research Report TR-301, Digital Equipment Corporations, Palo Alto, CA, USA, http://www.cse.wustl.edu/~jain/papers.html, 1984.

[Joh01] S. Johnson. *Emergence – The connected lives of ants, brains, cities, and software*. Scribner, New York, NY, UK, 2001.

[Jos00] C. Joslyn and L. Rocha. "Towards semiotic agent-based models of socio-technical organizations". In: *Proceedings of AI, Simulation and Planning in High Autonomy Systems 2000 (AIS 2000)*, pp. 70–79. Tucson, AZ, USA, 2000.

[Kar05] H. Karuna, P. Valckenaers, B. Saint-Germain, P. Verstraete, C. B. Zamfirescu, and H. v. Brussels. "Emergent Forecasting using a stigmergy approach in manufacturing coordination and control". In: S. A. Brueckner, G. D. Marzo-Serugendo, A. Karageorgos, and R. Nagpal (eds.), *Engineering Self-Organising Systems – Methodologies and Applications*, Lecture Notes in Artificial Intelligence, pp. 210–226. Springer, Berlin/Heidelberg, Germany, 2005.

[Kat99] P. A. Kattuman, J. W. Bialek, and N. Abi-Samra. "Electricity Tracing and Co-Operative Game Theory". In: *Proceedings of the 13th Power System Computation Conference*, pp. 238–243. Trondheim, Norway, Jun 1999.

[Kat06] R. T. v. Katwijk, B. d. Schutter, and J. Hellendoorn. "Traffic adaptive control of a single intersection – A taxonomy of approaches". In: *Proceedings of the 11th IFAC Symposium on Control in Transportation Systems*, pp. 227–232. Delft, The Netherlands, Aug 2006.

[Kau93] S. Kauffman. *The Origins of Order – Self-Organization and Selection in Evolution.* Oxford University Press, New York, NY, USA, 1993.

[Kau95] S. Kauffman. *At Home in the Universe – The search for the Laws of Self-Organization and Complexity.* Oxford University Press, New York, NY, USA, 1995.

[Kes09] A. Kesting, M. Treiber, and D. Helbing. "Agents for Traffic Simulation". In: A. Uhrmacher and D. Weyns (eds.), *Multi-Agent Systems – Simulation and Applications*, pp. 325–356. CRC, Boca Raton, FL, USA, 2009.

[Kes12] S. Keshav. *Mathmatical Foundations of Computer Networking.* Addison-Wesley, Upper Saddle River, NJ, USA, 2012.

[Koz92] J. R. Koza. *On the programming of computers by means of natural selection.* MIT Press, Cambrigde, MA, USA, 1992.

[Kum90] P. R. Kumar and T. I. Seidman. "Dynamic instabilities and stabilization methods in distributed real-time scheduling of manufacturing systems". *IEEE Transactions on Automatic Control*, 35(3), pp. 289–298, 1990.

[Lan94] P. T. Landsberg. "Self-Organization, Entropy and Order". In: *On Self-Organization*, pp. 157–184. Springer, Berlin/Heidelberg, Germany, 1994.

[Lee95] J. H. Lee, K. M. Lee, K. A. Seong, C. B. Kim, and H. Lee-Kwang. "Traffic control of ntersection group based on fuzzy logic". In: *Proceedings of the 6th interational Fuzzy Systems Association World Congress*, pp. 465–468. São Paulo, Brazil, 1995.

[Let07] T. Letia, S. Barbu, and F. Dinga. "Using Preemption For Dependable Urban Vehicle Traffic". In: *Proceedings of the 2007 International Multiconference on Computer Science and Information Technology*, pp. 923–932. Oct 2007.

[Lit61] J. D. C. Little. "A Proof for the Queuing Formula $L = \lambda W$". *Operations Research*, 9(3), pp. 383–387, 1961.

[Lit66] J. D. C. Little. "The Synchronization of Traffic Signals by Mixed-Integer Linear Programming". *Operations Research*, 14, pp. 568–594, 1966.

[Loo12] J. Loo, J. L. Mauri, and J. H. Oritz (eds.). *Mobile Ad Hoc Networks – Current Status and Future Trends.* CRC, Boca Raton, FL, USA, 2012.

[Luk84] J. Y. K. Luk and R. W. Stewart. "A Comparison Study of Three Urban Network Models – Saturn, TRANSYT-7F, and NETSIM". In: *Australasian Transport Research Forum Papers*, pp. 51–66. Adelaide, Australia, 1984.

[Lyn96] N. A. Lynch. *Distributed Algorithms.* Morgan Kaufmann, San Francisco, CA, USA, 1996.

[Läm06] S. Lämmer, H. Kori, K. Peters, and D. Helbing. "Decentralised control of material or traffic flows in networks using phase-synchronisation". *Physica A*, 363(1), pp. 39–47, 2006.

[Läm07] S. Lämmer. *Reglerentwurf zur dezentralen Online-Steuerung von Lichtsignalanlagen in Straßennetzwerken.* Doctoral Thesis, Technical University of Dresden, Germany, 2007.

[Läm09] S. Lämmer, J. Krimmling, and A. Hoppe. "Selbst-Steuerung von Lichtsignalanlagen in Straßennetzwerken – Regelungstechnischer Ansatz und Simulation". *Straßenverkehrstechnik*, 11, pp. 714–721, 2009.

[Läm12] S. Lämmer and M. Treiber. "Self-Healing Networks – Gridlock Prevention with Capacity Regulating Traffic Lights". In: A. Datta, M.-P. Gleizes, and I. Scholtes (eds.), *Proceedings of the Sixth IEEE International Conference on Self-Adaptive and Self-Organizing Systems*. Sep 2012.

[Mam05] M. Mamei, V. Vasirani, and F. Zambonelli. "Self-organizing spatial shapes in mobile particles – The TOTA approach". In: S. A. Brueckner, G. D. Marzo-Serugendo, A. Karageorgos, and R. Nagpal (eds.), *Engineering Self-Organising Systems – Methodologies and Applications*, Lecture Notes in Artificial Intelligence, pp. 138–153. Springer, Berlin/Heidelberg, Germany, 2005.

[Mam06] M. Mamei, R. Menezes, R. Tolksdorf, and F. Zambonelli. "Case Studies for Self-Organization in Computer Science – Nature-Inspired Applications and Systems". *Journal of Systems Architecture*, 52(8–9), pp. 443–460, 2006.

[Mar03] P. T. Martin, J. Perrin, B. R. Chilukuri, C. Jhaveri, and Y. Feng. *Adaptive Signal Control II*. Mountain-Plains Research Consortium Report No. 03-141, Salt Lake City, UT, USA, 2003.

[Mat94] M. J. Mataric. *Interaction and Intelligent Behavior*. PhD Thesis, Massachusetts Institute of Technology, Boston, MA, USA, 1994.

[May90] A. D. May. *Traffic Flow Fundamentals*. Prentice-Hall, Englewood Cliffs, NJ, USA, 1990.

[Mef12] K. Meffert et al. *JGAP – Java Genetic Algorithms and Genetic Programming Package*. SourceForge Project, http://jgap.sf.net, 2012. Retrieved Oct 2012.

[Mei09] H. Meinhardt, P. Prusinkiewicz, and D. R. Fowler. *The Algorithmic Beauty of Sea Shells*. Springer, Berlin/Heidelberg, Germany, 4th edn., 2009.

[Mic96] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, Berlin/Heidelberg, Germany, 3rd edn., 1996.

[Mik95] S. Mikami and Y. Kakazu. "Genetic reinforcement learning for cooperative traffic signal contr". In: *Proceedings of the First IEEE World Congress on Computational Intelligence*, pp. 223–228. Sapporo, Japan, 1995.

[Mim00] L. E. Y. Mimbela and L. A. Klein. *Summary Of Vehicle Detection And Surveillance Technologies Used In Intelligent Transportation Systems*. Joint Program Office for Intelligent Transportation Systems Project Report, Southwest Technology Development Institute at New Mexico State University, Las Cruces, NW, USA and U.S. Department of Transportation, Washington, DC, USA, 2000.

[Mir90] R. E. Mirollo and S. H. Strogatz. "Synchronization of pulse-coupled biological oscillators". *SIAM Journal on Applied Mathematics*, 50(6), pp. 1645–1662, 1990.

[MS06] G. D. Marzo-Serugendo, M.-P. Gleizes, and A. Karageorgos. "Self-Organisation and Emergence in MAS – An Overview". *Informatica*, 30(1), pp. 45–54, 2006.

[Möl07] D. Möller, J. Göbel, J. Wittmann, and B. Schroer. "A Mesoscopic Level Traffic Modeling Approach – Concept and Level of Detail". In: *Huntsville Simulation Conference 2007 (HSC'07)*, pp. 186–193. Huntsville, AL, USA, Oct 2007.

[Nag92] K. Nagel and M. Schreckenberg. "A cellular automaton model for freeway traffic". *Journal de Physique I*, 2, pp. 2221–2229, Dec 1992.

[Nag03] K. Nagel. "Traffic networks". In: S. Bornholdt and H. G. Schuster (eds.), *Handbook of Graphs and Networks – From the Genome to the Internet*. Wiley, New York, NY, USA, July 2003.

[Nak95] T. Nakatsuji, S. Seki, and T. Kaku. "Development of a self-organizing traffic control system using neural network models". In: *Proceedings of the 4th International Conference on Microcomputers in Transportation*, pp. 332–343. New York, NY, USA, 1995.

[Nor11] R. B. Northrop. *Introduction to Complexity and Complex Systems*. CRC, Boca Raton, FL, USA, 2011.

[Pag05] B. Page and W. Kreutzer. *The Java Simulation Handbook – Simulating Discrete Event Systems with UML and Java*. Shaker, Aachen, Germany, 2005.

[Pan05] C. G. Panayiotou, W. C. Howell, and M. Fu. "Online traffic light control through gradient estimation using stochastic fluid models". In: P. Zítek (ed.), *Proceedings of the 16th IFAC World Congress*. 2005.

[Pap84] M. Papageorgiou. "An integrated control approach for traffic corridors". *Transport Research C*, 3, pp. 19–30, 1984.

[Pap99] C. H. Papadimitriou and J. N. Tsitsiklis. "The complexity of optimal queuing network control". *Mathematics of Operations Research*, 24(2), pp. 293–305, 1999.

[Pap03] M. Papageorgiou, C. Diakaki, V. Dinopoulou, A. Kotsialos, and Y. Wang. "Review of Road Traffic Control Strategies". *Proceedings of the IEEE*, 91(12), pp. 2043–2067, 2003.

[Per12] J. L. F. Pereira and R. J. F. Rossetti. "An integrated architecture for autonomous vehicles simulation". In: *Proceedings of the ACM Symposium on Applied Computing 2012*, pp. 286–292. Riva del Garda, Italy, 2012.

[Poh10] T. Pohlmann. *New Approaches for Online Control of Urban Traffic Signal Systems*. Doctoral Thesis, Technical University of Braunschweig, Germany, 2010.

[Pol08] R. Poli, W. B. Langdon, and N. F. McPhee. *A Field Guide to Genetic Programming*. Lulu, Raleigh, NC, USA, 2008.

[Poo10] D. L. Poole and A. K. Mackworth. *Artificial Intelligence – Foundations of Computational Agents*. Cambridge University Press, New York, NY, USA, 2010.

[Por96] I. Porche, M. Sampath, Y. Chen, R. Sengupta, and S. Lafortune. "A decentralized scheme for realtime optimization of traffic signals". In: M. Prokopenko (ed.), *Proceedings of the 1996 IEEE International Conference on Control Applications*, pp. 582–589. Sep 1996.

[Por97] I. Porche and S. Lafortune. "Dynamic Traffic Control – Decentralized and Coordinated Methods". In: *Proceedings of the 1997 IEEE Conference on Intelligent Transportation Systems*, pp. 930–935. Sep 1997.

[Por98] I. Porche and S. Lafortune. "Coordination of local adaptive traffic signal controllers". In: *Proceedings of the 1998 American Control Conference vol. 3*, pp. 1833–1837. Jun 1998.

[Pri84] I. Prigogine. *Order Out of Chaos – Man's new Dialog with Nature*. Bantam, New York, NY, USA, 1984.

[Ram04] S. D. Ramchurn, D. Huynh, and N. Jennings. "Trust in multi-agent systems". *Knowledge Engineering Review*, 19(1), pp. 1–25, 2004.

[Res97] M. Resnick. *Turtles, Termites, and Traffic Jams – Explorations in Massively Parellel Microworlds*. MIT Press, Cambrigde, MA, USA, 1997.

[Res06] M. G. C. Resende and P. M. Pardalos (eds.). *Handbook of Optimization in Telecommunications*. Springer, New York, NY, USA, 2006.

[Rio09] V. Rious and P. Dessante. "Real gains from flow-based methods for allocating power transmission capacity in Europe". In: *Proceedings of the 6th International Conference on the European Energy Market*, pp. 1–6. Gif-sur-Yvette, France, May 2009.

[Ros11] D. A. Rosenblueth and C. Gershenson. "A model of city traffic based on elementary cellular automata". *Complex Systems*, 19(4), pp. 305–322, 2011.

[Rot84] M. H. Rothkopf and S. A. Smith. "There are No Undiscovered Priority Index Sequencing Rules for Minimizing Total Delay Costs". *Operations Research*, 32, pp. 451–456, 1984.

[Roy04] T. Royani, J. Haddadnia, and M. Alipoor. "Traffic signal control for isolated intersections based on fuzzy neural network and genetic algorithm". In: *Proceedings of the 10th WSEAS international conference on Signal processing, computational geometry and artificial vision*, pp. 87–91. Taipei, Taiwan, 2004.

[Sen97] S. Sen and K. L. Head. "Controlled optimization of phases at an intersection". *Transportation Research Part B – Methodological*, 31(1), pp. 5–17, 1997.

[Sha48] C. E. Shannon. "A Mathematical Theory of Communication". *Bell System Technical Journal*, 27(3), pp. 379–423, Jul 1948.

[Sha03] J. G. Shanthikumar, D. D. Yao, W. Henk, and M. Zijm (eds.). *Stochastic Modeling and Optimization of Manufacturing Systems and Supply Chains*. International Series in Operations Research & Management Science. Springer, Dordrecht, The Netherlands, 2003.

[She11] Z. Shen, K. Wang, and F. Zhu. "Agent-based traffic simulation and traffic signal timing optimization with GPU". In: A. Eskandarian (ed.), *Proceedings of the 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pp. 145–150. Washington, DC, USA, Oct 2011.

[Sie12] Siemens. *SITRAFFIC MOTION MX – Und der Verkehr kann wieder richtig fließen*. Offical Product Information, Siemens Intelligent Traffic Systems, Munich, Germany, http://www.mobility.siemens.com/shared/data/pdf/www/infrastructure_logistics/sitraffic_motion_mx-der_verkehr_flie_dft.pdf, 2012. Retrieved Oct 2012.

[Sle85] D. D. Sleator and R. T. Tarjan. "Amortized efficiency of list update and paging rules". *Communications of the ACM*, 28(2), pp. 202–208, 1985.

[SR05] B. Scholz-Reiter, M. Freitag, H. Rekersbrink, B. L. Wenning, C. Gorldt, and W. Echelmeyer. "Auf dem Weg zur Selbststeuerung in der Logistik – Grundlagenforschung und Praxisprojekte". In: G. Wäscher et al. (eds.), *Abschlussband zur 11. Magdeburger Logistiktagung "Intelligente Logistikprozesse – Konzepte, Lösungen, Erfahrungen"*, pp. 166–180. Logisch, Magdeburg, Germany, 2005.

[Sun06] Z. Sun. "On-road vehicle detection – a review". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(5), pp. 694–711, May 2006.

[Swa12] Swarco Mizar S.p.A. *UTOPIA – Optimizing Traffic Flows Across the Road Network*. Offical Product Information, Swarco Mizar S.p.A., Turin, Italy, http://www.swarco.com/en/layout/set/pdf/Products-Services/Traffic-Management/Urban-Traffic-Management/Urban-Traffic-Systems/UTOPIA, 2012. Retrieved Oct 2012.

[Tan04] A. C. Tan and R. O. Bowden. *The Virtual Transport System (VITS) – Final Report*. Department of Industrial Engineering, Mississippi State University, MS, USA, 2004.

[Tan10] P. Tannenbaum. *Excursions in Modern Mathematics*. Pearson, London, UK, 7th edn., 2010.

[Tie10] T. Tielert, M. Killat, H. Hartenstein, R. Luz, S. Hausberger, and T. Benz. "The Impact of Traffic-Light-to-Vehicle Communication on Fuel Consumption and Emissions". In: *Proceedings of the 2010 IEEE International Conference on Internet of Things.*, pp. 1–8. Tokyo, Japan, Nov 2010.

[Tig07] M. Tigrek. *Selbstorganisation als naturwissenschaftlicher Begriff und als Begriff in der Soziologie*. Doctoral Thesis, University of Münster, Germany, 2007.

[Tra00] Transportation Research Board. "Highway Capacity Manual". *Transportation Research Board*, Special Report 209, 2000.

[Tre10] M. Treiber and A. Kesting. "An Open-Source Microscopic Traffic Simulator". *IEEE Intelligent Transportation Systems Magazine, Special issue on ITS Traffic Simulators*, 2, pp. 6–13, 2010.

[Vas04] M. Vasirani, M. Mamei, and F. Zambonelli. "Experiments in Morphogenesis of Simple Mobile Robots". *Applied Artificial Intelligence*, 18, pp. 9–10, 2004.

[Vyg78] L. S. Vygotsky (ed.). *Mind in Society – The development of higher psychological processes*. Harvard University Press, Cambridge, MA, USA, 1978.

[Weg08] A. Wegener, M. Piórkowski, M. Raya, H. Hellbrück, S. Fischer, and J. Hubaux. "TraCI – an interface for coupling road traffic and network simulators". In: A. Ahmad and A. Bragg (eds.), *Proceedings of the 11th ACM Communications and Networking Simulation Symposium*, pp. 155–163. Ottawa, Canada, 2008.

[Weh78] A. Wehrl. "General properties of entropy". *Reviews of Modern Physics*, 50(2), pp. 221–260, Apr 1978.

[Wei09] E. Weingärtner, H. v. Lehn, and K. Wehrle. "A performance comparison of recent network simulators". In: *Proceedings of the IEEE International Conference on Communications 2009 (ICC 2009)*, pp. 1–5. Dresden, Germany, 2009.

[Wic99] H. H. Wickman and J. N. Korley. "Colloid crystal self-organization and dynamics at the air/water interface". *Nature*, 393, pp. 445–447, 1999.

[Wie04] M. Wiering, J. Vreeken, J. van Veenen, and A. Koopman. "Simulation and Optimization of Traffic in a City". In: *Proceedings of the 2004 IEEE Intelligent Vehicles Symposium*, pp. 453–458. Parma, Italy, 2004.

[Wit07] J. Wittmann, J. Göbel, D. Möller, and B. Schroer. "Refinement of the Virtual Intermodal Transportation System (VITS) and Adoption for Metropolitan Area Traffic Simulation". In: *2007 Summer Computer Simulation Conference (SCSC'07)*, pp. 587–592. San Diego, CA, USA, July 2007.

[Wol03] T. de Wolf and T. Holvoet. "Adaptive behaviour based on evolving thresholds with feedback". In: D. Kudenko, D. Kazakov, and E. Alonso (eds.), *Proceedings of the AISB'03 Symposium on Adaptive Agents and Multiagent Systems*, pp. 91–96. Aberystwyth, UK, 2003.

[Wol05] T. de Wolf and T. Holvoet. "Emergence Versus Self-Organisation – Different Concepts but Promising When Combined". In: S. A. Brueckner, G. D. Marzo-Serugendo, A. Karageorgos, and R. Nagpal (eds.), *Engineering Self-Organising Systems – Methodologies and Applications*, pp. 1–15. Springer, Berlin/Heidelberg, Germany, 2005.

[Woo93] K. Wood. *Urban Traffic Control Systems Review.* UK Transport Research Laboratory Project Report 41, Crowthorne, UK, 1993.

[Wu10] L. Wu, X. Zhang, and Z. Shi. "An Intelligent Fuzzy Control for Crossroads Traffic Light". In: *2010 Second WRI Global Congress on Intelligent Systems*, pp. 28–32. Dec 2010.

[Yag94] S. Yagar and B. Han. "A procedure for real-time signal control that considers transit interference and priority". *Transportation Research Part B – Methodological*, 28(4), pp. 315–331, 1994.

[Yi06] P. Yi, C. Shao, and L. Sheng. "Improved Signal Control for Oversaturated Intersection". In: *Proceedings of the 2006 IEEE Intelligent Transportation Systems Conference (IEEE ITSC 2006)*, pp. 81–84. Toronto, Canada, Sep 2006.

[Yi08] P. Yi, C. Shao, and Y. Wang. "Piecewise Optimum Delay Estimation for Improved Signal Control". *Transportation Research Record*, 2080, 2008.

[Zam04] F. Zambonelli, M.-P. Gleizes, M. Mamei, and R. Tolksdorf. "Spray Computers – Frontiers of Self-Organization for Pervasive Computing". In: *Proceedings of the 13th IEEE International Workshops on Enabling Technologies – Infrastructure for Collaborative Enterprises*, pp. 403–408. 2004.

[Zha95] B.-T. Zhang and H. Mühlenbein. "Balancing accuracy and parsimony in genetic programming". *Evolutionary Computation*, 3(1), pp. 17–38, 1995.

[Zho10] B. Zhou, J. Cao, X. Zeng, and H. Wu. "Adaptive Traffic Light Control in Wireless Sensor Network-Based Intelligent Transportation System ". In: H. Yanikomeroglu and J. Reid (eds.), *Proceedings of the 72nd IEEE Vehicular Technology Conference*, pp. 1–5. Ottawa, Canada, Sep 2010.