Universität Hamburg

Lösung ausgewählter Routenplanungsprobleme mit Hilfe der Self-Organizing Map

Dissertation

zur Erlangung des akademischen Grades
eines Doktors der Wirtschafts- und Sozialwissenschaften
(Dr. rer. pol.)

des Fachbereichs Wirtschaftswissenschaften
der Universität Hamburg

vorgelegt von

Dipl. Ing. oec. Martin Schwardt
23858 Reinfeld

Hamburg 2005

Mitglieder der Promotionskommission

Vorsitzender:	Prof. Dr. H. Stadtler
Erstgutachter:	Prof. Dr. H. Seelbach
Zweitgutachter:	Prof. Dr. K.-W. Hansmann

Das wissenschaftliche Gespräch fand am 25. Mai 2005 statt.

Inhaltsverzeichnis

Seite

Seite

Abbildungsverzeichnis

Seite

Seite

Tabellenverzeichnis

Seite

Symbolverzeichnis

Arabische Schriftzeichen

$a_j(t)$	:	Aktivierungszustand des Neurons *j* zum Zeitpunkt *t*
a_k	:	Auslastung des Fahrzeugs *k*
b_j	:	Schwellenwert des Neurons *j*
c	:	Zentrum
c_{mn}	:	Kostenbewertung der Kante / des Pfeils zwischen *m* und *n*
d_{mn}	:	Kostenbewertung der Kante / des Pfeils zwischen *m* und *n*
d_j	:	Distanz für Neuron j
$d_n(x,y)$	:	Distanz zwischen *W(x,y)* und P_n
d_j^k	:	Potential des zu Neuronenring *k* zugeordneten Neurons *j*
$d_{j^*}^{\min}$	:	Minimale Distanz zwischen Eingabevektor und Gewichtsvektoren, die bei Gewichtsvektor j^* auftritt
dp_m	:	Entfernung zwischen Depot und Knoten *m*
f	:	Parameter der modifizierten Savings-Funktion
f_{ij}	:	Verbindungsfunktion
f_j^{act}	:	Aktivierungsfunktion des Neurons *j*
f_j^{out}	:	Ausgabefunktion des Neurons *j*
f_j^{prop}	:	Propagierungsfunktion des Neurons *j*
g	:	Parameter der modifizierten Savings-Funktion
$h(t)$	:	Parameter der Auswahlwahrscheinlichkeit zum Zeitpunkt *t*, Ansatz von GHAZIRI
$h_{cj}(t)$	:	Stärke der Adaption des Gewichtsvektors des Neurons *j,* wenn Neuron *c* Zentrum der Iteration *t* ist
l_{0x}	:	x-Koordinate des Depot-Standortes
l_{0y}	:	y-Koordinate des Depot-Standortes
$net_j(t)$	:	Eingabesignal für das Neuron *j* zum Zeitpunkt *t*
$o_i(t)$	:	Ausgabe des Neurons *i* zum Zeitpunkt *t*
p_u	:	Wahrscheinlichkeit, mit welcher der Knoten bzw. Eingabevektor *u* präsentiert wird
q_k	:	Fahrzeug *k* zugeordnete Nachfrage
r_{jh}	:	Laterale Verbindung zwischen Neuron *j* und Neuron *h*
t	:	Zeitpunkt / Iterationszähler

$t_{\max}$	:	Vorgegebene (maximale) Anzahl an Iterationen
t_p	:	Zähler der Präsentationszyklen
$t_{p\max}$	:	Vorgegebene (maximale) Anzahl an Präsentationszyklen
$u_j(t)$	:	Häufigkeit, mit der Neuron *j* bis zum Zeitpunkt *t* als Zentrum ausgewählt wurde
w_n	:	Gewichtung der euklidischen Distanz für Knoten *n*
$\mathbf{w_j}(t)$	:	Gewichtsvektor des Neurons *j* zum Zeitpunkt *t*
$\mathbf{w}_j^k(t)$	:	Gewichtsvektor des zu Neuronenring *k* zugeordneten Neurons *j* zum Zeitpunkt *t*
$w_{ij}(t)$	:	Gewicht der Verbindung zwischen Neuron *i* und Neuron *j* zum Zeitpunkt *t*
win_j	:	Anzahl der von Neuron *j* gewonnenen Wettbewerbe
x	:	x-Koordinate
x^t	:	x-Koordinate zum Zeitpunkt t
x_i	:	*i*-te Komponente des Eingabevektors
$\mathbf{x}_u$	:	Eingabevektor u
x_{mn}	:	$\begin{cases} 1, & \textit{direkte Verbindung von m nach n ist Teil der Rundreise} \\ 0, & \textit{sonst} \end{cases}$
x_{mnk}	:	$\begin{cases} 1, & \textit{wenn Fahrzeug k Knoten n direkt im Anschluß an Knoten m besucht} \\ 0, & \textit{sonst} \end{cases}$
x_{0nk}	:	$\begin{cases} 1, & \textit{wenn Knoten n der erste von Fahrzeug k besuchte Knoten ist} \\ 0, & \textit{sonst} \end{cases}$
x_{m0k}	:	$\begin{cases} 1, & \textit{wenn Knoten m der letzte von Fahrzeug k besuchte Knoten ist} \\ 0, & \textit{sonst} \end{cases}$
y	:	y-Koordinate
$\mathbf{y}_j$	:	Koordinaten des Knotens *j*
y^t	:	y-Koordinate zum Zeitpunkt t
y_{mk}	:	$\begin{cases} 1, & \textit{wenn Knoten m Fahrzeug k zugeordnet wird} \\ 0, & \textit{sonst} \end{cases}$
B_n	:	Nachfrage des Knotens *n*
$\boldsymbol{C}(c_{mn})$	:	Kostenmatrix
C	:	Konstante
$Dist_{AVG}$	:	Durchschnittliche Länge der Fahrzeugtouren

$Dist_k$	:	Länge der Tour des Fahrzeugs k
$Dist_r$	:	Länge der Tour des Handlungsreisenden r
$\boldsymbol{D}(d_{mn})$	:	Kostenmatrix für das MTSP
E	:	Kanten- beziehungsweise Pfeilmenge
$F(.)$	:	Zur Multiplikation der euklidischen Distanz verwendeter Faktor
F_α	:	Am Neuron angreifende Kraft
F_β	:	Am Neuron angreifende Kraft
$F_c(\mathbf{W})$	:	Vom Zustand des Verbindungsnetzwerkes **W** abhängige Teilmenge von V
F_{mj}	:	Gravitationskraft zwischen Knoten m und Neuron j
G	:	Graph
$G(t_p)$	:	Parameter der Nachbarschaftsfunktion zum Zeitpunkt t_p
$H(.)$	:	Sprungfunktion
$H_{k_z m}$	:	$\begin{cases} 1 & , Fahrzeug\ k_z\ darf\ Knoten\ m\ bedienen \\ \infty & , sonst \end{cases}$
I	:	Anzahl Komponenten der Eingabevektoren = Anzahl Neuronen der Eingabeschicht
J	:	Anzahl Neuronen der Ausgabeschicht
K	:	Anzahl Fahrzeuge
L	:	Die Grenze der Nachbarschaft definierender Parameter der Nachbarschaftsfunktion
M	:	Anzahl Knoten
M_{mj}	:	$\begin{cases} 1 & , Kunde\ m\ ist\ Neuron\ j\ zugeordnet \\ 0 & , sonst \end{cases}$
N	:	Anzahl Knoten
$Norm_{ij}$	:	Zur Normalisierung verwendeter Koeffizient
$NMasse_j$	:	Gewicht des Neurons j
$KMasse_m$	:	Gewicht des Knotens m
$O(.)$	:	Zeitkomplexität
O_k	:	Auslastungsgrad des Fahrzeugs k
P	:	Anzahl eingesetzter Prozessoren
$P_n(x_n, y_n)$	:	Knoten n einer Instanz für das Weber-Problem
P_A^k	:	Auswahlwahrscheinlichkeit, mit der das Fahrzeug k einem Eingabevektor zugeordnet wird, Ansatz von GHAZIRI
P_B^k	:	Parameter der Auswahlwahrscheinlichkeit, Ansatz von GHAZIRI

Q_k	:	Kapazität des Fahrzeugs *k*
Q	:	Teilmenge von *V*
R	:	Anzahl Handlungsreisender
S_{mn}	:	Savings-Wert bei Realisierung der Fahrt von Knoten *m* zum Knoten *n*
T	:	Zeitobergrenze
$T(t)$	:	Parameter der Adaptionsvorschrift zum Zeitpunkt *t*, Ansatz von AMIN
U	:	Anzahl Eingabevektoren
V	:	Menge der Knoten
$V(\mathbf{W})$	:	Vom Zustand des Verbindungsnetzwerkes **W** abhängige Potentialfunktion
$\mathbf{W}$	:	Matrix, welche die Gewichte aller Verbindungen eines KNN enthält
$W(x^*, y^*)$	:	Lösung des Weber-Problems

Griechische Schriftzeichen

$\alpha(t)$	:	Lernrate zum Zeitpunkt *t*
$\beta(t)$	:	Parameter der Nachbarschaftsfunktion zum Zeitpunkt *t*
γ	:	Auslastungsgewichtung
δ	:	Parameter des Faktors zur Anpassung des Parameters $G(t_p)$, Ansatz von ANGENIOL ET AL.
$\varepsilon(t)$	:	Lernrate zum Zeitpunkt *t*
$\varepsilon_{t_{\max}} / \varepsilon_{t_{p\max}}$	:	Lernrate zum Zeitpunkt t_{max} / t_{pmax}
η_j	:	Gewinnfrequenz des Neurons *j*
λ	:	Gewichtung in der Adaptionsvorschrift
μ	:	Gewichtung in der Adaptionsvorschrift
$v(t)$	:	Parameter der Distanzfunktion zum Zeitpunkt *t*, Gewichtung des „Auslastungs-grads“
$v_{\max}$	:	Höchste Gewinnfrequenz
$v_{\min}$	:	Niedrigste Gewinnfrequenz
π_m, π_n	:	Hilfsvariablen zur Formulierung von Zyklusbedingungen
$\sigma(t)$	:	Parameter der Nachbarschaftsfunktion zum Zeitpunkt *t*
$\sigma_{t_{\max}} / \sigma_{t_{p\max}}$	:	Parameter der Nachbarschaftsfunktion zum Zeitpunkt t_{max} / t_{pmax}
$\varphi(t)$	:	Parameter der Adaptionsvorschrift zum Zeitpunkt *t*
Γ	:	Teilmenge von Γ_Z

Γ_Z	:	Menge der Zyklen
Δ_{cj}	:	Distanz zwischen Zentrum *c* und Neuron *j* im Neuronengitter
ΔE	:	Energiedifferenz
ΔQ_k	:	Differenz zwischen Fahrzeugkapazität und zugeordneter Nachfrage
$\Delta \mathbf{w}_j$	:	Differenz zwischen altem und neuem Gewichtsvektor des Neurons *j*
$\mathrm{E}(\Delta V(\mathbf{W}))$	:	Mittlere Abnahme der Potentialfunktion
$\mathrm{E}(\Delta \mathbf{w}_j \mid \mathbf{W})$	:	Bedingter Erwartungswert für die Änderung des Gewichtsvektors
Θ_i	:	Menge der Nachbarn des Neurons *i*
Λ_0	:	Menge der Indizes der Handlungsreisenden
Φ	:	Menge der Neuronen der Eingabeschicht
Ψ	:	Menge der Neuronen der Ausgabeschicht
Ω	:	Menge der Indizes u

Sonstige Zeichen und Symbole

∇	:	Nablaoperator
$\phi_{\mathbf{w}}$	:	Abbildung der Eingabevektoren auf die Neuronen der Ausgabeschicht
ù	:	Menge der natürlichen Zahlen
ø	:	Durchschnitt
$\forall$	:	für alle
$\in$	:	ist Element

Indizes

c	:	Zentrum
h	:	Neuronenindex
i	:	Neuronenindex
j	:	Neuronenindex
j^*	:	Neuron, das die geringste Distanz zum Eingabevektor aufweist
k	:	Fahrzeugindex
l	:	Knotenindex
m	:	Knotenindex
n	:	Knotenindex
r	:	Index der Handlungsreisenden

u	:	Index der Eingabevektoren
z	:	Index der Fahrzeuge
A	:	Index der Auswahlwahrscheinlichkeit, Ansatz von GHAZIRI
B	:	Index des Zwischenergebnisses, Ansatz von GHAZIRI
0	:	Index des Depots
α	:	Bezeichnung einer Kraft
β	:	Bezeichnung einer Kraft

Abkürzungsverzeichnis

Modellbezeichnungen

TSP	:	Traveling Salesman Problem
ETSP	:	Euklidisches Traveling Salesman Problem
MTSP	:	Multiple Traveling Salesman Problem
EMTSP	:	Euklidisches Multiple Traveling Salesman Problem
CVRP	:	Kapazitiertes Tourenplanungsproblem
ECVRP	:	Euklidisches (kapazitiertes) Tourenplanungsproblem
CELRP	:	Kontinuierliches euklidisches Location-Routing Problem

Sonstige Abkürzungen

ALA	:	Alternierendes Location-Allocation Verfahren
BBL	:	Beste bekannte Lösung
DEV1	:	Prozentuale Abweichung von der besten bekannten Lösung, LRPSOM-Variante.
DEV2	:	Prozentuale Abweichung von der besten bekannten Lösung, LRPSOMW-Variante.
DEV3	:	Prozentuale Angabe der Verbesserung des Zielfunktionswertes durch Verbesserung der Standortkoordinaten.
DEV4	:	Prozentuale Abweichung zwischen der besten bekannten Lösung und der Lösung, die mit Hilfe der Variante LRPSOMD ermittelt wurde.
DEV LRPSOM	:	Prozentuale Abweichung zwischen den Lösungen der Variante LRPSOM und der besten Lösung der Vergleichsverfahren
DEV LRPSOMW	:	Prozentuale Abweichung zwischen den Lösungen der Variante LRPSOMW und der besten Lösung der Vergleichsverfahren
EN	:	Elastic Net (Methode)
KNN	:	Künstliche(s) neuronale(s) Netz(e)
LRPSOM	:	Variante der Self-Organizing Map zur Lösung des CELRP
LRPSOMW	:	Variante der Self-Organizing Map zur Lösung des CELRP mit anschließender Verbesserung der Standortkoordinaten
LRPSOMD	:	LRPSOM-Variante mit Positionierung des Depots an den Originalkoordinaten
MNN	:	Multiple Nearest Neighbour
NNA	:	Nearest Neighbour Verfahren, Variante A
NNB	:	Nearest Neighbour Verfahren, Variante B

NP	:	Nondeterministic polynomial (time)
PSV	:	Parametrisiertes Savings-Verfahren
SOM	:	Self-Organizing Map
SV	:	Savings-Verfahren
TSPLIB	:	Online-Bibliothek für Testinstanzen
u.B.v.	:	unter Beachtung von
WGED	:	Weiszfeld-Algorithmus, mit Nachfragemengen gewichtete euklidische Distanzen
WUED	:	Weiszfeld-Algorithmus, mit Eins gewichtete euklidische Distanzen

1 Einleitung

1.1 Problemstellung und Zielsetzung der Arbeit

Das von GUTENBERG formulierte, auf die volle Ausnutzung von Gewinnchancen abzielende erwerbswirtschaftliche Prinzip[1] begründet verschiedene Optimierungsprobleme, mit denen Unternehmen konfrontiert werden können und die in der Bestimmung von Maxima oder Minima quantitativer Größen bestehen[2]. Zu den schwierigsten Problemen zählen dabei die sogenannten NP-schweren Optimierungsprobleme, für die bisher keine Verfahren bekannt sind, welche die exakte Lösung dieser Optimierungsprobleme mit Sicherheit mit polynomial begrenztem Zeitaufwand bestimmen können.[3] Diese Eigenschaft rechtfertigt den Einsatz heuristischer Verfahren, die unter Aufgabe der Forderung zur Bestimmung der Optimallösung in vertretbarem Zeitaufwand „möglichst gute" Lösungen bestimmen sollen[4]. Zu diesen heuristischen Verfahren können auch künstliche neuronale Netze gezählt werden[5], die sich am Aufbau und an der Funktionsweise biologischer Nervenzellverbände orientieren.

Die Idee, künstliche neuronale Netze (KNN) zur Lösung NP-schwerer (kombinatorischer) Optimierungsprobleme einzusetzen, geht auf einen Ansatz von HOPFIELD und TANK zurück, die 1985 in ihrem vieldiskutierten Artikel[6] ein KNN zur Lösung eines Traveling Salesman Problems verwenden. Neben einer Ausweitung der Anwendung künstlicher neuronaler Netze auf Optimierungsprobleme führte der Artikel unter anderem auf Grund der Tatsache, dass die veröffentlichten Ergebnisse nicht reproduziert werden konnten, zu einer kontroversen Debatte, die als einer der Gründe für die geringe Anerkennung der Erfolge der KNN auf dem Gebiet der kombinatorischen Optimierung angesehen wird[7]. Darüber hinaus wird hierfür die Tatsache verantwortlich gemacht[8], dass noch nicht in hinreichendem Ausmaß Zugang zu Hardware besteht, welche die durch neuronale Netze ermöglichte Parallelisierung der Rechenoperationen unterstützt und so zu einer Verminderung benötigter Rechenzeiten führen kann.

---

[1] Vgl. Gutenberg [1983], S. 464 f.
[2] Vgl. Gutenberg [1967], S. 18.
[3] Vgl. Maffioli [1979], S. 108.
[4] Vgl. Fuller [1978], S. 483.
[5] Vgl. zur Einordnung der künstlichen neuronalen Netze S. 5.
[6] Vgl. Hopfield und Tank [1985].
[7] Vgl. Smith [1999], S. 15. Darüber hinaus wurde gezeigt, dass das vorgeschlagene Netz häufig unzulässige Lösungen produziert und für größere Probleminstanzen nicht geeignet ist, vgl. Budinich [1997], S. 1618, Cochrane und Beasley [2003], S. 1500, und die dort angegebenen Quellen.
[8] Vgl. Smith [1999], S. 15.

Dennoch wurden mittlerweile eine Vielzahl an Netztypen entwickelt[9], die zur Lösung verschiedener Aufgaben[10] mit unterschiedlichen Erfolgen eingesetzt werden: Während einige Forscher Ergebnisse vorstellen, die darauf schließen lassen, dass KNN alternativen Techniken zur Lösung der jeweils untersuchten Problemstellungen überlegen sind, kommen andere Wissenschaftler zu genau dem entgegengesetzten Urteil.[11] Diese widersprüchlichen Ergebnisse sind auch für die weitverbreitete[12] Self-Organizing Map (SOM), ein 1982 von KOHONEN vorgestelltes KNN[13], zu beobachten[14].

Die Flexibilität der SOM eröffnet ein breites Spektrum von Anwendungsgebieten, zu denen neben statistischen Analysen, medizinischen und biologischen Anwendungen, Prozessanalysen und -kontrollen[15] auch das Gebiet der kombinatorischen Optimierung zu zählen ist[16].

Viele Probleme der betriebswirtschaftlichen Logistik stellen NP-schwere kombinatorische Optimierungsprobleme dar. Hierzu sind auch verschiedene Probleme der Routenplanung zu zählen, zu deren Lösung in der Literatur unterschiedliche SOM-Varianten vorgeschlagen werden. Die Zusammenstellung dieser Varianten und ein Vergleich der erzielten Ergebnisse stellen das erste Ziel dieser Arbeit dar.

Das zweite Ziel besteht in einer Erweiterung der SOM, die es erlaubt, den Algorithmus als Eröffnungsverfahren zur Lösung von Problemen der (kontinuierlichen) simultanen Standort- und Tourenplanung zu verwenden. Der simultane Ansatz dieser NP-schweren Probleme[17] trägt den in jüngerer Vergangenheit von verschiedenen Autoren identifizierten Interdependenzen zwischen den enthaltenen Teilproblemen der Bestimmung der Lage des / der Depots, der Zuordnung der Kunden zu Depots beziehungsweise Fahrzeugen und der Routenführung Rechnung, deren Vernachlässigung durch isolierte Betrachtung der Teilprobleme zu einer unnötigen Verschlechterung des Zielfunktionswertes

[9] Vgl. für eine Übersicht über unterschiedliche Netztypen Zell [1994].
[10] Vgl. für mögliche Anwendungsgebiete Burke und Ignizio [1992], eine ausführliche Literaturübersicht über die Anwendungen neuronaler Netze zur Lösung betriebswirtschaftlicher Probleme geben Wong et al. [1997] und Wong et al. [2000]. Neben verschiedenen Anwendungen beschreiben Smith und Gupta [2000] die am häufigsten verwendeten Netztypen und geben eine Übersicht über die historische Entwicklung der Erforschung künstlicher neuronaler Netze.
[11] Vgl. Wong et al. [1997], S. 308 und Smith [1999], S. 15.
[12] Weit über 4000 wissenschaftliche Veröffentlichungen dokumentieren die Verbreitung des Verfahrens, vgl. Kohonen [2002].
[13] Vgl. Kohonen [1982a] und Kohonen [1982b].
[14] Vgl. beispielsweise die gegensätzlichen Aussagen von Gendreau et al. [1997], S. 336, Potvin [1993], S. 344, Somhom et al. [1999], S. 404, Smith [1999], S. 28, Amin [1994], S. 132, und Modares et al. [1999], S. 604.
[15] Vgl. Kohonen [2002], S. 9. Eine Zusammenstellung aktueller Veröffentlichungen aus unterschiedlichen Forschungsgebieten geben jeweils Obermayer und Sejnowski [2001], Allinson et al. [2001], Oja und Kaski [1999] sowie Seiffert und Lakhmi [2002].
[16] Vgl. Smith [1999], S. 22 ff.
[17] Vgl. Laporte [1988], S. 163.

führen kann.[18] Viele Publikationen aus dem Gebiet der simultanen Standort- und Tourenplanung betrachten diskrete Problemstellungen[19]. Allerdings sind auch Probleme vorstellbar, die eine kontinuierliche Modellierung erfordern, beispielsweise Probleme der Planung von Standorten im Bereich der Luftfahrt[20] und Probleme aus dem Bereich der Halbleiterfertigung oder der Platinenbestückung[21]. Darüber hinaus kann die Lösung eines kontinuierlichen Modells als Näherungslösung für diskrete Problemstellungen und zur Einschränkung der Menge potentieller Standorte für diskrete Modellformulierungen verwendet werden[22].

Die Eignung des SOM-Algorithmus zur Lösung kontinuierlicher Problemstellungen der simultanen Standort- und Tourenplanung wird in dieser Arbeit anhand eines Vergleiches mit verschiedenen alternativen Heuristiken überprüft. Die Ergebnisse werden gemeinsam mit der Literaturauswertung herangezogen, um Probleme der Anwendung sowie die Leistungsfähigkeit der SOM zu untersuchen und so der umstrittenen Frage der Eignung des Verfahrens zur Lösung von Routenplanungsproblemen nachzugehen.

1.2 Gang der Untersuchung

Im zweiten Kapitel werden zunächst die Grundlagen für die folgenden Ausführungen gelegt. Im ersten Abschnitt des Kapitels wird eine mögliche Einteilung der Lösungsverfahren für kombinatorische Optimierungsprobleme und die Einordnung der verwendeten SOM vorgenommen. Abschnitt 2.2 widmet sich der Darstellung von Bestandteilen künstlicher neuronaler Netze, der dritte Abschnitt stellt auf dieser Grundlage die SOM vor und erläutert wesentliche Eigenschaften des Verfahrens. Darüber hinaus werden häufig vorgefundene Varianten und Probleme des Algorithmus beschrieben.

Das in vier Abschnitte untergliederte dritte Kapitel stellt Anwendungen der SOM auf betriebswirtschaftliche Optimierungsprobleme vor. Während in den ersten drei Abschnitten verschiedene Routenplanungsprobleme erläutert und die Anwendung der SOM anhand einer Literaturübersicht und eines Beispiels illustriert werden, wird im vierten Abschnitt eine Übersicht über weitere Optimierungsprobleme gegeben, die mit Hilfe der SOM gelöst werden können. Im vierten Kapitel wird zunächst ein Problem der simultanen, kontinuierlichen Standort- und Tourenplanung beschrieben. Anschließend

[18] Vgl. Sahli und Rand [1989], S. 15 f.
[19] Eine Einführung in Problemstellungen dieser Klasse gibt Laporte [1988].
[20] Vgl. Love et al. [1988], S. 146, und Amin et al. [1994], S. 121.
[21] Vgl. Fujimura et al. [2001].
[22] Kontinuierliche Modelle werden häufig eingesetzt, um mögliche Standort-Kandidaten für diskrete Modelle zu identifizieren, vgl. Love et al. [1988], S. 143 ff., und Lozano et al. [1998], S. 107.

wird eine Adaption der SOM vorgenommen, um eine Lösung für das formulierte Problem ableiten zu können. Zur Einschätzung der Qualität des entwickelten Verfahrens wird ein Vergleich der Ergebnisse der SOM-Variante für eine Reihe bekannter Testinstanzen mit den Ergebnissen verschiedener, sequentiell arbeitender Heuristiken vorgenommen, die ebenfalls zur Lösung des Problems implementiert wurden.

Das fünfte Kapitel, das die Untersuchungsergebnisse zusammenfasst, bildet den Abschluss der Arbeit.

2. Grundlagen

2.1 Lösungsverfahren für kombinatorische Optimierungsprobleme und Einordnung der Self-Organizing Map

Die Verfahren zur Lösung von kombinatorischen Optimierungsproblemen werden in der Regel in exakte und heuristische Verfahren unterteilt.[23] Zur Gruppe exakter Verfahren werden dabei solche Verfahren gezählt, die mit Sicherheit das globale Optimum für das untersuchte Problem ermitteln. Die Gruppe der Heuristiken[24] umfasst Methoden, die nicht mit Sicherheit das globale Optimum identifizieren. Heuristiken werden zur Suche nach einer möglichst guten Lösung meist speziell der Gestalt eines Optimierungsproblems angepasst. Innerhalb der Gruppe der Heuristiken können Eröffnungsverfahren und Verbesserungsverfahren unterschieden werden[25]. Während Eröffnungsverfahren eine erste zulässige Lösung ermitteln, versuchen Verbesserungsverfahren, eine zulässige Lösung durch lokale Suche hinsichtlich des Zielfunktionswertes zu verbessern[26]. Neben den beschriebenen Methoden existieren Verfahren, die problemunabhängige Vorgehensweisen zur Verfügung stellen, und die zur Ermittlung von Lösungen unterschiedlicher Optimierungsprobleme verwendet werden können. Zu diesen Metastrategien können auch künstliche neuronale Netze gezählt werden.[27] Die Self-Organizing Map (*selbstorganisierende Karte*) ist innerhalb der KNN der Gruppe der Netze zuzuordnen, deren Lernvorgang auf einem Wettbewerb zwischen den Neuronen beruht.[28]

Ein Algorithmus, der zur Lösung eines speziellen Problems einen problemunabhängigen Mechanismus einer Metastrategie integriert, kann aufgrund der problemspezifischen Ausgestaltung der Metastrategie der Gruppe der Heuristiken zugeordnet werden. Die in dieser Arbeit untersuchten Varianten der Self-Organizing Map können dabei zu den Eröffnungsverfahren gezählt werden[29], da sie das Ziel verfolgen, eine erste zulässige Lösung für das bearbeitete Problem zu bestimmen. Daher sind bei einer Evaluation des Algorithmus, die auf einem Vergleich mit alternativen Verfahren beruht, ebenfalls

[23] Vgl. Domschke und Drexl [2004], S. 128.
[24] Vgl. für eine aktuelle Übersicht über heuristische Lösungsverfahren Silver [2004].
[25] Vgl. Foulds [1983], S. 929.
[26] Eine ausführliche Übersicht über Verbesserungsverfahren findet sich in Aarts und Lenstra [1997].
[27] Vgl. Domschke [1997], S. 21.
[28] Die Einteilung der künstlichen neuronalen Netze erfolgt gemäß Kohonen [2001], S. 85. Der Vorgang des Wettbewerbslernens wird detailliert in Kapitel 2.3 beschrieben.
[29] Vgl. Burke [1996], S. 122, und Modares et al. [1999], S. 604.

Eröffnungsverfahren zu verwenden.[30] In Abbildung 2.1 wird eine mögliche Einteilung der Verfahren zur Lösung kombinatorischer Optimierungsprobleme vorgestellt[31], Abbildung 2.2 zeigt die mögliche Zuordnung künstlicher neuronaler Netze zur Gruppe der Metastrategien und die Einordnung der Self-Organizing Map innerhalb der künstlichen neuronalen Netze.

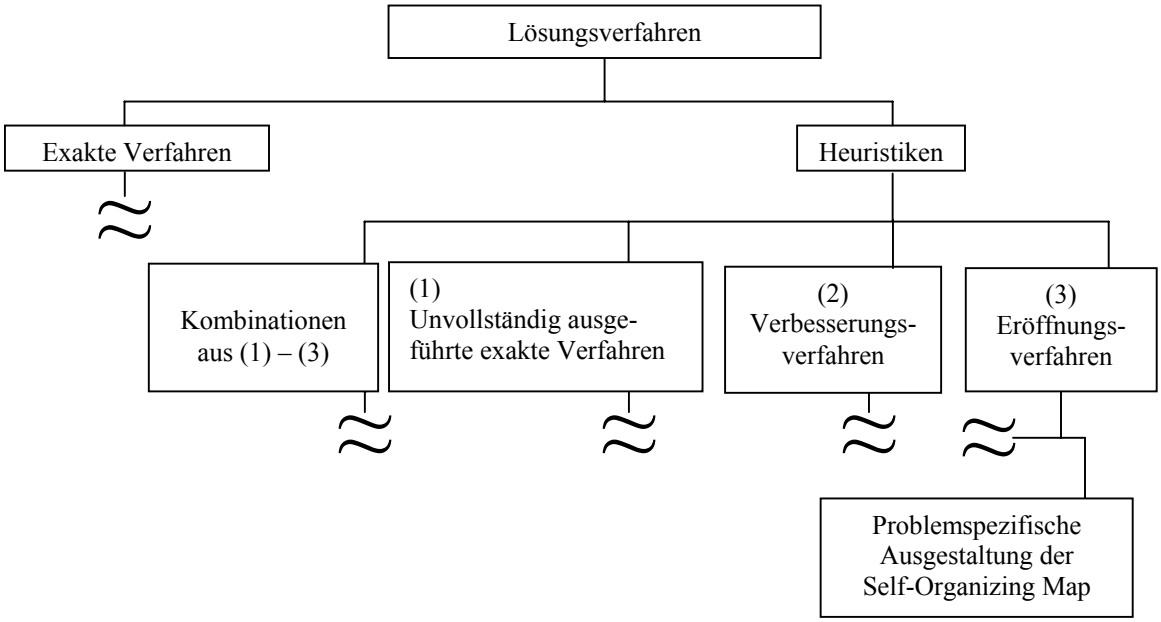


Abbildung 2.1: Einteilung der Lösungsverfahren und Einordnung der Self-Organizing Map

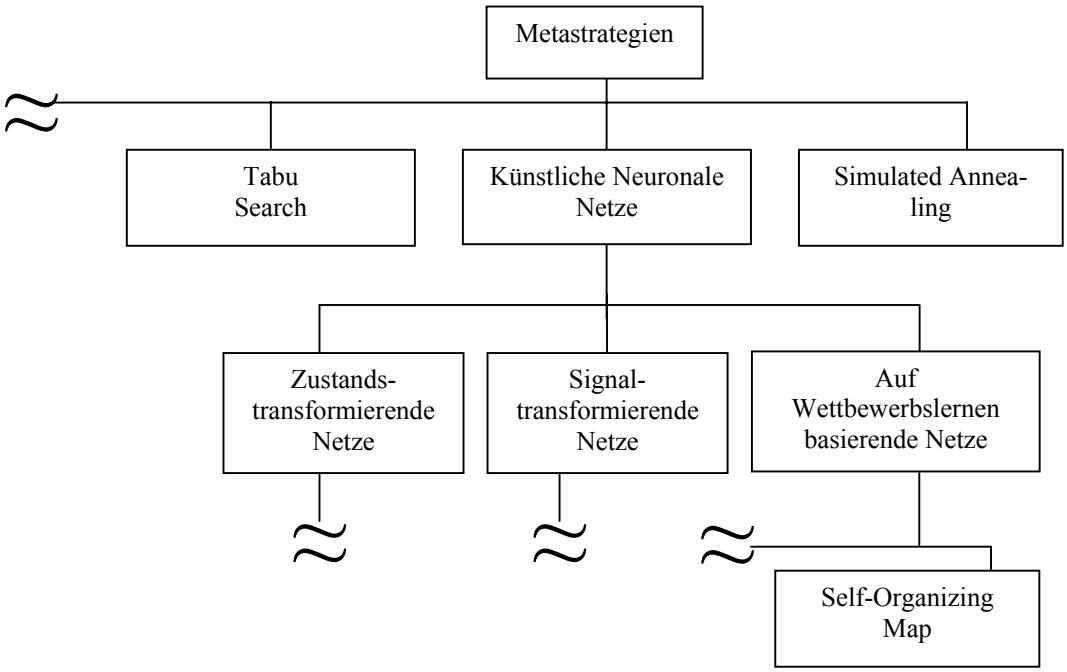


Abbildung 2.2: Einordnung der Self-Organizing Map innerhalb der künstlichen neuronalen Netze

[30] Vgl. Burke [1996], S. 122.
[31] Die Einteilung der Lösungsverfahren erfolgt auf Basis der von Domschke und Drexl vorgeschlagenen Unterteilung, vgl. Domschke und Drexl [2004], S. 126 ff.

2.2 Künstliche Neuronale Netze

2.2.1 Definition künstlicher neuronaler Netze

KNN können als biologisch inspirierte, informationsverarbeitende Systeme definiert werden, die aus einer großen Anzahl verarbeitender Elemente[32] (Neuronen) bestehen. Diese senden sich Informationen in Form der den Zustand der Neuronen beschreibenden Aktivierung über gerichtete Verbindungen, deren Gesamtheit als Verbindungsnetzwerk bezeichnet wird, zu[33]. Als zusätzlicher Bestandteil eines KNN kann die sogenannte Lernregel angesehen werden[34], mit deren Hilfe das KNN Eigenschaften auf der Basis der dem Netz präsentierten Informationen erlernt. Die Art der Informationsverarbeitung eines KNN wird durch Definition des Neuronenmodells, des Verbindungsnetzwerks und der Lernregel festgelegt, deren Gestaltungsmöglichkeiten in den folgenden Abschnitten beschrieben werden.

2.2.2 Neuronenmodell

Das erste formale Modell eines künstlichen Neurons beschreiben McCULLOCH und PITTS bereits 1943[35], die ein Neuron als logisches Schwellenwertelement mit zwei möglichen Zuständen abbilden. Grundsätzlich kann ein Neuron als Objekt betrachtet werden, das verschiedene Eigenschaften aufweist, die festlegen, wie die sogenannte Aktivierung $a_j(t+1)$ eines Neurons und die von der Aktivierung abhängige Ausgabe $o_j(t+1)$ durch Verarbeitung der Eingabe $net_j(t+1)$ zum Zeitpunkt *t+1* berechnet werden (s. Abbildung 2.3).

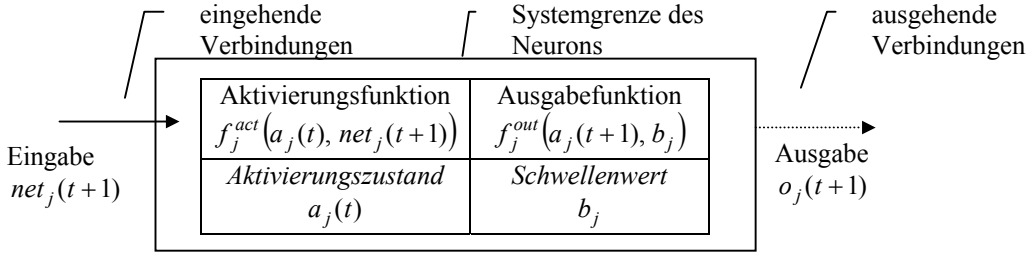


Abbildung 2.3 : Neuronenmodell

[32] Vgl. Kasabov [1996], S. 18.
[33] Vgl. Zell [1994], S. 23 und Lohrbach [1994], S. 9 ff.
[34] Vgl. Zell [1994], S. 72.
[35] Vgl. McCulloch und Pitts [1943].

Die Aktivierungsfunktion f_j^{act} dient zur Berechnung der aktuellen Aktivierung des Neurons und kann beispielsweise von der aktuellen Eingabe und dem bisherigen Aktivierungszustand abhängen[36]. Die Ausgabefunktion f_j^{out} ermittelt in Abhängigkeit von der berechneten Aktivierung die Ausgabe des Neurons, die über die vom Neuron ausgehenden Verbindungen weitergeleitet wird: Überschreitet die Aktivierung des Neurons den Schwellenwert b_j, der auch als Parameter der Funktionen angesehen werden kann[37], so weist die Ausgabe des Neurons einen von Null verschiedenen Wert auf. Wie in Abbildung 2.3 skizziert, besteht ein Neuron also aus Vorschriften zur Verarbeitung von Informationen und einem Speicher, in dem Informationen wie der Aktivierungszustand und der Schwellenwert über einen bestimmten Zeitraum erhalten werden können. Darüber hinaus können die eingehenden Verbindungen ebenfalls als Bestandteil eines Neurons angesehen werden.[38]

Neuronen verarbeiten Informationen weitgehend unabhängig voneinander.[39] Durch diese Eigenschaft ergibt sich die Möglichkeit, die Informationsverarbeitung zu parallelisieren und so die Geschwindigkeit der Verarbeitung zu erhöhen.

KNN werden oft anhand der Art und des Zeitpunktes der Berechnung der Aktivierung der Neuronen unterteilt:[40] Wenn jedes Neuron seine Aktivierung direkt aus den eingehenden Informationen berechnet, spricht man von deterministischen Netzen. Wird die Aktivierung eines Teils der Neuronen hingegen nur mit einer bestimmten Wahrscheinlichkeit berechnet, so bezeichnet man die Netze als stochastisch.

Wenn alle Neuronen oder bestimmte Mengen von Neuronen gleichzeitig ihre Aktivierung berechnen[41] (oder berechnen könnten), so bezeichnet man die Berechnung als synchron, in allen anderen Fällen als asynchron.

2.2.3 Verbindungsnetzwerk

Die das Verbindungsnetzwerk eines KNN bildenden Verbindungen dienen der Informationsübertragung. Auch Verbindungen können Eigenschaften zugewiesen werden, die

[36] Vgl. Zell [1994], S. 83.
[37] Vgl. Zell [1994], S. 82.
[38] Vgl. Kasabov [1996], S. 257. Dieses Vorgehen ist für die weiteren Betrachtungen sinnvoll, da im Verlauf der Arbeit die den eingehenden Verbindungen zugeordneten Gewichte als Koordinaten der Position eines Neurons in der Ebene interpretiert werden.
[39] Vgl. Lohrbach [1994], S. 9.
[40] Vgl. bspw. Lohrbach [1994], S. 27.
[41] Dies ist bei Verwendung parallel arbeitender Prozessoren möglich.

durch Funktionen abgebildet werden (s. Abbildung 2.4), welche die Ausgabe $o_i(t)$ eines Neurons *i* verarbeiten. Üblich ist die zeitabhängige Gewichtung der Ausgabe eines Neurons durch Multiplikation mit einem als Gewicht bezeichneten Skalar $w_{ij}(t)$[42]. Die Gewichte der in ein Neuron *j* eingehenden Verbindungen werden häufig in einem Gewichtsvektor $\mathbf{w}_j(t)$ zusammengefasst[43].

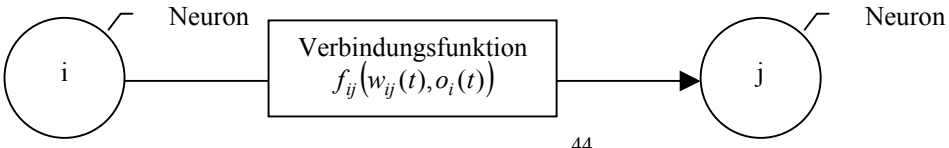


Abbildung 2.4 : Verbindungsfunktion[44]

Sollen die einem Neuron über einzelne Verbindungen zugesendeten Informationen zu einem Eingangssignal verarbeitet werden, so kann eine Propagierungsfunktion verwendet werden, welche die eingehenden Informationen zu einem Signal umwandelt (s. Abbildung 2.5).

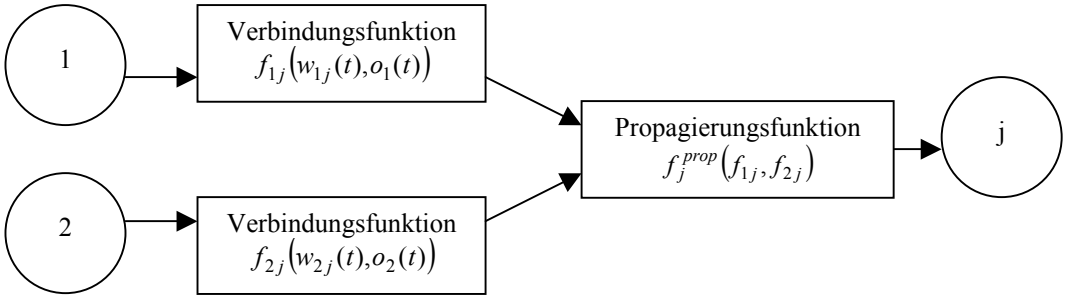


Abbildung 2.5 : Propagierungsfunktion[45]

Eine Verbindung wird als *exzitatorisch* bezeichnet, wenn die Ausgabe eines Neurons zur Erhöhung der Aktivierung eines anderen Neurons führt, eine Verbindung wird als *inhibitorisch* bezeichnet, wenn die Ausgabe zur Verminderung der Aktivierung eines anderen Neurons führt.[46]

Verbindungen sind gerichtet,[47] Informationen können also jeweils nur in eine Richtung übertragen werden, so dass zum Informationsaustausch zwischen zwei Neuronen gege-

[42] Vgl. Zell [1994], S. 72.
[43] Vgl. Nour und Madey [1996], S. 430.
[44] Vgl. Lohrbach [1994], S. 18 f.
[45] Vgl. Grauel [1992], S. 50 und Lohrbach [1994], S. 20. Es ist auch möglich, dass die Ausgabewerte und Verbindungsgewichte direkt in die Propagierungsfunktion einfließen und auf die Verbindungsfunktionen verzichtet wird.
[46] Vgl. Lohrbach [1994], S. 11.
[47] Vgl. Zell [1994], S. 72.

benenfalls zwei Verbindungen benötigt werden. Neben einer Verbindung zwischen zwei Neuronen und Verbindungen des Netzes mit der Systemumgebung[48], mit der das Netz Informationen austauscht, ist auch eine Verbindung eines Neurons mit sich selbst denkbar (s. Abbildung 2.6).

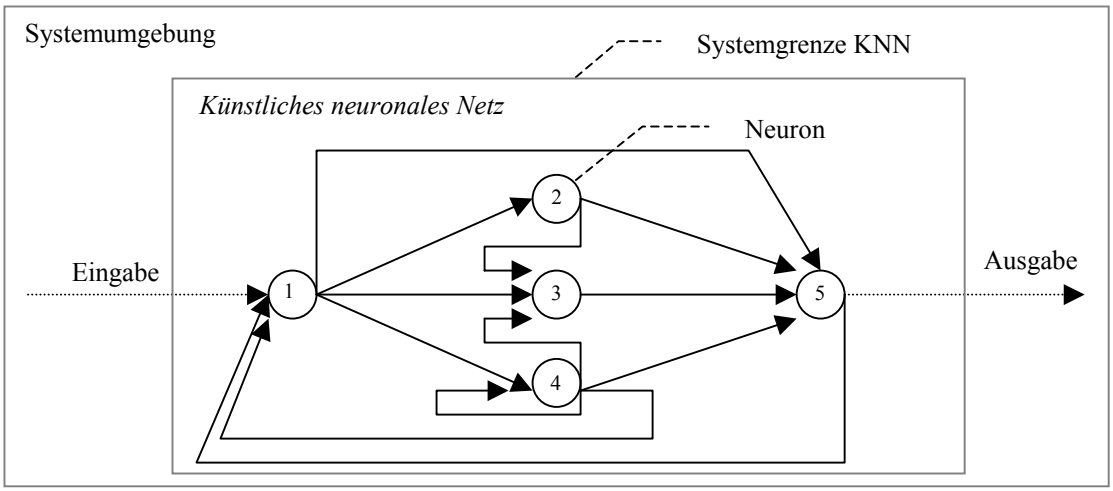


Abbildung 2.6: Mögliche Verbindungen in KNN

Durch die in einem Netz vorhandenen Verbindungen entsteht eine bestimmte Netzstruktur. Diese Struktur kann häufig durch eine Gruppierung von Neuronen, die durch ein gleichartiges Verhalten beziehungsweise identische Funktionen gekennzeichnet sind, in Neuronenschichten unterteilt werden[49]. Üblicherweise werden drei Schichttypen unterschieden: Die Eingabeschicht steht mit der Systemumgebung des KNN in Verbindung und nimmt die in einem sogenannten Eingabevektor zusammengefassten Eingangsinformationen auf, versteckte Schichten stehen hingegen nicht in direktem Kontakt mit der Systemumgebung und verarbeiten eingehende Informationen, die Ausgabeschicht dient (unter anderem) zur Übergabe des Verarbeitungsergebnisses an die Systemumgebung.[50] Abbildung 2.6 verdeutlicht, dass Informationen sowohl innerhalb einer Schicht, zwischen zwei aufeinanderfolgenden Schichten oder über Schichten hinweg ausgetauscht werden können. Vorwärtsgerichtete Verbindungen, die Informationen von einer Schicht zu einer nachgelagerten Schicht übertragen, werden als *Bottom-Up-Verbindungen*, rückwärtsgerichtete Verbindungen als *Top-Down-Verbindungen*, Verbindungen innerhalb einer Neuronenschicht als *laterale Verbindungen* bezeichnet.[51] Bei rückwärts gerichteten Verbindungen zu einer vorgelagerten Schicht ist zu beachten,

[48] Die Systemumgebung umfasst neben der Ein- und Ausgabecodierung weitere Instanzen der Daten-, Prozess- und Schnittstellenverwaltung, vgl. Kratzer [1993], S. 31 f.
[49] Vgl. Kratzer [1993], S. 27.
[50] Vgl. Kratzer [1993], S. 27.
[51] Vgl. Lohrbach [1994], S. 14 f.

dass nur Ausgabewerte zeitlich vorausgehender Berechnungen als Eingaben eingehen können, da der Ausgabewert der aktuellen Berechnung nicht vorliegen kann.

Die Bezeichnung der Schichten eines KNN soll hier von der Bezeichnung der Stufen eines KNN unterschieden werden (s. Abbildung 2.7), die durch die Anzahl der Schichten mit veränderbaren Gewichten zugehöriger Verbindungen definiert werden.[52]

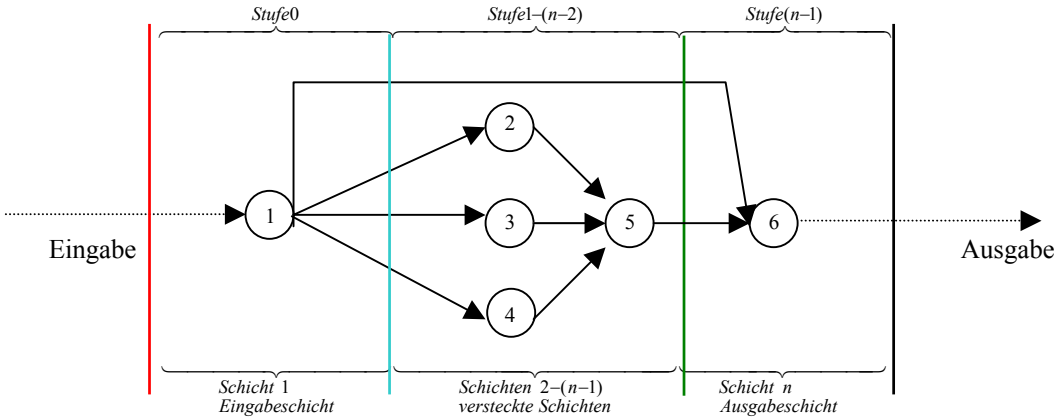


Abbildung 2.7: Schichten und Stufen eines KNN

Anhand der Richtung des Informationsflusses entlang der Verbindungen eines Netzes werden KNN in folgende Klassen eingeteilt[53]: Bei *Feed-Forward-Netzen* treten nur Verbindungen in Richtung nachgelagerter Schichten auf, zur Klasse der *Feed-Back-Netze* werden alle Netze gezählt, die auch laterale und / oder Top-Down-Verbindungen enthalten.

2.2.4 Lernregel

Die Lernregel eines KNN kann als Algorithmus angesehen werden, der festlegt, wie die Eingabevektoren das KNN beeinflussen. Unter dem Begriff „Lernen“ kann dabei

1. die Entwicklung neuer Verbindungen,
2. das Löschen existierender Verbindungen,
3. die Entwicklung neuer Neuronen,
4. das Löschen von Neuronen,
5. die Modifikation der Gewichtsvektoren,
6. die Modifikation der Schwellenwerte, oder
7. die Modifikation der Aktivierungs-, Verbindungs-, Propagierungs- oder Ausgabefunktion

[52] Vgl. Zell [1994], S. 73.
[53] Vgl. Lohrbach [1994], S. 26.

verstanden werden.[54] Es wird deutlich, dass die Lernregel bei Kombination der beschriebenen Vorgänge sowohl Eigenschaften vorhandener Bestandteile des KNN als auch die Struktur des KNN selbst, d.h. das Verbindungsnetzwerk oder die Anzahl an Neuronen, verändern kann.

Die Einteilung der Lernregeln wird in der Literatur nach unterschiedlichen Kriterien vorgenommen[55]. Alle Einteilungen lassen sich dabei auf die Frage reduzieren, ob für die Eingabeinformationen auch Ausgabeinformationen vorliegen, mit deren Hilfe der Lernprozess gesteuert wird.[56] Wird der Lernprozess in dieser Art gesteuert, so spricht man von überwachtem Lernen[57], andernfalls wird eine Lernregel als unüberwachtes Lernen[58] bezeichnet.

Bei der Variante des überwachten Lernens müssen neben Eingabevektoren auch die zugehörigen gewünschten Ausgabevektoren als externe Vorgabe vorliegen oder eine Bewertung der Netzausgabe[59] vorgenommen werden. Anhand eines Vergleiches zwischen der Netzausgabe und dem gewünschten Ausgabevektor bzw. der Bewertung der Netzausgabe erlernt das KNN, die erwünschte Netzausgabe für die Eingabevektoren zu produzieren.

Die Variante des unüberwachten Lernens verzichtet vollständig auf externe Vorgaben, es werden lediglich Eingabevektoren präsentiert. Die vorgegebene Lernregel wird dabei so gestaltet, dass die Präsentation der Eingabevektoren zu einer Anpassung des Netzes führt. Zu dieser Kategorie ist beispielsweise die 1949 von HEBB entwickelte Lernregel zu zählen, die zu einer Anpassung einer Verbindung (beziehungsweise des zugeordneten Gewichtes) führt, wenn die der Verbindung zugeordneten Neuronen zur gleichen Zeit einen von Null verschiedenen Aktivierungszustand aufweisen.[60]

In die Klasse des unüberwachten Lernens ist auch das von KOHONEN 1982 entwickelte Lernverfahren der Self-Organizing Map[61] einzuordnen, die das sogenannte Wettbewerbslernen[62] verwendet. Bei dieser Form des Lernens werden den verschiedenen Neu-

---

[54] Vgl. Zell [1994], S. 84.
[55] Vgl. für eine Übersicht Lohrbach [1994], S. 30 ff.
[56] Vgl. Lohrbach [1994], S. 35.
[57] Diese Form wird auch als *supervised learning, Lernen mit Lehrer* oder *beaufsichtigtes Lernen* bezeichnet, vgl. Lohrbach [1994], S. 32.
[58] Diese Form wird auch als *unsupervised* bzw. *self-supervised learning, self-organized learning* oder *unbeaufsichtigtes Lernen* bezeichnet, vgl. Lohrbach [1994], S. 32.
[59] Dieser Fall wird auch als *bestärkendes Lernen* oder *Graded Learning* bezeichnet, vgl. Zell [1994], S. 93 f. und Lohrbach [1994], S. 33.
[60] Vgl. Hebb [1949].
[61] Vgl. Kohonen [1982a].
[62] Vgl. für eine ausführliche Darstellung des Wettbewerbslernens Rumelhart und Zipser [1985] und Kohonen [2001], S. 86.

ronen identische Informationen zur Verfügung gestellt und ein Wettbewerb zwischen den Neuronen durchgeführt. Das Neuron, das diesen Wettbewerb - beispielsweise auf Grund der stärksten Aktivierung - gewinnt, weist meistens als einziges Neuron der Ausgabeschicht eine von Null verschiedene Ausgabe auf.

Tabelle 2.1 fasst abschließend die in den vorangegangenen Abschnitten beschriebenen Komponenten künstlicher neuronaler Netze zusammen.

Komponente	**Bestandteile / Symbol**	**Beschreibung**
Neuronen	*Aktivierungszustand* $a_j(t)$	Grad der Aktivierung des Neurons *j* zum Zeitpunkt *t*.
	Aktivierungsfunktion $a_j(t+1) = f_j^{act}\left(a_j(t),\, net_j(t)\right)$	Vorschrift zur Berechnung des Aktivierungszustandes des Neurons *j*.
	Ausgabefunktion $f_j^{out}\left(a_j(t),\, b_j\right)$	Vorschrift zur Berechnung der von der Aktivierung abhängigen Ausgabe $o_j(t)$ eines Neurons *j* zum Zeitpunkt *t*.
Verbindungsnetzwerk	*Der Zustand des Verbindungsnetzwerks kann durch die Matrix* $\mathbf{W}$ *der Gewichte aller Verbindungen beschrieben werden.*	Ein KNN kann als gerichteter Graph angesehen werden, wobei die Verbindungen der Neuronen als Kanten des Graphen und die Neuronen als Knoten interpretiert werden. Das Symbol $w_{ij}(t)$ beschreibt die zeitabhängige Kantenbewertung der Kante zwischen Neuron *i* und Neuron *j* zum Zeitpunkt *t*.
	Verbindungsfunktion $f_{ij}\left(w_{ij}(t), o_i(t)\right)$	Vorschrift zur Berechnung der von der Gewichtung $w_{ij}(t)$ der Verbindung zwischen Neuron *i* und Neuron *j* abhängigen Information, die zum Zeitpunkt *t* an Neuron *j* gesendet wird.
	Propagierungsfunktion $f_j^{prop}\left(f_{ij}\left(w_{ij}(t), o_i(t)\right), \forall i\right)$	Vorschrift zur Zusammenfassung der Verbindungsfunktionswerte zum Zeitpunkt *t* zur Bestimmung der Neuroneneingabe $net_j(t)$ für Neuron *j*.
Lernregel	*Algorithmus*	Anweisungen, nach deren Vorschrift Eigenschaften des KNN verändert werden.

Tabelle 2.1 : Komponenten künstlicher neuronaler Netze

2.2.5 Ablaufdiagramm zum Einsatz künstlicher neuronaler Netze

Bevor in Kapitel 2.3 näher auf das Verfahren des Wettbewerbslernens eingegangen wird, soll zunächst in einem Ablaufdiagramm eine mögliche Vorgehensweise zum Einsatz künstlicher neuronaler Netze vorgestellt werden. Die Entwicklung und der Einsatz eines KNN zur Lösung einer spezifischen Aufgabenstellung können wie in Abbildung 2.8 dargestellt vorgenommen werden.[63]

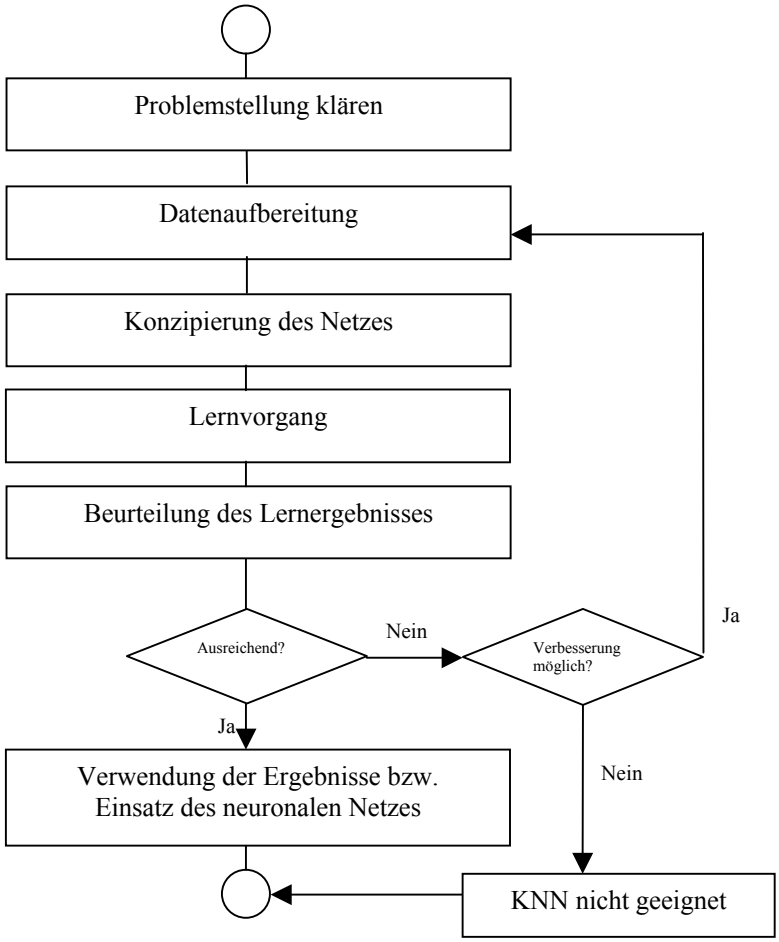


Abbildung 2.8: Ablaufdiagramm zum Einsatz von KNN

Die einzelnen Schritte des Ablaufdiagramms können durch die folgenden Aufgaben charakterisiert werden:[64] Der Klärung der Problemstellung, zu der auch die Abbildung des Problems in einem formalen Modell gezählt werden kann, schließt sich die Datenaufbereitung an. In diesem Schritt ist die Eignung der vorliegenden Datenmengen hinsichtlich quantitativer und qualitativer Anforderungen festzustellen und eine Aufbereitung der Eingabe- und Ausgabedaten vorzunehmen. Hierzu ist die benötigte Form der

[63] Vgl. Corsten und May [1996], S. 7.
[64] Vgl. Corsten und May [1996], S. 6 ff.

Datenpräsentation festzulegen, gegebenenfalls ist eine Aufteilung der vorliegenden Daten in eine Trainings- und eine Testdatenmenge vorzunehmen. Im Rahmen der Konzipierung des KNN sind der zu verwendende Netztyp, das Neuronenmodell, das Verbindungsnetzwerk und die Lernregel festzulegen. Der mit Erreichen eines definierten Abbruchkriteriums endende Lernvorgang besteht in der Präsentation der Eingabedaten und der Verarbeitung der Daten durch das KNN. Das Ergebnis des Lernvorgangs ist anschließend, beispielsweise durch Verwendung der Testdatenmenge, zu bewerten. Sind die Lernergebnisse nicht zufriedenstellend, so kann eine andere Datenaufbereitung oder Konfiguration des Netzes vorgenommen und der Lernvorgang - gegebenenfalls mit anderen Parametern - wiederholt werden. Lässt sich kein zufriedenstellendes Ergebnis erzielen, d.h., das am besten geeignete Netz führt im Sinne der Problemstellung und Zielsetzung zu einer nicht hinreichenden Ergebnisqualität, so ist die Aufgabenstellung für KNN nicht geeignet. Andernfalls können die Ergebnisse des Lernvorgangs verwendet beziehungsweise das KNN zur Erfüllung der Aufgabenstellung angewendet werden.

2.3 Die Self-Organizing Map

2.3.1 Motivation und Herleitung

Die Self-Organizing Map, ein 1982 von KOHONEN entwickelter Netztyp,[65] gehört zu den selbstorganisierenden KNN. Sie orientiert sich an Vorgängen im Neokortex des Gehirns, also der Hirnrinde, einer stark gefalteten Schicht aus Nervenzellen, in der ähnliche Eingabesignale auch zur Aktivierung von Neuronen in benachbarten Regionen führen.[66,67] Die lokale Aktivierung kann dabei durch inhibitorische und exzitatorische laterale Verbindungen in den Neuronenschichten der Hirnrinde erklärt werden, welche die Aktivierung weit entfernter Neuronen vermindern beziehungsweise die Aktivierung benachbarter Neuronen erhöhen. Eingabesignale werden häufig von Rezeptoren in einer Sinnesoberfläche an das Gehirn geleitet, wobei benachbarte Neuronen Signale von benachbarten Rezeptoren erhalten, es besteht also eine Abbildung der Sinnesoberfläche auf dem zugeordneten Rindenfeld.[68] Diese Abbildungen werden auch als *Karten (Maps)* bezeichnet, deren Ausbildung durch eine auf Erfahrung beruhende Selbstorganisation[69] der Neuronenverbindungen erklärt werden kann.

Ein solcher Prozess der Selbstorganisation kann beispielsweise durch den folgenden Mechanismus begründet werden[70]: Die Neuronen werden im ersten Schritt mit einem eingehenden Signal anhand der hervorgerufenen Aktivierung der Neuronen verglichen. Durch einen Wettbewerb der Neuronen wird dann eine Gruppe benachbarter Neuronen bestimmt, die aufgrund der Ähnlichkeit zum Eingabesignal am stärksten aktiviert wurden. Anschließend werden die Eigenschaften der zugeordneten Neuronenverbindungen in Abhängigkeit von der Aktivierung der Neuronen verändert, so dass diese bei erneuter Präsentation des eingehenden Signals noch stärker aktiviert werden. Werden viele solcher Lernschritte für verschiedene Eingabesignale wiederholt, so nehmen die den Neuronen der Ausgabeschicht zugeordneten Verbindungen schließlich Eigenschaften an, die in ähnlicher Relation zueinander stehen wie die Eingabesignale, d.h. es entsteht eine

[65] Vgl. Kohonen [1982a] und Kohonen [1982b].
[66] Vgl. Retzko [1996], S. 24.
[67] So lässt sich die Hirnrinde in unterschiedliche Bereiche aufteilen, die verschiedene Signale verarbeiten, vgl. Ritter et al. [1994], S. 17 ff. bzw. Retzko [1996], S. 25. Beispiele für solche Bereiche stellen der sogenannte somatosensorische Kortex, der für die Tastwahrnehmung verantwortlich ist, oder der visuelle Kortex dar, der für die visuelle Wahrnehmung verantwortlich ist.
[68] Vgl. Ritter et al. [1994], S. 22.
[69] Ausführliche Überlegungen zur Selbstorganisation von Neuronen werden in der Veröffentlichung von Willshaw und von der Malsburg [1976] angestellt.
[70] Vgl. Kohonen [1982a], S. 60 und Kohonen [2001], S. 102f.

topologieerhaltende Abbildung der Eingabesignale. Abbildung 2.9 stellt ein Modell für zwei miteinander verbundene Neuronenschichten dar[71], mit dem der beschriebene Prozess der Selbstorganisation umgesetzt werden kann.

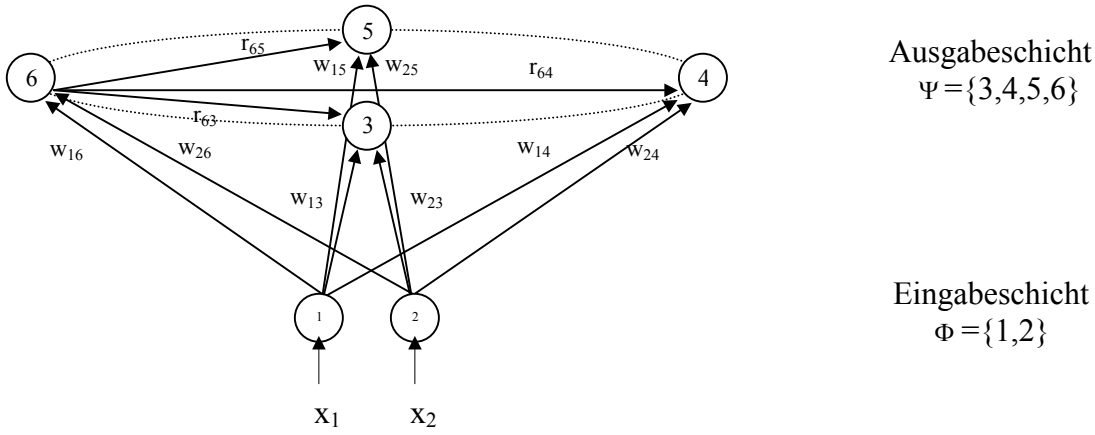


Abbildung 2.9: Darstellung einer Self-Organizing Map mit ringförmigem Neuronengitter

Typischerweise wird eine vollständige Verbindung der beiden Schichten vorgenommen, d.h., jedes Neuron der zum Einlesen von Informationen dienenden[72] Eingabeschicht (Rezeptorschicht) hat eine unidirektionale Verbindung zu jedem Neuron der Ausgabeschicht. Die Neuronen der Ausgabeschicht werden in einem Gitter angeordnet, das Nachbarschaftsbeziehungen der Neuronen in der Hirnrinde widerspiegelt und so die Nachbarschaftsbeziehungen in der Ausgabeschicht bestimmt. Die Komponenten x_i der Eingabevektoren $\mathbf{x}$ stellen die Eingabesignale dar, die den einzelnen Neuronen der Eingabeschicht präsentiert werden. Diese Informationen werden über die Verbindungen zu jedem Neuron der Ausgabeschicht übermittelt und rufen deren Aktivierung hervor. Einer Verbindung zwischen einem Neuron der Eingabeschicht und einem Neuron der Ausgabeschicht wird ein Gewicht w_{ij} $(i \in \Phi, i = 1(1)I, j \in \Psi, j = I+1(1)I+J)$ zugeordnet. Zusätzlich sind die Neuronen der Ausgabeschicht durch exzitatorische (bei im Neuronengitter benachbarten Neuronen) und inhibitorische (bei weiter entfernten Neuronen) laterale, mit den Gewichten r_{jh} $(j,h \in \Psi)$ bewertete Verbindungen untereinander verbunden, welche die lokale Aktivierung der Ausgabeschicht bewirken[73,74]. Die Eingabe für ein Neuron der Ausgabeschicht bei Präsentation eines Eingabesignals zum Zeitpunkt *t* kann dann beispielsweise durch die Funktion

[71] Vgl. beispielsweise Kinnebrock [1992], S. 79.
[72] Vgl. Retzko [1996], S. 25.
[73] Vgl. Kohonen [1990], S. 1467 und Ritter et al. [1994], S. 69.
[74] In der Abbildung werden stellvertretend nur die lateralen Verbindungen zwischen Neuron 6 und den übrigen Neuronen der Ausgabeschicht dargestellt. Die Verbindung zwischen Neuron 6 und Neuron 4 könnte beispielsweise inhibitorisch, die übrigen Verbindungen könnten exzitatorisch modelliert werden.

$$(2.1) \quad net_j(t) = \sum_{i=1}^{I} o_i \cdot w_{ij} + \sum_{\substack{h=I+1,\\ h\neq j}}^{I+J} o_h \cdot r_{jh}$$

berechnet werden.[75] Die Aktivierung des Neurons, die zur Bestimmung der Ähnlichkeit des Eingabesignals und des Neurons herangezogen wird, kann somit nur durch Lösung des Gleichungssystems[76]

$$(2.2) \quad a_j(t) = f_j^{act}\left(\sum_{i=1}^{I} o_i \cdot w_{ij} + \sum_{\substack{h=I+1,\\ h\neq j}}^{I+J} o_h\left(a_h(t)\right) \cdot r_{jh} \right), \forall\, j$$

bestimmt werden.[77] Um diesen aufwendigen Schritt zu vermeiden, vernachlässigt KOHONEN zunächst den Einfluss der lateralen Verbindungen innerhalb der Ausgabeschicht[78] und bestimmt die Ähnlichkeit zwischen Eingabevektor und Neuron *j* ausschließlich auf Basis der euklidischen Distanz zwischen Eingabevektor und Gewichtsvektor des Neurons.[79] Die maximale Aktivierung weist bei KOHONEN dabei das Neuron auf, für das die euklidische Distanz zwischen Gewichtsvektor und Eingabevektor minimal ist. Im folgenden Schritt des selbstorganisierenden Prozesses wird nun der Wettbewerb zwischen den Neuronen der Ausgabeschicht ausgetragen, den das Neuron *c* mit der maximalen Aktivierung gewinnt. Damit wird eine vom Zustand des Verbindungsnetzwerks abhängige Abbildung

$$(2.3) \quad \phi_{\mathbf{w}} : \mathbf{x}(t) \mapsto c = \phi_{\mathbf{w}}\left(\mathbf{x}(t)\right)$$

des Eingabevektors auf die Ausgabeschicht vorgenommen.[80] Das Bild *c* des Eingabevektors wird auch als Zentrum bezeichnet. Schließlich wird im letzten Schritt der Selbstorganisation eine von der Aktivierung $h_{cj} = a_j(t)$ eines Neurons *j* abhängige Anpassung der Gewichte der zugeordneten Verbindungen gemäß

$$(2.4) \quad \Delta w_{ij} = h_{cj} \cdot (x_i - w_{ij}), \forall i$$

vorgenommen. Die von der Nachbarschaftsbeziehung zu *c* abhängige Aktivierung eines Neurons *j* wäre allerdings wieder nur durch Lösung des Gleichungssystems *(2.2)* bestimmbar[81]. Zur Vereinfachung nähert KOHONEN h_{cj} deshalb durch eine vorgegebene Nachbarschaftsfunktion mit einem Maximum bei h_{cc} an, die für große Abstände Δ_{cj} zwischen Neuron *c* und Neuron *j* gegen Null strebt, z. B. durch die Gauß-Funktion

[75] Vgl. Retzko, S. 26.
[76] Vgl. Kohonen [1982b], S. 66 und Ritter et al. [1994], S. 69. Der Schwellenwert des Neurons wurde hier auf den Wert Null gesetzt.
[77] Vgl. Ritter et al. [1994], S. 71.
[78] Vgl. Kohonen [1982b], S. 61.
[79] Vgl. Kohonen [2001], S. 110.
[80] Vgl. Ritter et al. [1994], S. 72.
[81] Vgl. Ritter et al. [1994], S. 73, und Retzko [1996], S. 23.

$$(2.5) \quad h_{cj}(t) = \alpha(t) \cdot \exp\left(-\frac{\Delta_{cj}^{\,2}}{2 \cdot \sigma^2(t)} \right).$$ [82]

Die beschriebene Funktion wird mit Hilfe der Parameter $\alpha(t)$ und $\sigma(t)$ zeitabhängig modelliert, so dass die Zahl der adaptierten Neuronen in der Umgebung des Zentrums und die Stärke der Adaption mit zunehmendem Iterationsfortschritt sinkt.[83] Durch diese Vorgehensweise wird erreicht, dass sich die Neuroneneigenschaften im Verlaufe des Verfahrens immer weniger verändern und schließlich in einer endgültigen Karte resultieren. Das entwickelte Modell der *Self-Organizing Map* kann als deterministisches, einstufiges Netz interpretiert werden, da nur eine Neuronenschicht mit trainierbaren Verbindungen existiert. Die Berechnung der Aktivierung der Neuronen der Eingabe- bzw. Ausgabeschicht kann jeweils synchron erfolgen.[84]

2.3.2 Pseudocode und Darstellung der Funktionsweise des Algorithmus

Die dargestellten Mechanismen werden im Folgenden in einem Pseudocode[85,86] eines Standard-Algorithmus[87] für die SOM zusammengefasst (Abbildung 2.10), anhand der Abbildung 2.11 wird die Funktionsweise des Algorithmus veranschaulicht. Die Interpretation der im Pseudocode verwendeten Symbole zeigt Tabelle 2.2.

Symbol	**Interpretation**
$\alpha(t)$	Parameter der Nachbarschaftsfunktion
$\sigma(t)$	Parameter der Nachbarschaftsfunktion
c	Zentrum
Δ_{cj}	Distanz zwischen Zentrum c und Neuron j
$h_{cj}(t)$	Stärke der Adaption des Gewichtsvektors des Neurons j, wenn Neuron c Zentrum der Iteration ist.
$d_j(\mathbf{x}(t), \mathbf{w_j}(t))$	Distanzmaß
$\mathbf{w}_j(t)$	Gewichtsvektor des Neurons j in Iteration t. Enthält die Gewichte der Verbindungen, die zum Neuron führen.
w_{ij}	i-te Komponente des Gewichtsvektors j
$\mathbf{x}(t)$	Eingabevektor in Iteration t
x_i	i-te Komponente des Eingabevektors
Φ	Menge der Neuronen $i = 1,..., I$ der Eingabeschicht
Ψ	Menge der Neuronen $j = I+1,..., I+J$ der Ausgabeschicht

Tabelle 2.2 : Symbole des Pseudocodes der SOM

[82] Vgl. Kohonen [2001], S. 111. Die Funktion wird in Kapitel 2.3.4 detailliert beschrieben.
[83] Vgl. Kohonen [1990], S. 1467.
[84] Eine Übersicht über mögliche parallele Implementierungen der SOM findet sich in Hämäläinen [2002].
[85] In Anlehnung an Kohonen [2001], S. 109 ff., Ritter et al. [1994], S. 74 und Kasabov [1996], S. 296.
[86] Programmcodes finden sich z. B. in Schmitter [1991], S. 212, und Kinnebrock [1992], S. 158 ff.
[87] Es existieren je nach verwendeter Metrik unterschiedliche algorithmische Beschreibungen für die SOM. Für eine Übersicht vergleiche Kohonen [1993], S. 1148 ff.

INITIALISIERUNG

Setze $t = 0$ und definiere die Parameter der Nachbarschaftsfunktion $\sigma(t = 0)$ und $0 < \alpha(t = 0) < 1$.

Initialisiere den Gewichtsvektor $\mathbf{w}'_j(t = 0) = \begin{pmatrix} w_{1j} & \dots & w_{Ij} \end{pmatrix}$ für jedes Neuron $j \in \Psi$.

REPEAT

Präsentiere dem Netz einen Eingabevektor $\mathbf{x}'(t) = \begin{pmatrix} x_1 & \dots & x_I \end{pmatrix}$.

$\forall j \in \Psi$: Vergleiche Eingabevektor und Neuron j durch Berechnung der Distanz $d_j\left(\mathbf{x}(t), \mathbf{w}_j(t)\right)$ zwischen $\mathbf{x}(t)$ und Gewichtsvektor $\mathbf{w}_j(t)$.

Deklariere das Neuron j^*, das $d_{j^*}^{\min} = \min\left\{d_j\left(\mathbf{x}(t), \mathbf{w}_j(t)\right)\right\}$ erfüllt, als Zentrum c.

$\forall j \in \Psi$: Adaptiere die Gewichtsvektoren gemäß

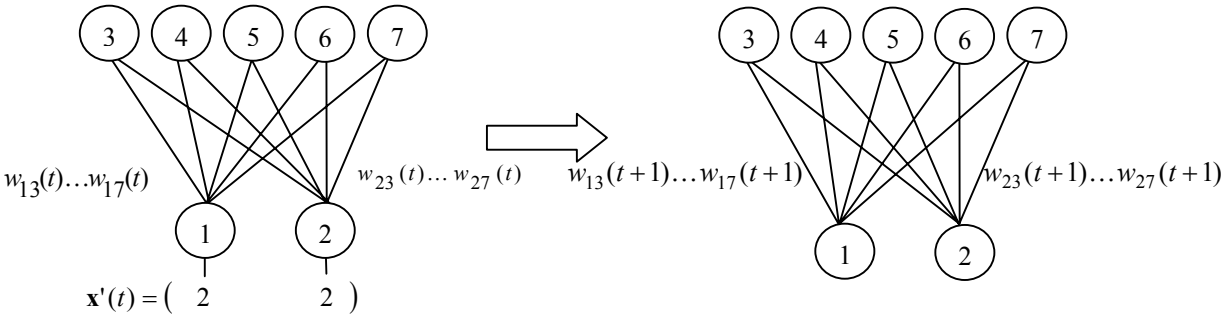
$\mathbf{w}_j(t+1) = \mathbf{w}_j(t) + h_{cj}(t) \cdot \left(\mathbf{x}(t) - \mathbf{w}_j(t)\right) = \left(1 - h_{cj}(t)\right) \cdot \mathbf{w}_j(t) + h_{cj}(t) \cdot \mathbf{x}(t)$,

mit $h_{cj}(t) = \alpha(t) \cdot \exp\left(-\frac{\Delta_{cj}^2}{2 \cdot \sigma^2(t)}\right)$, wobei $h_{cj}(t) \to 0$ für $t \to \infty$.

Aktualisiere t, $\alpha(t)$ und $\sigma(t)$.

UNTIL *Abbruchkriterium* IS TRUE

Abbildung 2.10 : Pseudocode eines Self-Organizing Map Algorithmus



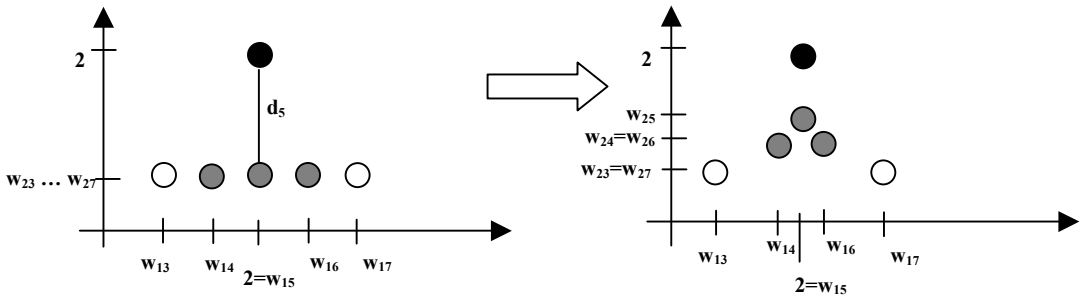


Abbildung 2.11 (a) Abbildung 2.11 (b)

Abbildung 2.11 : Funktionsweise des Self-Organizing Map Algorithmus

Die Eingabeschicht der SOM soll - wie in Abbildung 2.11 skizziert - im Beispiel aus zwei Neuronen bestehen ($\Phi = \{1,2\}$). In diesem Fall können die Gewichtsvektoren der Neuronen der Ausgabeschicht als Positionen der Neuronen in der Ebene (also deren Ortsvektoren) interpretiert werden[88]. Unter der Annahme, dass die Initialisierung des Algorithmus zu einer wie in Abbildung 2.11 (a) dargestellten Anordnung der Gewichtsvektoren der Neuronen der Ausgabeschicht $\Psi = \{3,4,5,6,7\}$ geführt hat, wird der Eingabeschicht der Eingabevektor $\mathbf{x}(t)' = \begin{pmatrix} 2 & 2 \end{pmatrix}$ präsentiert. Für jedes Neuron der Ausgabeschicht wird nun die euklidische Distanz zwischen Eingabevektor und Gewichtsvektor berechnet. Im Beispiel wird das Neuron 5 anschließend als Zentrum *c* deklariert, da dessen Gewichtsvektor die geringste euklidische Distanz $d_{j^*}^{\min} = \min\left\{d_j\left(\mathbf{x}(t), \mathbf{w}_j(t)\right)\right\}$ zum Eingabevektor aufweist. Wird angenommen, dass nur für die direkten Nachbarn des Zentrums (Neuron 4 und Neuron 6) die Stärke der Gewichtsanpassung $h_{cj}(t)$ von Null verschieden ist, so werden nur die Gewichtsvektoren der in der Abbildung grau gefärbten Neuronen 4, 5 und 6 durch Linearkombination des Eingabe- und des jeweiligen Gewichtsvektors in Richtung des Eingabevektors $\mathbf{x}(t)' = \begin{pmatrix} 2 & 2 \end{pmatrix}$ verschoben (s. Abbildung 2.11 (b)). Anschließend werden der als Lernrate bezeichnete Parameter $\alpha(t)$ und der Parameter $\sigma(t)$ der Nachbarschaftsfunktion $h_{cj}(t)$ verringert, der Iterationszähler *t* wird um eine Einheit erhöht. Diese Prozedur wird wiederholt, bis das Abbruchkriterium erfüllt wird.

2.3.3 Übersicht über die Bestandteile der Self-Organizing Map

Tabelle 2.3 fasst die grundsätzliche Ausgestaltung der Bestandteile künstlicher neuronaler Netze für die beiden Schichten der Self-Organizing Map zusammen.

[88] Vgl. Kinnebrock [1992], S. 87.

Komponente	Bestandteile / Symbol	Interpretation / Bemerkung
Neuronen i der Eingabeschicht	*Aktivierungszustand* $a_i(t) = x_i$	Der Aktivierungszustand eines Neurons der Eingabeschicht entspricht der zugehörigen Komponente des Eingabevektors.
	Aktivierungsfunktion $f_i^{act}\left(net_i(t)\right) = x_i$	Der Wert der Aktivierungsfunktion ist ausschließlich von der Eingabe zum Zeitpunkt *t* abhängig und verändert die Eingabeinformation nicht.
	Ausgabefunktion $f_i^{out}\left(a_i(t)\right) = a_i(t) = x_i$	Die Ausgabe entspricht der Aktivierung.
Neuronen j der Ausgabeschicht	*Aktivierungszustand* $a_j(t) = f_j^{act}\left(net_j(t)\right)$	Der Aktivierungszustand eines Neurons *j* entspricht dem aktuellen Wert der zugehörigen Aktivierungsfunktion.
	Aktivierungsfunktion $f_j^{act}\left(net_j(t)\right)$	Je geringer der Wert der Propagierungsfunktion (s. u.), umso größer ist der Wert der Aktivierungsfunktion.
	Ausgabefunktion $f_j^{out}\left(a_j(t)\right) = \begin{cases} 1 & , f_j^{act} > f_h^{act}, \forall h \neq j \\ 0 & , sonst \end{cases}$	Wenn die Aktivierung des Neurons *j* größer als die Aktivierung aller anderen Neuronen der Ausgabeschicht ist, dann ist die Ausgabe des Neurons gleich Eins und das Neuron wird als Zentrum *c* bezeichnet.
Verbindungsnetzwerk	Vollständige Vernetzung der Eingabe- und Ausgabeschicht, d.h. jedes Neuron der Eingabeschicht ist mit jedem Neuron der Ausgabeschicht unidirektional durch *Bottom-Up-Verbindungen* verbunden.	Die *Bottom-Up-Verbindungen* dienen zur Übertragung der Eingangsinformationen an alle Neuronen der Ausgabeschicht.
	Verbindungsfunktion $f_{ij}\left(o_i(t)\right) = o_i(t) = x_i$	Die Verbindungsfunktion gibt die Ausgabe eines Neurons der Eingabeschicht unverändert weiter.

	Propagierungsfunktion $net_j(t) = f_j^{prop}\left(f_{1j},...,f_{ij},w_{1j},...,w_{ij},b_j\right)$ $= f_j^{prop}\left(x_1,...,x_i,w_{1j},...,w_{ij},b_j\right)$ $= d_j\left(\mathbf{x}(t),\mathbf{w}_j(t),b_j\right)$	Die Propagierungsfunktion berechnet die Eingabe des Neurons *j* in Abhängigkeit von den Komponenten der Eingabevektoren und gegebenenfalls dem Schwellenwert des Neurons. Die Funktion berechnet die Distanz zwischen Eingabe- und Gewichtsvektor.
Lernregel	*Wettbewerbslernen,* *Modifikation der Gewichte der Verbindungen.*	Die Gewichte der Verbindungen stellen das mit zunehmendem Iterationsfortschritt erlernte „Wissen“ des Netzes dar.

Tabelle 2.3 : Bestandteile der SOM

Die Eigenschaften der Neuronen der Eingabeschicht zeigen, dass bei der Programmierung des Algorithmus keine explizite Modellierung der Funktionen vorgenommen werden muss, da durch diese keine Veränderung der Eingabeinformationen vorgenommen werden. Aus demselben Grund müssen auch die Verbindungsfunktionen nicht explizit modelliert werden. Der Schwellenwert fließt abweichend von den bisherigen Ausführungen in die Propagierungsfunktion ein, da dieser im Folgenden häufig zur Bestimmung der Ähnlichkeit eines Neurons zum präsentierten Eingabevektor herangezogen wird.

2.3.4 Spezielle Ausprägungen und Probleme bei der Anwendung der Self-Organizing Map auf Routenplanungsprobleme

Zunächst wird in diesem Abschnitt in Tabelle 2.4 eine vereinfachte Darstellung des in Abbildung 2.10 vorgestellten Pseudocodes vorgenommen. Anschließend wird für die einzelnen Schritte des vereinfachten Pseudocodes allgemein dargestellt, wie diese häufig bei der Anwendung der Self-Organizing Map auf die in Kapitel 3 und Kapitel 4 untersuchten Probleme umgesetzt werden. Darüber hinaus wird erläutert, welche Probleme bei der Anwendung auftreten können.

Schritt	Bezeichnung
1.	Initialisierung.
2.	Präsentation eines Eingabevektors.
3.	Berechnung der Aktivierung der Neuronen durch Vergleich des Eingabevektors mit den Gewichtsvektoren der Neuronen der Ausgabeschicht anhand einer Distanzfunktion.
4.	Bestimmung des Zentrums.
5.	Anpassung der Gewichtsvektoren der Neuronen der Nachbarschaft und des Zentrums.
6.	Anpassung der Parameter.
7.	Überprüfung des Abbruchkriteriums.

Tabelle 2.4 : Vereinfachte Darstellung des SOM-Algorithmus

Schritt 1. Initialisierung

Neben den Parametern der Nachbarschaftsfunktion[89] sind in diesem Schritt auch die Gewichtsvektoren zu initialisieren. Tabelle 2.5 stellt einige Möglichkeiten vor und fasst mögliche Auswirkungen der Gewichtsinitialisierung zusammen[90]:

Gewichtsinitialisierung	Mögliche negative Auswirkungen
Äquidistante Anordnung der Gewichte entlang eines Kreises oder einer Kette.	-
Initialisierung gemäß der Merkmalsausprägung zufällig ausgewählter Datensätze.	Überrepräsentation eines Clusters innerhalb der Daten.
Initialisierung gemäß einer (zufälligen) Konvexkombination der minimalen und maximalen Merkmalsausprägungen.	-

Tabelle 2.5 : Mögliche Vorgehensweisen der Gewichtsinitialisierung

Wie in Kapitel 3 gezeigt wird, erfolgt die Initialisierung der Gewichte bei der Anwendung auf die in dieser Arbeit vorgestellten Optimierungsprobleme meist problembezogen, beispielsweise wird bei der Anwendung auf das Traveling Salesman Problem eine Anordnung entlang eines Kreises vorgenommen. Allerdings können auch in diesem

[89] Die Nachbarschaftsfunktionen werden im Zusammenhang mit den Ausführungen zu Schritt 5 diskutiert.
[90] Vgl. Nour und Madey [1996], S. 433.

Fall unterschiedliche Gewichtsinitialisierungen bei gleicher Präsentationsreihenfolge der Eingabevektoren zu unterschiedlichen Anpassungsprozessen und somit möglicherweise auch zu unterschiedlichen Orientierungen der resultierenden Karte führen, die unter Umständen „fehlerhafte“ topologische Abbildungen darstellen.

Schritt 2. Präsentation der Eingabevektoren

Unterschiedliche Präsentationsreihenfolgen bewirken unterschiedliche Wettbewerbsprozesse in der Ausgabeschicht und eine voneinander abweichende Entwicklung der Gewichtsvektoren, die teilweise ebenfalls zu topologisch fehlerhaften Abbildungen führen können[91]. Wenn große Datenmengen vorliegen, können zudem Eingabevektoren, die am Ende des Lernvorgangs präsentiert werden, aufgrund der zeitlich angepassten Parameter der Nachbarschaftsfunktion keine starke Veränderung der Gewichtsvektoren hervorrufen und werden somit unter Umständen nicht ausreichend berücksichtigt. Als Folge ist ebenfalls eine fehlerhafte Abbildung der Eingabevektoren in Betracht zu ziehen. Einen Ansatz zur Problemlösung stellt die mehrfache Präsentation der Eingabedaten in sogenannten Präsentationszyklen mit (zufälliger) Variation der Reihenfolge der Eingabevektoren dar. Diese Vorgehensweise wird häufig bei den im dritten und vierten Kapitel untersuchten Anwendungen der SOM, bei denen die Eingabevektoren die Position von Knoten in der Ebene beschreiben, umgesetzt.

Schritt 3. Berechnung der Aktivierung der Neuronen durch Vergleich des Eingabevektors mit den Gewichtsvektoren der Neuronen der Ausgabeschicht anhand einer Distanzfunktion

Die Distanzfunktion dient als Maß für die Wettbewerbsfähigkeit der Neuronen der Ausgabeschicht. In Tabelle 2.6 werden vier Funktionen aufgeführt, die bei der Anwendung der SOM auf Routenplanungsprobleme häufig verwendet werden.[92]

[91] Vgl. Nour und Madey [1996], S. 434.
[92] Eine Übersicht über verschiedene Funktionen zur Beurteilung der Ähnlichkeit von Mustern (bzw. der Ähnlichkeit von Eingabevektoren und Gewichtsvektoren) findet sich beispielsweise in Kohonen [2001], S. 17 ff.

Nr.	Berechnungsvorschrift	Interpretation	Anwendung
(2.6)	$d_j = d(\mathbf{x}(t), \mathbf{w}_j(t)) = \lVert \mathbf{x}(t) - \mathbf{w_j}(t) \rVert = \sqrt{\sum_{i=1}^{I} (x_i - w_{ij})^2}$	Euklidische Distanz.	Entfernungsmessung Luftlinie.
(2.7)	$d_j = d(\mathbf{x}(t), \mathbf{w}_j(t)) = \lVert \mathbf{x}(t) - \mathbf{w_j}(t) \rVert^2 = \sum_{i=1}^{I} (x_i - w_{ij})^2$	Quadrierte euklidische Distanz.[93]	Entfernungsmessung Luftlinie.
(2.8)	$d_j = d(\mathbf{x}(t), \mathbf{w}_j(t)) = F(.) \cdot \lVert \mathbf{x}(t) - \mathbf{w_j}(t) \rVert^2 = F(.) \cdot \sum_{i=1}^{I} (x_i - w_{ij})^2$	Künstliche Erhöhung / Verminderung der quadrierten euklidischen Distanz.	Berücksichtigung von Restriktionen.
(2.9)	$d_j = d(\mathbf{x}(t), \mathbf{w}_j(t)) = \lVert \mathbf{x}(t) - \mathbf{w_j}(t) \rVert^2 + \gamma \cdot b_j = \gamma \cdot b_j + \sum_{i=1}^{I} (x_i - w_{ij})^2$	Künstliche Erhöhung / Verminderung der quadrierten euklidischen Distanz.	Berücksichtigung von Restriktionen.

Tabelle 2.6 : Für Routenplanungsprobleme häufig verwendete Distanzfunktionen

Wie Tabelle 2.6 entnommen werden kann, werden hierbei überwiegend Distanzfunktionen verwendet, die auf der euklidischen Distanz zwischen Eingabevektor und Gewichtsvektor beruhen. Oft werden dabei die Modifikationen gemäß *(2.8)* und *(2.9)* vorgenommen, um verschiedenen problemspezifischen Restriktionen Rechnung zu tragen:

Distanzfunktion (2.8)

Bei dieser Variante der Distanzfunktion wird die euklidische Distanz mit einem Faktor $F(.)$ multipliziert, der die Einhaltung spezieller Restriktionen unterstützen soll. Wird die SOM beispielsweise zur Lösung eines Tourenplanungsproblems eingesetzt, so müssen Kapazitätsrestriktionen für die Fahrzeuge berücksichtigt werden. Der Faktor $F(.)$ könnte nun beispielsweise den gewichteten Auslastungsgrad eines Fahrzeugs bei der aktuell gültigen Zuordnung der Nachfrage zu den Fahrzeugen widerspiegeln, mit dem die euklidische Distanz zwischen Eingabe- und Gewichtsvektor der dem Fahrzeug eindeutig zugeordneten Neuronen der Ausgabeschicht multipliziert wird. Je größer der Auslastungsgrad eines Fahrzeuges, umso größer wird folglich die resultierende gewichtete euklidische Distanz. Durch dieses Vorgehen könnten die einem zweiten Fahrzeug zugeordneten Neuronen, deren euklidische Distanz zwar größer ist als die des betrachte-

[93] Die Verwendung der quadrierten euklidischen Distanz erspart gegenüber Gleichung *(2.6)* die Operation des Radizierens. Die Wahl des Zentrums wird hierdurch nicht verändert.

ten Neurons, trotzdem als Zentrum ausgewählt werden, wenn der Auslastungsgrad des zweiten Fahrzeuges geringer ist.

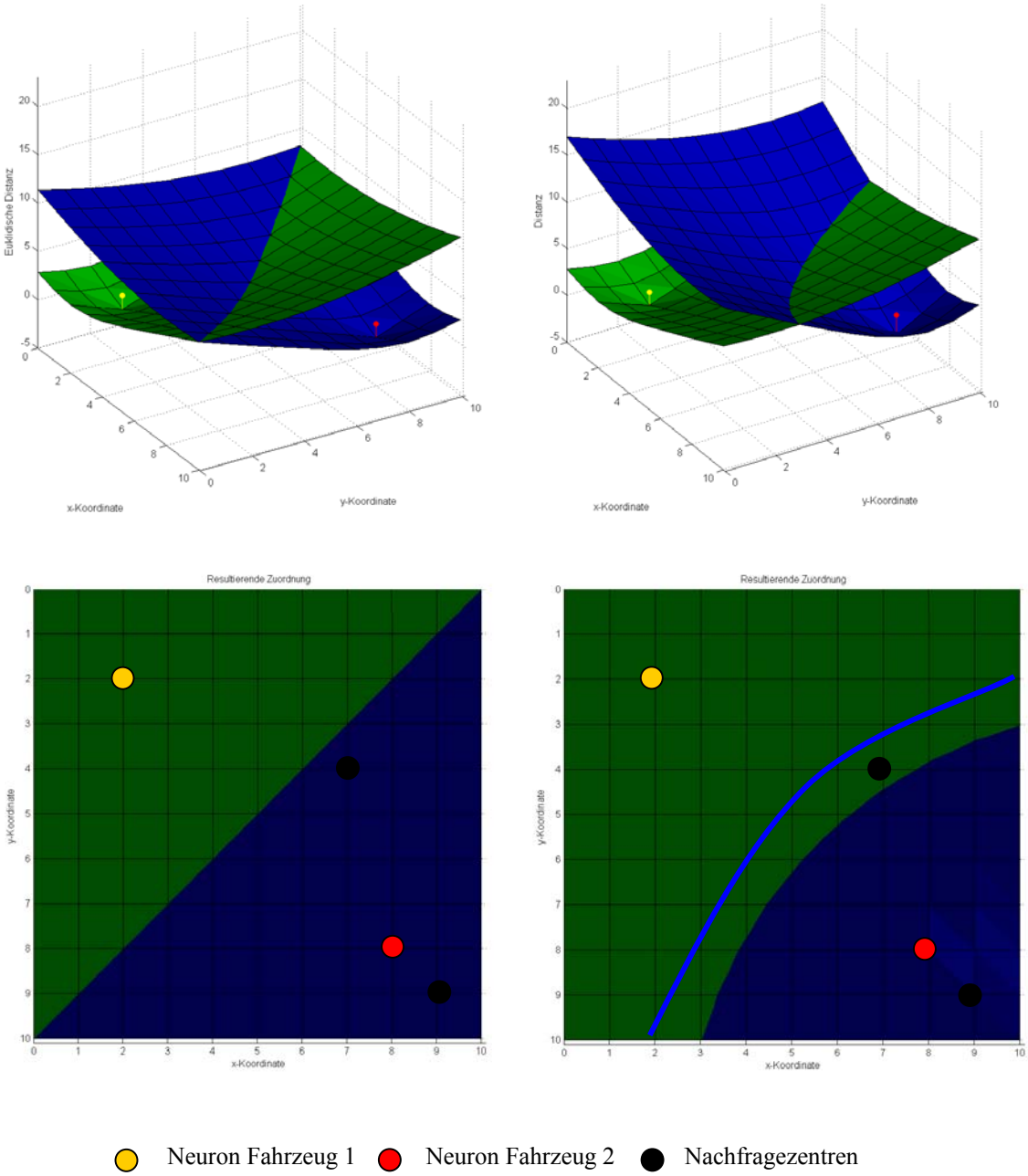
Distanzfunktion (2.9)

Bei dieser Variante wird der gewichtete Schwellenwert eines Neurons zur euklidischen Distanz addiert, um die Einhaltung der Restriktionen zu unterstützen. Der Schwellenwert könnte für das oben beschriebene Problem ebenfalls als Auslastungsgrad der Fahrzeuge modelliert werden. Je größer der Wert des mit dem Faktor γ gewichteten Auslastungsgrads, umso stärker wird folglich die euklidische Distanz erhöht, so dass Neuronen eines zweiten Fahrzeugs mit geringerem Auslastungsgrad auch bei größerer euklidischer Distanz zum Eingabevektor als Zentrum ausgewählt werden könnten.

Die Verwendung der beschriebenen Funktionen garantiert nicht zwangsläufig die Einhaltung der Restriktionen. Wird beispielsweise die Gewichtung des Schwellenwertes oder der Faktor in *(2.8)* für die bearbeitete Probleminstanz zu klein gewählt, so ergibt sich unter Umständen keine Veränderung bei der Wahl des Zentrums.
Abbildung 2.12 verdeutlicht diese Problematik. In Abbildung 2.12 (a) wird die euklidische Entfernung für zwei Neuronen zu den Punkten in der *x*-*y*-Ebene dargestellt und das resultierende Voronoi-Diagramm[94] angedeutet. In Abbildung 2.12 (b) werden die Distanzen für die zwei Neuronen zu den Punkten der *x*-*y*-Ebene dargestellt, wenn die euklidische Distanz des zweiten Neurons mit dem gewichteten Auslastungsgrad (hier 1,5) multipliziert wird. Wie ersichtlich, kann dieses Vorgehen als Verschiebung der Grenze des Voronoi-Diagramms interpretiert werden. Wird die euklidische Distanz verwendet, so werden beide Nachfragezentren dem Neuron des Fahrzeugs 2 zugeordnet, Fahrzeug 1 wird kein Nachfragezentrum zugeordnet. Geht nun zusätzlich der Faktor in die Berechnungen ein, so wird ein Nachfragezentrum dem Neuron des Fahrzeugs 1 zugeordnet. Ist allerdings die Gewichtung des Auslastungsgrads zu gering, so wird die Grenze des Voronoi-Diagramms - angedeutet durch die hellblaue Linie - nicht ausreichend stark verschoben, so dass keine Veränderung bei der Wahl des Zentrums beziehungsweise bei der Zuordnung der Nachfragezentren stattfindet. Die Gewichtung ist daher für verschie-

[94] Ein Voronoi-Diagramm unterteilt einen Raum in Regionen, die durch Begrenzungslinien voneinander getrennt werden. Diese Begrenzungslinien werden dabei so in den Raum gelegt, dass jede Region einen vorgegebenen Referenzvektor (hier die Ortsvektoren der Neuronen) enthält, der die kürzeste (euklidische) Entfernung zu jedem beliebigen Vektor aus der Region aufweist, vgl. hierzu Kohonen [2001], S. 59. Ein Neuron wird also dann zum Zentrum, wenn es die minimale Distanz zum Eingabevektor aufweist, d.h. dieser innerhalb der Region liegt, für die das entsprechende Neuron Referenzvektor ist.

dene Probleminstanzen unterschiedlich einzustellen. Eine allgemeine Vorschrift ist hierzu allerdings nicht verfügbar, so dass Testläufe mit unterschiedlichen Gewichtungen durchgeführt werden müssen.



(a) Voronoi-Diagramm für euklidische Distanzen

(b) „Voronoi-Diagramm“ für Distanzfunktion der Form (2.8) und (2.9)

Abbildung 2.12 : Auswirkung der Modifikationen der Distanzfunktion

Die Verwendung der euklidischen Distanz kann auch ohne die beschriebenen Modifikationen zu topologischen Defekten führen, die anhand der Abbildung 2.13 verdeutlich werden können.

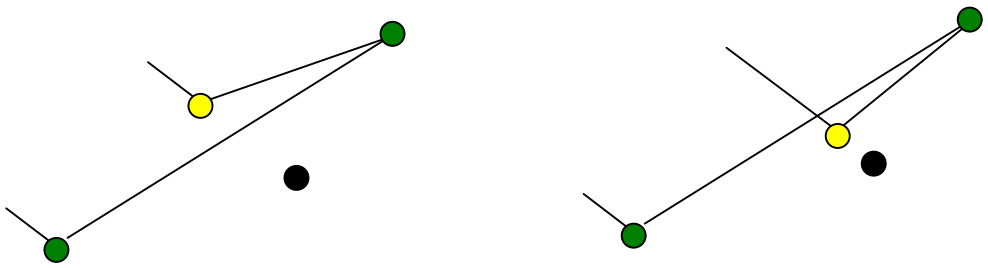


Abbildung 2.13: Entstehung topologischer Defekte[95]

Wird der SOM der schwarz markierte Eingabevektor präsentiert, so wird das gelb markierte Neuron als Zentrum ausgewählt, da dieses die minimale Entfernung zum Eingabevektor aufweist. Durch Adaption des Gewichtsvektors kommt es zu einer Überschneidung, die beispielsweise zu unnötig hohen Zielfunktionswerten bei Rundreise- bzw. Tourenplanungsproblemen führen kann.

Schritt 4. Bestimmung des Zentrums

Als Zentrum wird in der Regel das Neuron mit der geringsten Distanz zum präsentierten Eingabevektor $\mathbf{x}(t)$ bezeichnet. Die Methode zur Suche nach dem Zentrum beeinflusst die Geschwindigkeit des Algorithmus, was zur Entwicklung verschiedener Suchtechniken geführt hat, welche das Ziel verfolgen, die benötigte Rechenzeit zur Bestimmung des Zentrums zu vermindern.[96]

Schritt 5. Anpassung der Gewichtsvektoren der Neuronen der Nachbarschaft und des Zentrums

Die Anpassung der Gewichtsvektoren erfolgt meist für das Zentrum und Neuronen, die zur durch die Nachbarschaftsfunktion $h_{cj}(t)$ definierten Nachbarschaft des Zentrums zählen. Diese Funktionen legen fest, welche Neuronen j zur Nachbarschaft zählen und wie stark deren Gewichtsvektoren adaptiert werden. Die verwendete Nachbarschaftsfunktion muss zur Gewährleistung der Konvergenz die Eigenschaft $h_{cj}(t) \to 0$ für

[95] In Anlehnung an Plebe und Anile [2001], S. 447.
[96] Vgl. hierzu Nour und Madey [1996], S. 435 f., und Amin [1994].

$t \to \infty$ erfüllen.[97] Die Bestimmung der Größe der Nachbarschaft ist ein kritischer Parameter für die Qualität der Ergebnisse. Die in Tabelle 2.7 zusammengestellten negativen Auswirkungen, die bei einer unangemessenen Wahl der Nachbarschaftsfunktion resultieren können, sind zu berücksichtigen:

Größe der Nachbarschaft / Stärke der lateralen Interaktion	**Mögliche negative Auswirkung**	**Erläuterung**
zu klein	Unzureichende globale Anordnung der Gewichtsvektoren[98]	Wird beispielsweise nur der Gewichtsvektor des Gewinner-Neurons angepasst, so ist es möglich, dass einige Neuronen der Ausgabeschicht den Wettbewerb niemals gewinnen und sich deren Gewichtsvektor nicht verändert. Eine zu geringe Stärke der lateralen Interaktion kann dazu führen, dass einige wenige Neuronen auf einen zu großen Bereich des Eingaberaums reagieren und die resultierende Karte keine topologische Abbildung generiert.[99]
zu groß	Lernzeiten zu hoch	In großen Karten muss für eine zu hohe Anzahl an Neuronen eine Gewichtsanpassung vorgenommen werden.
	Kollabieren der Karte[100]	Die Gewichtsvektoren aller Neuronen nehmen die gleichen oder sehr ähnliche Werte an.

Tabelle 2.7 : Einfluss der Nachbarschaftsfunktion

In Tabelle 2.8 werden zwei Nachbarschaftsfunktionen aufgeführt, die häufig zur Lösung der in dieser Arbeit untersuchten Optimierungsprobleme verwendet werden:

Nr.	**Formale Beschreibung**
(2.10)	$h_{cj}(t) = \begin{cases} \alpha(t) \cdot \exp\left(-\frac{\Delta_{cj}^{\,2}}{2 \cdot \sigma^2(t)} \right) & , \Delta_{cj} < L \\ 0 & , sonst \end{cases}$
(2.11)	$h_{cj}(t) = \begin{cases} \left(1 - \frac{\Delta_{cj}}{L} \right)^{\beta(t)} & , \Delta_{cj} < L \\ 0 & , sonst \end{cases}$

Tabelle 2.8 : Nachbarschaftsfunktionen

Beschreibung der Funktion (2.10)

Der Parameter $\alpha(t)$ dieser Gauß-Funktion sinkt in den meisten Ansätzen mit zunehmender Iterationszahl *t*. Der ebenfalls mit zunehmender Iterationszahl sinkende Parameter $\sigma(t)$ steuert die räumliche Ausdehnung (Breite) der resultierenden Funktion[101]. Beide Parameter müssen jeweils auf die entsprechende Probleminstanz und das verwendete

[97] Vgl. Kohonen [2001], S. 111.
[98] Vgl. Kohonen [1982b], S. 68 und Kohonen [1990], S. 1469.
[99] Vgl. Kohonen [1982b], S. 68.
[100] Vgl. Kohonen [1982b], S. 68.
[101] Vgl. Kinnebrock [1992], S. 80.

KNN angepasst werden. Der Parameter Δ_{cj} beschreibt die Entfernung eines Neurons *j* vom Zentrum *c*. Er wird meist als kardinale Distanz zum Zentrum definiert, welche die minimale Anzahl an Verbindungen zwischen *c* und *j* beschreibt, deren Wert sich aus der durch das Neuronengitter definierten Nachbarschaftsstruktur ergibt (s. Abbildung 2.14).

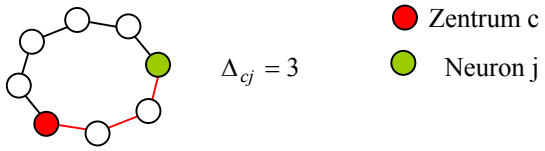


Abbildung 2.14 : Entfernung zwischen Neuronen

Abbildung 2.15 stellt beispielhaft den rot markierten Verlauf der Funktion (2.10) dar. Es wird deutlich, dass mit zunehmenden Iterationsfortschritt immer weniger Neuronen in der Umgebung des Zentrums immer geringer adaptiert werden, bis schließlich in der letzten Iteration t_{max} kein Neuron mehr in Richtung des Eingabevektors verschoben wird[102]. Um eine Beschleunigung des Algorithmus zu erreichen, kann zusätzlich eine Obergrenze *L* für die Distanz zwischen *c* und *j* definiert werden, so dass die Anzahl der zu adaptierenden Neuronen auf ein vorgegebenes Maximum beschränkt wird.[103]

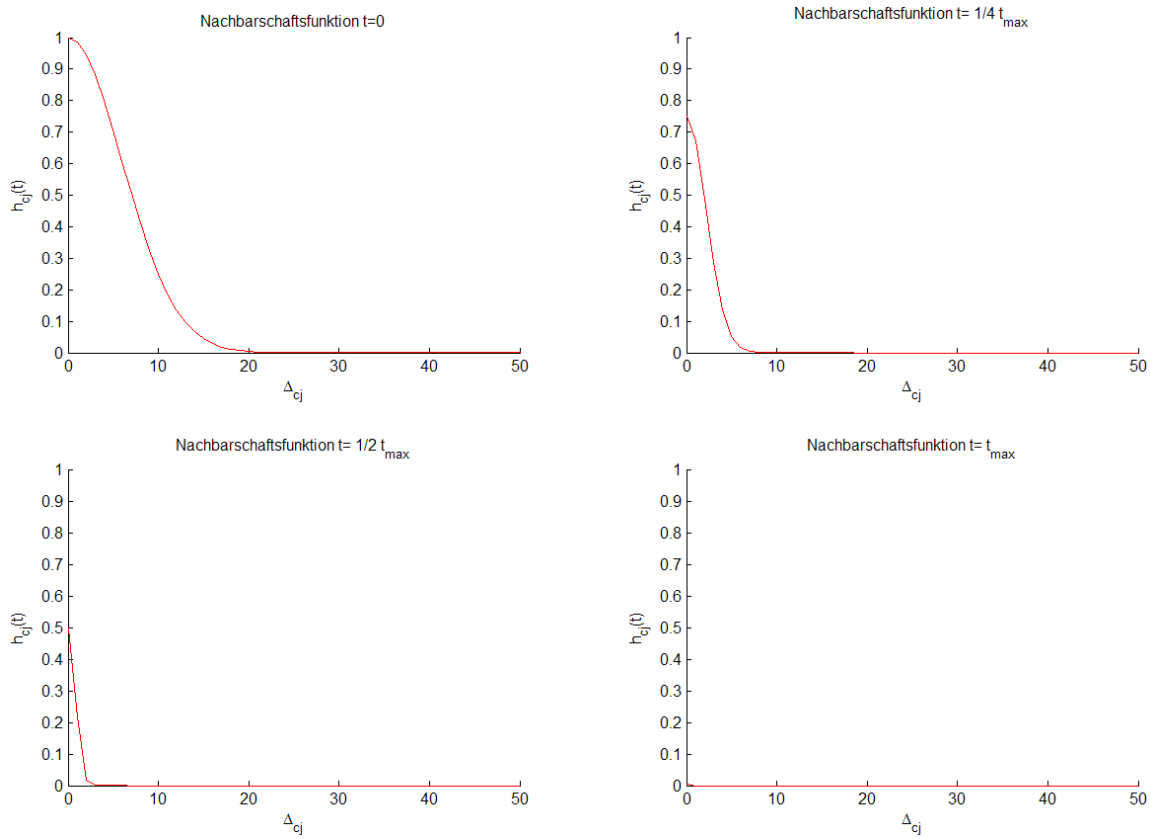


Abbildung 2.15 : Beispielhafter Verlauf der Funktion *(2.10)* bei zunehmendem Iterationszähler

[102] S. Abbildung 2.15, der Parameter $h_{cj}(t_{max})$ weist praktisch für alle Δ_{cj} den Wert Null auf.

[103] Vgl. Retzko [1996], S. 84 ff.

Beschreibung der Funktion (2.11) für den Bereich $\Delta_{cj} < L$

Wie auch bei der Gauß-Funktion, beschreibt der Parameter Δ_{cj} dieser Funktion häufig die Distanz zwischen Neuron j und Zentrum c im Neuronengitter. Der Parameter $\beta(t)$ wird meist zeitabhängig modelliert, um eine Verringerung der Stärke der Adaption der Gewichtsvektoren mit zunehmendem Iterationsfortschritt zu erreichen, d.h., der Wert des Parameters steigt mit zunehmender Iterationszahl[104]. Die Anzahl der zu adaptierenden Gewichtsvektoren kann ebenfalls mit Hilfe des Parameters L auf ein Maximum beschränkt werden, um eine Beschleunigung des Algorithmus zu erreichen. Der Verlauf der Funktion für den Bereich $\Delta_{cj} < L$ wird in Abbildung 2.16 dargestellt.

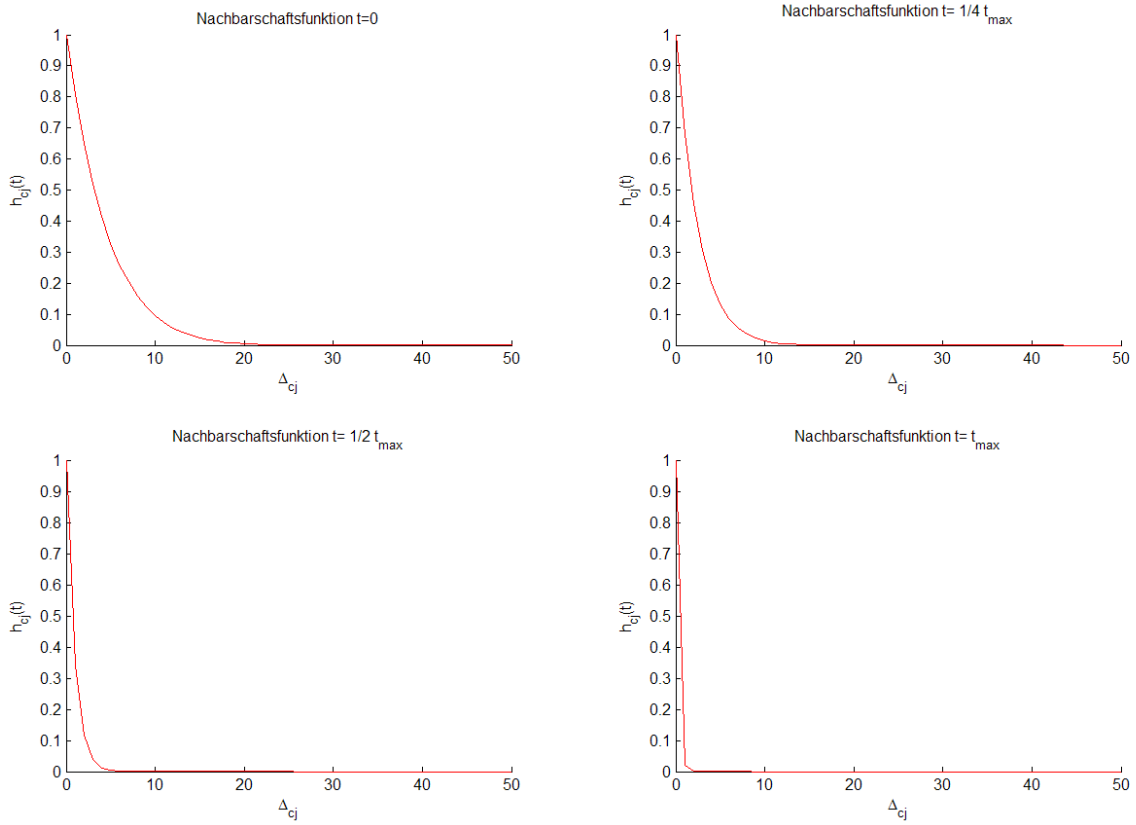


Abbildung 2.16 : Beispielhafter Verlauf der Funktion *(2.11)* bei zunehmendem Iterationszähler

Adaptionsvorschriften

Die Adaption der Gewichte wird bei den in dieser Arbeit vorgestellten Varianten der SOM meist gemäß der in Tabelle 2.9 dargestellten Vorschriften durchgeführt. Neben diesen Vorschriften sind in der Literatur einige Spezialfälle zu finden, die jedoch nur selten verwendet werden.

[104] Burke [1992] verwendet zu Beginn des Algorithmus bspw. $\beta = 10$ und erhöht den Wert jeweils nach einer vorgegebenen Iterationszahl auf das 1,1-fache des Ausgangswertes.

	Adaptionsvorschrift
(2.12)	$\mathbf{w}_j(t+1) = \mathbf{w}_j(t) + h_{cj}(t) \cdot \left(\mathbf{x}(t) - \mathbf{w}_j(t)\right) = \left(1 - h_{cj}(t)\right) \cdot \mathbf{w}_j(t) + h_{cj}(t) \cdot \mathbf{x}(t)$
(2.13)	$\mathbf{w}_j(t+1) = \underbrace{\mathbf{w}_j(t) + h_{cj}(t) \cdot \left(\mathbf{x}(t) - \mathbf{w}_j(t)\right)}_{F_\alpha} + \underbrace{\varphi(t) \cdot \left(\left(\mathbf{w}_{j+1}(t) - \mathbf{w}_j(t)\right) + \left(\mathbf{w}_{j-1}(t) - \mathbf{w}_j(t)\right)\right)}_{F_\beta}$

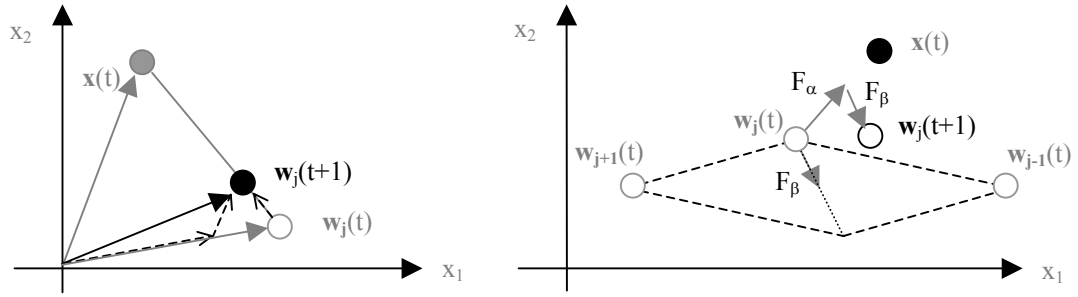
Tabelle 2.9 : Für Routenplanungsprobleme häufig verwendete Adaptionsvorschriften

Adaptionsvorschrift (2.12)

Der neue Gewichtsvektor eines Neurons ergibt sich hier als Linearkombination aus dem bisher gültigen Gewichtsvektor und dem Eingabevektor (vgl. Abbildung 2.17 (a)).

Adaptionsvorschrift (2.13)

Diese Adaptionsvorschrift beruht auf der Modellierung zweier Kräfte, die auf den Gewichtsvektor wirken. Die Bewegung eines Neurons *j* wird durch die am Neuron angreifenden Kräfte F_α und F_β verursacht,[105] wobei die Parameter $h_{cj}(t)$ und $\varphi(t)$ den Einfluss der Kräfte steuern. Die Richtung und der Betrag der Kraft F_α bestimmen sich wie auch in Gleichung *(2.12)* durch Linearkombination aus dem alten Gewichtsvektor und dem Eingabevektor, Richtung und Betrag der Kraft F_β werden durch die Gewichtsvektoren des jeweiligen Neurons und dessen direkt benachbarter Neuronen (s. Abbildung 2.17 (b)) bestimmt. Durch diesen Ansatz wird eine rückstellende Kraft modelliert, die der Entfernung des Neurons von seinen Nachbarn entgegenwirkt. Die Idee zu dieser Adaptionsvorschrift wurde ursprünglich in dem von DURBIN und WILLSHAW verwendeten Elastic Net-Ansatz verwendet[106].



(a) Linearkombination aus Eingabe- und Gewichtsvektor

(b) Darstellung der an einem Neuron angreifenden Kräfte

Abbildung 2.17 : Darstellung der Wirkung der Adaptionsvorschriften *(2.12)* und *(2.13)*

[105] Vgl. Potvin [1993], S. 340.
[106] Vgl. Durbin und Willshaw [1987].

Schritt 6. Anpassung der Parameter

In diesem Schritt werden die Parameter der Nachbarschaftsfunktion und der Iterationszähler angepasst. Da die Parameter häufig unterschiedlich modelliert werden, wird eine detaillierte Beschreibung jeweils in Kapitel drei vorgenommen.

Schritt 7. Abbruchkriterien

Tabelle 2.10 fasst einige Abbruchkriterien zusammen, die auch in Kombination miteinander angewendet werden[107]. Die zeitabhängigen Parameter des Algorithmus werden überwiegend in Abhängigkeit von der Iterationszahl modelliert, damit die Nachbarschaftsfunktion den erwünschten Verlauf aufweist. Demzufolge wird das Erreichen einer vorgegebenen Iterationszahl sehr oft als Abbruchkriterium verwendet.

Abbruchkriterium	**Mögliche negative Auswirkungen**
Erreichen einer vorgegebenen maximalen Zeitgrenze *T*.	Keine zufriedenstellende topologische Anordnung der Ausgabeschicht aufgrund vorzeitigen Abbruchs oder hohe Rechenzeiten bei zu hoher Zeitgrenze.[108]
Erreichen einer vorgegebenen Anzahl an Iterationen $t_{\max}$ beziehungsweise Präsentationszyklen $t_{p\max}$.	Keine zufriedenstellende topologische Anordnung der Ausgabeschicht aufgrund vorzeitigen Abbruchs oder „Überlernen“ und hohe Rechenzeiten bei zu großer Iterationszahl.
Wahl der Zentren für die Eingabevektoren verändert sich nicht mehr.	Hohe Rechenzeiten.
Gewichtsvektoren der Neuronen befinden sich in akzeptabler Entfernung von den Eingabevektoren.[109]	Hohe Rechenzeiten.

Tabelle 2.10 : Abbruchkriterien

[107] Vgl. beispielsweise Burke und Damany [1992], S. 262.
[108] Vgl. Nour und Madey [1996], S. 433.
[109] Vgl. Modares et al. [1999], S. 598.

2.3.5 Konvergenzverhalten des Algorithmus

Obwohl das Verfahren aus relativ einfachen Mechanismen besteht, erweist sich eine umfassende mathematische Analyse als schwierig. Bisher sind Konvergenzbeweise nur für wenige spezielle Fälle bekannt[110], für den Fall diskret verteilter Eingabedaten (beispielsweise der unveränderlichen Koordinaten der auf einer Rundreise zu besuchenden Orte) konnten RITTER ET AL. eine Beschreibung des Konvergenzverhaltens entwickeln[111]. Aufgrund des Umfanges der mathematischen Analyse sollen hier nur wesentliche Merkmale des Verhaltens, das für beliebige Topologien gilt[112], kurz skizziert werden, da diesen Hinweise zur Parametrisierung des Algorithmus entnommen werden können.

Der Algorithmus kann formal durch einen Markov-Prozess beschrieben werden, dessen Zustände durch die Gewichtsvektoren der Neuronen beschrieben werden und dessen Übergänge durch Präsentation der Eingabevektoren ausgelöst werden.[113] RITTER ET AL. zeigen, dass sich der vom Zustand des Verbindungsnetzwerks $\mathbf{W}$ abhängige Erwartungswert für die Änderung des Gewichtsvektors $\Delta\mathbf{w}_j = \mathbf{w}_j(t+1) - \mathbf{w}_j(t)$ des Neurons $j \in \Psi$ in einem Lernschritt zu

$$(2.14)\quad \mathrm{E}\left(\Delta\mathbf{w}_j \middle| \mathbf{W}\right) = \alpha \cdot \sum_{c=I+1}^{I+J} h_{cj} \cdot \sum_{u \in \Omega} p_u \cdot \left(\mathbf{x}_u - \mathbf{w}_j(t)\right) = -\alpha \cdot \nabla_{\mathbf{w}_j(t)} V(\mathbf{W})$$

ergibt[114]. Dabei beschreibt p_u die Wahrscheinlichkeit, dass der Eingabevektor dem Vektor $\mathbf{x}_u$ aus der Menge $V = \{\mathbf{x}_1, ..., \mathbf{x}_U\}$ entspricht, wobei $\sum_{u=1}^{U} p_u = 1$ gilt. Die Menge

$$(2.15)\quad \Omega = \left\{u \middle|\ \mathbf{x}_u \in F_c(\mathbf{W})\right\}$$

enthält die Indizes der Eingabevektoren, für die Neuron *c* am stärksten erregt wird, also derjenigen Eingabevektoren, die in der vom Netzwerkzustand abhängigen Menge

$$(2.16)\quad F_c(\mathbf{W}) = \{\mathbf{x}_u \in V \left| \ \|\mathbf{x}_u - \mathbf{w}_c(t)\| \le \|\mathbf{x}_u - \mathbf{w}_j(t)\| \ \forall j \in \Psi\right\}$$

enthalten sind.

Einem Zustand $\mathbf{W}$ des Verbindungsnetzwerks wird durch

---

[110] Eine aktuelle Liste der bekannten Konvergenzbeweise findet sich in Kohonen [2002], S. 7.

[111] Vgl. Ritter et al. [1994], S. 107 ff. und S. 251 ff. Weitere Untersuchungen zum Konvergenzverhalten finden sich in Ritter und Schulten [1988b], Kohonen [1991] und Kohonen [2001].

[112] Vgl. Ritter et al. [1994], S. 107.

[113] Vgl. Ritter und Schulten [1988b], S. 59.

[114] Die Lernrate α wird hier vor dem Summenzeichen notiert, ist also nicht mehr in h_{cj} enthalten, vgl. Ritter et al. [1994], S. 108.

$$(2.17)\quad V(\mathbf{W}) = \frac{1}{2} \cdot \sum_{c=I+1}^{I+J} \sum_{j=I+1}^{I+J} \left(h_{cj} \cdot \sum_{u \in \Omega} p_u \cdot \left(\mathbf{x}_u - \mathbf{w}_j(t) \right)^2 \right)$$

ein Potentialfunktionswert zugewiesen.[115] Die durch die Lernschritte hervorgerufene Veränderung des Netzwerkzustandes führt im Mittel zu der Abnahme

$$(2.18)\quad \mathrm{E}(\Delta V(\mathbf{W})) = -\alpha \cdot \sum_{j=I+1}^{I+J} \left\| \nabla_{\mathbf{w}_j(t)} V(\mathbf{W}) \right\|^2$$

des Potentialfunktionswertes[116], ein einzelner Lernschritt kann allerdings auch zur Zunahme von $V(\mathbf{W})$ führen, so dass für $\alpha > 0$ ein lokales Minimum verlassen werden kann[117]. Die Anzahl solcher lokaler Minima von $V(\mathbf{W})$ hängt von h_{cj} ab: Für konstante $h_{cj} = h$ [118] wird $V(\mathbf{W})$ zu

$$(2.19)\quad V(\mathbf{W}) = \frac{h}{2} \cdot \sum_{u=1}^{U} \sum_{j=I+1}^{I+J} p_u \cdot \left(\mathbf{x}_u - \mathbf{w}_j(t) \right)^2$$

mit einem Minimum bei

$$(2.20)\quad \mathbf{w}_j(t) = \sum_{u=1}^{U} p_u \cdot \mathbf{x}_u \ .$$

Bei kürzerer Reichweite der Funktion h_{cj} treten weitere lokale Minima hinzu.

Die Minimierung von $V(\mathbf{W})$ kann als Umsetzung der Aufgabe interpretiert werden, eine Abbildung der Eingabevektoren auf die Neuronen der Ausgabeschicht vorzunehmen, so dass die durch die Funktion h_{cj} definierten Nachbarschaftsrelationen der Neuronen der Ausgabeschicht die Nachbarschaftsrelationen der Eingabevektoren möglichst gut wiedergeben[119]. $V(\mathbf{W})$ besitzt bei Funktionen h_{cj} mit kurzer Reichweite in der Regel allerdings eine große Anzahl lokaler Minima[120], und bei geringen Werten α kann der Algorithmus diese nicht überwinden. Es ist daher für das Auffinden möglichst guter Lösungen angebracht, mit weitreichenden Funktionen h_{cj}, für die $V(\mathbf{W})$ nur ein Minimum besitzt, zu beginnen und die Reichweite langsam zu vermindern. Durch dieses Vorgehen entstehen neue lokale Minima in der aktuellen Talsohle der Potentialfunktion, in die der Algorithmus aufgrund von *(2.18)* fallen wird.[121]

---

[115] Vgl. Ritter et al. [1994], S. 108.
[116] Vgl. Ritter und Schulten [1988a], S. 114.
[117] Vgl. Ritter et al. [1994], S. 108.
[118] Ritter et al. bezeichnen diesen Fall als „unbegrenzte Reichweite“ der Nachbarschaftsfunktion.
[119] Vgl. Ritter et al. [1994], S. 109.
[120] Vgl. für eine Abschätzung Ritter et al. [1994], S. 109 f.
[121] Vgl. Ritter et al. [1994], S. 110.

Zusammenfassend ist zu erwähnen, dass auch für diskrete Daten kein Beweis zur Konvergenz des Algorithmus gegen das globale Minimum der Potentialfunktion erbracht werden konnte. Die dem Verfahren zugrundeliegenden Mechanismen können jedoch zur Identifikation „guter" Lösungen führen, die durch lokale Minima von $V(\mathbf{W})$ dargestellt werden. Es ist allerdings klar herauszustellen, dass mit der Minimierung des Potentialfunktionswertes nicht zwangsläufig die Zielfunktionswerte der Optimierungsprobleme, die mit der Self-Organizing Map gelöst werden, minimiert werden.

2.3.6 Zeitkomplexität der Self-Organizing Map

Ein Vorteil künstlicher neuronaler Netze wird in der Parallelisierbarkeit der Berechnungen der einzelnen Neuronenmodelle gesehen, die eine Verringerung der Rechenzeiten der Algorithmen ermöglicht. Aus diesem Grund soll in diesem Abschnitt die Zeitkomplexität der SOM für den Idealfall paralleler Implementierung dargestellt werden.[122] Das Symbol *I* beschreibt dabei die Anzahl der Komponenten der Eingabevektoren, das Symbol *J* die Anzahl der Neuronen der Ausgabeschicht und das Symbol *P* die Anzahl der eingesetzten Recheneinheiten. Tabelle 2.11 zeigt die Zeitkomplexität O(.) für die einzelnen Schritte des Algorithmus und die resultierende Zeitkomplexität für eine Iteration einer SOM mit einer zweidimensionalen Ausgabeschicht.
Die in den folgenden Kapiteln vorgestellten Anwendungen weisen überwiegend Eingabeschichten auf, die aus zwei Neuronen bestehen. Die Anzahl der Neuronen der Ausgabeschicht variiert mitunter stark, es sind häufig Ausgabeschichten zu finden, deren Neuronenzahl mit der Zahl der Eingabevektoren *N* übereinstimmen oder ein Vielfaches (in der Regel das Zwei- bis Dreifache) dieser Zahl aufweisen.

[122] Vgl. Hämäläinen [2002], S. 251 f.

Schritt	Bezeichnung	Zeitkomplexität im Idealfall
2.	Kopieren der Eingabevektoren für alle Neuronen.	*O(I)*
	Simultane Verteilung der Komponenten der Eingabevektoren an alle Neuronen.	*O(1)*
3.	Parallele Berechnung der Distanzen.	*O(I)*
4.	Bestimmung des Zentrums.	*O(J)*
5.	Bestimmung der Nachbarschaft und	*O(J/P)*
	Anpassung der Gewichtsvektoren der Neuronen der Nachbarschaft.	*O(I)*
Insgesamt	-	*O(I+J)*

Tabelle 2.11 : Zeitkomplexität der SOM im Idealfall

Ist die Anzahl durchzuführender Iterationen ebenfalls von der Zahl der Eingabevektoren abhängig, so ergibt sich insgesamt eine Zeitkomplexität der Ordnung $O(N^2)$. Gegenwärtig wird das Verfahren überwiegend auf nicht parallel arbeitender Hardware verwendet, was im Vergleich zu anderen Eröffnungsverfahren zu verhältnismäßig hohen Rechenzeiten führt. Viele vorgeschlagene SOM-Varianten zur Bearbeitung der untersuchten Optimierungsprobleme weisen einen Rechenaufwand der Größenordnung $O(N^3)$ auf[123], während z.B. der alternativ einsetzbare Spacefilling Curve-Ansatz nur einen Aufwand von *O(NlogN)* aufweist[124]. Der Vergleich der Zeitkomplexitäten kann als Begründung für die Aussage, dass Self-Organizing Maps erst mit zunehmender Verbreitung parallel arbeitender Hardware als Eröffnungsverfahren für kombinatorische Optimierungsprobleme an Relevanz gewinnen können, herangezogen werden. Die abschließende Beurteilung eines Algorithmus sollte neben der Zeitkomplexität allerdings auch Kriterien wie die Lösungsqualität und -wahrscheinlichkeit, sowie den Speicherplatzbedarf in Betracht ziehen.[125]

[123] Vgl. Burke [1996], S. 123 und S. 126. Um diese Schwäche des Algorithmus zu überwinden, entwickeln Fritzke und Wilke [1992] eine SOM-Variante, die einen linear mit der Problemgröße anwachsenden Rechenaufwand aufweist. Darüber hinaus existieren verschiedene Modifikationen, die ein quadratisches Anwachsen der Rechenzeit aufweisen, vgl. Budinich [1996], Amin [1994] und Amin et al. [1994].
[124] Vgl. Bartholdi und Platzman [1982], S. 123. Vgl. zur Darstellung des Spacefilling Curve-Ansatzes Platzman und Bartholdi [1989].
[125] Eine Übersicht über mögliche Bewertungskriterien geben Fischer und Kruschwitz [1980].

2.4 Vergleichsverfahren

In diesem Abschnitt erfolgt eine kurze Skizzierung der Grundideen von Algorithmen, die häufig als Vergleichsverfahren zur Einschätzung der Lösungsqualität der SOM herangezogen werden.

Elastic Net-Ansatz

1987 wird von DURBIN und WILLSHAW das Elastic Net (EN) zur Lösung des Rundreiseproblems mit euklidischen Distanzen vorgestellt[126]. Zu Beginn des Algorithmus werden *j (j=1(1)N)* Knoten entlang eines Kreises um den Schwerpunkt der Koordinaten $\mathbf{x}_i$ der zu besuchenden Orte *i (i=1(1)M)* angeordnet. Diese Anordnung kann als Pfad interpretiert werden, der im Verlauf des Algorithmus so angepasst wird, dass er eine Tour durch die Orte beschreibt. Die Bewegung eines Knotens *j* wird durch zwei am Knoten angreifende Kräfte[127] F_α und F_β verursacht, die eine Veränderung der Position $\mathbf{y}_j$ des Knotens gemäß

$$\Delta \mathbf{y}_j = \underbrace{C_1 \cdot \sum_i Norm_{ij} \cdot \left(\mathbf{x}_i - \mathbf{y}_j\right)}_{F_\alpha} + \underbrace{C_2 \cdot \sigma(t)\left(\left(\mathbf{y}_{j+1} - \mathbf{y}_j\right) + \left(\mathbf{y}_{j-1} - \mathbf{y}_j\right)\right)}_{F_\beta} \qquad (2.21)$$

bewirken. Die Konstanten C_1 und C_2 steuern den Einfluss der Kräfte, welche durch die beiden Terme in Gleichung *(2.21)* beschrieben werden. Der zur Normalisierung des Einflusses eines Ortes *i* auf einen Knoten *j* verwendete Koeffizient

$$Norm_{ij} = \frac{f\left(\left|\mathbf{x}_i - \mathbf{y}_j\right|, \sigma(t)\right)}{\sum_j f\left(\left|\mathbf{x}_i - \mathbf{y}_j\right|, \sigma(t)\right)} \qquad (2.22)$$

wird in Abhängigkeit von der Distanz $\left|\mathbf{x}_i - \mathbf{y}_j\right|$ zwischen $\mathbf{x}_i$ und $\mathbf{y}_j$ und vom Parameter $\sigma(t)$ modelliert. Die Autoren verwenden für $f\left(\left|\mathbf{x}_i - \mathbf{y}_j\right|, \sigma(t)\right)$ eine Gauß-Funktion der Form

$$f\left(\left|\mathbf{x}_i - \mathbf{y}_j\right|, \sigma(t)\right) = \exp\left(\frac{-\left|\mathbf{x}_i - \mathbf{y}_j\right|^2}{2 \cdot \sigma(t)^2}\right). \qquad (2.23)$$

Der Parameter $\sigma(t)$ wird mit zunehmender Iterationszahl *t* schrittweise verringert, so dass der Einfluss des zweiten Terms im Verlauf des Algorithmus sinkt.

---

[126] Vgl. Durbin und Willshaw [1987].
[127] Vgl. Potvin [1993], S. 340.

Es wird deutlich, dass im Unterschied zur SOM bei der Adaption nicht nur ein Eingabevektor verwendet wird, sondern alle zu besuchenden Orte einen Einfluss auf die Verschiebung haben. Darüber hinaus ist das Verfahren im Gegensatz zur SOM als deterministisches Verfahren zu klassifizieren. VAKHUTINSKY und GOLDEN erweitern den EN-Ansatz 1994[128], um eine Anwendung auf kapazitierte Tourenplanungsprobleme zu ermöglichen, 1995 entwickeln die gleichen Autoren einen hierarchischen EN-Ansatz für das Rundreiseproblem[129]. Eine detaillierte Analyse des Verfahrens von DURBIN und WILLSHAW findet sich in DURBIN ET AL.[130].

Farthest Insertion

Dieses Einfügeverfahren kann zur Lösung von Routenplanungsproblemen verwendet werden und weist eine Zeitkomplexität der Größenordnung $O(N^2)$ auf.[131] Ausgehend von einer geschlossenen Subtour, die nur einen Teil der zu besuchenden Kunden enthält, werden sukzessive die bisher nicht in die Tour aufgenommenen Kunden eingefügt. Der in einer Iteration des Eröffnungsverfahrens[132] in die Tour einzufügende Kunde kann anhand verschiedener Regeln identifiziert werden.[133] Die *Farthest Insertion*-Variante wählt dabei denjenigen Kunden aus, der am weitesten von den Kunden der bestehenden Subtour entfernt ist. Für alle möglichen Einfügepositionen wird die resultierende Verlängerung der Subtour ermittelt und der Kunde anschließend an derjenigen Position eingefügt, für welche die Verlängerung minimal wird[134] (vgl. Abbildung 2.18).

Abbildung 2.18 : Schema des Farthest Insertion-Verfahrens für ein Rundreiseproblem

[128] Vgl. Vakhutinsky und Golden [1994].
[129] Vgl. Vakhutinsky und Golden [1995].
[130] Vgl. Durbin et al. [2001].
[131] Vgl. Vahrenkamp [2003], S. 208.
[132] Vgl. Golden und Stewart, S. 215 ff.
[133] Eine Übersicht über unterschiedliche Varianten gibt Vahrenkamp [2003], S. 209 f.
[134] Vgl. Golden und Stewart [1985], S. 217, und Vahrenkamp [2003], S. 208.

Nearest Neighbor-Verfahren

Dieses Eröffnungsverfahren[135] für Routenplanungsprobleme fügt, ausgehend von einem aus der Menge der Kunden V gewählten Startkunden, sukzessive Kunden in die Tour eines Fahrzeuges ein, wobei immer derjenige noch nicht eingeplante Kunde der Tour hinzugefügt wird, der die geringste Entfernung zum aktuellen Kunden aufweist[136] (vgl. Abbildung 2.19). Die Komplexität des Verfahrens beträgt für Probleminstanzen mit N Knoten $O(N^2)$[137]. Wird das Verfahren N-mal mit jeweils unterschiedlichen Startkunden durchlaufen (häufig wird diese Variante als Multiple Nearest Neighbor Verfahren bezeichnet), ergibt sich eine Zeitkomplexität $O(N^3)$[138].

Abbildung 2.19 : Schema des Nearest Neighbor-Verfahrens für ein Rundreiseproblem

Neuronales Netz von HOPFIELD und TANK

Das von HOPFIELD und TANK 1985 vorgeschlagene KNN[139] besteht aus vollständig miteinander verbundenen Neuronen[140] (Abbildung 2.20 zeigt stellvertretend ein Netz mit vier Neuronen), wobei die Verbindungsgewichte zwischen zwei Neuronen jeweils identische Werte aufweisen. Die Verwendung der Ausgabe eines Neurons als Eingabesignal für die übrigen Neuronen (Rückkopplung) des KNN führt dabei zu einem iterativen Anpassungsprozess der Neuronenausgaben[141], die für binäre HOPFIELD-TANK-Netze[142] jeweils den Wert Null oder Eins annehmen können. Letzterer wird immer dann angenommen, wenn die Aktivierung den Schwellenwert des Neurons überschreitet.[143] Zur Bestimmung der Aktivierung wird die Summe der gewichteten Eingangssignale berechnet, zu denen neben Rückkopplungen auch externe Netzeingaben zählen können.

[135] Vgl. Vahrenkamp [2003], S. 204.
[136] Vgl. Johnson und McGeoch [1997], S. 221.
[137] Vgl. Johnson und McGeoch [1997], S. 221.
[138] Vgl. Burke [1995], S. 124.
[139] Vgl. Hopfield und Tank [1985].
[140] Ein Neuron ist dabei nicht mit sich selbst verbunden.
[141] Vgl. Retzko [1996], S. 21.
[142] Eine kontinuierliche Modellierung ist auch möglich, vgl. Zell [1994], S. 201.
[143] Vgl. Zell [1994], S. 198.

Zur Anwendung auf Rundreiseprobleme mit N Kundenorten werden N^2 Neuronen benötigt, die als Einträge in einer $N \times N$ Matrix interpretiert werden können. Nimmt die Ausgabe des Neurons *ij (i, j=1(1)N)* den Wert Eins an, so wird der Ort *i* an Position *j* der Rundreise besucht, nimmt die Ausgabe den Wert Null an, so wird der Ort *i* nicht an Position *j* der Rundreise besucht[144].

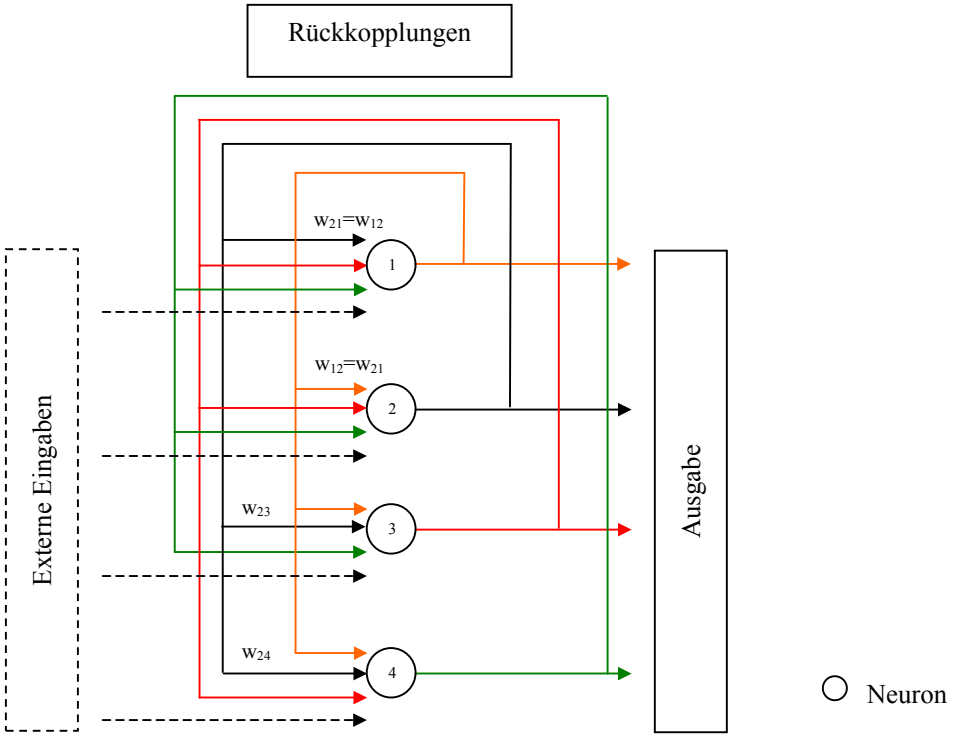


Abbildung 2.20 : Schema des KNN von HOPFIELD und TANK[145]

Es lässt sich zeigen, dass der Zustand des KNN durch eine Energiefunktion beschrieben werden kann, die von den aktuellen Neuronenausgaben, den externen Eingaben und den Verbindungsgewichten abhängt[146] und deren Wert im Verlauf des Anpassungsprozesses nicht zunehmen kann[147]. Das KNN stabilisiert sich darüber hinaus genau dann in einem Netzwerkzustand (d. h., die Neuronenausgaben verändern sich nicht mehr), wenn ein lokales Minimum der Energiefunktion erreicht wird.[148]

Zur Repräsentation des Rundreiseproblems durch das KNN wird die Energiefunktion so festgelegt, dass diese insbesondere dann geringe Werte aufweist, wenn alle Nebenbedingungen des Problems erfüllt werden und die Länge der Rundreise möglichst gering

[144] Vgl. Hopfield und Tank [1985], S. 146 f., und Zell [1994], S. 202.
[145] Vgl. Zell [1994], S. 198. Stellvertretend werden zugunsten der Übersichtlichkeit hier nur die Verbindungsgewichte der Rückkopplungen zwischen Neuron 2 und den übrigen Neuronen dargestellt. Zur Verdeutlichung der Symmetrie der Verbindungen zwischen zwei Neuronen wurde zusätzlich das Gewicht der Rückkopplung zwischen Neuron 1 und Neuron 2 in der Abbildung verzeichnet.
[146] Vgl. Zell [1994], S. 200, und die dort angegebenen Quellen.
[147] Vgl. Zell [1994], S. 201, und Retzko [1996], S. 23.
[148] Vgl. Retzko [1996], S. 23.

ausfällt.[149] Detaillierte Darstellungen des KNN finden sich in ZELL [1994], Anwendungsbeispiele geben neben HOPFIELD und TANK [1985] beispielsweise WACHOLDER ET AL. [1989] und XU und TSAI [1991].

Savings-Verfahren

Das Savings-Verfahren wurde ursprünglich von CLARKE und WRIGHT zur Lösung von Tourenplanungsproblemen entwickelt[150]. Das Eröffnungsverfahren kann allerdings auch zur Lösung von Rundreiseproblemen verwendet werden[151] und weist in diesem Fall eine Zeitkomplexität der Größenordnung $O(N^2 logN)$ auf.[152]

Bei Anwendung auf Tourenplanungsprobleme wird ausgehend von einer Lösung, in der die zu besuchenden Kunden jeweils auf Pendeltouren besucht werden, eine sogenannte Savings-Liste erstellt, welche die Ersparnisse der Gesamttourenlänge bei Zusammenlegung zweier Touren enthält. Die Berechnung der Savings für symmetrische Entfernungen zwischen den Kunden gemäß

$$(2.24) \quad S_{mn} = d_{0n} + d_{0m} - d_{mn}$$

wird in Abbildung 2.21 verdeutlicht:

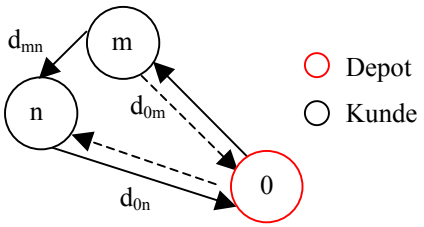


Abbildung 2.21 : Berechnung der Savings

Positive Savings S_{mn} werden in der Liste nach nicht ansteigenden Werten geordnet und die Listeneinträge anschließend in der entstandenen Reihenfolge abgearbeitet. Dabei werden die Routen der Kunden *m* und *n* miteinander verbunden, wenn

1. die Kunden *m* und *n* Endkunden zweier verschiedener Routen sind und
2. etwaige Zeit-, Tourlängen- und Kapazitätsrestriktionen durch die erweiterte Tour nicht verletzt werden.[153]

Das Verfahren bricht ab, wenn alle Listeneinträge überprüft wurden.

[149] Vgl. hierzu bspw. Zell [1994], S. 203.
[150] Vgl. Clarke und Wright [1964].
[151] Vgl. Golden und Stewart [1985], S. 240.
[152] Vgl. Johnson und McGeoch [1997], S. 222.
[153] Vgl. Domschke [1997], S. 245.

Simulated Annealing

1983 stellen KIRKPATRICK ET AL. das Verfahren Simulated Annealing (*simuliertes Abkühlen*) zur Lösung kombinatorischer Optimierungsprobleme vor[154], das sich an einem Prozess orientiert, der eingesetzt wird, um niedrige Energiezustände eines physikalischen Systems (beispielsweise des Atomgitters eines Kristalls) zu erreichen. Der Prozess besteht aus zwei Stufen: Zunächst wird die Temperatur des physikalischen Systems auf dessen Schmelzpunkt erhöht. Anschließend wird die Temperatur langsam abgesenkt, so dass sich die Atome in einer Struktur mit niedrigem Energiezustand anordnen können. Zur Übertragung des physikalischen Vorbildes auf kombinatorische Optimierungsprobleme kann die Zielfunktion des Problems als Energiefunktion E und eine Lösung des Problems als Zustand des physikalischen Systems interpretiert werden. Die aktuelle Lösung wird in einer Iteration nach zu definierenden Regeln modifiziert, d.h. es wird eine Lösung aus der Nachbarschaft der aktuellen Lösung ausgewählt[155]. Die Wahrscheinlichkeit, mit der die gewählte Lösung der Nachbarschaft in der folgenden Iteration als Ausgangslösung verwendet wird, ist durch die Funktion

(2.25)

$$P_{\sigma(t)}(\Delta \mathrm{E}) = \begin{cases} 1 & ,\Delta E \le 0 \\ \exp\left(\dfrac{-\Delta E}{\sigma(t)}\right) & ,\Delta E > 0 \end{cases}$$

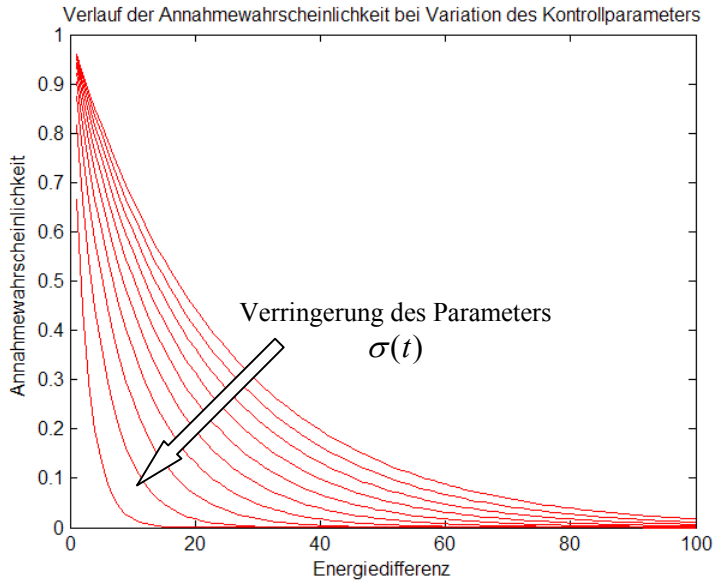


Abbildung 2.22 : Verlauf der Annahmewahrscheinlichkeit bei Variation des Kontrollparameters

[154] Vgl. Kirkpatrick et al. [1983].
[155] Vgl. Aarts et al. [1997], S. 92.

vorgegeben, d.h. eine Verschlechterung des Zielfunktionswertes wird mit einer von der Höhe der Verschlechterung abhängigen Wahrscheinlichkeit ermöglicht, so dass das Verfahren in der Lage ist, lokale Optima zu überwinden. Der Parameter $\sigma(t)$ stellt dabei einen Kontrollparameter dar, der nach einem vordefinierten Schema vermindert wird und das „Abkühlen" des Systems, d.h. die Verringerung der Annahmewahrscheinlichkeit bei positiven Energiedifferenzen ΔE, steuert (vgl. Abbildung 2.22). Zum Erreichen guter Lösungen ist es notwendig, den Abkühlungsprozess langsam verlaufen zu lassen und auf einer durch $\sigma(t)$ definierten „Temperaturstufe" eine ausreichende Zahl an Iterationen durchzuführen.[156]

Eine detaillierte Analyse des Verfahrens findet sich in AARTS ET AL. [1997]. Das Verfahren ist als Metastrategie zu charakterisieren, da es eine problemunabhängige Vorgehensweise zur Verfügung stellt. Darüber hinaus kann Simulated Annealing als Verbesserungsverfahren klassifiziert werden, da eine lokale Suche in der Nachbarschaft der aktuellen Lösung durchgeführt wird.

Spacefilling Curve

1982 schlagen BARTHOLDI und PLATZMAN den sogenannten Spacefilling Curve Ansatz zur Lösung von Rundreiseproblemen vor[157], der eine Komplexität von O(*NlogN*) aufweist. Die Spacefilling Curve (Raumfüllende Kurve) entsteht durch einen rekursiven Prozess, der einen gegebenen Raum in einer Iteration in mehrere Gebiete unterteilt und die Figur der raumfüllenden Kurve aus der letzten Iteration in jedes dieser Gebiete „kopiert" und die Teilfiguren miteinander verbindet (vgl. Abbildung 2.23). Sind *N* Knoten als auf einer Rundreise zu besuchende Orte gegeben, so können diese in der Reihenfolge angefahren werden, in der sie auf der raumfüllenden Kurve angeordnet sind.[158] Eine detaillierte Analyse des Eröffnungsverfahrens findet sich in PLATZMAN und BARTHOLDI [1989].

[156] Vgl. Kirkpatrick et al. [1983], S. 673.
[157] Vgl. Bartholdi und Platzman [1982].
[158] Vgl. Platzman und Bartholdi [1989], S. 719.

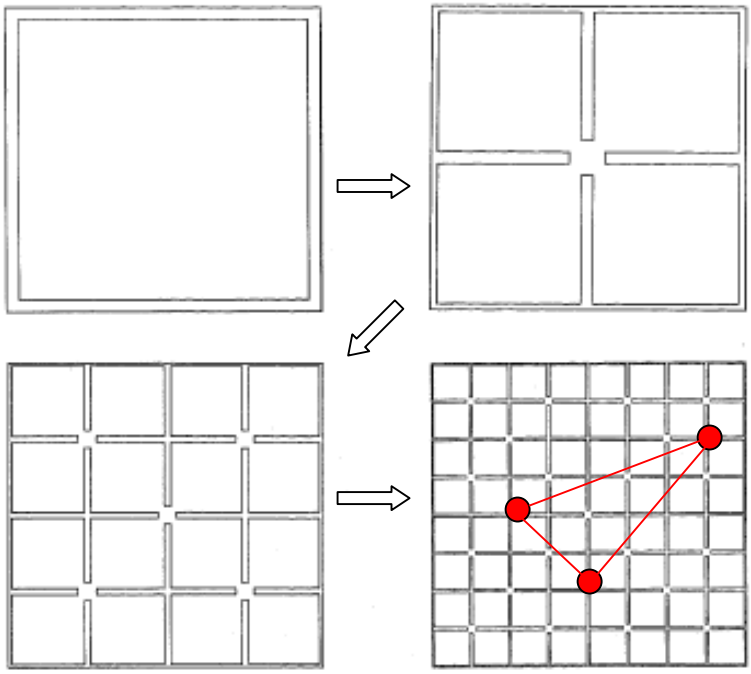


Abbildung 2.23[159]:
Raumfüllende Kurve und resultierende Rundreise durch auf der Kurve angeordnete Knoten

Tabu Search-Verfahren

Diese Metastrategie[160] kann der Klasse der Verbesserungsverfahren zugeordnet werden[161]. Das Verfahren generiert in einer Iteration ausgehend von einer Ausgangslösung durch Anwendung zu definierender Regeln (sogenannter Züge) eine Menge benachbarter Lösungen, die beispielsweise mit Hilfe des zugehörigen Wertes der Zielfunktion des zu lösenden Optimierungsproblems bewertet werden. Abbildung 2.24 verdeutlicht eine mögliche Definition der Nachbarschaft einer Lösung für ein Rundreiseproblem. Dabei werden benachbarte Lösungen durch Positionswechsel zweier Kunden in der Rundreise generiert.

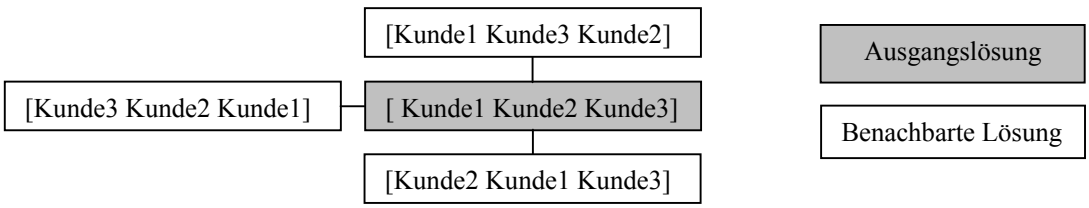


Abbildung 2.24 : Mögliche Definition der Nachbarschaft einer Lösung für ein Rundreiseproblem

[159] Entnommen aus Platzman und Bartholdi [1989], S. 723.
[160] Vgl. zur Einordnung des Verfahrens Glover und Laguna [1997], S. 18.
[161] Vgl. Domschke [1997], S. 21.

Aus der Menge der Nachbarschaftslösungen wird in der Regel diejenige Lösung ausgewählt, welche die größte Verbesserung beziehungsweise die geringste Verschlechterung des Zielfunktionswertes aufweist. Die Vorgehensweise, auch Lösungen zu akzeptieren, die einen schlechteren Zielfunktionswert als die Ausgangslösung aufweisen, soll eine Überwindung lokaler Optima ermöglichen. Zusätzlich wird eine Tabu-Liste definiert, die ein „Kreisen" des Algorithmus um lokale Optima verhindern soll. Diese Liste, die häufig als „Kurzzeitgedächtnis" interpretiert wird[162], soll eine erneute Wahl von Ausgangslösungen vorhergehender Iterationen für eine zu definierende Anzahl an Iterationen durch Verbot bestimmter Züge ausschließen.

Optionale Bestandteile des Lösungsverfahrens stellen Aspirationskriterien[163] dar, die den Tabu-Status eines Zuges aufheben und so die Lösungsqualität des Verfahrens verbessern können[164]. Intensivierungs- und Diversifikationsstrategien[165] können ebenfalls zu einer Verbesserung der Lösungsqualität beitragen. Erstere modifizieren die Auswahlregeln dahingehend, dass Züge oder Eigenschaften von Lösungen, die sich in den bisher durchgeführten Iterationen als erfolgreich erwiesen haben, bei der Auswahl bevorzugt werden. Dieses Vorgehen unterstützt die Intensivierung der Suche in erfolgsversprechenden Regionen des Lösungsraumes. Im Gegensatz dazu sollen Diversifikationsstrategien dazu beitragen, bisher nicht oder nur wenig durchsuchte Bereiche des Lösungsraumes zu untersuchen. Die häufig als „Langzeitgedächtnis" interpretierten Strategien können zur Suche nach möglichst guten Lösungen abwechselnd eingesetzt werden. Tabu Search ist als eines der erfolgreichsten Verbesserungsverfahren zur Lösung von Tourenplanungsproblemen anzusehen[166]. Detaillierte Darstellungen der Metastrategie finden sich in GLOVER und LAGUNA [1997] .

2-Opt-Verfahren

Dieses von CROES 1958 vorgeschlagene Verfahren[167] kann der Klasse der Verbesserungsverfahren zugeordnet werden[168]. Ausgehend von einer zulässigen Lösung für das zugrundeliegende Routenplanungsproblem wählt das Verfahren jeweils zwei nicht be-

[162] Vgl. Glover und Laguna [1997], S. 25 ff.
[163] Ein weitverbreitetes Aspirationskriterium stellt die Aufhebung des Tabu-Status eines Zuges dar, wenn dieser eine Lösung generiert, die einen besseren Zielfunktionswert aufweist als die bisher beste ermittelte Lösung.
[164] Vgl. Glover und Laguna [1997], S. 50 ff.
[165] Vgl. Glover und Laguna [1997], S. 8 und S. 93 ff..
[166] Vgl. Gendreau et al. [1997], S. 336.
[167] Vgl. Croes [1958].
[168] Vgl. Johnson und McGeoch [1997], S. 230.

nachbarte Kanten einer Tour und entfernt diese.[169] Die entstehenden Teilpfade der Tour werden durch zwei andere Kanten wieder miteinander verbunden, wenn dies zu einer Verkürzung der Tour führt (vgl. Abbildung 2.25).

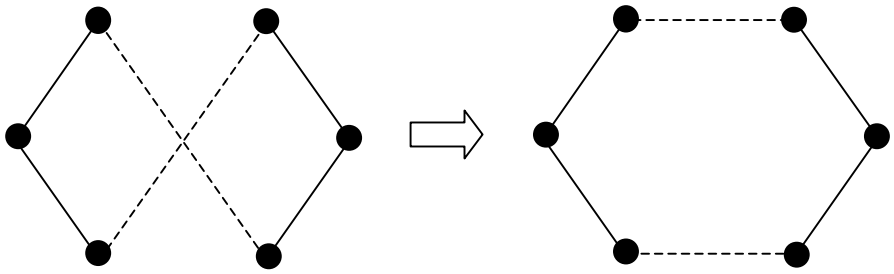


Abbildung 2.25 : Darstellung des Austauschverfahrens 2-Opt

Der Algorithmus bricht ab, wenn keine weitere Verbesserung durch Austausch von Kanten erzielt werden kann. Detaillierte Darstellungen des Verfahrens geben JOHNSON und McGEOCH [1997].

[169] Vgl. Vahrenkamp [2003], S. 217.

3. Anwendung der Self-Organizing Map auf ausgewählte Routenplanungsprobleme

In diesem Kapitel wird für ausgewählte Routenplanungsprobleme in den Abschnitten 3.1-3.3 anhand einer Literaturübersicht und eines Anwendungsbeispiels[170] gezeigt, wie die Self-Organizing Map zur Ableitung von möglichst guten Lösungen eingesetzt wird und welche Grenzen der Anwendung gesetzt sind. Abschnitt 3.4 gibt einen Überblick über weitere Problemstellungen, die mit dem Verfahren gelöst werden können.

3.1 Euklidische Traveling Salesman Probleme

3.1.1 Beschreibung der Traveling Salesman Probleme

Der Begriff „Traveling Salesman Problem“ (TSP) wurde vermutlich von WHITNEY um 1931-1932 geprägt, erste Beschreibungen der Problemstellung werden bereits 1832 von VOIGT veröffentlicht.[171] 1954 veröffentlichen schließlich DANTZIG, FULKERSON und JOHNSON einen Artikel[172], der sowohl den Nachweis über die Optimalität einer Lösung eines TSP mit 49 Orten präsentiert als auch erste Ansätze zur Verwendung von relaxierten Problemen und Branch-and-Bound-Techniken enthält und somit als grundlegend für die Entwicklung der Erforschung kombinatorischer Probleme angesehen wird[173]. Für eine detaillierte Übersicht über die wesentlichen Entwicklungen zum TSP und über verschiedene Lösungsverfahren wird auf LAWLER ET AL.[174] und den 1992 von LAPORTE veröffentlichten Artikel[175] verwiesen.

Das TSP besteht in der Aufgabe, die (kosten-)günstigste Rundreise durch eine vorgegebene Anzahl an Knoten zu bestimmen. Als *Rundreise* bezeichnet man dabei einen Zyklus[176] eines bewerteten, schlichten[177], ungerichteten[178] Graphen[179] oder eines bewerteten

[170] Die in den Beispielen verwendete Instanz weist insgesamt 83 unregelmäßig verteilte Knoten auf. Eine Übersicht über die Koordinaten der Knoten befindet sich im Anhang der Arbeit.
[171] Vgl. Hoffman und Wolfe [1985], S. 5.
[172] Vgl. Dantzig et al. [1954].
[173] Vgl. Hoffman und Wolfe [1985], S. 6.
[174] Vgl. Lawler et al. [1985].
[175] Vgl. Laporte [1992].
[176] Als Zyklus bezeichnet man einen geschlossenen Weg, der mit Ausnahme von Anfangs- und Endpunkt lauter verschiedene Knoten enthält. Vgl. Domschke [1995], S. 8.
[177] Ein Graph ist schlicht, wenn er keine parallelen Pfeile bzw. Kanten und Schlingen besitzt, vgl. Domschke [1995], S. 2.
[178] Ein Graph ist ungerichtet, wenn das durch die Inzidenzabbildung jedem Element der Kantenmenge E zugeordnete Paar aus der Knotenmenge V nicht geordnet ist. Ein Graph ist gerichtet, wenn das jedem Element aus E zugeordnete Paar aus V geordnet ist. Vgl. Domschke [1995], S. 1.

Digraphen[180] G, der jeden der N Knoten von G genau einmal enthält.[181] Als Bewertung c_{mn} der Kante beziehungsweise des Pfeils zwischen Knoten m und Knoten n kann beispielsweise die kürzeste Distanz zwischen den Knoten verwendet werden.

Ist der Graph G gerichtet, spricht man von einem asymmetrischen TSP, ist der Graph G ungerichtet, so lautet die Bezeichnung symmetrisches TSP. Symmetrische und asymmetrische TSPe gehören zur Klasse „NP-schwerer" Probleme[182]. Bisher sind keine Lösungsverfahren mit polynomialem Zeitaufwand bekannt, mit deren Hilfe die optimale Lösung mit Sicherheit bestimmt werden kann[183], weshalb die Verwendung von Heuristiken zur Ableitung einer Lösung für größere Probleminstanzen gerechtfertigt ist.

Zur Beschreibung des TSP wurden unterschiedliche Modellformulierungen entwickelt, so existieren beispielsweise lineare und quadratische Modellformulierungen[184], die sich hinsichtlich des Grades der Zielfunktion, der Anzahl verwendeter Variablen und der Anzahl an Nebenbedingungen unterscheiden.

Mit der in Tabelle 3.2 angegebenen Formulierung[185] kann sowohl das asymmetrische als auch das symmetrische TSP beschrieben werden. Das Letztere zeichnet sich im Gegensatz zum asymmetrischen TSP durch eine symmetrische Kostenmatrix $\boldsymbol{C}(c_{mn})$ aus, d.h., $c_{mn}=c_{nm}$ gilt für alle Paarungen der Knoten m und n. Für symmetrische Problemstellungen existieren allerdings Modellformulierungen, durch welche die Anzahl benötigter Variablen reduziert werden kann[186].

---

[179] Ein Graph G besteht aus der nichtleeren Knotenmenge V, der Kanten- bzw. Pfeilmenge E und einer Inzidenzabbildung, die jedem Element aus E genau ein Knotenpaar zuordnet, vgl. Domschke [1995], S. 1.

[180] Als einen Digraphen bezeichnet man einen endlichen, schlichten, gerichteten Graphen. Endlich ist ein gerichteter Graph G, wenn Pfeil- und Knotenmenge endlich sind. Vgl. Domschke [1995], S. 3.

[181] Vgl. Domschke [1997], S. 101.

[182] Dies gilt auch für das TSP mit euklidischen Distanzen, vgl. Garey und Johnson [1979], S. 211 f.

[183] Vgl. Papadimitriou [1995], S. 13

[184] Vgl. Domschke [1997], S. 104 ff.

[185] Vgl. Domschke [1997], S.105 ff.

[186] Auf eine Darstellung dieser Modellformulierungen wird verzichtet, vgl. hierzu Domschke [1997], S. 107 ff.

Symbol	Interpretation
$m = 1(1)N$	Index der Knoten.
$n = 1(1)N$	Index der Knoten.
$x_{mn} = \begin{cases} 1, & \textit{direkte Verbindung von m nach n ist Teil der Rundreise} \\ 0, & \textit{sonst} \end{cases} \quad \forall m,n$	Beschreibt, ob die Kante / der Pfeil zwischen *m* und *n* in der Rundreise verwendet wird.
$c_{mn} = \begin{cases} c(m,n), & \textit{falls} \quad (m,n) \in E \\ \infty, & \textit{sonst} \end{cases} \quad \forall m,n$	Kostenbewertung der Kante / des Pfeils zwischen *m* und *n*.
$\pi_m,\ m = 1(1)N$	Hilfsvariablen zur Formulierung von Kurzzyklenbedingungen.

Tabelle 3.1 : Symbole der Modellformulierung des TSP

		Formale Beschreibung	Interpretation
Minimiere	*(3.1)*	$\sum_{m=1}^{N} \sum_{\substack{n=1, \\ n \neq m}}^{N} c_{mn} x_{mn}$	Die Gesamtkosten der Rundreise entsprechen der Summe der Kostenbewertungen der verwendeten Kanten.
u.B.v.:	*(3.2)*	$\sum_{\substack{n=1, \\ n \neq m}}^{N} x_{mn} = 1 \ , \ m = 1(1)N$	Jeder Knoten *m* muss genau einmal in Richtung eines Knoten *n* verlassen werden.
	(3.3)	$\sum_{\substack{m=1, \\ m \neq n}}^{N} x_{mn} = 1, \ n = 1(1)N$	Jeder Knoten *n* muss genau von einem Knoten *m* aus angefahren werden.
	(3.4)	$\pi_m - \pi_n + N * x_{mn} \leq N - 1 \quad \forall m,n = 2(1)N, \quad m \neq n$	Nebenbedingung zur Verhinderung von Kurzzyklen[187].
	(3.5)	$x_{mn} \in \{0,1\} \quad \forall m,n = 1(1)N, \quad m \neq n$	Binärbedingungen.
	(3.6)	$\pi_m \geq 0 \qquad m = 2(1)N$	Forderung der Nichtnegativität für die Hilfsvariablen.

Tabelle 3.2 : Mögliche Modellformulierung für das TSP

Für Zyklusbedingungen sind alternative Formulierungen denkbar, die sich in der Anzahl der benötigten Nebenbedingungen unterscheiden. Hier werden die nach den Erstautoren benannten *Miller-Tucker-Zemlin-Bedingungen* verwendet[188], die insgesamt $(N-1)\cdot(N-2)$ zusätzliche Bedingungen erfordern[189]. Diese verhindern Kurzzyklen, die den Knoten 1 nicht enthalten. Isoliert betrachtet werden durch (*3.4)* allerdings keine Kurzzyklen verhindert, die den Knoten 1 enthalten, erst in Verbindung mit den Gleichungen (*3.2)* und (*3.3)* werden sämtliche Kurzzyklen ausgeschlossen und es verbleiben

[187] Als Kurzzyklus bezeichnet man einen Zyklus, der weniger als alle *N* Knoten eines Graphen jeweils einmal enthält, vgl. Domschke [1997], S. 101.
[188] Vgl. Miller et al. [1960]. Für alternative Formulierungen vgl. bspw. Domschke [1997], S. 105 ff.
[189] Vgl. Domschke [1997], S. 106.

nur Rundreisen als zulässige Lösungen des Modells. Die Werte, welche die Hilfsvariablen π_m in einer zulässigen Lösung annehmen, können als Position des Knotens in der Rundreise interpretiert werden.[190] Bei Vernachlässigung der Binärbedingungen für die Entscheidungsvariablen ergeben sich insgesamt $2 \cdot N + (N-1) \cdot (N-2)$ Nebenbedingungen, das Modell verwendet N^2 binäre Entscheidungs- und $(N-1)$ Hilfsvariablen.

Bei den im Folgenden vorgestellten Anwendungen der SOM auf das symmetrische TSP wird vorausgesetzt, dass die Kostenbewertung einer Kante der euklidischen Distanz zwischen den der Kante zugeordneten Knoten entspricht. Das Problem wird in diesem Fall als euklidisches TSP (ETSP) bezeichnet[191], bei dessen Bearbeitung folgendes technisches Problem zu berücksichtigen ist[192]: Bei den Distanzen zwischen zwei Knoten kann es sich theoretisch um irrationale Zahlen handeln, welche eine unendliche Präzision der Distanzmatrix erfordern, so dass die Entfernungen implizit, beispielsweise durch Verwendung der Koordinaten der Knoten, vorgegeben werden müssen. Dennoch tritt das Problem spätestens bei der konkreten Berechnung einer Entfernung auf. Es kann jedoch umgangen werden, indem die Distanzen unter Inkaufnahme eines gewissen Präzisionsverlustes beispielsweise auf ganzzahlige Werte aufgerundet werden. Das ETSP kann formal durch die oben angegebene Modellformulierung beschrieben werden, die Kostenmatrix ist in diesem Fall symmetrisch und beinhaltet die (gerundeten) euklidischen Entfernungen zwischen den Knoten der Knotenmenge *V*. Sie erfüllt die Dreiecksungleichung, da bei euklidischen Problemen

$$(3.7) \quad c_{mn} + c_{nl} \geq c_{ml} \;\; \forall m,n,l \in V, m \neq n, l \neq m, l \neq n$$

gilt[193]. Es sind auch Problemstellungen vorstellbar, für welche die Entfernungen zwischen den Knoten nicht der euklidischen Distanz entsprechen[194]. Liegen die zu besuchenden Knoten allerdings in einem Gebiet, für das die Multiplikation der euklidischen Entfernungen mit einem konstanten Umwegfaktor[195] gute Näherungen für die realen Distanzen erzielt, können dennoch euklidische Modellierungen verwendet werden. Außerdem kann eine solche Modellierung zur Ableitung von Näherungslösungen verwendet werden, wenn die Ermittlung realer Distanzen aufgrund des mit der Größe der Probleminstanz verbundenen Aufwands ausscheidet.

---

[190] Vgl. Domschke [1997], S. 107.
[191] Vgl. Johnson und Papadimitriou [1985], S. 60.
[192] Vgl. Johnson und Papadimitriou [1985], S. 60. Dies gilt auch für andere euklidische Probleme.
[193] Vgl. Laporte [1992], S. 231.
[194] Bspw. bei Entfernungen in Straßennetzwerken oder unüberwindbaren Barrieren auf der direkten Verbindung.
[195] Vgl. hierzu bspw. Berens und Körling [1983], S. 68.

3.1.2 Eingesetzte Varianten der Self-Organizing Map

FORT zeigt bereits 1988, wie die Self-Organizing Map auf das ETSP angewendet werden kann.[196] Die grundsätzliche Idee besteht darin, eine „zirkulare" Matrix so auf die Koordinaten der zu besuchenden Knoten zu „legen", dass zwei in der Matrix benachbarte Einträge, die Punkte in der Ebene beschreiben, ebenfalls im durch die Koordinaten der N Orte aufgespannten Raum hinsichtlich ihrer euklidischen Entfernung benachbart sind. Als zirkular bezeichnet FORT dabei eine Matrix, welche die folgende Nachbarschaftsstruktur aufweist[197]: Sei

$$(3.8) \quad L \in \mathbb{N}, L << N.$$

Dann sind diejenigen Einträge h Nachbarn von Eintrag j, deren Indizes in der Menge

$$(3.9) \quad \Theta_j = \left\{ h \middle| \; h \in [j - L, j + L] \text{modulo } N \right\}$$

enthalten sind, d.h. es existiert eine „zirkulare" Nachbarschaftsstruktur innerhalb der Indizes. Die von FORT verwendete Matrix beinhaltet die Gewichte der den einzelnen Neuronen zugehörigen Verbindungen, die als Koordinaten der Positionen der Neuronen im Eingaberaum interpretiert werden können. Die im Laufe des Verfahrens entstehende Anordnung der Neuronen kann somit als Rundreise interpretiert werden. Die Anzahl der verwendeten Neuronen in der Ausgabeschicht entspricht für kleinere Instanzen dem Zweifachen der Knotenzahl, für größere Instanzen wählt der Autor das 2,5-fache dieser Zahl. Dieses Vorgehen soll verhindern, dass einem Neuron mehrere Knoten zugeordnet werden und somit keine eindeutige Reihenfolge der Knoten in der Rundreise angegeben werden kann. Abbildung 3.1 verdeutlicht die von FORT vorgeschlagene Vorgehensweise.

[196] Vgl. Fort [1988].
[197] Vgl. Fort [1988], S. 34.

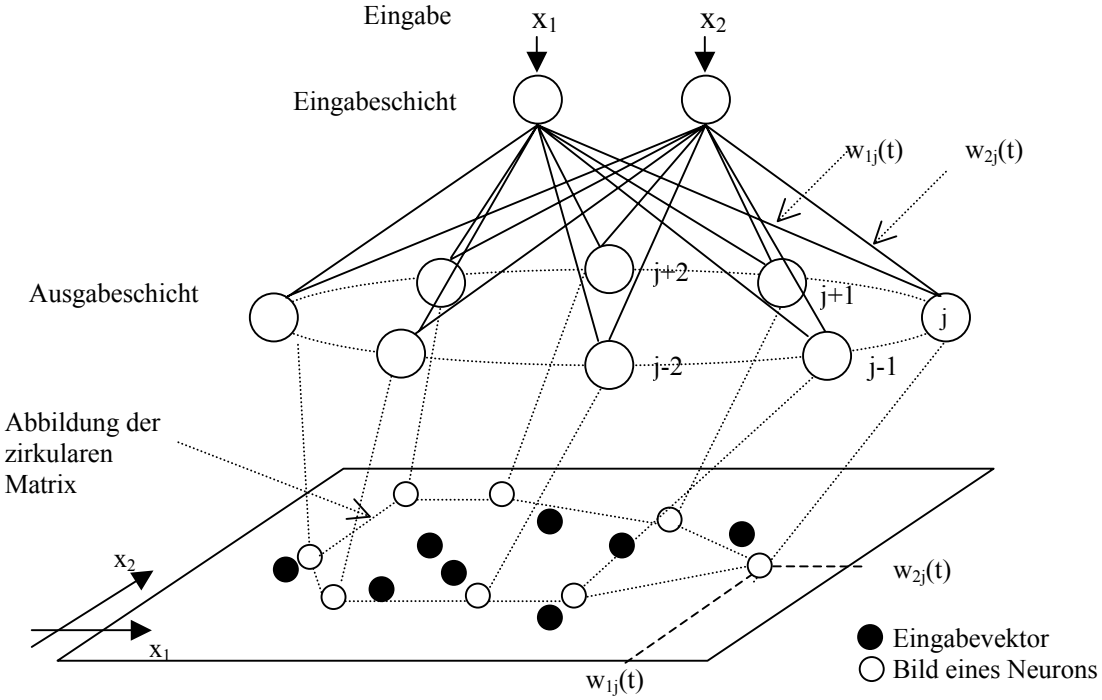


Abbildung 3.1 : Zirkulare Gewichtsmatrix und Raum der Eingabevektoren

Die Anzahl der Knoten der von FORT zur Einschätzung des Lösungsverhaltens verwendeten Testinstanzen variiert zwischen 10 und 400. Der Autor zeigt darüber hinaus die Möglichkeit auf, ausgehend von einer mit der SOM ermittelten Lösung für ein Problem mit *N* Knoten, Lösungen für das um einige Knoten erweiterte Problem in kurzer Zeit durch erneute Anwendung des Algorithmus zu bestimmen. Zur Einschätzung der Lösungsqualität vergleicht FORT die mittlere Länge der ermittelten Routen mit den Ergebnissen einer Simulated Annealing-Prozedur[198]:

	Mittlere Länge	**Varianz**
FORT	6,035	0,385
Simulated Annealing	5,735	0,290

Tabelle 3.3 : Ergebnisse von FORT[199]
(50 Instanzen mit jeweils 50 Knoten)

[198] Vgl. zum Verfahren des Simulated Annealing Kirkpatrick et al. [1983] und Aarts et al. [1997]. Das Verfahren kann als Verbesserungsverfahren klassifiziert werden und ist aus diesem Grund nur bedingt als Vergleichsverfahren geeignet.
[199] Vgl. Fort [1988], S. 39.

RITTER und SCHULTEN veröffentlichen ebenfalls 1988 einen Tagungsbeitrag[200], in dem die Anwendung des Algorithmus zur Lösung eines ETSP mit 30 Orten demonstriert wird. Darüber hinaus stellen die Autoren eine mathematische Analyse für diskrete Eingabedaten vor (vgl. Abschnitt *2.3.5*). RITTER und SCHULTEN verwenden als Nachbarschaftsfunktion eine Gauß-Funktion der Form

(3.10) $0 \le h_{cj}(t) = \exp\left(-\frac{\Delta_{cj}^{\ 2}}{2 \cdot \sigma^2(t)}\right) \le 1$, (Verlauf ähnlich zur Abbildung 2.15)

welche die Stärke der Anpassung eines Neurons *j* der Ausgabeschicht in Abhängigkeit von der Entfernung zum Zentrum *c* in Iteration *t* bestimmt. Die Anpassung der Gewichtsvektoren wird dann gemäß

(3.11) $\mathbf{w}_j(t+1) = \mathbf{w}_j(t) + \varepsilon(t) \cdot h_{cj}(t) \cdot \left(\mathbf{x}(t) - \mathbf{w}_j(t)\right)$, wobei

(3.12) $\varepsilon(t) = \varepsilon_{t=0} \cdot \left(\frac{\varepsilon_{t_{\max}}}{\varepsilon_{t=0}}\right)^{\frac{t}{t_{\max}}}$ und

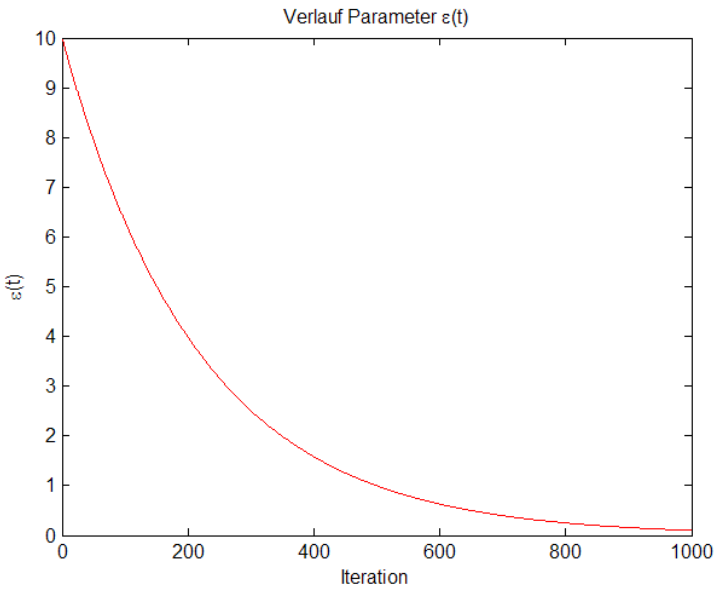


Abbildung 3.2 : Verlauf Parameter $\varepsilon(t)$, RITTER und SCHULTEN

(3.13) $\sigma(t) = \sigma_{t=0} \cdot \left(\frac{\sigma_{t_{\max}}}{\sigma_{t=0}}\right)^{\frac{t}{t_{\max}}}$ (Verlauf ähnlich zur Abbildung 3.2),

[200] Vgl. Ritter und Schulten [1988a].

vorgenommen.[201] Die Ausgabeschicht der verwendeten SOM besteht im Beispiel aus 100 Neuronen, also dem $3\frac{1}{3}$-fachen der Knotenanzahl.

HUETER veröffentlicht im gleichen Jahr einen Tagungsbeitrag[202], in dem Ergebnisse für den in Kapitel 2.3.2 vorgestellten Algorithmus von KOHONEN und die drei Varianten „*Smearing*“, „*Attention*“ und „*Reallocation*“ für Instanzen mit 10 beziehungsweise 30 Knoten vorgestellt werden. Die Anzahl der Neuronen der Ausgabeschicht entspricht in allen Varianten der Anzahl an Knoten *N* der verwendeten Instanz. Der Autor beobachtet für die nicht modifizierte Version der SOM, dass einige Neuronen keinem Knoten zugeordnet werden, während andere Neuronen zwischen zwei Knoten oszillieren. Als Nachbarschaftsfunktion wird jeweils die monoton abnehmende Funktion

$$h_{cj}(t) = \alpha(t) \cdot \begin{cases} \left(1 - \frac{\Delta_{cj}}{L}\right)^{\beta(t)} & ,\Delta_{cj} < L = \frac{N}{3} \\ 0 & ,sonst \end{cases} \quad \text{(s. Abbildung 2.16)} \tag{3.14}$$

mit der Entfernung

$$\Delta_{cj} = \min\left(|c-j|, N - |c-j|\right) \quad \text{(s. Abbildung 2.14)} \tag{3.15}$$

zwischen Neuron *c* und *j* verwendet, wobei der Exponent $\beta(t)$ alle *N* Iterationen um 0,1 erhöht wird, um die Stärke der Anpassung der Neuronen der Nachbarschaft mit zunehmender Iterationszahl zu vermindern.[203] Der Lernparameter $\alpha(t)$ wird im Verlauf des Algorithmus von einem geringen Wert ausgehend schrittweise erhöht[204], so dass das Zentrum immer stärker in Richtung des Eingabevektors verschoben wird.

Die Variante „*Smearing*“ präsentiert anstatt fest vorgegebener Koordinaten für die Positionen der Knoten zufällig bestimmte Koordinaten aus der mit zunehmender Iterationszahl *t* schrumpfenden Umgebung der tatsächlichen Position des Knotens. HUETER verwendet hierzu eine Wahrscheinlichkeitsfunktion, mit deren Hilfe die Entfernung des Eingabevektors vom eigentlichen Standort des Knotens zufällig bestimmt wird.[205] Um den „verschmierten“ Eingabevektor zu erhalten, wird mit Hilfe einer Gleichverteilung zusätzlich der Winkel bestimmt, in dessen Richtung die Entfernung vom Standort aus abzutragen ist.

[201] Vgl. Ritter und Schulten [1988a], S. 110.
[202] Vgl. Hueter [1988].
[203] Vgl. Hueter [1988], S. 86.
[204] Vgl. hierzu Hueter [1988], S. 87.
[205] Vgl. Hueter [1988], S. 86.

In der Variante „*Attention*“ ist die Chance eines Neurons, als Zentrum ausgewählt zu werden, abhängig davon, wie oft das Neuron bereits als Zentrum ausgewählt wurde. HUETER verwendet einen Faktor η_j (bezeichnet als „Gewinn-Frequenz“) in der Distanzfunktion, der die euklidische Distanz für weniger oft ausgewählte Neuronen durch Multiplikation vermindert, so dass diese Neuronen eine höhere Chance erhalten, als Zentrum ausgewählt zu werden. Die Gewinn-Frequenz beschreibt das Verhältnis zwischen der um eine Konstante C erhöhten Anzahl $u_j(t)$, für die das Neuron j bereits als Zentrum ausgewählt wurde, und der mit dem Kehrwert der Anzahl an Neuronen $J=N$ der Ausgabeschicht multiplizierten Summe der entsprechenden Werte für alle Neuronen:

$$(3.16)\quad \eta_j = \frac{u_j(t)+C}{\frac{1}{N}\sum_j \left(u_j(t)+C\right)}.$$

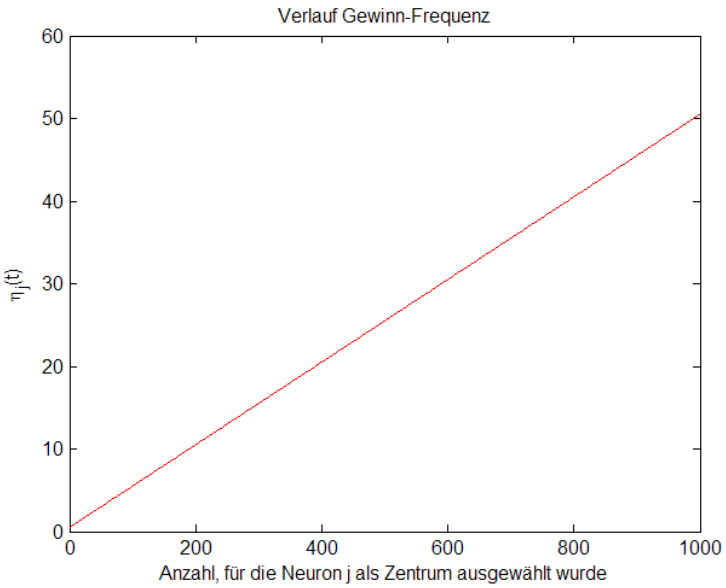


Abbildung 3.3 : Verlauf Gewinn-Frequenz, HUETER

Die Variante „*Reallocation*“ verwendet ebenfalls die Gewinn-Frequenz. Überschreitet die Differenz zwischen der höchsten ($v_{\max}$) und der niedrigsten ($v_{\min}$) Gewinn-Frequenz den Grenzwert

$$(3.17)\quad C \cdot \sqrt{v_{\max}},$$

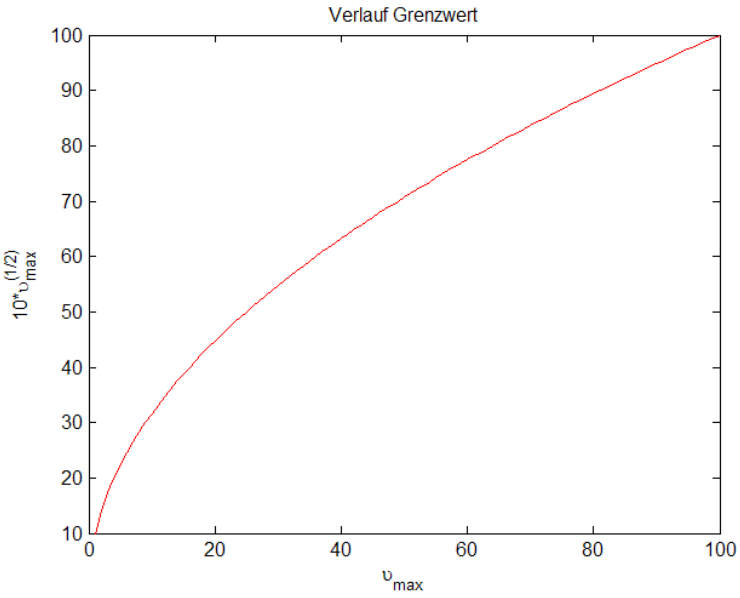


Abbildung 3.4 : Verlauf Grenzwert, HUETER

so wird das Neuron mit der geringsten Frequenz entfernt und das Neuron mit der höchsten Frequenz dupliziert, um die Verteilung der Neuronen an die Verteilung der Knoten anzupassen. Alle untersuchten Erweiterungen weisen eine höhere Lösungsgüte als die Grundversion auf und verbessern das Konvergenzverhalten. Die Variante „*Attention*“ wird von HUETER für die Instanz mit 10 Knoten als die beste Variante hinsichtlich des Konvergenzverhaltens identifiziert.[206] Auf die Instanz mit 30 Knoten wendet der Autor eine Variante der Self-Organizing Map an, die alle oben beschriebenen Erweiterungen enthält. Für diese konnte eine kürzere Route als mit dem Spacefilling Curve-Algorithmus[207] bestimmt werden, allerdings ermittelt eine Simulated Annealing-Prozedur eine bessere Lösung.[208]

ANGENIOL ET AL.[209] entwickeln einen Prozess, der Neuronen zu der ringförmig angeordneten Ausgabeschicht des Netzes hinzufügt beziehungsweise löscht. Der Algorithmus startet zunächst mit nur einem Neuron in der Ausgabeschicht. Die zu besuchenden Knoten werden dem Netz in Zyklen präsentiert. Innerhalb eines solchen Präsentationszyklus wird jeder Knoten genau einmal als Eingabevektor ausgewählt. Die Reihenfolge der Präsentation wird zu Beginn des Algorithmus zufällig bestimmt und bleibt in jedem Zyklus identisch. Ein Neuron wird dann als direkter Nachbar eines Neurons eingefügt,

[206] Vgl. Hueter [1988], S. 87.
[207] Vgl. hierzu Bartholdi und Platzman [1982] sowie Platzman und Bartholdi [1989].
[208] Vgl. Hueter [1988], S. 88.
[209] Vgl. Angeniol et al. [1988].

wenn es innerhalb eines Präsentationszyklus als Zentrum für zwei verschiedene Knoten ausgewählt wird. Ein Neuron wird entfernt, wenn es innerhalb von drei Zyklen nicht als Zentrum für einen Knoten ausgewählt wurde. Als Distanzfunktion wird die quadrierte euklidische Distanz verwendet[210], zur Anpassung der Elemente der Gewichtsvektoren wird die Entfernung zwischen zwei Neuronen des Ringes gemäß Gleichung *(3.15)* bestimmt. Die Nachbarschaftsfunktion

$$(3.18)\quad h_{cj}\left(G(t_p)\right)=\left(\frac{1}{\sqrt{2}}\right)\cdot\exp\left(\frac{-\Delta_{cj}^{\,2}}{G(t_p)^2}\right)$$ (Verlauf ähnlich zur Abbildung 2.15)

verwendet den Parameter

$$(3.19)\quad G(t_p+1)=(1-\delta)\cdot G(t_p),\ \delta>0,$$

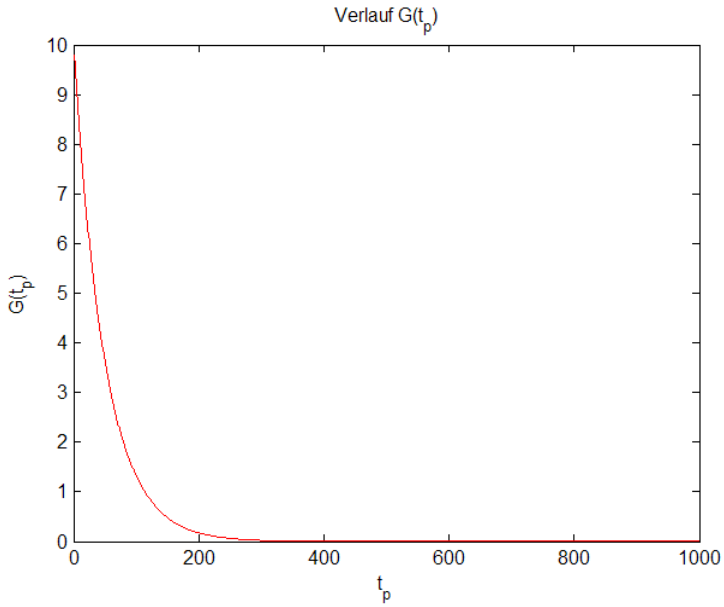


Abbildung 3.5: Verlauf Parameter $G(t_p)$, ANGENIOL ET AL.

der jeweils am Ende eines Präsentationszyklus vermindert wird. Die Anpassung der Elemente der Gewichtsvektoren erfolgt schließlich gemäß

$$(3.20)\quad \mathbf{w}_j(t+1)=\mathbf{w}_j+h_{cj}\left(G(t_p)\right)\cdot\left(\mathbf{x}(t)-\mathbf{w}_j(t)\right)$$ (s. Abbildung 2.17 (a)).

Die Ergebnisse des Algorithmus für fünf verschiedene Testinstanzen mit jeweils 50 Orten werden mit der besten bekannten Lösung und der Elastic Net-Methode[211] (EN) verglichen. Der vorgeschlagene Algorithmus weist dabei eine der EN-Methode überlegene Lösungsgüte auf, wenn man die jeweils beste ermittelte Lösung zum Vergleich

[210] Dieses Vorgehen erspart eine Rechenoperation (Radizieren), führt aber zum gleichen Ergebnis bei der Wahl des Zentrums.

[211] Vgl. zum Elastic Net Durbin und Willshaw [1987]. Eine detaillierte Analyse des Verfahrens für das TSP geben Durbin et al. [2001], einen hierarchischen Ansatz präsentieren Vakhutinsky und Golden [1995].

heranzieht, jedoch eine geringere Güte, wenn man die durchschnittlich ermittelten Rundreiselängen betrachtet:

Instanz (jeweils 50 Knoten)	**Beste bekannte Lösung**	**Elastic Net**	**Bestes Ergebnis**	**Bestes Ergebnis in 10 Läufen** δ =0,2	**Mittlere Länge** δ =0,2	**Mittlere Länge** δ =0,02
1	5,84	5,98	5,836	5,95	6,06	5,99
2	5,99	6,09	5,995	6,07	6,25	6,18
3	5,57	5,70	5,575	5,69	5,83	5,77
4	5,70	5,86	5,596	5,72	5,87	5,96
5	6,17	6,49	6,194	6,52	6,70	6,58

Tabelle 3.4 : Ergebnisse von ANGENIOL ET AL.[212]

FAVATA und WALKER verwenden in ihrem Artikel[213] eine Normalisierungskomponente als dritte Komponente des Eingabevektors, die Eingabeschicht weist somit drei Neuronen auf. Die Normalisierungskomponente stellt dabei sicher, dass alle Eingabevektoren die gleiche Länge aufweisen und keine Paarung zweier Eingabevektoren kollinear ist. Die Autoren fassen die Koordinaten der Knoten so als zweidimensionale Projektion eines dreidimensionalen Vektors auf, der einen Punkt auf einer Kugel beschreibt. Die Verschiebung der Gewichtsvektoren der Neuronen der Nachbarschaft des Neurons mit der maximalen Aktivierung erfolgt gemäß

$$\mathbf{w}_j(t+1) = \frac{\mathbf{w}_j(t) + \alpha(t) \cdot \mathbf{x}(t)}{\left\| \mathbf{w}_j(t) + \alpha(t) \cdot \mathbf{x}(t) \right\|} \quad (3.21)$$

[214]

Bei dieser Adaptionsvorschrift wird der mit dem im Iterationsverlauf sinkenden Skalar $\alpha(t)$ multiplizierte Eingabevektor zum bisherigen Gewichtsvektor addiert und der resultierende Vektor normiert. Diese Variante mit ringförmiger Anordnung der Ausgabeschicht wird anhand verschiedener Testinstanzen mit 30 bis 10.000 Knoten getestet. Mit zunehmender Anzahl an Iterationen sinken die durchschnittliche Länge der ermittelten Lösungen und die zugehörige Standardabweichung:[215]

[212] Entnommen aus Angeniol et al. [1988], S. 291.
[213] Vgl. Favata und Walker [1991].
[214] Die Normalisierungsoperation führt zu einer signifikanten Erhöhung der Rechenzeit, vgl. Kohonen [1990], S. 1470.
[215] Vgl. Favata und Walker [1991], S. 466.

Iterationen	∅-Länge	Standardabweichung	Bestes Ergebnis
2.500	103,5	11,0	75,60
5.000	39,1	2,3	36,75
10.000	30,5	0,75	29,42
25.000	29,1	0,6	28,05
50.000	28,2	0,5	27,31
100.000	27,6	0,3	26,82
Simulated Annealing	-	-	25,95

Tabelle 3.5 : Ergebnisse von FAVATA und WALKER für eine Instanz mit 1.000 Knoten

Die Autoren stellen außerdem fest, dass das Verhältnis zwischen der Anzahl Iterationen, die notwendig ist, um ein definiertes, hinreichendes Ergebnis zu erzielen, und der Problemgröße *N* mit zunehmender Problemgröße sinkt.[216] Im Vergleich mit dem Verfahren des Simulated Annealing ergibt sich eine im Mittel um 5% längere Rundreise bei Verwendung der Self-Organizing Map, allerdings berichten FAVATA und WALKER von einer geringeren Laufzeit des Algorithmus. So benötigt Simulated Annealing beispielsweise die 10-fache Laufzeit bei Anwendung auf eine Testinstanz mit 10.000 Orten.[217]

FRITZKE und WILKE veröffentlichen 1991 Untersuchungen der von den Autoren entwickelten SOM-Variante „FLEXMAP“, die einen Rechenzeitbedarf der Ordnung *O(N)* aufweist.[218] Ihr Algorithmus startet mit drei Neuronen in der Ausgabeschicht. Nach einer fest vorgegebenen Anzahl an Iterationen wird ein weiteres Neuron in einem Gebiet eingefügt, in dem das Verhältnis zwischen Neuronen und zu besuchenden Knoten minimal ist. Jedem Neuron der Ausgabeschicht wird hierzu eine Variable zugeordnet, die das Verhältnis zwischen der Anzahl an Neuronen und der Anzahl an zu besuchenden Knoten in der Umgebung des Neurons beschreibt. Jedem Knoten wird außerdem das Neuron zugeordnet, das als Zentrum für diesen Knoten ausgewählt wurde. Die Linearisierung der Zeitkomplexität wird dadurch erreicht, dass die Suche nach dem Zentrum für einen Knoten auf das bisherige Zentrum und dessen Nachbarschaft eingegrenzt wird. Darüber hinaus wird ein Neuron einem Knoten endgültig zugeordnet und nimmt nicht mehr an

[216] Vgl. Favata und Walker [1991], S. 466 f. Als hinreichendes Ergebnis wird hier eine Lösung definiert, deren Zielfunktionswert um weniger als 10 Prozent von der Lösung einer Simulated Annealing- Prozedur abweicht.
[217] Vgl. Favata und Walker [1991], S. 467.
[218] Vgl. Fritzke und Wilke [1991].

folgenden Adaptionsprozessen teil, wenn es für eine vorgegebene Anzahl als Zentrum für diesen Knoten ausgewählt wurde. Adaptionsprozesse werden auf das Zentrum und dessen direkte Nachbarn beschränkt. Ein Vergleich mit der Rechenzeit eines 2-Opt-Verbesserungsverfahrens, das die Zeitkomplexität $O(N^2)$ aufweist, zeigt, dass ab einer Problemgröße von 700 Orten das vorgeschlagene Verfahren geringere Rechenzeiten aufweist.[219] Die Lösungsqualität des 2-Opt-Verfahrens ist allerdings für größere Instanzen überlegen, die maximale Abweichung des Zielfunktionswertes der SOM-Lösungen vom Zielfunktionswert der Optimallösung beträgt etwa 10%.[220]

BURKE und DAMANY[221] stellen 1992 das „Guilty Net" vor, bei dem die Anzahl der Neuronen der Ausgabeschicht mit der Anzahl N der Knoten der Testinstanz übereinstimmt. Der Algorithmus bricht ab, wenn sich die Neuronen in einer vordefinierten Umgebung der Knoten befinden und sich die Gewinner der Wettbewerbe innerhalb einer vorgegebenen Iterationszahl[222] nicht mehr verändern oder eine vorgegebene Anzahl an Iterationen erreicht wird.
Das Guilty Net verwendet eine Distanzfunktion, die zur euklidischen Distanz das mit einer Konstanten C gewichtete Verhältnis zwischen der Anzahl der vom Neuron gewonnenen Wettbewerbe win_j und der um eine Einheit erhöhten Anzahl der bisher durchgeführten Wettbewerbe addiert[223] :

$$(3.22)\quad d_j = \left\| \mathbf{x}(t) - \mathbf{w}_j(t) \right\| + C \cdot \left(\frac{win_j}{1 + \sum_{j=I+1}^{I+J} win_j} \right).$$

[219] Es sind allerdings 2-Opt Implementierungen möglich, die einen geringeren Rechenaufwand verursachen, vgl. Johnson und McGeoch [1997], S. 238 ff. und S. 308.
[220] Vgl. Fritzke und Wilke [1991], S. 932 f.
[221] Vgl. Burke und Damany [1992].
[222] Eine vorgegebene Iterationszahl wird nicht explizit angegeben. Da dieses Vorgehen aber als notwendig erscheint, wird hier davon ausgegangen, dass eine entsprechende Iterationszahl vorgegeben werden muss.
[223] Vgl. Burke und Damany [1992], S. 261.

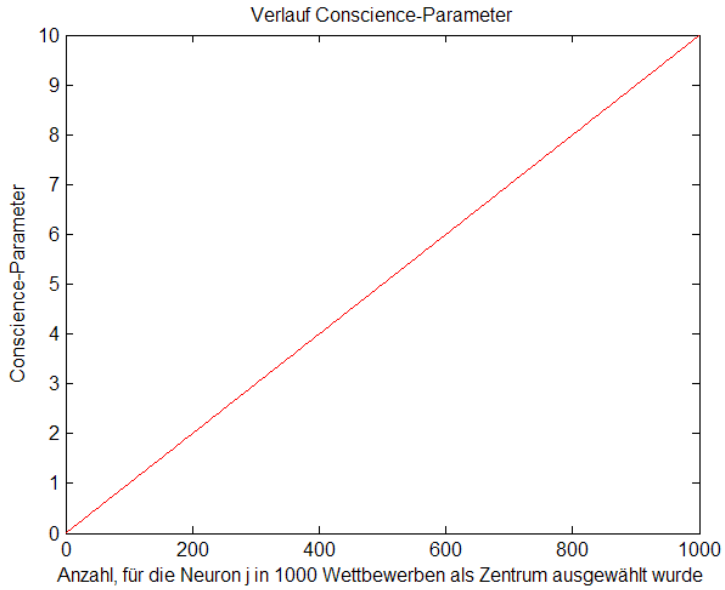


Abbildung 3.6 : Verlauf Conscience-Parameter, BURKE und DAMANY (*C=10*)

Der zusätzliche Term wird als Conscience-Parameter bezeichnet. Der vorgeschlagene Mechanismus erhöht die euklidische Distanz für Neuronen, die im bisherigen Verlauf des Algorithmus häufiger gewonnen haben stärker als für Neuronen, die bisher selten einen Wettbewerb gewonnen haben. Diese Form des „Erinnerungsvermögens“ soll zu einer Separation der Neuronen der Ausgabeschicht beitragen, so dass jedes Neuron eindeutig einem Knoten zugewiesen wird.[224] Die Definition der Nachbarschaftsfunktion erfolgt in Analogie zur Definition von HUETER[225] mit $L=N/2$, der Parameter $\beta(t)$ wird am Ende eines Präsentationszyklus auf das 1,1-fache des jeweiligen Ausgangswertes erhöht. Die Ergebnisse der Untersuchungen zeigen, dass das Guilty Net im Vergleich mit alternativen Lösungsverfahren nur eine mittelmäßige Lösungsqualität aufweist (vgl. Zielfunktionswerte in Tabelle 3.6), dafür aber eine weitaus geringere Anzahl an Iteration benötigt (Simulated Annealing: 1.000.000 Iterationen, Elastic Net: 7000 Iterationen, Guilty Net: 500 Iterationen):[226]

[224] Die Grundidee basiert auf der von DeSieno [1988] vorgeschlagenen Modifikation der SOM, in der erstmals ein „Erinnerungsvermögen“ modelliert wird. Im Gegensatz zu Hueter [1988], der die Gewinn-Frequenz multiplikativ mit der euklidischen Distanz verknüpft, wird bei DeSieno ebenfalls ein Term addiert.
[225] Vgl. S. 56.
[226] Vgl. Burke und Damany [1992], S. 257 und S. 264.

Verfahren	Anzahl Knoten			
	10	30	50	100
Theoretisches Optimum[227]	2,43	4,21	5,43	7,68
Simulated Annealing	2,74	4,75	6,13	8,40
Spacefilling Curve	3,14	5,45	7,03	9,56
Hopfield-Tank-KNN[228]	2,92	4,12	6,64	-
Elastic Net	2,45	4,51	5,58	8,23
Guilty Net	2,78	4,81	6,21	8,81

Tabelle 3.6 : Ergebnisse von BURKE und DAMANY

MATSUYAMA[229] verwendet eine mit zunehmender Iterationszahl ansteigende, nach oben beschränkte Lernrate mit dem Ziel, auch für Instanzen mit einer hohen Anzahl an Knoten und bei unregelmäßiger Verteilung der Knoten gute Lernergebnisse sicherzustellen. Verbindungen zwischen den direkt benachbarten Neuronen führen dazu, dass die Verschiebung eines Neurons auch von den Positionen der Nachbarn abhängt. MATSUYAMA verwendet als Distanzfunktion die quadrierte euklidische Distanz und die folgende Anpassungsvorschrift für die Gewichtsvektoren der Neuronen der Nachbarschaft:

$$\mathbf{w}_j(t+1) = \mathbf{w}_j(t) + \varepsilon(t)\cdot h_{cj}(t)\cdot H\left(\Delta_{cj} \le L(t)\right)\cdot\left(\mathbf{x}(t) - \mathbf{w}_j(t)\right) + \lambda(t)\cdot\left[\left(\mathbf{w}_{j+1}(t) - \mathbf{w}_j(t)\right) + \left(\mathbf{w}_{j-1}(t) - \mathbf{w}_j(t)\right)\right] \quad (3.23)$$

Hier stellt j den Index des zu verschiebenden Gewichtsvektors dar, $L(t)$ eine mit zunehmender Iterationszahl t sinkende maximale Entfernung zum Zentrum c, welche die Nachbarschaft begrenzt: Ist die Entfernung Δ_{cj} vom Zentrum größer als

$$L(t) = 3\cdot\sigma(t), \text{ mit} \quad (3.24)$$

$$\sigma(t) = \sigma(0)\cdot(1-C_1)^{\lfloor t/N \rfloor}, \quad (3.25)$$

[227] Mit Hilfe der Näherungsformel $0{,}765\cdot\sqrt{N}$ geschätzter Wert. Vgl. hierzu Stein [1977] und Johnson und McGeoch [1977], S. 225.
[228] Vgl. hierzu Hopfield und Tank [1985].
[229] Vgl. Matsuyama [1991] und Matsuyama [1992].

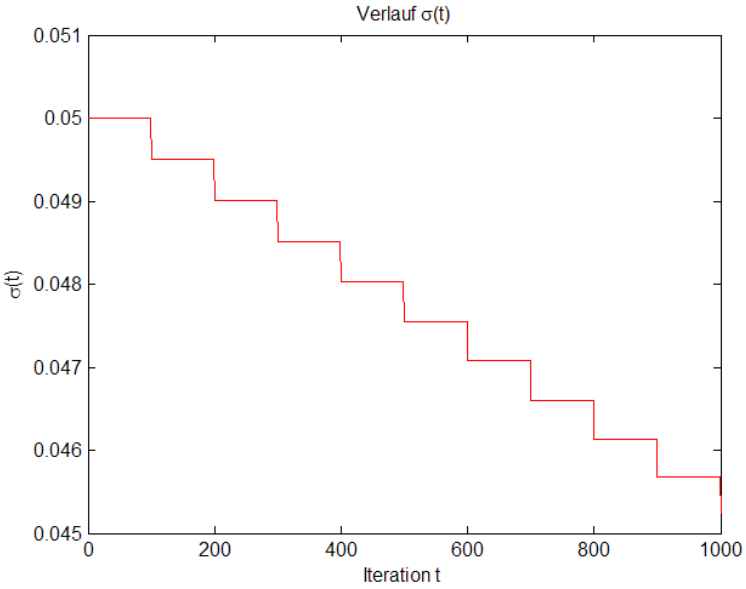


Abbildung 3.7 : Verlauf $\sigma(t)$, MATSUYAMA

so nimmt die Sprungfunktion $H\left(\Delta_{cj} \le L(t)\right)$ den Wert Null, ansonsten den Wert Eins an. Der Parameter

$$(3.26)\quad \lambda(t) = \max\left\{\lambda(0) - C_2 \cdot t, \lambda_{\min}\right\} > 0$$

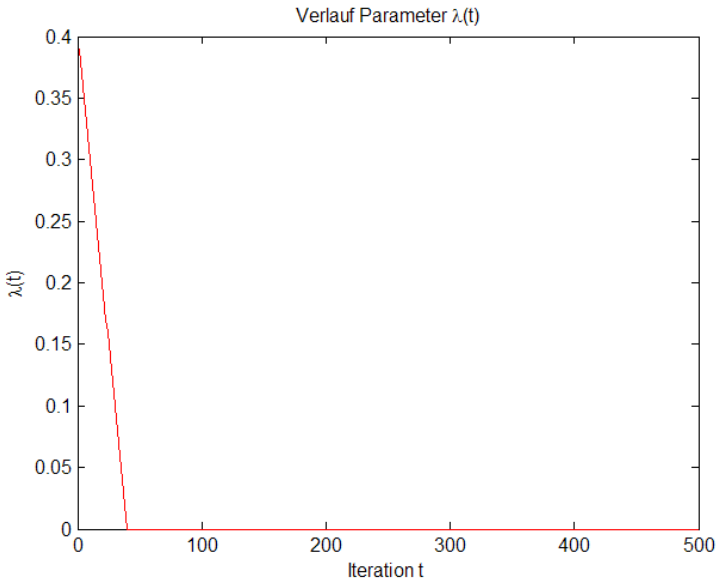


Abbildung 3.8 : Verlauf Parameter $\lambda(t)$, MATSUYAMA

sinkt mit zunehmender Iterationszahl t auf ein vorgegebenes Minimum $\lambda_{\min}$ ab, während der Parameter

$$(3.27)\quad \varepsilon(t) = \min\left\{\varepsilon(0) \cdot (1 + C_3)^{\lfloor t/N \rfloor}, \varepsilon_{\max}\right\}$$

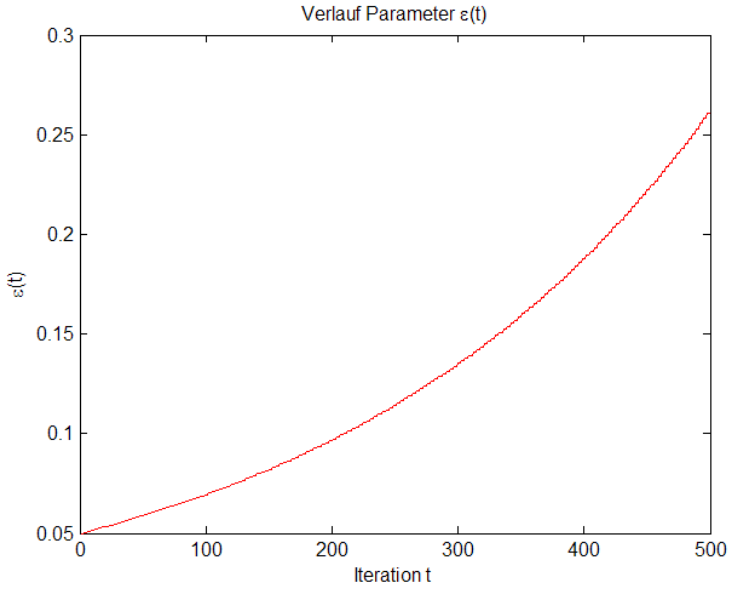


Abbildung 3.9: Verlauf Parameter $\varepsilon(t)$, MATSUYAMA

auf ein vorgegebenes Maximum $\varepsilon_{\max}$ ansteigt. C_1, C_2 und C_3 stellen positive Konstanten dar, die vorgegeben werden müssen $(C_1, C_2, C_3 << 1)$. Die ersten beiden Terme der Anpassungsvorschrift stehen in Analogie zu den bereits vorgestellten Varianten der SOM, der letzte Term in dieser Gleichung wird auch in der Elastic Net-Methode verwendet. Er kann wiederum als Kraft interpretiert werden, die eine „Spannung" innerhalb des Neuronenringes definiert,[230] deren Einfluss im Laufe des Verfahrens zügig abgesenkt wird. Als Nachbarschaftsfunktion $h_{cj}(t)$ verwendet MATSUYAMA eine Gauß-Funktion der Form *(3.14)*, die Präsentation der Eingabevektoren erfolgt zufällig ohne Verwendung von Präsentationszyklen. Der Algorithmus bricht ab, wenn jedem Knoten genau ein Neuron der Ausgabeschicht zugeordnet wird, das keinem anderen Knoten zugeordnet wurde. Insgesamt entspricht die Zahl der Neuronen etwa dem Vierfachen der Knotenanzahl. Während für 500 Testinstanzen mit gleichverteilten Orten Simulated Annealing eine geringfügig bessere Lösungsqualität als die SOM aufweist, ist die Lösungsqualität der Variante von MATSUYAMA der Elastic Net-Methode in 203 Fällen über- und in 175 Fällen unterlegen, in 122 Fällen wird eine identische Lösungsqualität erreicht.[231] Für ein Problem mit ungleichmäßig verteilten Orten weist die SOM-Variante eine bessere Lösungsgüte als das Elastic Net und Simulated Annealing auf, die Gesamtlänge der Optimallösung wird um 3,74% überschritten. MATSUYAMA veröffentlicht

[230] Vgl. Abbildung 2.17 (b), S. 32.
[231] Vgl. Matsuyama [1991], S. 387 f. und Matsuyama [1992], S. 106.

1995 einen Tagungsbeitrag[232], in dem eine Variante der Self-Organizing Map vorgestellt wird, die im wesentlichen mit der 1992 vorgestellten Variante der Self-Organizing Map übereinstimmt. Erneut vergleicht MATSUYAMA die Ergebnisse mit dem Simulated Annealing-Ansatz. In den meisten Fällen ergibt sich eine ähnliche Lösungsgüte, insgesamt ist die SOM-Variante jedoch unterlegen.

BURKE veröffentlicht 1994 in einem Tagungsbeitrag[233] eine empirische Untersuchung für zwei Varianten der SOM, das bereits beschriebene Guilty Net und das „Vigilant Net". Auch beim Vigilant Net wird ein Netz verwendet, bei dem die Anzahl der Neuronen der Ausgabeschicht der Anzahl der Knoten der Testinstanz entsprechen. Beide Varianten brechen ab, wenn sich die einem Knoten zugeordneten Neuronen in einer vordefinierten Umgebung der Koordinaten der Knoten befinden und sich die Gewinner der Wettbewerbe innerhalb einer vorgegebenen Iterationszahl nicht mehr verändern[234]. Das Vigilant Net verwendet im Unterschied zum Guilty Net für jedes Neuron der Ausgabeschicht einen zusätzlichen „Vigilant"-Parameter, der festhält, wie oft ein Neuron in einem Präsentationszyklus bereits als Zentrum ausgewählt wurde. Überschreitet der Wert eine festgelegte Obergrenze, so kann es nicht mehr als Zentrum ausgewählt werden.[235] Vor dem Start des folgenden Präsentationszyklus werden die Einträge aller Vigilant-Parameter auf den Wert Null gesetzt. Dieser Ansatz wird - ähnlich wie der Conscience-Mechanismus des Guilty Net-Ansatzes - eingesetzt, um eine Separation der Neuronen im Eingaberaum zu erreichen. Für Instanzen mit gleichmäßig verteilten Orten unterliegt das Vigilant Net im Vergleich mit dem Guilty Net, für Instanzen mit nicht gleichmäßig verteilten Orten weisen die Ansätze eine ähnliche Lösungsqualität auf.[236] Im Vergleich zwischen Vigilant Net und einer SOM-Variante ohne Separationsmechanismus zeigt sich, dass der Vigilant Net-Ansatz bessere Ergebnisse erzielt.[237] BURKE untersucht darüber hinaus die Abhängigkeit der Lösungsgüte von den in der Initialisierungsphase des Algorithmus festgelegten Parameterwerten und der Anordnung der Gewichtsvektoren der Neuronen der Ausgabeschicht. Die Autorin leitet aus den Untersuchungsergebnissen

[232] Vgl. Matsuyama [1995].
[233] Vgl. Burke [1994].
[234] Die Vorgabe der Iterationszahl wird wiederum unterstellt.
[235] Burke lässt zunächst alle Neuronen am Wettbewerb teilnehmen. Wenn das gewinnende Neuron den vorgegebenen Wert des Vigilant-Parameters erreicht, wird der Wettbewerb unter Ausschluss des Neurons wiederholt. Auf diese Operation kann verzichtet werden, wenn alle Neuronen, die den Vigilant-Parameterwert erreichen würden, vom Wettbewerb ausgeschlossen werden.
[236] Vgl. Burke [1994], S. 688.
[237] Vgl. Burke [1994], S. 687.

ab, dass die Anordnung der Gewichtsvektoren entlang der konvexen Hülle um die zu besuchenden Knoten eine Verbesserung der Lösungsqualität bewirkt.
Verbesserungen des Vigilant Net untersucht die Autorin 1995.[238] Im entsprechenden Artikel werden die Parameterwerte des Netzes optimiert und ein selbststeuernder Mechanismus zur dynamischen Festlegung des Vigilant-Parameters in Abhängigkeit von der Veränderung der Rundreisenlänge untersucht. Die Ergebnisse für acht verschiedene Testinstanzen werden mit dem Spacefilling Curve-Ansatz[239] und einer Multiple Nearest Neighbor[240] (MNN) Heuristik verglichen. Das Vigilant Net ermittelt für alle Instanzen kürzere Touren als die Spacefilling Curve (höheren Rechenzeiten des Vigilant Net) und längere Touren als der MNN-Ansatz (bei geringeren Rechenzeiten des Vigilant Net für große Testinstanzen)[241].

In AMINs Variante[242] der SOM stimmt die Anzahl der Neuronen in der Ausgabeschicht zu Beginn des Algorithmus mit der Zahl der Knoten der Testinstanz überein. Die Anzahl der Neuronen wird allerdings - ähnlich zum Ansatz von ANGENIOL ET AL.[243] - dynamisch verändert. Im Unterschied zu ANGENIOL ET AL. weisen die Präsentationszyklen unterschiedliche (zufällig bestimmte) Reihenfolgen auf. Der Autor verwendet zur Bestimmung des Zentrums die Gravitationskraft

$$(3.28)\quad F_{mj} = C \cdot \frac{KMasse_m \cdot NMasse_j}{\left\| \mathbf{x}_m - \mathbf{w}_j \right\|^2},$$

[238] Vgl. Burke [1996].
[239] Vgl. Platzman und Bartholdi [1989].
[240] Vgl. Burke [1996], S. 123.
[241] Vgl. Burke [1996], S. 128.
[242] Vgl. Amin [1994].
[243] Vgl. S. 58.

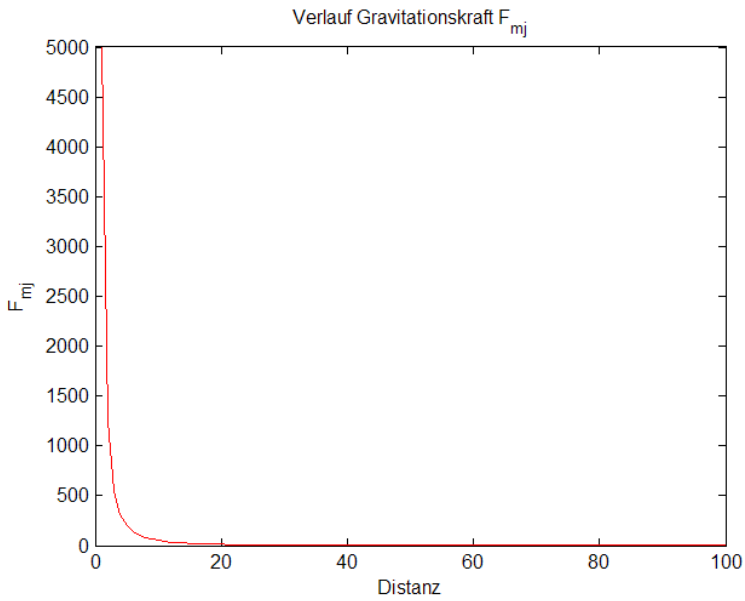


Abbildung 3.10 : Beispielhafter Verlauf Gravitationskraft, AMIN

wobei $KMasse_m$ ein dem Knoten m und $NMasse_j$ ein dem Neuron j zugeordnetes Gewicht und C eine Konstante darstellt. Die Adaption des Gewichtsvektors des Zentrums erfolgt dann durch

$$\mathbf{w}_j(t+1) = \mathbf{w}_j(t) + \frac{1}{\sqrt{2}} \cdot \left(\mathbf{x}(t) - \mathbf{w}_j(t)\right), \tag{3.29}$$

die Anpassung von insgesamt sechs Neuronen der Nachbarschaft erfolgt gemäß

$$\mathbf{w}_{j+h}(t+1) = \mathbf{w}_j(t) + \frac{1}{h} e^{\frac{h}{T(t)}} \cdot \left(\mathbf{x}(t) - \mathbf{w}_j(t)\right), \text{ für } h = 1(1)3, \text{ bzw.}$$

$$\mathbf{w}_{j-h}(t+1) = \mathbf{w}_j(t) + \frac{1}{h} e^{\frac{h}{T(t)}} \cdot \left(\mathbf{x}(t) - \mathbf{w}_j(t)\right), \text{ für } h = 1(1)3, \text{ wobei} \tag{3.30}$$

$$T(t+1) = C \cdot T(t), C < 1. \tag{3.31}$$

AMIN wendet das Verfahren auf eine Reihe von Testinstanzen mit bis zu 30.000 Knoten an. Bei einem Vergleich für Testinstanzen mit bis zu 3000 Knoten unterliegt das Simulated Annealing-Verfahren in allen Instanzen sowohl hinsichtlich der Lösungsqualität als auch bezüglich der Rechenzeiten. Es zeigt sich außerdem, dass im Gegensatz zum Simulated Annealing, bei dem mit zunehmender Größe der Probleminstanz eine Verschlechterung der Lösungsqualität eintritt, der mit dem SOM-Ansatz produzierte Fehler bei wachsender Problemgröße geringere Zuwächse aufweist.[244]

[244] Vgl. Amin [1994], S. 132.

BUDINICH[245] präsentiert eine Variante der SOM zur Lösung von TSPen mit gleichmäßig verteilten Orten. Die Neuronenzahl in der Ausgabeschicht ist mit der Anzahl der Knoten N der untersuchten Instanz identisch. Der Lernparameter nimmt mit zunehmender Entfernung vom Zentrum zunächst von 0,8 ausgehend linear ab und wird ab einer bestimmten, im Laufe des Verfahrens immer geringer werdenden Entfernung auf den Wert Null gesetzt. Gleichzeitig wird auch der Maximalwert des Lernparameters im Verlauf des Verfahrens vermindert, bis er schließlich in der letzten Iteration den Wert Null annimmt. Insgesamt werden $50 \cdot N$ Iterationen durchgeführt.
Ein wesentliches Unterscheidungsmerkmal des Ansatzes ist, dass ein Knoten nicht zwangsläufig einem Neuron des Ringes zugeordnet wird, sondern einer beliebigen Position auf dem Ring. Dies erreicht BUDINICH, indem er eine Kombination der mit den zugehörigen Aktivitäten gewichteten Koordinaten des Zentrums und den Koordinaten der benachbarten Neuronen bildet.[246]
Ein Vergleich mit Simulated Annealing zeigt, dass die ermittelten Rundreiselängen bis zu einer Instanzgröße von 500 Knoten maximal um +5% abweichen, ab dieser Grenze ermittelt die SOM-Variante durchgehend bessere Ergebnisse, die in etwa einem Zehntel der Rechenzeit des Simulated Annealing Algorithmus ermittelt werden können. Ein Vergleich mit der Elastic Net-Methode für fünf unterschiedliche Instanzen zeigt, dass die Verfahren etwa gleichwertige Ergebnisse erzielen.

SOMHOM ET AL.[247] benutzen zur Lösung des TSP einen Neuronenring, dessen Neuronenzahl der doppelten Knotenzahl der jeweiligen TSP-Instanz entspricht. Innerhalb eines Präsentationszyklus mit zufällig bestimmter Präsentationsreihenfolge kann jedes Neuron nur einmal als Zentrum ausgewählt werden. Die Autoren verwenden eine Gauß-Funktion zur Bestimmung der Stärke der Anpassung der Gewichtsvektoren der Nachbarschaft, wobei die Nachbarschaft auf eine vorgegebene Distanz-Obergrenze beschränkt wird. Der Algorithmus bricht ab, wenn sich die Gewichtsvektoren der Neuronen in hinreichend geringer Entfernung zu den Eingabevektoren befinden. Ein Vergleich mit der Lösungsqualität des Guilty Net, des Elastic Net und der 1992 von MATSUYAMA vorgeschlagenen SOM-Variante zeigt für die untersuchten Testinstanzen, die

[245] Vgl. Budinich [1996].
[246] Vgl. Budinich [1996], S. 420.
[247] Vgl. Somhom et al. [1997].

der TSPLIB-Bibliothek[248] entnommen wurden, dass die vorgeschlagene Vorgehensweise den übrigen Methoden überlegen ist.[249] Die Abweichungen der ermittelten Gesamtlänge von der jeweils besten bekannten Lösung für die Instanzen betragen im Mittel 1,22% und im schlechtesten Fall 2,17%, während sich für das Vorgehen von MATSUYAMA eine mittlere Abweichung von 5,13% und für das Guilty Net von 6,26% ergeben. Der Elastic Net-Ansatz ist allen anderen Methoden unterlegen. Der vorgeschlagene Ansatz weist außerdem geringere Rechenzeiten als die Vergleichsverfahren auf. Die Autoren stellen fest, dass die zufällig bestimmte Reihenfolge der Präsentation der Eingabevektoren im Vergleich zu einer festen Präsentationsreihenfolge im Mittel zu einer 2% kürzeren Gesamtlänge führt. Identische Ergebnisse werden auch in MODARES ET AL.[250] präsentiert.

ARAS ET AL. integrieren statistische Eigenschaften der Menge der Eingabevektoren in den Adaptionsprozess der Gewichtsvektoren[251], der aus zwei Modulen besteht. Während durch das *attraction module* das Zentrum und die Neuronen der Nachbarschaft in Richtung des präsentierten Eingabevektors verschoben werden, führt das *dispersion module* zu einer Verschiebung der Neuronen, die nicht zur Nachbarschaft gezählt werden. Diese Verschiebung wird für die SOM-Variante KNIES-TSP so gestaltet, dass der Mittelwert der Neuronenpositionen mit dem Mittelwert der Eingabevektoren, die durch die Neuronen der Nachbarschaft repräsentiert werden, übereinstimmt. Die zweite Variante KNIES-TSP-GLOBAL verwendet ein *dispersion module*, das den Mittelwert der Neuronenpositionen an den Mittelwert aller Eingabevektoren anpasst. Die Autoren integrieren den Einfüge- und Vernichtungsprozess von ANGENIOL ET AL. und verwenden eine Nachbarschaftsfunktion der Form *(3.18)*.[252] Die besten ermittelten Ergebnisse für fünfzehn Testinstanzen der TSPLIB-Bibliothek weisen im Mittel ein Abweichung der Rundreisenlänge von der Optimallösung von 2,73% für die Variante KNIES-TSP beziehungsweise 3,17% für die Variante KNIES-TSP-GLOBAL auf. Ein Vergleich mit dem Guilty Net, der Variante von ANGENIOL ET AL. und einer Variante ohne Einfüge- und Vernichtungsprozedur zeigt, dass der Ansatz von ARAS ET AL. eine überlegene Lösungsgüte aufweist.[253]

[248] Eine Beschreibung der Bibliothek TSPLIB (*Traveling Salesman Problem Library)* findet sich in Reinelt [1991].
[249] Vgl. Somhom et al. [1997], S. 925 f, und Modares et al. [1999], S. 601 f.
[250] Vgl. Modares et al. [1999].
[251] Vgl. Aras et al [1999].
[252] Vgl. S. 58 f.
[253] Vgl. Aras et al. [1999], S. 1281.

COCHRANE und BEASLEY stellen in ihrem 2003 veröffentlichten Artikel[254] das Co-Adaptive Net, eine weitere Variante der SOM mit einer Zeitkomplexität der Ordnung $O(N^2 logN)$, vor. Die Autoren verwenden eine Ausgabeschicht, deren Neuronenzahl dem 2,5-fachen der Knotenzahl der jeweiligen Probleminstanz entspricht. Weitere Merkmale der Variante stellen die Beschränkung bei der Suche nach dem Zentrum auf das bisherige Zentrum für den Eingabevektor und Neuronen in dessen Umgebung[255] sowie die Verwendung von Präsentationszyklen mit zufälliger Präsentationsfolge der Knoten dar. Wird innerhalb des Wettbewerbs der Neuronen durch Vergleich der euklidischen Distanzen zum Eingabevektor ein Neuron als Zentrum ausgewählt, das innerhalb des aktuellen Präsentationszyklus bisher nicht als Zentrum ausgewählt wurde, werden die Gewichtsvektoren des entsprechenden Neurons und der Neuronen in dessen Nachbarschaft adaptiert. Wurde das Neuron bereits einmal als Zentrum deklariert, so werden nur die Gewichtsvektoren der Neuronen in dessen Nachbarschaft adaptiert, wurde das Neuron bereits mehr als einmal als Zentrum ausgewählt, werden keinerlei Anpassungen der Gewichtsvektoren vorgenommen. Der beschriebene Prozess wird nur solange durchgeführt, bis der sogenannte Gain-Parameter G_t, der die Stärke der Gewichtsadaption beeinflusst und im Verlaufe des Verfahrens vermindert wird, eine vorgegebene Grenze G_{cross} unterschreitet. In dem Fall wird der Wettbewerbsprozess durch einen als *Kooperation* bezeichneten Prozess ersetzt, der dadurch gekennzeichnet ist, dass Adaptionsprozesse für das Zentrum und dessen benachbarte Neuronen nur ausgeführt werden, wenn das Neuron zum ersten Mal im aktuellen Präsentationszyklus als Zentrum ausgewählt wurde. COCHRANE und BEASLEY modellieren die Stärke der Anpassung eines Gewichtsvektors in Abhängigkeit von der euklidischen Distanz zwischen Gewichtsvektor und dem präsentierten Eingabevektor, um die Entfaltung des Neuronenringes zu beschleunigen[256]. Am Ende eines Präsentationszyklus bestimmen die Autoren jeweils durch Zuweisung nicht eindeutig zugeordneter Knoten zu Neuronen, die innerhalb des Präsentationszyklus nicht als Zentrum ausgewählt wurden, eine zulässige Rundreise und speichern die Lösung, wenn diese die beste bisher ermittelte Lösung darstellt. Der Algorithmus bricht ab, wenn sich die Neuronen des Neuronenringes hinreichend an die Eingabevektoren der Instanz angenähert haben, sich die Neuronenpositionen nicht von deren Positionen am Ende des vorangegangen Zyklus unterscheiden oder der Parameter G_t eine definierte Grenze unterschreitet. Die Autoren testen die Variante anhand von 91

[254] Vgl. Cochrane und Beasley [2003].
[255] Vgl. hierzu auch Amin [1994].
[256] Vgl. für eine detaillierte Darstellung Cochrane und Beasley [2003], S. 1510.

Testinstanzen für das ETSP mit bis zu 85.900 Knoten. Die Testergebnisse[257] zeigen, dass die Lösungsqualität des vorgeschlagenen Ansatzes dem Guilty Net, dem Elastic Net und den Ansätzen von MATSUYAMA, SOMHOM ET AL. und ARAS ET AL. überlegen ist. Darüber hinaus weist das Co-Adaptive Net geringere Rechenzeiten als die übrigen Varianten auf. Als wesentliche Bestandteile für die Verbesserung der Lösungsqualität identifizieren die Autoren durch Variation der Bestandteile des Algorithmus die zufällige Bestimmung der Präsentationsreihenfolgen, durch welche die Abweichung des Zielfunktionswertes von der besten bekannten Lösung für die Testinstanzen um etwa 2% vermindert wird. Die Einführung des Kooperationsmechanismus führt ebenfalls zu einer geringen Verbesserung der Lösungsqualität um etwa 0,8%.

[257] Vgl. Cochrane und Beasley [2003], S. 1511 ff.

3.1.3 Zusammenfassung und Diskussion

Tabelle 3.7 fasst wesentliche Merkmale der unterschiedlichen SOM-Varianten zur Lösung des ETSP und die Ergebnisse der in der Literatur vorgefundenen Verfahrensvergleiche zusammen:

Quelle	**Wesentliche Merkmale**	**Vergleichsverfahren**[258]
FORT [1988]	Entwicklung der Idee der Anwendung der SOM zur Lösung des ETSP	Simulated Annealing (+)
RITTER und SCHULTEN [1988]	Verwendung einer Gauß-Nachbarschaftsfunktion	-
HUETER [1988]	Verwendung „unscharfer“ Eingabevektoren, Multiplikation der euklidischen Distanz mit einem Faktor, Duplikation von Neuronen	Simulated Annealing (+) Spacefilling Curve (-)
ANGENIOL ET AL. [1988]	Neuronenvernichtung und –duplikation	Elastic Net (+), Kriterium: durchschnittlich ermittelte Ergebnisse Elastic Net (-), Kriterium: bestes Ergebnis
FAVATA und WALKER [1991]	Normalisierung der Eingabevektoren	Simulated Annealing (+)
FRITZKE und WILKE [1991]	Rechenzeitbedarf der Ordnung O(*N*)	2-Opt (+)
BURKE und DAMANY [1992]	Addition des *Conscience-Parameters* zur euklidischen Distanz	Simulated Annealing (+) Elastic Net (+) Spacefilling Curve (-) Hopfield-Tank-KNN (-)
MATSUYAMA [1992] / [1995]	Integration der Adaptionsvorschrift des Elastic Net, ansteigende Lernrate	Simulated Annealing (+) Elastic Net (-)
BURKE [1994]/[1995]	Ausschluss häufig gewinnender Neuronen vom Wettbewerb	Guilty Net (+) Multiple Nearest Neighbor (+) Standard-SOM (-) Spacefilling Curve (-)
AMIN [1994]	Verwendung einer Gravitationskraft bei Adaption der Gewichtsvektoren	Simulated Annealing (-)
BUDINICH [1996]	Abbildung der Knoten auf beliebige Positionen in der Rundreise	Simulated Annealing (+/-)

[258] Angaben in Klammern weisen darauf hin, ob ein Vergleichsverfahren hinsichtlich der Lösungsqualität überlegene (+) oder unterlegene (-) Ergebnisse erzielt.

SOMHOM ET AL. [1997] und MODARES ET AL. [1999]	Wahl eines Neurons als Zentrum innerhalb eines Präsentationszyklus nur einmal möglich	Guilty Net (-) Elastic Net (-) MATSUYAMA [1992] (-)
ARAS ET AL. [1999]	Berücksichtigung statistischer Eigenschaften der Menge der Eingabevektoren	Guilty Net (-) Standard-SOM (-) ANGENIOL ET AL. (-)
COCHRANE und BEASLEY [2003]	Kooperationsmechanismus, Bestimmung einer Lösung am Ende eines Präsentationszyklus	Guilty Net (-) Elastic Net (-) MATSUYAMA [1992] (-) SOMHOM ET AL. (-) ARAS ET AL. (-) FRITZKE und WILKE (-) ANGENIOL ET AL. (-)

Tabelle 3.7: Übersicht über SOM-Varianten zur Lösung des ETSP

Bei den präsentierten Varianten zur Lösung des ETSP wird jedem Knoten im Verlauf des Algorithmus in der Regel ein Neuron zugeordnet, das häufig auch als Bild des zugeordneten Knotens bezeichnet wird. Die Schwierigkeiten, die bei der Zuordnung von Knoten zu Neuronen auftreten, wenn die Anzahl der Neuronen der Ausgabeschicht mit der Anzahl der Knoten der jeweiligen Testinstanzen übereinstimmt, kann durch Verwendung von Einfüge- und Neuronenvernichtungsprozeduren beziehungsweise durch die Verwendung einer Ausgabeschicht mit einer größeren Anzahl an Neuronen umgangen werden[259]. Dabei ist insbesondere im letzten Fall eine Erhöhung der Laufzeit des Algorithmus in Kauf zu nehmen. Werden einem Neuron dennoch mehrere Knoten zugeordnet, so wird meist eine beliebige Reihenfolge der Knoten in der Rundreise gewählt[260]. Dieses Vorgehen stellt sicher, dass jeder Knoten genau einmal besucht und genau einmal wieder verlassen wird, d.h. die Gleichungen *(3.2)* und *(3.3)* werden erfüllt. Da stets ein geschlossener Neuronenring verwendet wird, ist sichergestellt, dass keine Kurzzyklen entstehen, die Ungleichung *(3.4)* wird also ebenfalls erfüllt.

Die Lösungsqualität der SOM-Varianten wird häufig mit den Ergebnissen des Simulated Annealing-Verfahrens verglichen. Dies ist vermutlich darauf zurückzuführen, dass es sich ebenfalls um ein stochastisches Verfahren handelt und die erste Untersuchung von FORT eben diesen Vergleich anführt[261]. Beim Vergleich mit dem Verbesserungsver-

[259] Diese Zahl kann für große Instanzen allerdings sehr groß werden, vgl. Aras et al. [1999], S. 1281.
[260] Vgl. Favata und Walker [1991], S. 465.
[261] Vgl. S. 53 f.

fahren ergibt sich, dass für kleinere Testinstanzen die Güte der durch Simulated Annealing bestimmten Lösungen der Qualität der SOM-Lösungen überlegen ist. Es ist darauf hinzuweisen, dass die Vorteilhaftigkeit der Verfahren in mehreren Untersuchungen ab einer bestimmten Größe der Testinstanz (etwa 500 Knoten) wechselt.

Im Vergleich zu anderen Eröffnungsverfahren zeigt sich, dass die SOM konkurrenzfähige Lösungen bezüglich der Ergebnisqualität ermittelt. Während der Spacefilling Curve-Ansatz in allen Untersuchungen schlechtere Ergebnisse produziert, werden die Ergebnisse des Elastic Net teilweise als überlegen, teilweise als unterlegen bezeichnet. Dies ist darauf zurückzuführen, dass einige Autoren die Durchschnittswerte der Rundreisenlängen zum Vergleich verwenden, während andere Untersuchungen die jeweils besten ermittelten Ergebnisse heranziehen. Da die Ergebnisse neben der verwendeten Testinstanz auch von der jeweiligen Implementierung des Verfahrens abhängen können, soll hier eine eindeutige Wertung unterbleiben. Insgesamt kann die Güte der Lösungen der SOM und des Elastic Net als vergleichbar angesehen werden.

Während im Vergleich mit Simulated Annealing und dem Elastic Net die SOM bezüglich des Laufzeitverhaltens überlegen scheint, ist dies bei Vergleich mit den übrigen herangezogenen Verfahren nicht der Fall. So weisen beispielsweise Nearest Neighbor-Verfahren und der Spacefilling Curve-Ansatz geringere Laufzeiten auf. In diesem Zusammenhang ist der bezüglich der Zeitkomplexität konkurrenzfähige Ansatz von FRITZKE und WILKE herauszustellen[262], der verwendet werden kann, wenn die Lösung möglichst schnell zu ermitteln ist. Außerdem ist auch auf die Parallelisierbarkeit der SOM hinzuweisen, die eine Beschleunigung des Verfahrens bewirken kann.

Die mit Hilfe der SOM ermittelten Lösungen weisen je nach Testinstanz und verwendeter SOM-Variante Zielfunktionswerte auf, deren prozentuale Abweichungen von der besten bekannten Lösung in der Regel im einstelligen Bereich liegen. Die Ansätze von SOMHOM ET AL., MODARES ET AL.[263] und COCHRANE und BEASLEY[264] sind als die bisher erfolgreichsten Varianten hervorzuheben.

Zusammenfassend kann festgehalten werden, dass das Verfahren im Hinblick auf die Lösungsqualität anderer Eröffnungsverfahren für das ETSP nicht unterlegen ist. Vergleichsweise hohe Rechenzeiten der meisten SOM-Varianten und die Tatsache, dass eine Erweiterung des Verfahrens auf asymmetrische bzw. nicht-euklidische Probleminstanzen bisher nicht erfolgreich umgesetzt wurde, zeigen allerdings Grenzen der An-

[262] Vgl. Fritzke und Wilke [1991].
[263] Vgl. S. 70 f.
[264] Vgl. Cochrane und Beasley [2003].

wendung auf. Ein Vergleich der Lösungsqualität und der benötigten Rechenzeiten mit den derzeit besten bekannten Verbesserungsverfahren zur Lösung des ETSP zeigt außerdem, dass die bisher entwickelten selbstorganisierenden Netze als unterlegen einzuschätzen sind. [265]

[265] Vgl. beispielsweise Johnson und McGeoch [1997], S. 308 f., und Cochrane und Beasley, S. 1522.

3.1.4 Anwendungsbeispiel

Zur Illustration wurde vom Verfasser eine Variante des Algorithmus implementiert. Da sich die Verwendung von Präsentationszyklen mit zufälliger Präsentationsreihenfolge in verschiedenen Veröffentlichungen als erfolgreich erwiesen hat[266], wurde diese Vorgehensweise auch hier übernommen. Als Nachbarschaftsfunktion wurde die in der Literatur überwiegend eingesetzte Gauß-Funktion verwendet. Die Variante entspricht somit im wesentlichen dem von SOMHOM ET AL. vorgeschlagenen Algorithmus[267]. Weitere Merkmale der Variante und der verwendeten Probleminstanz können Tabelle 3.8 entnommen werden.

Merkmal	**Wert**	**Bemerkung**
Anzahl Knoten	83	Eingabevektoren
Anzahl Neuronen der Eingabeschicht = Anzahl Komponenten des Eingabevektors	2	-
Anzahl Neuronen der Ausgabeschicht	300	-
Präsentationszyklen	$t_{p\max} = 200$ Zyklen	Der Zykluszähler wird im Anschluss an die Präsentation von 83 zufällig bestimmten Orten erhöht.
Iterationen	16.600	$t_{\max} = t_{p\max} \cdot 83$
Verlauf des Parameters der Nachbarschaftsfunktion	$\sigma(t_p) = \sigma_{t_p=0} \cdot \left(\frac{\sigma_{t_{p\max}}}{\sigma_{t_p=0}} \right)^{\frac{t_p}{t_{p\max}}}$	Steuert die Breite der Nachbarschaftsfunktion.
Initialisierungswert des Parameters $\sigma(t_p)$	$\sigma_{t_p=0} = 30$	-
Verlauf der Lernrate	$\alpha(t_p) = \max\left\{1 - \frac{t_p}{t_{p\max}}, 0{,}1\right\}$	Steuert die Stärke der Adaption der Gewichtsvektoren.
Distanzfunktion	$\left\|\mathbf{x}(t) - \mathbf{w}_j(t)\right\|^2$	Quadrierte euklidische Distanz.
Reihenfolge der Präsentation der Eingabedaten	Zufällig.	Auswahlwahrscheinlichkeit gemäß einer Gleichverteilung.
Initialisierung der Gewichtsvektoren	Entlang eines Kreises.	-
Topologie der Ausgabeschicht	Ring	-
Nachbarschaftsfunktion	$h_{cj}(t_p) = \alpha(t_p) \cdot \exp\left(-\frac{\Delta_{cj}^{\,2}}{2 \cdot \sigma^2(t_p)} \right)$	Zur Bestimmung der Entfernung zwischen zwei Neuronen wird die minimale Anzahl an Verbindungen zwischen den Neuronen verwendet.
Lernregel	$\mathbf{w}_j(t+1) = \left(1 - h_{cj}(t_p)\right) \cdot \mathbf{w}_j(t) + h_{cj}(t_p) \cdot \mathbf{x}(t)$	-

Tabelle 3.8 : Merkmale des Anwendungsbeispiels für das ETSP

[266] Vgl. Cochrane und Beasley [2003] und Somhom et al. [1997].
[267] Vgl. Somhom et al. [1997].

In der folgenden Abbildung werden die Koordinaten der zu besuchenden Knoten blau, die Neuronenkoordinaten grün und die resultierende Route rot gefärbt dargestellt. Um die Entwicklung der Anpassung der Neuronengewichte aufzuzeigen, können neben der Initialisierungsanordnung jeweils die Anordnungen der Neuronengewichte nach ¼ der Iterationen, nach ¾ der Iterationen und schließlich die Anordnung nach der vollen Zahl der Iterationen entnommen werden.

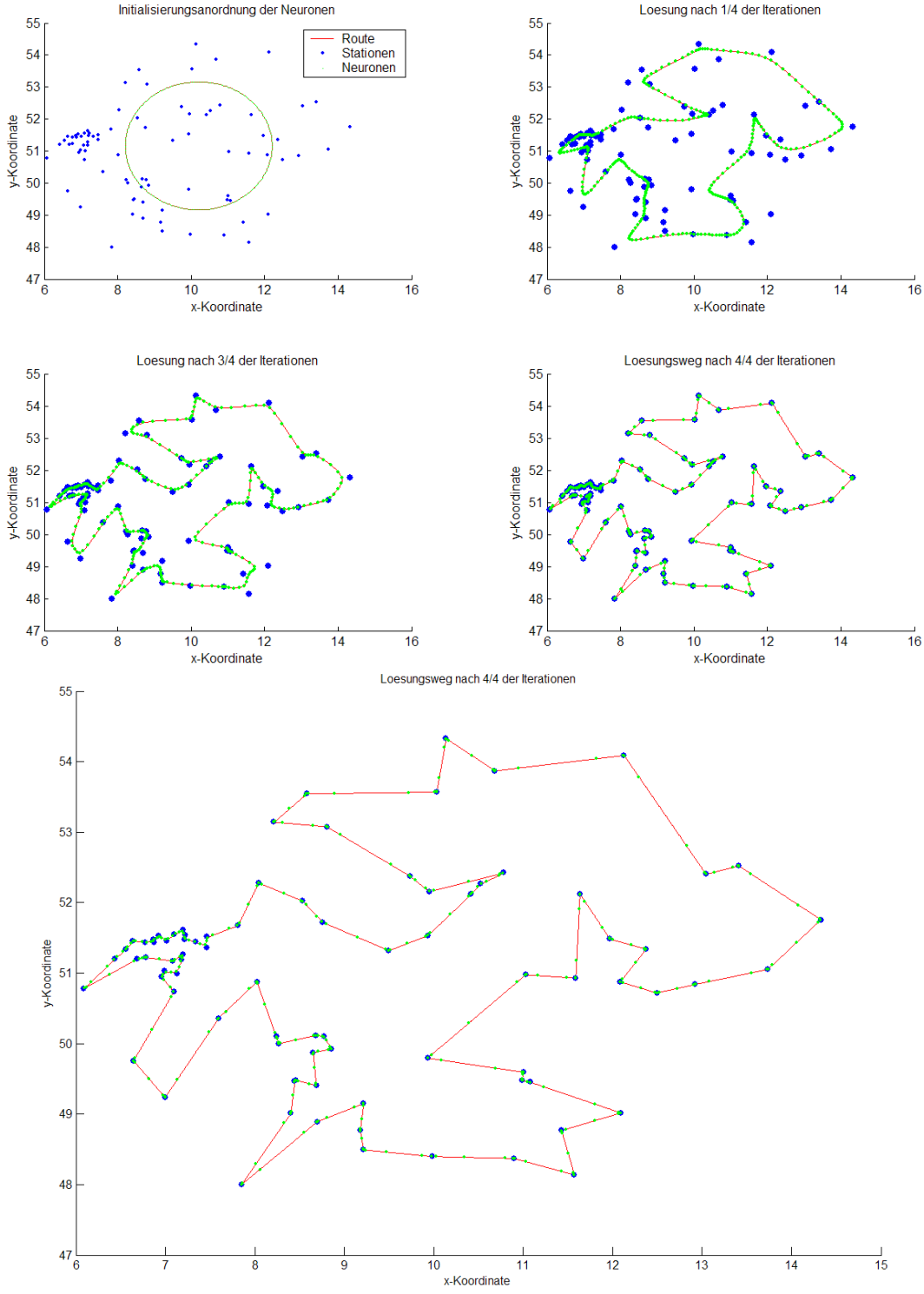


Abbildung 3.11 : Entfaltung der Self-Organizing Map, ETSP-Beispiel

Der folgenden Abbildung können die Graphen der eingesetzten Nachbarschaftsfunktion der ersten und letzten Iteration sowie die Graphen des Verlaufs der Parameterwerte mit zunehmender Anzahl an Präsentationszyklen entnommen werden:

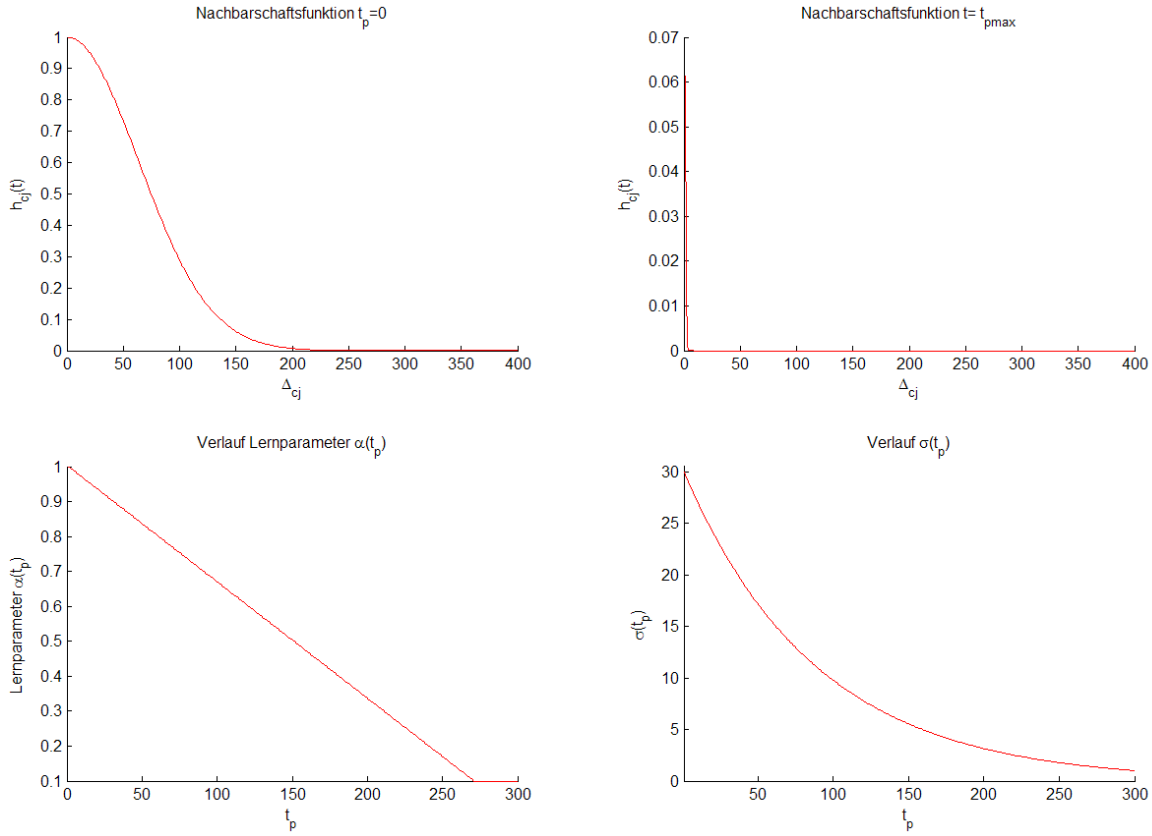


Abbildung 3.12 : Verlauf der Nachbarschaftsfunktion und der Parameter, ETSP-Beispiel

3.2 Euklidische Multiple Traveling Salesman Probleme

3.2.1 Beschreibung der Multiple Traveling Salesman Probleme

Klassische Multiple Traveling Salesman Probleme (MTSP) beschreiben eine Generalisierung des TSP (und sind somit ebenfalls als NP-schwere Probleme zu klassifizieren), in der Routen für *R* Handlungsreisende so festzulegen sind, dass die Summe der von allen Handlungsreisenden zurückgelegten Distanzen minimal wird. 1973 beschreiben SVESTKA und HUCKFELDT[268] die im Folgenden vorgestellte Modellierung durch Transformation in ein TSP, bei dem die Gesamtkosten der Rundreisen der Handlungsreisenden minimiert werden sollen.

Eine allgemeine Formulierung des MTSP für Graphen lautet[269]: Die Elemente der Mengen Γ_Z seien Zyklen, deren Längen bekannt sind. Bei gegebenem Graphen *G* mit den Knoten *N* wird die Menge $\Gamma \in \Gamma_Z$ gesucht, für welche die Summe der Zyklenlängen minimal ist. Dabei sind folgende Bedingungen zu erfüllen:

1. Jeder Zyklus enthält den sogenannten Depotknoten und
2. jeder andere (Kunden-)Knoten ist genau einmal in genau einem Zyklus enthalten.

Bei der angegebenen Modellformulierung wird eine Kostenmatrix $\boldsymbol{D}(d_{mn})$ verwendet (vgl. Abbildung 3.13), die aus der Kostenmatrix $\boldsymbol{C}(c_{mn})$ für das TSP abgeleitet wird, indem *R-1* zusätzliche Zeilen und Spalten angefügt werden, die jeweils der ersten Zeile und Spalte der ursprünglichen Matrix entsprechen[270]:

∞	Kopie der Zeile 1
Kopie der Spalte 1	Matrix $\boldsymbol{C}$

Abbildung 3.13 : Erweiterung der ursprünglichen Kostenmatrix

Das resultierende Modell wird in Tabelle 3.10 formuliert[271], die verwendeten Symbole werden in Tabelle 3.9 beschrieben. Bei der Interpretation einer Lösung für die gegebene Formulierung ist darauf zu achten, dass *R-1* Hilfsknoten eingeführt wurden, die das Depot repräsentieren. Dadurch werden auch die Indizes der ursprünglich vorhandenen Knoten, welche die zu besuchenden Knoten repräsentieren, verändert.

---

[268] Vgl. Svestka und Huckfeldt [1973].
[269] Vgl. Domschke [1997], S. 162 f.
[270] Vgl. Svestka und Huckfeldt [1973], S. 791 f. Es wird angenommen, dass die erste Zeile beziehungsweise Spalte der Matrix $\boldsymbol{C}(c_{mn})$ Bewertungen der vom Depot ausgehenden beziehungsweise eingehenden Kanten enthält.
[271] Die von Svestka und Huckfeldt vorgeschlagenen Bedingungen zur Verhinderung von Kurzzyklen wurden nicht verwendet, da sie nicht alle unzulässigen Lösungen ausschließen, vgl. Anhang E. Die hier formulierten Kurzzyklenbedingungen lehnen sich an die bereits vorgestellten *Miller-Tucker-Zemlin*-Bedingungen an, vgl. S. 51 und Anhang E.

Symbol	Interpretation
m	Index der Knoten
n	Index der Knoten
R	Anzahl Handlungsreisender
Λ_0	Menge der Indizes der Handlungsreisenden
$x_{mn} = \begin{cases} 1, & \text{direkte Verbindung von } m \text{ nach } n \text{ ist Teil einer Rundreise} \\ 0, & \text{sonst} \end{cases}, \forall m,n$	s. Abbildung 3.14
π_m	Hilfsvariablen zur Formulierung von Kurzzyklenbedingungen.
d_{mn}	Kantenbewertungen

Tabelle 3.9 : Symbole der Modellformulierung des klassischen MTSP

	Formale Beschreibung	Interpretation
Minimiere	*(3.32)* $\sum_{m=1}^{N+R-1} \sum_{\substack{n=1, \\ n \neq m}}^{N+R-1} d_{mn} x_{mn}$	Die Gesamtkosten entsprechen der Summe der Bewertungen der verwendeten Kanten.
u.B.v.:	*(3.33)* $\sum_{\substack{n=1, \\ n \neq m}}^{N+R-1} x_{mn} = 1 \;,\; m = 1(1)N+R-1$	Jeder Knoten *m* muss einmal in Richtung eines Knoten *n* verlassen werden.
	(3.34) $\sum_{\substack{m=1, \\ m \neq n}}^{N+R-1} x_{mn} = 1,\; n = 1(1)N+R-1$	Jeder Knoten *n* muss von einem Knoten *m* aus angefahren werden.
	(3.35) $\pi_m - \pi_n + N \cdot x_{mn} \leq N-1 \qquad \forall m \neq n \wedge m,n \notin \Lambda_0 = \{1,2,...,R\}$	Nebenbedingung zur Verhinderung von unzulässigen Kurzzyklen.
	(3.36) $x_{mn} \in \{0,1\} \qquad \forall m,n = 1(1)N+R-1, \quad m \neq n$	Binärbedingungen.
	(3.37) $\pi_m \geq 0 \qquad m = R(1)N+R-1$	Nichtnegativitätsbedingungen für die Hilfsvariablen.

Tabelle 3.10 : Mögliche Modellformulierung für das klassische MTSP

Die angegebene Modellformulierung verwendet $(N+R-1)^2-(N+R-1)$ binäre Entscheidungs- und $(N-1)$ reelle Hilfsvariablen. Insgesamt werden unter Vernachlässigung der Binärbedingungen $2 \cdot (N+R-1) + (N-1) \cdot (N-2) + (N-1)$ Nebenbedingungen benötigt. Es ist darauf hinzuweisen, dass im Fall symmetrischer Distanzen Modellformulierungen verwendet werden können, die eine geringere Anzahl an Variablen benötigen.[272] Bei Verwendung euklidischer Distanzen als Kantenbewertung wird das MTSP im Folgen-

[272] Vgl. bspw. Laporte und Nobert [1980], S. 1018.

den als euklidisches MTSP (EMTSP) bezeichnet. Abbildung 3.14 zeigt, wie die Lösung des Modells zu interpretieren ist.

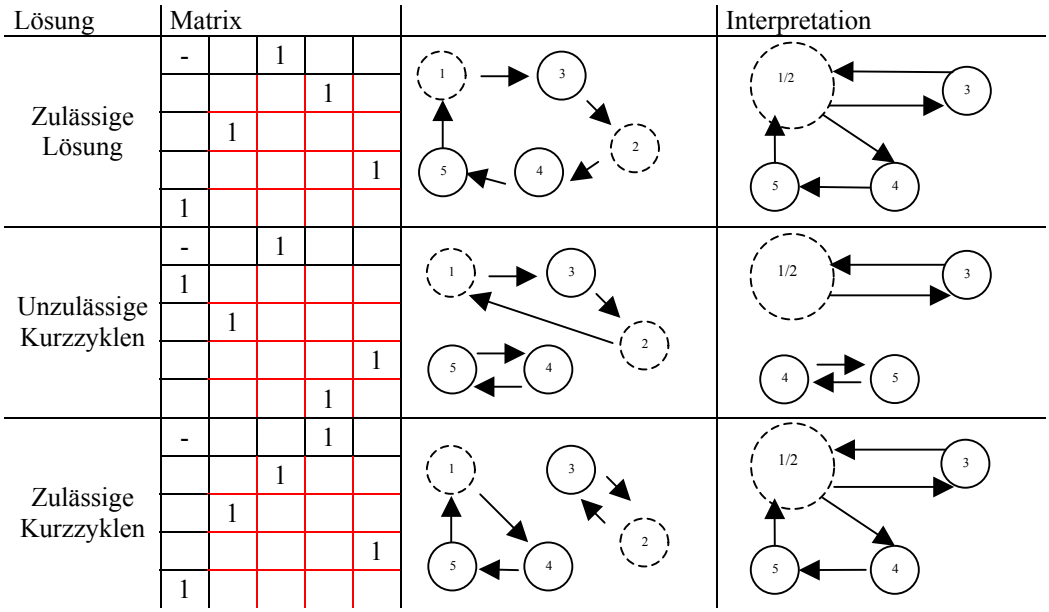


Abbildung 3.14 : Interpretation der Lösung der MTSP-Formulierung[273]

Da die ersten beiden Zeilen und Spalten der Matrix das Depot (Knoten 1 und Knoten 2) repräsentieren, werden jeweils zwei Rundreisen abgebildet. In der ersten Zeile der Tabelle wird eine zulässige Lösung beschrieben, die zweite Zeile stellt aufgrund des unzulässigen Kurzzyklus zwischen den Knoten 4 und 5 eine unzulässige Lösung für das MTSP dar. Die dritte Zeile beschreibt eine Lösung, die einen Kurzzyklus enthält, der aufgrund der mehrfachen Repräsentation des in der Abbildung jeweils gestrichelt dargestellten Depotknotens für das MTSP zulässig ist.

3.2.2 Eingesetzte Varianten der Self-Organizing Map

Anhand der Interpretation der Lösung in Abbildung 3.14 lässt sich bereits erahnen, wie der Algorithmus der SOM auf die Problemstellung angewendet werden kann: Für jeden Salesman wird ein Neuronenring modelliert, der dessen Rundreise abbildet. Die Zielfunktion *(3.32)* wird in der Literatur allerdings häufig durch die Zielsetzung „Minimierung der maximalen Länge der Rundreisen“ ersetzt (sogenanntes „MINMAX“-MTSP),

[273] In Anlehnung an Svestka und Huckfeldt [1973], S. 793.

um eine möglichst gleichmäßige Auslastung der Handlungsreisenden zu gewährleisten.[274]

1990 veröffentlicht GOLDSTEIN einen Tagungsbericht[275], in dem er die von ANGENIOL ET AL. vorgeschlagene SOM-Variante für das ETSP auf das EMTSP erweitert. GOLDSTEIN verwendet Präsentationszyklen mit identischer Reihenfolge der zu präsentierenden Eingabevektoren und eine mit der Anzahl an Handlungsreisenden R übereinstimmende Anzahl an Neuronenringen. Wird dem neuronalen Netz ein Eingabevektor $\mathbf{x}(t)$ präsentiert, so wird für jedes Neuron j der Ringe der Ausgabeschicht das Potential

$$(3.38)\quad d_j^r = \left\|\mathbf{x}(t) - \mathbf{w}_j^r(t)\right\|^2 \cdot \left[1 + \left(\frac{Dist_r - Dist_{AVG}}{Dist_{AVG}}\right)\right]$$

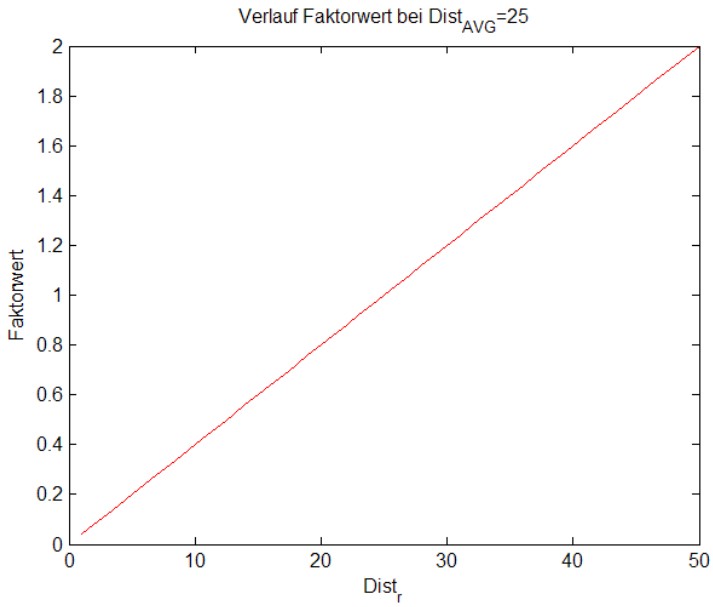


Abbildung 3.15 : Verlauf Faktorwert, GOLDSTEIN

berechnet und das Zentrum durch

$$(3.39)\quad d_{j^*}^r = \min_{j,r}\left\{d_j^r\right\}$$

bestimmt. Dabei beschreibt $Dist_r$ die Länge der Tour des Handlungsreisenden r $(r=1(1)R)$ und $Dist_{AVG}$ die durchschnittliche Tourenlänge der R Touren. Gilt $Dist_r = Dist_{AVG}$, ergibt sich als Potential die quadrierte euklidische Distanz, gilt $Dist_r < Dist_{AVG}$, wird dieses Potential verkleinert, sonst erhöht. Anschließend werden

[274] Vgl. hierzu bspw. Johnson und Papadimitriou [1985], S. 169, und Somhom et al. [1999], S. 397.
[275] Vgl. Goldstein [1990].

die Gewichtsvektoren des Zentrums $\mathbf{w}^r_{j^*}(t)$ und dessen direkter Nachbarn im Ring r gemäß der von ANGENIOL ET AL. vorgeschlagenen Anpassungsvorschrift verändert. Wird dem Netz ein Ortsvektor eines zu besuchenden Knotens präsentiert, so werden also nur Neuronen des Ringes verschoben, der das Zentrum enthält. Zusätzlich zu den Ortsvektoren der Kundenknoten wird in einem Präsentationszyklus auch der Ortsvektor des Depotknotens R-mal jeweils einem Neuronenring präsentiert, so dass Neuronen jedes Ringes in Richtung des Depots verschoben werden. Der Autor berichtet, dass durch die vorgeschlagene Distanzfunktion eine Angleichung der Länge der R Touren erreicht wird.

PIETSCH und TEUBNER schlagen 1996 eine Variante zur Lösung des EMTSP vor[276], bei der die euklidische Distanz zwischen Eingabevektor und Gewichtsvektor mit dem Verhältnis zwischen der Länge der Rundreise des Handlungsreisenden r, zu dessen Neuronenring der Gewichtsvektor gehört, und der durchschnittlichen Länge der Rundreisen aller Handlungsreisenden multipliziert wird:

$$(3.40)\quad \frac{Dist_r}{Dist_{AVG}} = \frac{Dist_{AVG} + Dist_r - Dist_{AVG}}{Dist_{AVG}} = 1 + \left(\frac{Dist_r - Dist_{AVG}}{Dist_{AVG}} \right).^{277}$$

Der vorgeschlagene Faktor kann, wie gezeigt, leicht in den von GOLDSTEIN vorgeschlagenen Faktor umgeformt werden.

MATSUYAMA erweitert seinen 1992 für das ETSP vorgeschlagenen Algorithmus[278] auch auf das EMTSP, wobei Beschränkungen bezüglich der Zuordnung der Handlungsreisenden zu Kundenorten integriert werden, d.h., nicht jeder Knoten kann von jedem Handlungsreisenden besucht werden.[279] Auch in diesem Ansatz werden in einer Iteration jeweils nur die Neuronen des Ringes, der das Zentrum enthält, adaptiert. Ein Vergleich mit alternativen Verfahren wird nicht durchgeführt.

SOMHOM ET AL. verwenden eine Variante der SOM, um eine Lösung für das „MIN-MAX“-EMTSP mit dem Ziel der Minimierung der maximalen Tourlänge abzuleiten.[280] Die Autoren nutzen ebenfalls R Neuronenringe, welche die Touren der R Handlungsreisenden repräsentieren, wobei jeweils ein Neuron das Depot darstellt und an den Koordi-

[276] Vgl. Pietsch und Teubner [1996].
[277] Vgl. Pietsch und Teubner [1996], S. 231.
[278] Vgl. S. 64 f.
[279] Vgl. Matsuyama [1992], S. 107 ff.
[280] Vgl. Somhom et al. [1999].

naten des Depot-Standortes platziert wird. Die Distanzfunktion entspricht der von GOLDSTEIN vorgeschlagenen Funktion. Die Eingabevektoren werden in Zyklen präsentiert, wobei die Reihenfolge der Präsentation der Knoten für jeden Zyklus zufällig bestimmt wird. Die Adaptionsvorschrift stimmt mit der von RITTER und SCHULTEN verwendeten Vorschrift zur Lösung des TSP überein[281], es werden jeweils nur Neuronen des Ringes, der das Zentrum enthält, angepasst. Die Autoren vergleichen die Lösungsgüte des Algorithmus anhand von bekannten Testinstanzen für Tourenplanungsprobleme unter Vernachlässigung der Kapazitätsrestriktionen. Ein Vergleich der Lösungsgüte mit den Ergebnissen für zwei heuristische Verfahren (ein Farthest Insertion-Verfahren und ein Nearest Neighbor-Verfahren, jeweils erweitert um eine 2-Opt-Prozedur) und der Elastic Net-Methode zeigt, dass die SOM-Variante für alle Instanzen bessere Ergebnisse erzielt.[282] Identische Ergebnisse werden auch in MODARES ET AL. [1999] vorgestellt[283]. SOMHOM ET AL. erweitern die SOM zusätzlich um ein 2-Opt-Verbesserungsverfahren und vergleichen die Ergebnisse mit einer adaptiven Tabu Search-Prozedur. Mit der um das Verbesserungsverfahren erweiterten SOM-Variante lassen sich mit geringerem Zeitaufwand (durchschnittlich 31%) Lösungen ableiten, deren Gesamtlänge nicht mehr als 3% von den mit Hilfe der Tabu Search-Prozedur ermittelten Lösungen nach oben abweichen.[284] Die Autoren folgern, dass hybride Algorithmen auf der Basis selbstorganisierender neuronaler Netze eine sinnvolle Vorgehensweise zur Lösung kombinatorischer Optimierungsprobleme darstellen.

[281] Vgl. S. 55.
[282] Vgl. Somhom et al. [1999], S. 404.
[283] Vgl. Modares et al. [1999], S. 604.
[284] Vgl. Somhom et al. [1999], S. 405.

3.2.3 Zusammenfassung und Diskussion

Tabelle 3.11 fasst die SOM-Varianten zur Lösung des EMTSP zusammen:

Quelle	**Wesentliche Merkmale**	**Vergleichsverfahren[285]**
GOLDSTEIN [1990]	Verwendung eines Faktors in der Distanzfunktion zur Angleichung der Rundreisenlängen	-
MATSUYAMA [1992]	Integration zusätzlicher Nebenbedingungen, die Zuordnungen bestimmter Knoten zu bestimmten Handlungsreisenden verbieten	-
MODARES ET AL. [1999]	Präsentationszyklen mit zufälliger Reihenfolge	Elastic Net (-) Farthest Insertion (-) Nearest Neighbor (-)
SOMHOM ET AL. [1999]	Präsentationszyklen mit zufälliger Reihenfolge, Erweiterung um 2-Opt-Prozedur	Elastic Net (-) Farthest Insertion (-) Nearest Neighbor (-) Tabu Search Prozedur (+)

Tabelle 3.11: Übersicht über SOM-Varianten zur Lösung des EMTSP

Jedem Knoten wird im Verlauf des Algorithmus ein Neuron zugeordnet. Dies stellt wiederum sicher, dass jeder Knoten aus der Richtung des vorgelagerten Neurons genau einmal besucht und genau einmal in Richtung des nachgelagerten Neurons wieder verlassen wird, d.h. die Gleichungen *(3.33)* und *(3.34)* werden erfüllt. Da geschlossene Neuronenringe verwendet werden und bei Präsentation des Depotknotens für jeden Neuronenring ein Zentrum bestimmt und entsprechende Adaptionsprozesse durchgeführt werden, ist sichergestellt, dass keine Kurzzyklen entstehen; die Ungleichung *(3.35)* wird also ebenfalls erfüllt. Da nur eine geringe Anzahl an Verfahrensvergleichen vorliegt, soll hier auf eine endgültige Bewertung des Verfahrens zur Lösung von EMTSPen verzichtet werden und stattdessen lediglich auf die positiven Ergebnisse von MODARES ET AL. und SOMHOM ET AL. hingewiesen werden. Die Autoren zeigen, dass der SOM-Ansatz zur Lösung des MINMAX-EMTSP verschiedenen Eröffnungsverfahren auch dann überlegen ist, wenn diesen eine 2-Opt-Prozedur angeschlossen wird.[286] Wird ein solches Verfahren auch der SOM angeschlossen, so unterliegt das Verfahren zwar einer Tabu Search-Prozedur bezüglich der Lösungsqualität, weist jedoch eine geringere Rechenzeit auf.

[285] Angaben in Klammern weisen darauf hin, ob ein Vergleichsverfahren hinsichtlich der Lösungsqualität überlegene (+) oder unterlegene (-) Ergebnisse erzielt.
[286] Vgl. Modares et al. [1999], S. 604, und Somhom et al. [1999], S. 405.

3.2.4 Anwendungsbeispiel

Tabelle 3.12 fasst die Merkmale des illustrierenden Beispiels zusammen. In der vom Verfasser implementierten Version werden wieder Präsentationszyklen mit zufälliger Präsentationsreihenfolge verwendet, die eingesetzte Distanzfunktion entspricht der von GOLDSTEIN vorgeschlagenen Variante[287]. Bei Präsentation des Depots wird für jeden Ring ein Zentrum bestimmt und entsprechende Gewichtsadaptionen ausgeführt.

Merkmal	**Wert**	**Bemerkung**
Anzahl Knoten	83	Eingabevektoren.
Anzahl Neuronen der Eingabeschicht	2	-
Anzahl Salesmen R	3	Annahme.
Anzahl Neuronen der Ausgabeschicht	240	80 Neuronen je Ring.
Präsentationszyklen	$t_{p\max} = 200$ Zyklen	-
Anzahl präsentierter Eingabevektoren	16.600	$t_{\max} = t_{p\max} \cdot 83$
Verlauf des Parameters der Nachbarschaftsfunktion	$\sigma(t_p) = \sigma_{t_p=0} \cdot \left(\frac{\sigma_{t_{p\max}}}{\sigma_{t_p=0}} \right)^{\frac{t_p}{t_{p\max}}}$	Steuert die Breite der Nachbarschaftsfunktion.
Initialisierungswert des Lernparameters $\sigma(t_p)$	$\sigma_{t_p=0} = 6$	-
Verlauf der Lernrate	$\alpha(t_p) = \max\left\{1 - \frac{t_p}{t_{p\max}}, 0{,}1\right\}$	Steuert die Stärke der Adaption der Gewichtsvektoren.
Distanzfunktion	$\left\|\mathbf{x}(t) - \mathbf{w}_j^r(t)\right\|^2 \cdot \left[1 + \left(\frac{Dist_r - Dist_{AVG}}{Dist_{AVG}} \right)\right]$	-
Bestimmung des Zentrums	Neuron mit geringster Distanz.	Bei Präsentation des Depots werden für jeden Ring Gewichtsadaptionen ausgeführt.
Reihenfolge der Präsentation der Eingabevektoren	Zufällig.	-
Initialisierung der Gewichtsvektoren	Jeweils Ringanordnung der zu einem Handlungsreisenden zugeordneten Neuronen.	Die Kreismittelpunkte der Ringe werden zufällig bestimmt.
Topologie der Ausgabeschicht	Ringe.	-
Nachbarschaftsfunktion	$h_{cj}(t_p) = \alpha(t_p) \cdot \exp\left(-\frac{\Delta_{cj}^2}{2 \cdot \sigma^2(t_p)} \right)$	-
Lernregel	$\mathbf{w}_j(t+1) = \left(1 - h_{cj}(t_p)\right) \cdot \mathbf{w}_j(t) + h_{cj}(t_p) \cdot \mathbf{x}(t)$	-

Tabelle 3.12 : Merkmale des Anwendungsbeispiels für das EMTSP

[287] Vgl. Goldstein [1990].

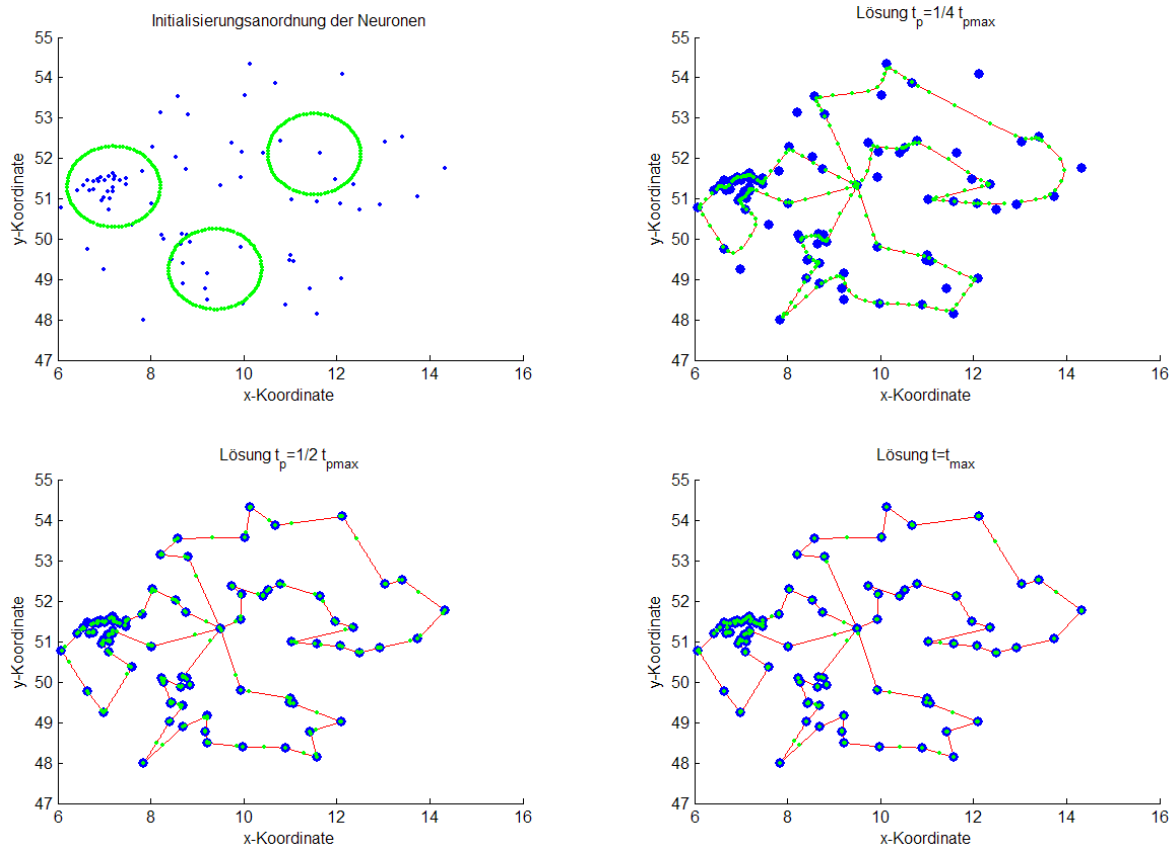


Abbildung 3.16 : Entfaltung der Self-Organizing Map, EMTSP-Beispiel

Abbildung 3.16 kann die Anpassung der Neuronpositionen im Verlaufe des Verfahrens entnommen werden. Es ist zu erkennen, dass die Wahl eines Zentrums für jeden Neuronenring bei Präsentation des Depots dazu führt, dass die Rundreisen aller Handlungsreisender durch den Depotknoten führen.

3.3 Euklidische Tourenplanungsprobleme

3.3.1 Beschreibung eines Standardproblems der Tourenplanung

Tourenplanungsprobleme beschreiben eine Klasse von Problemen, deren Gegenstand die Planung des Besuchs von Kunden durch mehrere kapazitierte Fahrzeuge darstellt.[288] Einen Überblick über verschiedene Probleme dieser Klasse geben beispielsweise ASSAD[289] und DETHLOFF[290], eine Untersuchung exakter und heuristischer Lösungsverfahren für Tourenplanungsprobleme mit Zeitfensterrestriktionen findet sich in WEISSERMEL[291]. Ein allgemeiner Überblick über erfolgreiche moderne und klassische Heuristiken, die zur Lösung von Tourenplanungsproblemen verwendet werden können, findet sich in der Arbeit von CORDEAU ET AL.[292] DANTZIG und RAMSER beschreiben bereits 1959 das „Truck Dispatching Problem"[293], das heute auch als Standardproblem der Tourenplanung *(Capacitated Vehicle Routing Problem, kurz CVRP)* bezeichnet wird. Dabei soll eine Flotte von *K* kapazitierten Fahrzeugen eingesetzt werden, um die unteilbare Nachfrage B_n der Knoten *n (n=1(1)N)* zu befriedigen. Der zu minimierende Wert der Zielfunktion beschreibt die Gesamtlänge des Tourenplans, wobei die Tour jedes Fahrzeugs am Depot beginnen und enden muss. Tabelle 3.14 enthält ein Modell zur Beschreibung des NP-schweren Problems[294]. Die Definitionen der verwendeten Symbole können Tabelle 3.13 entnommen werden.

Symbol	Definition
N	Anzahl der Knoten *m (n), m, n=1(1)N*
K	Anzahl der Fahrzeuge *k=1(1)K*
0	Index des Depotknotens
c_{mn}	Kosten der Fahrt von Knoten *m* zu Knoten *n*
Q_k	Kapazität des Fahrzeuges *k*
B_n	Nachfrage des Knotens *n*
x_{mnk}	$x_{mnk} = \begin{cases} 1, & \text{wenn Fahrzeug } k \text{ Knoten } n \text{ direkt im Anschluss an Knoten } m \text{ besucht} \\ 0, & \text{sonst} \end{cases}, \forall m,n,k$
y_{mk}	$y_{mk} = \begin{cases} 1, & \text{wenn Knoten } m \text{ Fahrzeug } k \text{ zugeordnet wird} \\ 0, & \text{sonst} \end{cases}, \forall m,k$
V	Menge der Knoten einschließlich des Depotknotens
Q	Teilmenge von V

Tabelle 3.13 : Symbole der Modellformulierung des CVRP

[288] Vgl. Christofides et al. [1979], S. 315.
[289] Vgl. Assad [1988].
[290] Vgl. Dethloff [1994].
[291] Vgl. Weissermel [1999].
[292] Vgl. Cordeau et al. [2002].
[293] Vgl. Dantzig und Ramser [1959].
[294] Vgl. Gendreau et al. [1997], S. 312.

		Formale Beschreibung[295]		Interpretation
Minimiere	*(3.41)*	$\sum_{m=1}^{N}\sum_{n=1}^{N}\left(c_{mn}\sum_{k=1}^{K}x_{mnk}\right)$		Ziel ist die Minimierung der Gesamtlänge der Touren.
u.B.v.:	*(3.42)*	$\sum_{n=1}^{N}B_n y_{nk} \le Q_k \;,\; k = 1(1)K$		Die aggregierte Nachfrage der einem Fahrzeug zugeordneten Knoten darf die Fahrzeugkapazität nicht überschreiten.
	(3.43)	$\sum_{k=1}^{K}y_{nk} = 1 \;,\; n = 1(1)N$		Jeder Knoten muss genau einem Fahrzeug zugeordnet werden.
	(3.44)	$y_{nk} \in \{0,1\}, n = 1(1)N,\; k = 1(1)K$		Binärbedingung für die Zuordnungsvariablen
	(3.45)	$\sum_{m=0}^{N}x_{mnk} = y_{nk} \;,\; n = 0(1)N$	$k = 1(1)K$	Wenn der Knoten *n* dem Fahrzeug *k* zugeordnet wurde, muss das Fahrzeug den Knoten genau einmal anfahren. Wenn der Knoten *n* dem Fahrzeug nicht zugeordnet wurde, darf das Fahrzeug den Knoten nicht anfahren.
	(3.46)	$\sum_{n=0}^{N}x_{mnk} = y_{mk} \;,\; m = 0(1)N$		Wenn der Knoten *m* dem Fahrzeug *k* zugeordnet wurde, muss das Fahrzeug den Knoten genau einmal verlassen. Wenn der Knoten *m* dem Fahrzeug nicht zugeordnet wurde, darf das Fahrzeug den Knoten nicht verlassen.
	(3.47)	$\sum_{m\in Q}\sum_{n\in Q}x_{mnk} \le \lvert Q\rvert - 1 \;,\; \forall Q \subseteq V - \{0\},\quad 2 \le \lvert Q\rvert \le N-1$		Bedingung zur Verhinderung von Kurzzyklen zwischen den Kunden.
	(3.48)	$x_{mnk} \in \{0,1\},\; m,n = 0(1)N$		Binärbedingungen.
	(3.49)	$y_{0k} = 1$		Der Depotknoten (Index 0) wird jedem Fahrzeug zugeordnet.

Tabelle 3.14 : Mögliche Modellformulierung für das CVRP

[295] Vgl. Domschke [1997], S. 215 f. Bei der Formulierung wird unterstellt, dass die Kostenmatrix in der Hauptdiagonalen hinreichend große Einträge c_{mm} enthält, die verhindern, dass die Variablen x_{mmk} für $m > 0$ den Wert Eins annehmen.

Das Modell benötigt unter Vernachlässigung der Binärbedingungen insgesamt $K + N + K \cdot \left(2 \cdot (N+1) + 1 + \sum_{|Q|=2}^{N-1} \binom{N}{|Q|} \right)$ Nebenbedingungen, die Anzahl der Binärvariablen beträgt $\left((N+1)^2 - N\right) \cdot K + N \cdot K$. Soll eine homogene Fahrzeugflotte verwendet werden, so kann der Index *k* auf der rechten Seite der Kapazitätsrestriktion *(3.42)* entfallen. Anhand der Modellformulierung wird deutlich, dass das CVRP ein verallgemeinertes Zuordnungsproblem mit maximal *K* Traveling Salesman Problemen verknüpft. Hieraus leitet sich auch die Anwendung der SOM in Analogie zum EMTSP ab, wobei – wie im Folgenden gezeigt – Mechanismen in den Algorithmus zu integrieren sind, welche die Einhaltung der Kapazitätsrestriktionen für die Fahrzeuge unterstützen. Die vorgestellte Modellierung kann sowohl zur Beschreibung symmetrischer als auch asymmetrischer Probleme verwendet werden. Wie auch für das TSP, finden sich in der Literatur alternative Formulierungen für asymmetrische[296] und symmetrische[297] CVRPe, die mitunter eine geringere Anzahl an Variablen benötigen. Die Anwendung der SOM beschränkt sich überwiegend auf Probleme mit euklidischen Distanzen, weshalb das zugehörige Tourenplanungsproblem im Folgenden durch ECVRP abgekürzt wird.

3.3.2 Eingesetzte Varianten der Self-Organizing Map

MATSUYAMA veröffentlicht 1991 in einem Tagungsbeitrag[298] eine Erweiterung der SOM zur Behandlung euklidischer Tourenplanungsprobleme. Der Autor verwendet mehrere Neuronenringe in der Ausgabeschicht mit insgesamt *J* Neuronen, welche die Touren der Fahrzeuge repräsentieren. Jeder Neuronenring besteht dabei aus $\frac{J}{K}$ Neuronen. Zusätzlich zu den Kapazitätsrestriktionen wird das Problem um Restriktionen erweitert, die festlegen, welche der Knoten *m (m=1(1)N)* von einem Fahrzeug *k (k=1(1)K)* bedient werden dürfen. Zur Bestimmung des Zentrums verwendet der Autor die Funktion

$$\min_{0<k_z<K,\,0<j<\frac{J}{K}} \left[H_{k_z m} \cdot \frac{Dist_{k_z}^2}{\sum_k Dist_k \left(\sum_k Dist_k - Dist_{k_z} \right)} \cdot \frac{q_{k_z}^2}{\sum_k q_k \left(\sum_k q_k - q_{k_z} \right)} \cdot \left\| \mathbf{w}_j^k(t) - \mathbf{x}(t) \right\|^2 \right], \tag{3.50}$$

[296] Vgl. Domschke [1997], S. 213 f.
[297] Vgl. Domschke [1997], S. 216 f.
[298] Vgl. Matsuyama [1991].

wobei

$$H_{k_z m} = \begin{cases} 1 & , Fahrzeug\ k_z\ darf\ Knoten\ m\ bedienen \\ \infty & , sonst \end{cases} . \tag{3.51}$$

Darf das Fahrzeug k_z $(z \in \{1,...,K\})$ den Knoten m bedienen, so nimmt der Faktor $H_{k_z m}$ den Wert Eins an, sonst wird die quadrierte euklidische Distanz zwischen Eingabevektor und Gewichtsvektor des Neurons j mit einer hinreichend großen Zahl multipliziert. Dadurch nimmt die resultierende Distanz einen hohen Wert an, der verhindert, dass Neuronen des Fahrzeugringes k_z als Zentrum ausgewählt werden. Der zweite Faktor dient dazu, die Längen der Fahrzeugtouren anzugleichen. Je größer das Verhältnis zwischen der quadrierten Länge der Tour $Dist_{k_z}$ des Fahrzeuges k_z und der Summe der mit den Längen $Dist_k$ der Fahrzeugtouren k multiplizierten Differenz der Gesamtlänge aller Touren zur Länge der betrachteten Tour k_z, um so größer wird durch die multiplikative Verknüpfung die resultierende Distanz ausfallen.

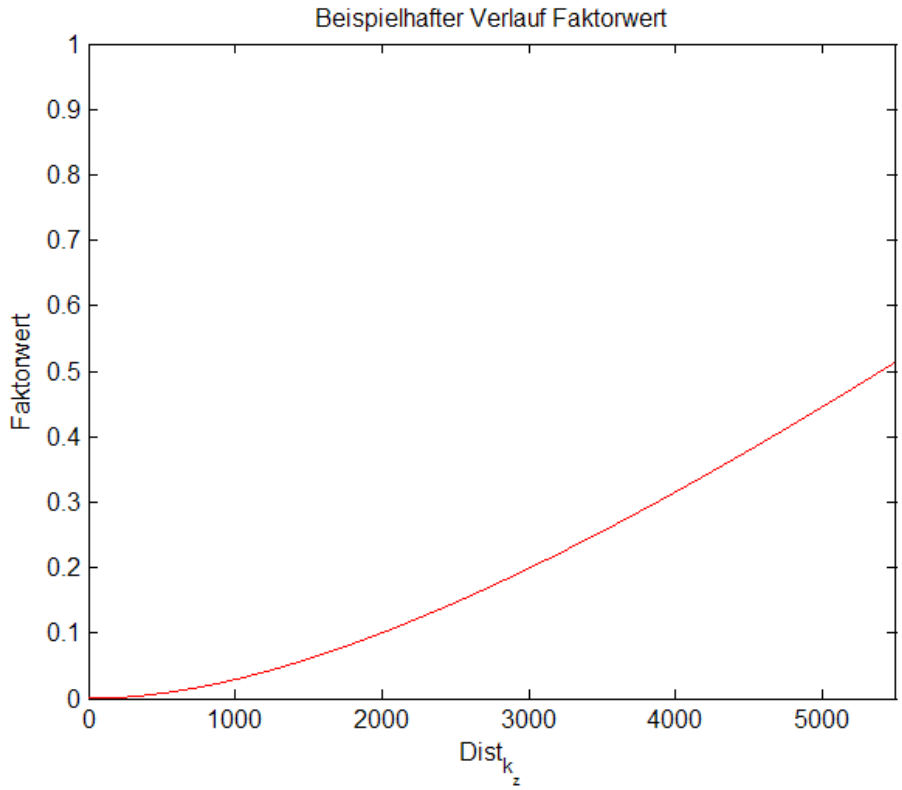


Abbildung 3.17: Verlauf Faktorwert ECVRP, MATSUYAMA

Analog zu diesem Vorgehen wird der dritte Faktor multiplikativ mit der quadrierten euklidischen Distanz verknüpft, um eine Angleichung der Kapazitätsauslastungen der Fahrzeuge zu bewirken. Dabei beschreibt q_{k_z} die dem Fahrzeug k_z zugeordnete Nachfrage, q_k die einem Fahrzeug k zugeordnete Nachfrage. Die Anpassung der Neuronen

des Neuronenrings, der das Zentrum enthält, wird gemäß Gleichung *(3.23)* vorgenommen. Die Präsentation der Eingabevektoren erfolgt zufällig. Wird der Depotknoten präsentiert, so wird für jeden Neuronenring ein Zentrum bestimmt und die Gewichtsvektoren der Neuronen der Nachbarschaft ebenfalls gemäß Gleichung *(3.23)* angepasst. MATSUYAMA implementiert außerdem einen Mechanismus, der ein Neuron dupliziert, wenn es zu häufig als Zentrum ausgewählt wird. Der Algorithmus bricht ab, wenn jedem Knoten ein Neuron eindeutig zugeordnet wurde. Die Ergebnisse für das erweiterte ECVRP zeigen, dass die Verwendung der genannten Funktion zur Bestimmung des Zentrums eine Angleichung der Kapazitätsauslastung der Fahrzeuge und der resultierenden Tourlängen bei gleichzeitiger Erhöhung der Gesamtlänge des Tourenplans bewirkt.[299]

GHAZIRI entwickelt 1991 die SOM-Variante HDN (*Hierarchical Deformable Net*), in der zunächst für jeden Ring das Neuron c ermittelt wird, das die geringste euklidische Entfernung zum präsentierten Eingabevektor $\mathbf{x}(t)$ aufweist. Anschließend wird jedem Neuronenring k durch[300]

$$P_A^k = \frac{P_B^k}{\sum_k P_B^k}, \text{ mit} \qquad (3.52)$$

$$\Delta Q_k = Q_k - q_k \text{ und} \qquad (3.53)$$

$$P_B^k = \frac{e^{-\frac{\left\|\mathbf{w}_c^k(t)-\mathbf{x}(t)\right\|}{h(t)}}}{1+e^{\frac{-\Delta Q_k}{h(t)}}} \qquad (3.54)$$

eine Auswahlwahrscheinlichkeit P_A^k zugewiesen, mit der das entsprechende Fahrzeug dem Eingabevektor zugeordnet wird. Der Parameter Q_k beschreibt dabei die Kapazität des Fahrzeugs k, der Parameter q_k die dem Fahrzeug aktuell zugeordnete Nachfrage. Je größer die Distanz zwischen Eingabevektor und Gewichtsvektor des Neurons c im Ring k, umso geringer wird der Wert des Zählers in *(3.54)* ausfallen. Mit zunehmender freier Kapazität ΔQ_k des Ringes k wird somit der Wert des Parameters P_B^k steigen. Der Parameter $h(t)$ wird im Verlauf des Algorithmus langsam von hohen Werten ausgehend abgesenkt, um die Auswahlwahrscheinlichkeit für Ringe, welche die Kapazitätsbedingung verletzen, im Verlauf des Algorithmus zu vermindern. Im Vergleich mit dem Sa-

[299] Vgl. Matsuyama [1991], S. 388.
[300] Für eine Darstellung der Funktionen siehe Ghaziri [1991], S. 832.

vings-Verfahren[301] und dem Sweep-Verfahren[302] für drei Testinstanzen ermittelt der Algorithmus Ergebnisse vergleichbarer Güte (Anzahl Fahrzeuge in Klammern)[303]:

Instanz	Savings-Verfahren	Sweep-Verfahren	Ghaziri 1991	Ghaziri[304] 1993	Beste bekannte Lösung[305]
1	1079 (5)	1266 (7)	1133 (7)	1133,4	1042,11
2	831 (10)	937 (10)	836 (10)	836,6	819,56
3	585 (6)	532 (5)	586 (6)	545,1	524,61

Tabelle 3.15 : Ergebnisse GHAZIRI

GHAZIRI erweitert den Ansatz 1996, so dass auch Zeitrestriktionen berücksichtigt werden können (Variante HDNT). Es zeigt sich in einem Vergleich mit dem Savings-, dem Sweep-Verfahren und einem 1976 von MOLE und JAMESON vorgeschlagenen Verfahren[306], dass die Lösungsqualität der Variante HDNT ähnlich ist. Im Vergleich mit einem Tabu Search-Verfahren wird deutlich, dass die Lösungsgüte des Verbesserungsverfahrens überlegen ist, allerdings die 1,6 bis 15,2-fache Rechenzeit benötigt wird.[307]

POTVIN und ROBILLARD[308] verwenden eine Variante der SOM in der Initialisierungsphase eines heuristischen Konstruktionsverfahrens für Tourenplanungsprobleme mit Zeitfenstern, um sogenannte Seed-Kunden, die sich als Ausgangspunkt für die verwendete Einfügeheuristik eignen, zu identifizieren. Dazu wird eine SOM verwendet, deren Anzahl an Neuronen *j (j=1(1)J)* der Ausgabeschicht mit der Anzahl der zu verwendenden Fahrzeuge *K* übereinstimmt. Die Distanzfunktion

$$(3.55) \quad d_j = \left\| \mathbf{x}(t) - \mathbf{w}_j(t) \right\| \cdot u_j(t)$$

beschreibt die mit der Anzahl $u_j(t)$, für die Neuron *j* bis zum Zeitpunkt *t* als Zentrum ausgewählt wurde, multiplizierte euklidische Distanz zwischen Eingabevektor $\mathbf{x}(t)$, der die Koordinaten des Knotens *m (m=1(1)N)* enthält, und Gewichtsvektor $\mathbf{w}_j(t)$ des

[301] Vgl. zum Savings-Verfahren Clarke und Wright [1964].
[302] Vgl. zum Sweep-Verfahren Gillet und Miller [1974].
[303] Vgl. Ghaziri [1991], S. 834.
[304] Die zugehörige Quelle lag dem Verfasser zum Zeitpunkt der Bearbeitung leider nicht vor. Die Werte wurden Gendreau et al. [1997], S. 335, entnommen. Die Anzahl der Fahrzeuge wurde dort nicht angegeben.
[305] Entnommen aus Gendreau et al. [1997], S. 335.
[306] Vgl. Mole und Jameson [1976].
[307] Vgl. Ghaziri [1996], S. 658. Die Abweichungen der Rechenzeiten unterscheiden sich bei verschiedenen Testinstanzen.
[308] Vgl. Potvin und Robillard [1995].

Neurons *j*. Durch dieses Vorgehen soll verhindert werden, dass ein Neuron zu selten gewinnt und so im Verlauf des Algorithmus der zu Beginn zufällig gewählte Gewichtsvektor nicht adaptiert wird. Die Güte der Lösung wird mit Hilfe der Potentialfunktion

$$(3.56) \quad \frac{1}{N} \cdot \sum_{m=1}^{N} \sum_{j=1}^{J} M_{mj} \left\| \mathbf{x}_m - \mathbf{w}_j(t) \right\|^2 ,$$

$$(3.57) \quad M_{mj} = \begin{cases} 1 & , \textit{Kunde } m \textit{ ist Neuron } j \textit{ zugeordnet} \\ 0 & , \textit{sonst} \end{cases}$$

bewertet. Diese Funktion berechnet die durchschnittliche quadrierte euklidische Distanz zwischen den Gewichtsvektoren der Neuronen und den ihnen zugeordneten Knoten. Der Algorithmus bricht ab, wenn die Verbesserung des Potentials eine vorgegebene Grenze unterschreitet. Die Anpassung der Gewichtsvektoren erfolgt durch Linearkombination des Eingabe- und Gewichtsvektors, wobei der Linearkombinationsparameter stets kleiner Eins ist und mit zunehmender Iterationszahl abgesenkt wird. Als Seed-Kunde wird schließlich derjenige Knoten ausgewählt, dessen Entfernung zum zugeordneten Neuron der Ausgabeschicht minimal ist. Durch das Vorgehen wird die Lösungsqualität der verwendeten Heuristik verbessert, jedoch steigt im Gegenzug die zur Ermittlung der Lösung benötigte Rechenzeit an.[309]

Umfassende Untersuchungen zur Anwendung selbstorganisierender Karten auf verschiedene Tourenplanungsprobleme führt RETZKO 1996 durch. Zur Berücksichtigung der Kapazitätsrestriktion erweitert der Autor die Distanzfunktion zu

$$(3.58) \quad \left\| \mathbf{x}(t) - \mathbf{w}_j^k(t) \right\| + \gamma \cdot a_k ,$$

die zur euklidischen Distanz den mit einem konstanten Faktor γ gewichteten Auslastungsgrad a_k des einem Neuron zugeordneten Fahrzeugs *k* addiert.[310] Durch den linearen Ansatz des zweiten Terms wird die euklidische Distanz für Neuronen des zum Fahrzeug gehörigen Neuronenringes umso stärker erhöht, je höher die Auslastung des Fahrzeugs bei der aktuellen Zuordnung der Knoten zu den Fahrzeugen ausfällt. Wird durch die Zuordnung des zufällig präsentierten Eingabevektors dennoch eine Kapazitätsrestriktion verletzt, so wird die Zuordnung wieder aufgehoben und die Auswahlwahrscheinlichkeit des entsprechenden Eingabevektors erhöht, damit möglichst wenig Eingabevektoren keine Zuordnung zu Fahrzeugen aufweisen. Der Autor untersucht neben der vorgestellten Variante unter anderem die Dynamisierung des Auslastungsge-

[309] Vgl. Potvin und Robillard [1995], S. 135 ff.
[310] Vgl. Retzko [1996], S. 125.

wichts und Ansätze zur Bearbeitung von Problemen, die keine euklidischen Distanzen aufweisen, Möglichkeiten zur (zusätzlichen) Berücksichtigung von zeitkritischen Restriktionen und Multidepot-Tourenplanungsprobleme. Die Vielzahl der Untersuchungen verhindert eine detaillierte Darstellung, deshalb sollen hier lediglich die von RETZKO abgeleiteten Schlussfolgerungen kurz zusammengefasst werden[311]: Die Flexibilität der SOM genügt nach Ansicht des Autors weitgehenden Anforderungen, die an Algorithmen zur Lösung von Tourenplanungsproblemen mit unterschiedlichen Restriktionstypen gestellt werden. RETZKO hebt den vergleichsweise geringen Aufwand zur Anpassung des Standard-Algorithmus an unterschiedliche Problemstellungen hervor. Die Lösungsqualität wird im Vergleich zu alternativen Verfahren[312] als konkurrenzfähig bezeichnet, wobei weitere Potentiale zur Verbesserung des Lösungsverhaltens durch Hybridisierung mit Verbesserungsverfahren vermutet werden.

TORKI ET AL.[313] untersuchen 1997 die Anwendung einer SOM-Variante mit Präsentationszyklen und vergleichen die Ergebnisse mit der von VAKHUTINSKY und GOLDEN[314] vorgeschlagenen Erweiterung der Elastic Net-Methode zur Lösung des ECVRP. Fast identische Untersuchungen wurden 1999 auch von MODARES ET AL.[315] veröffentlicht. Die Anzahl der Neuronen der Ausgabeschicht J entspricht jeweils dem Zweifachen der Zahl der in der jeweiligen Probleminstanz vorhandenen Knoten N. Die von TORKI ET AL. vorgeschlagene Distanzfunktion

$$d_j^k = \left\| \mathbf{x}(t) - \mathbf{w}_j^k(t) \right\| + \nu(t) \cdot O_k \quad \text{mit} \tag{3.59}$$

$$O_k = \left(\frac{q_k}{1 + Q_k} \right)^2 = \left(\frac{Fahrzeug\ k\ zugewiesene\ Nachfrage}{1 + (Kapazität\ Fahrzeug\ k)} \right)^2 \tag{3.60}$$

beziehungsweise bei MODARES ET AL.

$$O_k = \frac{q_k}{Q_k^2} = \frac{Fahrzeug\ k\ zugewiesene\ Nachfrage}{(Kapazität\ Fahrzeug\ k)^2} \tag{3.61}$$

[311] Vgl. Retzko [1996], S. 281 ff.
[312] Retzko untersucht mehrere Verfahren, der überwiegende Teil der durchgeführten Untersuchungen vergleicht die Lösungsqualität der SOM-Varianten mit dem Savings- und dem Sweep-Verfahren.
[313] Vgl. Torki et al. [1997].
[314] Vgl. Vakhutinsky und Golden [1994].
[315] Vgl. Modares et al. [1999].

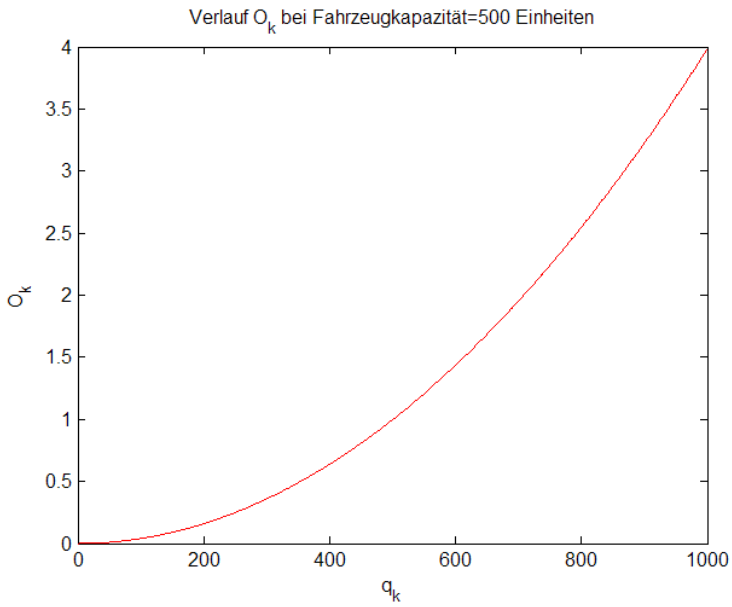


Abbildung 3.18 : Verlauf O_k TORKI ET AL. (Q_k =500)

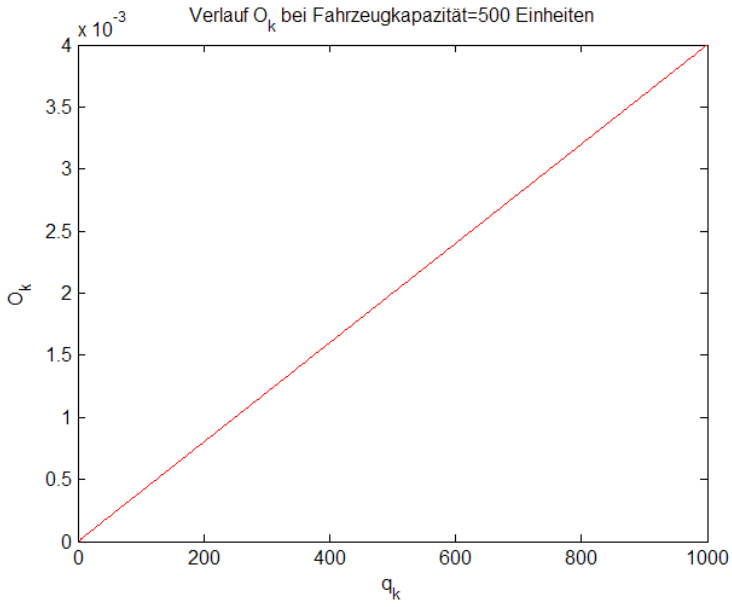


Abbildung 3.19 : Verlauf O_k MODARES ET AL. (Q_k =500)

berechnet die um den mit dem Parameter $\nu(t)$ gewichteten „Auslastungsgrad“ erhöhte euklidische Distanz zum Eingabevektor. Das Neuron, für das diese Distanz minimal ist, wird wieder als Zentrum deklariert. Der Wert des Parameters $\nu(t)$ wird im Verlauf des Algorithmus gemäß

$$(3.62) \quad \nu(t+1) = \frac{\nu(t)}{\left(1 + C \cdot \nu(t)\right)}$$

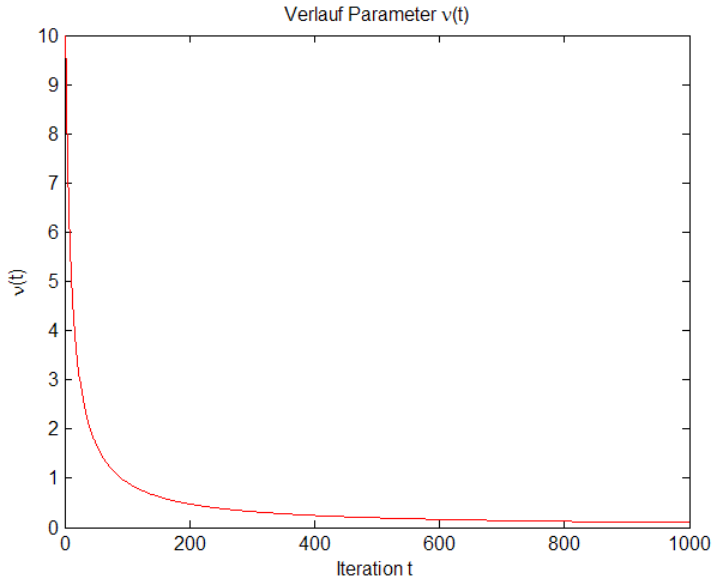


Abbildung 3.20 : Verlauf Parameter $\nu(t)$, TORKI ET AL. und MODARES ET AL.

vermindert (C stellt eine Konstante dar), da die euklidische Distanz zwischen Neuronen und Eingabevektoren mit zunehmender Iterationszahl sinkt und der zweite Term sonst zunehmenden Einfluss gewinnen würde.[316] Die Anpassung der Neuronen erfolgt mit Hilfe der Adaptionsvorschrift

$$\mathbf{w}_j^k(t+1) = \mu \cdot h_{cj} \cdot \left(\mathbf{x}(t) - \mathbf{w}_j^k(t)\right) + \lambda \cdot \left[\left(\mathbf{w}_{j+1}^k(t) - \mathbf{w}_j^k(t)\right) + \left(\mathbf{w}_{j-1}^k(t) - \mathbf{w}_j^k(t)\right)\right], \quad (3.63)$$

wobei

$$h_{cj}\left(G(t_p), \Delta_{cj}\right) = \begin{cases} e^{\frac{-\Delta_{cj}^2}{G(t_p)^2}} & , \Delta_{cj} < 0{,}2 \cdot J \\ 0 & , sonst \end{cases} . \quad (3.64)$$ [317]

Die einzelnen Terme der Adaptionsvorschrift können mit Hilfe der Parameter μ und λ gewichtet werden. Die vorgeschlagene SOM-Variante weist in beiden Veröffentlichungen im Vergleich mit der von VAKHUTINSKY und GOLDEN untersuchten Elastic Net-Variante in allen Testinstanzen eine überlegene Lösungsgüte auf,[318] die durchschnittliche Abweichung von der jeweils besten bekannten Lösung beträgt 3,9%.

[316] Vgl. Modares et al. [1999], S. 600.
[317] Vgl. zur Definition des Parameters $G(t_p)$ Gleichung *(3.19)*, S. 59.
[318] Vgl. Modares et al. [1999], S. 605.

3.3.3 Zusammenfassung und Diskussion

Tabelle 3.16 gibt eine zusammenfassende Übersicht über SOM-Varianten zur Lösung des ECVRP:

Quelle	**Wesentliche Merkmale**	**Vergleichsverfahren**[319]
MATSUYAMA [1991]	Integration zusätzlicher Nebenbedingungen, die Zuordnungen der Knoten zu bestimmten Handlungsreisenden verbieten; Einführung von Faktoren in der Distanzfunktion, um die Einhaltung der Kapazitätsrestriktionen und eine Angleichung der Tourenlängen zu bewirken	-
GHAZIRI [1991]	Zuweisung einer Auswahlwahrscheinlichkeit eines Neuronenringes bei Präsentation eines Eingabevektors	Savings-Verfahren (+) Sweep-Verfahren (-)
RETZKO [1996]	Verschiedene Erweiterungen zur Lösung von Problemstellungen der Tourenplanung	überwiegend Savings-Verfahren Sweep-Verfahren[320]
GHAZIRI [1996]	Integration von Zeitrestriktionen	Savings-Verfahren (-) Sweep-Verfahren (+) Algorithmus von MOLE und JAMESON (-) Tabu Search Prozedur (+)
TORKI ET AL. [1997] und MODARES ET AL. [1999]	Addition des gewichteten Auslastungsgrades, unterschiedliche Definition des Auslastungsgrades	Erweiterung des Elastic Net (-)

Tabelle 3.16: Übersicht über SOM-Varianten zur Lösung des ECVRP

Die Literaturübersicht zeigt, dass die SOM überwiegend zur Lösung von Tourenplanungsproblemen mit euklidischen Distanzen eingesetzt wird. Die Verwendung der SOM zur Identifizierung von Seed-Kunden kann als Ausnahme angesehen werden; Aus diesem Grunde soll in der folgenden Diskussion der Schwerpunkt auf die Anwendung zur Bestimmung einer Lösung für das ECVRP gelegt werden.

[319] Angaben in Klammern weisen wieder darauf hin, ob ein Vergleichsverfahren hinsichtlich der Lösungsqualität überlegene (+) oder unterlegene (-) Ergebnisse erzielt.
[320] Eine vergleichende Aussage soll hier nicht vorgenommen werden, da je nach untersuchter Problemstellung unterschiedliche Ergebnisse bzgl. der Vorteilhaftigkeit der Vergleichsverfahren erzielt wurden.

Alle vorgestellten Varianten legen zu Beginn die Anzahl der eingesetzten Fahrzeuge fest, d.h., der Algorithmus muss gegebenenfalls mehrfach mit unterschiedlicher Zahl an Neuronenringen ausgeführt werden. Der von MATSUYAMA vorgeschlagene Ansatz kann aufgrund fehlender Vergleiche mit anderen Verfahren nicht beurteilt werden. Die von GHAZIRI vorgeschlagene Variante führt im Vergleich mit bekannten Eröffnungsverfahren zu Lösungen ähnlicher Güte, zumindest eines der alternativen Verfahren ermittelt allerdings stets eine bessere Lösung. Stellt man die Ergebnisse der SOM-Ansätze den Ergebnissen der besten verfügbaren Verbesserungsverfahren gegenüber, so unterliegen die ermittelten Lösungen erwartungsgemäß.[321] Der Ansatz für das MINMAX-EMTSP von SOMHOM ET AL., in dem die SOM um ein Verbesserungsverfahren erweitert wird, könnte auch bei der Lösung des ECVRP zu einer Verbesserung der Konkurrenzfähigkeit gegenüber Verbesserungsverfahren führen.

Während bei den Anwendungen der SOM auf das ETSP und das EMTSP stets zulässige Lösungen ermittelt wurden, ist dies bei Anwendung auf Tourenplanungsprobleme nicht zwangsläufig der Fall. Wird beispielsweise die Gewichtung der Auslastung der Fahrzeuge in der Distanzfunktion von RETZKO, MODARES ET AL. und TORKI ET AL. zu gering gewählt, so ergeben sich - insbesondere für Probleme mit scharfen Kapazitätsrestriktionen bei Verwendung der minimal notwendigen Fahrzeugzahl - gegebenenfalls unzulässige Lösungen, welche die Kapazitätsrestriktionen verletzen. Wird hingegen das Auslastungsgewicht zu hoch gewählt, so ergeben sich unter Umständen zulässige Lösungen, die einen unnötig hohen Zielfunktionswert aufweisen.[322] Somit stellt sich bei der Anwendung das Problem, geeignete Gewichtungen zu identifizieren, welche einen hinreichend geringen Anteil an unzulässigen Lösungen, dafür aber auch zulässige Lösungen mit einem möglichst geringen Zielfunktionswert aufweisen. Zusätzlich ist ein linearer Ansatz des Auslastungsgrades zu kritisieren, da - gerade bei scharfen Kapazitätsrestriktionen - Konstellationen von zulässigen und unzulässigen Fahrzeugauslastungen auftreten können, die nur geringfügig voneinander abweichen, so dass die resultierende Erhöhung des Distanzwertes für das überlastete Fahrzeug im Vergleich zu einem Fahrzeug, dessen Kapazitätsrestriktion gerade noch erfüllt wird, zu klein ausfällt. Auch bei einer quadratischen Modellierung des Auslastungsgrades sind Konstellationen denkbar, die gerade bei geringen Kapazitätsüberschreitungen zu einer nicht ausreichenden Erhöhung der Distanz führen können. Einen Ausweg könnte hier ein Ansatz darstel-

[321] Vgl. beispielsweise die in Gendreau et al. [1997] präsentierten Ergebnisse für Simulated Annealing (S. 315) und verschiedene Tabu Search-Varianten (S. 317).
[322] Vgl. Retzko [1996], S. 141.

len, der den Exponenten der Fahrzeugauslastung in Abhängigkeit von der prozentualen Überschreitung der Kapazität anpasst. So könnte für sehr geringe Überschreitungen der Exponent stark erhöht werden, um die resultierende Gesamtdistanz ebenfalls stärker zu erhöhen. Im Vergleich mit alternativen Tourkonstruktions- beziehungsweise Eröffnungsverfahren ist die Lösungsqualität des Verfahrens auf der Grundlage der Literaturergebnisse insgesamt als vergleichbar einzuschätzen.

Abschließend ist zu bemerken, dass eine Integration von *zusätzlichen* Bedingungen wie beispielsweise Zeitfensterrestriktionen zu Schwierigkeiten führen kann, da gegebenenfalls gegenläufige Wirkungen zusätzlicher Terme beziehungsweise Faktoren in der Distanzfunktion auftreten können.

3.3.4 Anwendungsbeispiel

Tabelle 3.17 gibt einen Überblick über das Anwendungsbeispiel für das ECVRP. Zur Berücksichtigung der Kapazitätsrestriktionen wird in der vom Verfasser implementierten Variante die gewichtete Fahrzeugauslastung in der Distanzfunktion berücksichtigt.

Merkmal	**Wert**	**Bemerkung**
Anzahl Knoten	83	Präsentierte Eingabevektoren.
Anzahl Neuronen der Eingabeschicht = Anzahl Komponenten des Eingabevektors	2	Koordinaten der Knoten.
Anzahl Fahrzeuge K = Anzahl Neuronenringe	4	Annahme.
Kapazität je Fahrzeug	230 Kapazitätseinheiten	Homogene Fahrzeugflotte, heterogene Flotte möglich.
Gesamtnachfrage	830 Kapazitätseinheiten	Nachfrageverteilung heterogen.
Kapazitätsüberschuss	10,8%	-
Anzahl Neuronen der Ausgabeschicht	321	Je Neuronenring ergibt sich eine Anzahl von 81 Neuronen.
Präsentationszyklen	$t_{p\max} = 300$ Zyklen	-
Anzahl präsentierter Eingabevektoren	24.900	$t_{\max} = t_{p\max} \cdot 83$
Verlauf des Parameters $\sigma(t_p)$	$\sigma(t_p) = \sigma_{t_p=0} \cdot \left(\frac{\sigma_{t_{p\max}}}{\sigma_{t_p=0}} \right)^{\frac{t_p}{t_{p\max}}}$	Steuert die Breite der Nachbarschaftsfunktion.
Initialisierungswert des Parameters $\sigma(t_p)$	$\sigma_{t_p=0} = 6$	-
Verlauf der Lernrate	$\alpha(t_p) = \max\left\{1 - \frac{t_p}{t_{p\max}}, 0{,}05\right\}$	Steuert die Stärke der Adaption der Gewichtsvektoren.
Distanzfunktion	$\left\|\mathbf{x}(t) - \mathbf{w}_j^k(t)\right\|^2 + \gamma \cdot \left(\frac{q_k}{Q_k} \right)$	Die euklidische Distanz wird um den gewichteten Auslastungsgrad des Fahrzeuges, zu dessen Ring das Neuron gehört, erhöht.
Auslastungsgewichtung	$\gamma = 10$	-
Bestimmung des Zentrums	Neuron mit geringster Distanz.	Bei Präsentation des Depots wird für jeden Neuronenring ein Zentrum bestimmt und entsprechend für jeden Ring Gewichtsadaptionen ausgeführt.
Reihenfolge der Präsentation der Eingabedaten	Zufällig.	-
Initialisierung der Gewichtsvektoren	Jeweils Ringanordnung der einem Fahrzeug zugeordneten Neuronen.	Die Wahl der Kreismittelpunkte der Ringe erfolgt zufällig.
Topologie der Ausgabeschicht	Ringe.	Abbildung der Touren.
Nachbarschaftsfunktion	$h_{cj}(t_p) = \alpha(t_p) \cdot \exp\left(-\frac{\Delta_{cj}^2}{2 \cdot \sigma^2(t_p)} \right)$	-
Lernregel	$\mathbf{w}_j^k(t+1) = \left(1 - h_{cj}(t_p)\right) \cdot \mathbf{w}_j^k(t) + h_{cj}(t_p) \cdot \mathbf{x}(t)$	-

Tabelle 3.17 : Merkmale des Anwendungsbeispiels für das ECVRP

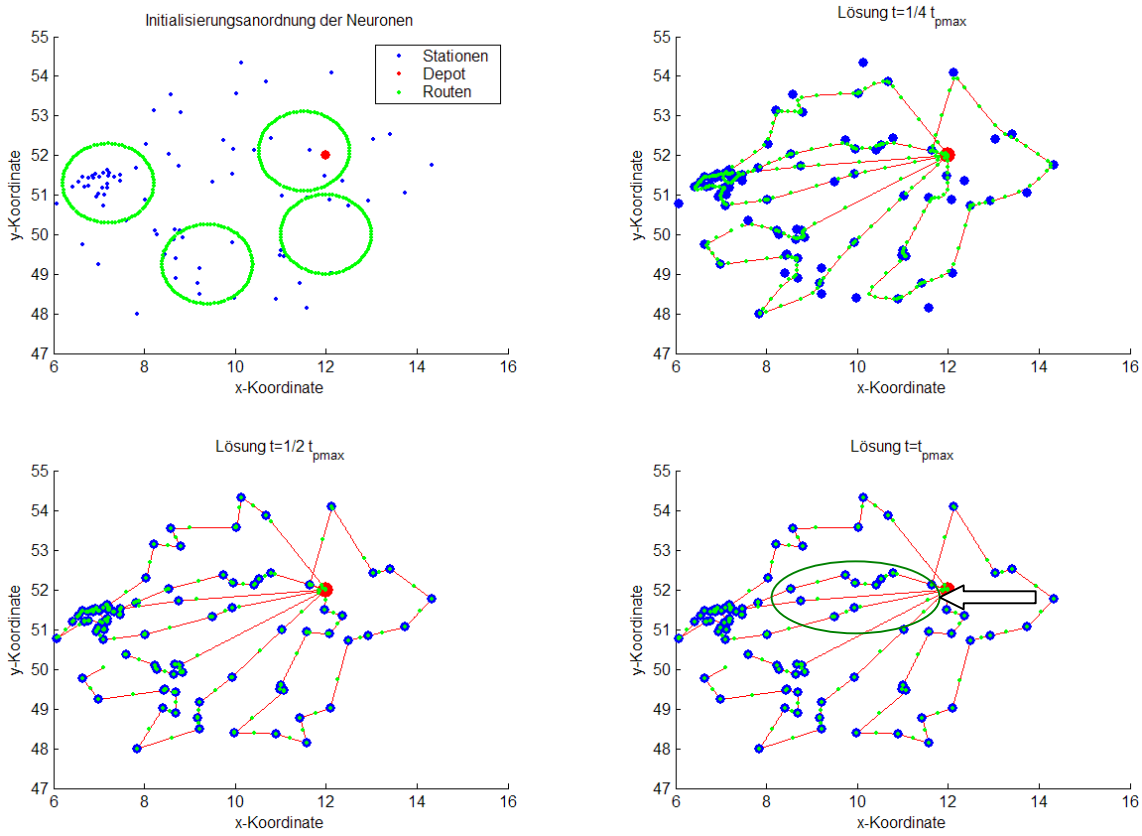


Abbildung 3.21 : Entfaltung der Self-Organizing Map, ECVRP-Beispiel

Wie in der Abbildung ersichtlich (siehe Markierung) überschneiden sich die Neuronenringe in der ermittelten Lösung. Die Ausbildung solcher topologischer Defekte bezüglich der Abbildung der Nachbarschaftsbeziehung der Knoten wird unter Umständen durch die modifizierte Distanzfunktion verstärkt: Weist ein Neuronenring gegenüber einem zweiten Ring einen hohen Auslastungsgrad auf, so wird die euklidische Distanz vergrößert, und einem Neuron des Ringes mit der geringeren Auslastung wird gegebenenfalls trotz einer größeren euklidischen Distanz zum Eingabevektor insgesamt eine geringere Gesamtdistanz zum Eingabevektor zugewiesen. Folglich wird dieses Neuron als Zentrum ausgewählt. Durch die anschließende Anpassung der Gewichtsvektoren der Neuronen des entsprechenden Ringes können dann Überschneidungen der Fahrzeugtouren auftreten. Die ermittelte Lösung wird hierdurch nicht unzulässig, weist aber unter Umständen unnötig hohe Zielfunktionswerte auf.

3.4 Weitere Optimierungsprobleme der betriebswirtschaftlichen Logistik

In diesem Abschnitt sollen weitere Ansätze zur Lösung verschiedener Optimierungsprobleme der betriebswirtschaftlichen Logistik, die für die im Kapitel 4 folgenden Ausführungen nur eine untergeordnete Rolle spielen, lediglich kurz beleuchtet werden, um die Breite des Anwendungsspektrums der SOM zu dokumentieren.

Es existieren Veröffentlichungen, welche die Reihenfolge von *Pick-Up and Delivery*-Aufträgen für automatisch gesteuerte Transportfahrzeuge mit Hilfe von Self-Organizing Maps festlegen[323]. Dabei sollen meist möglichst kurze Routen für die Fahrzeuge ermittelt werden. Als wesentlicher Unterschied zu den bereits beschriebenen Ansätzen ist die Anzahl der Neuronen der Eingabeschicht anzusehen, die aus vier Neuronen besteht. Diese dienen neben der Aufnahme der Quellkoordinaten eines Auftrages auch der Aufnahme der Zielkoordinaten für die Ablieferung des im Quellort aufgenommenen Transportgutes. Darüber hinaus werden verschiedene Modifikationen bei der Auswahl des Zentrums und der Bestimmung der Nachbarschaftsbeziehungen vorgenommen. Vergleiche mit Nearest Neighbor-Verfahren führen zu dem Ergebnis, dass die SOM-Ansätze meist bessere Lösungen bei höherer Laufzeit ermitteln, deren Zielfunktionswerte zwischen 2% und 28% von der Optimallösung der jeweiligen Probleminstanz abweichen.[324]

SMITH entwickelt 1988 eine SOM-Variante zur Lösung des in der innerbetrieblichen Standortplanung auftretenden *Quadratischen Zuordnungsproblems*, die nicht auf der Grundidee eines elastischen Ringes aufbaut und auch nicht im euklidischen Raum operiert.[325] Vielmehr werden dem Netz Zeilen einer Permutationsmatrix präsentiert, die jeweils genau in einer Spalte den Wert Eins enthalten (alle übrigen Spalten enthalten den Wert Null). Verschiedene Operationen, auf die hier nicht vertiefend eingegangen werden soll, führen schließlich zur Ableitung einer zulässigen Lösung des Problems. SMITH gelingt es, in 30% der Testläufe die Optimallösung einer Testinstanz geringen Umfangs zu bestimmen, die übrigen 70% der Lösungen stellen lokale Optima dar.[326] Der wesentliche Unterschied des Ansatzes ist in der Abkehr von geometrisch orientierten Operationen zu sehen, die es ermöglicht, die vorgeschlagene Modifikation auf eine Reihe unterschiedlicher Optimierungsprobleme anzuwenden, die keine euklidischen Distanzen aufweisen.

---

[323] Vgl. Hao und Lai [1996] und Soylu et al. [2000].
[324] Vgl. z.B. Soylu et al. [2000], S. 133 f.
[325] Vgl. Smith [1988] und Kwok und Smith [2004].
[326] Vgl. Smith [1988], S. 1878.

LOZANO ET AL. stellen 1998[327] die Anwendung der SOM auf *Location-Allocation Probleme*[328] vor, bei denen neben den Standorten von neu zu errichtenden Einrichtungen (beispielsweise Depots oder Fabriken) simultan eine Zuordnung von Güterflüssen zwischen den neuen und bereits existierenden Einrichtungen (beispielsweise Kunden oder Fabriken) zu bestimmen ist[329]. Neben der Untersuchung diskreter Nachfrageverteilungen zeigen die Autoren auch Anwendungsmöglichkeiten des Algorithmus auf Problemstellungen mit kontinuierlich in der Ebene verteilter Nachfrage auf. Die Eingabevektoren, welche im Fall der diskreten Nachfrageverteilung die Koordinaten der Nachfrageknoten enthalten, werden zufällig gemäß einer durch die Nachfrage der Kunden spezifizierten Wahrscheinlichkeitsverteilung präsentiert. Die Autoren schlagen unter anderem ein zweistufiges Vorgehen vor, in dem die SOM zunächst verwendet wird, um eine Zuordnung der Nachfrageknoten zu den Depotknoten, die durch die Neuronen der Ausgabeschicht abgebildet werden, zu ermitteln. In einem zweiten Schritt werden dann die mit der SOM ermittelten Standorte und Zuordnungen unter Verwendung des von COOPER vorgeschlagenen alternierenden Location-Allocation-Verfahrens (ALA) angepasst[330]. Die Ergebnisse der Untersuchungen zeigen, dass die mit der SOM abgeleiteten Lösungen durch den Anschluss des ALA geringfügig verbessert werden können[331].

ALTINEL ET AL. verwenden eine Variante, die den Mittelwert der Verteilung der Neuronen nach jedem Adaptionsschritt durch Adaptionen der Gewichtsvektoren der Neuronen, die nicht zur Nachbarschaft zählen, an den Mittelwert der Eingabevektoren anpasst. Die Variante wird zur Lösung des Problems der Bestimmung eines *Hamiltonschen Pfades* verwendet. Die Qualität der abgeleiteten Lösung im Vergleich mit Lösungen einer Reihe alternativer Verfahren wird als sehr gut bezeichnet.[332]

GHAZIRI und OSMAN[333] entwickeln eine SOM-Variante, die verwendet werden kann, um eine *Erweiterung des TSP* zu lösen, in der neben N_1 zu beliefernden Knoten zusätzlich N_2 Knoten existieren, in denen im Anschluss an die Bedienung aller zu beliefernden Knoten Güter abgeholt werden müssen. Zur Lösung des Problems verwenden die Autoren zwei Neuronenketten. Während Neuronen der ersten Kette nur auf Knoten reagieren, in denen Güter abgeliefert werden müssen, reagieren die Neuronen der zweiten

[327] Vgl. Lozano et al. [1998].
[328] Eine Übersicht über unterschiedliche Probleme dieser Klasse und geeignete Lösungsverfahren befindet sich in Love et al. [1988].
[329] Vgl. Love et al. [1988], S. 3.
[330] Vgl. Cooper [1963].
[331] Vgl. Lozano et al. [1998], S. 113.
[332] Vgl. Altinel et al. [2000], S. 461, und S. 57.
[333] Vgl. Ghaziri und Osman [2003]

Kette auf die Knoten, in denen Güter abgeholt werden müssen. Um eine Rundreise zu generieren, wird eine Interaktion zwischen den beiden Ketten modelliert, die dazu führt, dass sich das Ende der ersten und der Anfang der zweiten Kette annähern. Hierzu wird der ersten Kette die Position des ersten Neurons in der zweiten Kette präsentiert. Als Zentrum wird das letzte Neuron der ersten Kette bestimmt und die Gewichtsvektoren entsprechend angepasst. Ebenso wird der zweiten Kette das letzte Neuron der ersten Kette präsentiert, wobei das erste Neuron in der zweiten Kette als Zentrum deklariert wird. Um sicherzustellen, dass die Rundreise im Depot startet und endet, wird der Depotknoten beiden Ketten präsentiert. Als Zentrum werden das erste Neuron der ersten Kette und das letzte Neuron der zweiten Kette deklariert. Durch die Adaptionsprozesse verbinden sich die Neuronenketten im Laufe des Verfahrens zu einem Neuronenring und somit zu einer geschlossenen Tour. Vergleiche mit den derzeit besten Eröffnungs- und Verbesserungsverfahren zur Lösung des Problems zeigen, dass die SOM-Variante bei längeren Rechenzeiten bezüglich der Lösungsqualität konkurrenzfähige Ergebnisse erzielt[334].

Neben den beschriebenen Anwendungen finden sich in der Literatur auch Ergebnisse praxisorientierter Untersuchungen aus dem Bereich der *Robotersteuerung*[335] und der *Platinenbestückung*. Hierbei sind meist komplexere Zielfunktionen zu berücksichtigen, die beispielsweise neben der Minimierung der Länge der Leiter zwischen den Bauteilen den Einfluss elektromagnetischer Störfelder[336] oder komplexerer Maschinenbestückungstechniken berücksichtigen[337].

[334] Vgl. Ghaziri und Osman [2003], S. 279. Bei Vergleich mit Verbesserungsverfahren wird eine Variante verwendet, die um eine 2-Opt Prozedur erweitert wurde.
[335] Vgl. Plebe und Anile [2001].
[336] Vgl. beispielsweise Ang et al. [2002].
[337] Vgl. beispielsweise Tokutaka und Fujimura [1999].

4. Erweiterung der Self-Organizing Map zur Lösung simultaner Standort- und Tourenplanungsprobleme

4.1 Motivation und Beschreibung der Probleme der simultanen Standort- und Tourenplanung

Simultane Ansätze zur Bestimmung der Lage des / der Depots, der Zuordnung zu besuchender Kunden zu Depots und Fahrzeugtouren und der Routenführung sind in der Lage, Interdependenzen der Teilprobleme, deren Vernachlässigung zu einer unnötigen Verschlechterung des Zielfunktionswertes führen kann[338], zu berücksichtigen.

Erste Ansätze zur Berücksichtigung der Interdependenzen zwischen Standort- und Transportentscheidungen entwickelt COOPER 1972. Er verbindet ein Location-Allocation Problem mit einem Transportproblem und schlägt verschiedene Lösungsverfahren für das resultierende Gesamtproblem vor[339]. Wie auch das im folgenden Abschnitt 4.1.1 beschriebene Weber-Problem berücksichtigt diese Vorgehensweise allerdings nicht, dass die Verteilung der Transportgüter auf Touren vorgenommen werden kann. Kann ein Fahrzeug jedoch mehrere Kunden bedienen, ohne zum Depot zurückkehren zu müssen, so stellt sich simultan zum Problem der Standortbestimmung ein Problem der Routenplanung generell und insbesondere der Tourenplanung, die hier aufgegriffen werden soll.

Abschnitt 4.1.2 gibt zunächst einen Überblick über die Interdependenzen zwischen den Teilproblemen simultaner Ansätze. Im Anschluss wird ein Problem formuliert, das in der Bestimmung des Standortes eines in der Ebene anzuordnenden Depots, der Zuordnung der Kunden zu Fahrzeugtouren und der Routenführung der Fahrzeuge selbst besteht.

Da schon das in dieser Formulierung enthaltene Tourenplanungsproblem zur Klasse NP-schwerer Probleme zu zählen ist, ist eine Verwendung von Heuristiken zur Ableitung einer Lösung für über kleinere Demonstrationsbeispiele hinausgehende Probleminstanzen notwendig. Aus Kapitel 3 wird deutlich, dass die SOM sowohl zur Lösung von Tourenplanungsproblemen als auch zur Lösung von Zuordnungs- und Standortpla-

[338] Vgl. Sahli und Rand [1989], S. 15 f.
[339] Vgl. Cooper [1972].

nungsproblemen[340] in der Ebene geeignet ist. Es bietet sich daher an, das Verfahren zur Lösung simultaner Problemstellungen zu nutzen.[341]

In Abschnitt 4.2 wird deshalb ein Versuch der Erweiterung der SOM unternommen, die eine Lösung des in Abschnitt 4.1.2 formulierten Problems ermöglicht. Ein Anwendungsbeispiel des entwickelten Algorithmus wird in Abschnitt 4.3 präsentiert, Abschnitt 4.4 stellt schließlich Testergebnisse für eine Reihe von Testinstanzen vor.

4.1.1. Kontinuierliche Standortplanung bei Stichfahrten

Ein bekanntes Problem der kontinuierlichen Standortplanung stellt das sogenannte Weber-Problem[342] dar. Dieses besteht darin, einen Punkt $W(x^*, y^*)$ in der Ebene zu bestimmen, der die Summe der gewichteten euklidischen Distanzen von diesem Punkt zu *N* vorgegebenen Punkten $P_n(x_n, y_n)$ *(n=1(1)N)* minimiert. Bezeichnet man die Gewichtung für den Punkt P_n mit w_n und die zugehörige euklidische Entfernung zwischen $W(x, y)$ und P_n mit

$$(4.1) \quad d_n(x,y) = \sqrt{(x - x_n)^2 + (y - y_n)^2} ,$$

dann kann das Problem durch

$$(4.2) \quad \min_{x,y} \left\{ W(x,y) = \sum_{n=1}^{N} w_n \cdot d_n(x,y) \right\}$$

beschrieben werden.[343]

Im weiteren Verlauf der Arbeit wird das folgende, von WEISZFELD entwickelte[344] und später von MIEHLE verbesserte[345] Näherungsverfahren für das Weber-Problem verwendet: Für gegebene Koordinaten (x^t, y^t) des Punktes W können in der *t*-ten Iteration die Koordinaten (x^{t+1}, y^{t+1}) durch

[340] Einen Überblick über verschiedene diskrete und kontinuierliche Standortplanungsprobleme geben Drezner und Hamacher [2002]. Avella et al. [1998] diskutieren den Stand der Forschung und geben Hinweise auf noch offene Fragen.
[341] Die Untersuchung der Erweiterung der Self-Organizing Map zur Lösung simultaner Problemstellungen wird beispielsweise auch von Retzko [1996], S. 282, empfohlen.
[342] Vgl. Weber [1909].
[343] Vgl. Drezner et al. [2002], S. 2.
[344] Vgl. Weiszfeld [1937].
[345] Vgl. Miehle [1958].

$$(4.3) \quad \left(x^{t+1}, y^{t+1}\right) = \left(\frac{\sum_{n=1}^{N} \frac{w_n \cdot x_n}{d_n(x^t, y^t)}}{\sum_{n=1}^{N} \frac{w_n}{d_n(x^t, y^t)}}, \frac{\sum_{n=1}^{N} \frac{w_n \cdot y_n}{d_n(x^t, y^t)}}{\sum_{n=1}^{N} \frac{w_n}{d_n(x^t, y^t)}} \right)$$

berechnet werden[346], wobei $\left(x^{t+1}, y^{t+1}\right)$ in der folgenden Iteration *(t+1)* die Koordinaten $\left(x^t, y^t\right)$ ersetzt, bis ein vorgegebenes Abbruchkriterium erfüllt wird.[347] Es ist darauf hinzuweisen, dass die Lösung des Weber-Problems nicht zwangsläufig eindeutig sein muss. Darüber hinaus versagt die beschriebene Prozedur, wenn die Koordinaten $\left(x^t, y^t\right)$ mit den Koordinaten eines Punktes $P_n\left(x_n, y_n\right)$ übereinstimmen, solange kein Fehler-Term zu der in den Nennern verwendeten Distanz addiert wird. Interpretiert man die Gewichtungen w_n als Nachfrage von Kunden, deren Standorte sich in den Koordinaten der Punkte P_n befinden, so kann der Algorithmus verwendet werden, um den Standort eines Depots zu bestimmen, der die Summe der gewichteten euklidischen Distanzen zu den Kunden minimiert. Das Verfahren unterstellt somit, dass die Kunden auf Stichfahrten besucht werden und auf jeder Fahrt eine Produkteinheit geliefert wird. Wie bereits zu Beginn des Kapitels erwähnt, ist es allerdings vorstellbar, dass ein Fahrzeug mehrere Kunden bedienen kann, ohne zum Depot zurückkehren zu müssen. Folglich stellen sich Probleme der simultanen Standort- und Tourenplanung, die im folgenden Abschnitt verdeutlicht werden.

[346] Vgl. Drezner et al. [2002], S. 9.
[347] Ein Beweis der Konvergenz des Verfahrens erbringt Kuhn [1973].

4.1.2 Simultane Standort- und Tourenplanung

Zur Erläuterung der grundsätzlichen Problemstellung der simultanen Standort- und Tourenplanung werden zunächst die Interdependenzen zwischen Standort- und Tourenplanung[348] in der Abbildung 4.1 verdeutlicht:

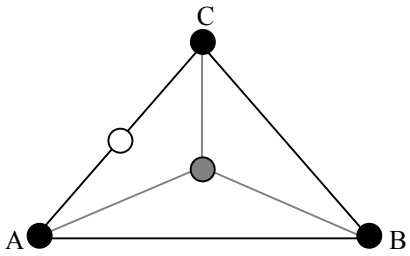


Abbildung 4.1 : Interdependenzen zwischen Standort- und Tourenplanung[349]

Wird für den Standort beispielsweise die Lösung des Weber-Problems für die drei Kunden A, B und C verwendet, so wird der Standort im grau markierten Punkt errichtet. Sollen die Kunden allerdings innerhalb einer Tour beliefert werden, so wäre unter der Zielsetzung der Minimierung der zurückgelegten Gesamtdistanz jeder Punkt auf einer der Verbindungslinien zwischen den Kunden A, B und C vorteilhaft. Setzt man voraus, dass langfristige Kundenbeziehungen existieren und keine kurzfristigen Schwankungen der in die Tourenplanung einfließenden Daten auftreten, so erscheint eine simultane Standort- und Tourenplanung also im Sinne der zugrunde liegenden Zielsetzung notwendig.

Abbildung 4.2 verdeutlicht zusammenfassend die Interdependenzen der Teilentscheidungen, die zur Ableitung einer Lösung für Probleme der simultanen Standort- und Tourenplanung zu treffen sind.

[348] Vgl. beispielsweise Srivastava [1993], S. 498 und Salhi und Rand [1989], S. 150.
[349] Vgl. Salhi und Rand [1989], S. 151.

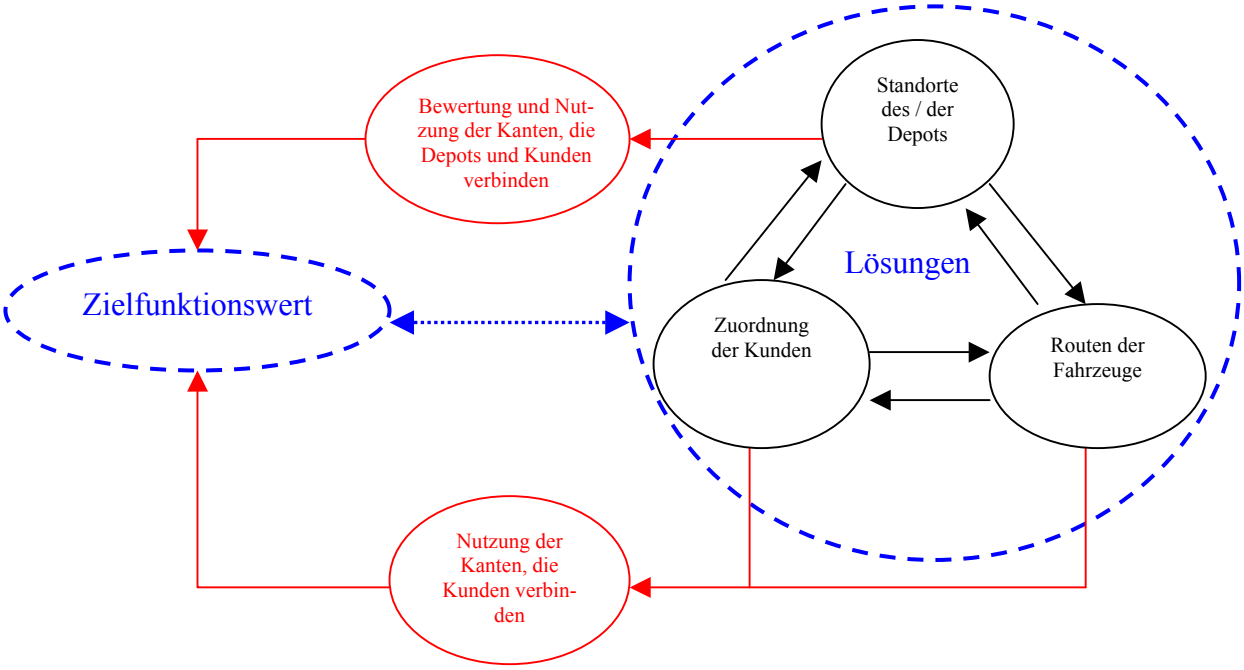


Abbildung 4.2 : Interdependenzen zwischen Standortwahl, Zuordnung und Routenführung

Zulässige Kombinationen der folgenden Entscheidungen bilden die Menge der zulässigen Lösungen simultaner Problemstellungen:

1. Die Entscheidung über den / die Standort(e) der / des Depots.
2. Die Entscheidung über die Zuordnung der Kunden zu Depots.
3. Die Entscheidung über die Zuordnung der Kunden zu Fahrzeugen.
4. Die Entscheidung über die Routen der Fahrzeuge.

Eine zulässige Lösung für das Problem wird mit Hilfe der Zielfunktion beurteilt. Aus dem von GUTENBERG formulierten erwerbswirtschaftlichen Prinzip[350] kann als geeignete Zielsetzung unternehmerischer Aktivitäten die Maximierung des Gewinns, also die Maximierung der Differenz zwischen Erlösen und Kosten, abgeleitet werden[351]. Stellt die Höhe der erzielbaren Erlöse für das untersuchte Planungsproblem eine feststehende Größe dar, so kann die Kostenminimierung als äquivalente Zielsetzung verwendet werden.[352] Die Voraussetzung feststehender Erlöse ist für viele Tourenplanungsprobleme erfüllt. Da in der Regel allerdings kein direkter funktionaler Zusammenhang zwischen der Gestalt eines Tourenplans und der durch diesen Plan verursachten Kosten hergestellt werden kann, müssen Ersatzzielsetzungen für die Kostenminimierung zur Beurteilung

[350] Vgl. Gutenberg [1983], S. 464 f.
[351] Vgl. bspw. Weissermel [1999], S. 10 f.
[352] Vgl. bspw. Dethloff [1994], S. 10.

der Güte eines Tourenplans herangezogen werden.[353] So kann die Zielfunktion beispielsweise von zur Distanz zwischen den Knoten proportionalen Kantenbewertungen abhängen. Die Bewertung der Kanten, welche die Depots mit den Kunden verbinden, ist dabei im Gegensatz zur Bewertung der Kanten, die Kunden verbinden, von der Wahl der Depot-Standorte abhängig.

Auch in der quantitativen Standortplanung wird typischerweise das Ziel der Kostenminimierung verfolgt[354]. Als Folge finden sich für simultane Standort- und Tourenplanungsprobleme ebenfalls überwiegend Zielfunktionen, die beispielsweise von den Kosten des Gütertransportes, den Betriebskosten für die Depots oder einer Kombination beider Kostenarten abhängen.[355]

Einen vertiefenden Überblick über Interdependenzen zwischen Standort- und Tourenplanung geben SAHLI und RAND[356], die einen zweistufigen Prozess entwickeln, in dem zunächst mit Hilfe unterschiedlicher Heuristiken eine Lösung für das Standortproblem, das ein Zuordnungsproblem für die Kunden beinhaltet, ermittelt wird. Anschließend wird ein heuristisches Verfahren zur Tourenplanung verwendet. Die Untersuchungen bestätigen, dass die in der ersten Stufe des Lösungsverfahrens ermittelten Standorte häufig von den für die Lösung des Tourenplanungsproblems bestmöglichen Standorten abweichen, d.h., die isolierte Betrachtung der Teilprobleme verursacht Lösungen mit unnötig hohen Zielfunktionswerten.

Viele Publikationen aus dem Gebiet der simultanen Standort- und Tourenplanung betrachten diskrete Problemstellungen[357], d.h., die Standorte können nur aus einer vorgegebenen, endlichen Menge potentieller Standorte ausgewählt werden.[358] Tatsächlich sind jedoch auch unterschiedliche Problemstellungen vorstellbar, die eine kontinuierliche Modellierung mit euklidischen Entfernungen erfordern, beispielsweise Probleme der Planung von Standorten im Bereich der Luftfahrt[359] und Probleme aus dem Bereich

[353] Vgl. Weissermel [1999], S. 12, und die dort angegebenen Quellen.
[354] Vgl. Love et al. [1988], S. 1 und Domschke [1996], S. 10. Einen Überblick über quantitative Standortfaktoren gibt Hansmann [2001], S. 103 f., ein Katalog nicht-quantifizierbarer Standortfaktoren findet sich bspw. in Hansmann [1974], S. 139 ff.
[355] Vgl. Laporte [1988], S. 167.
[356] Vgl. Sahli und Rand [1989].
[357] Eine Einführung in Problemstellungen dieser Klasse gibt Laporte [1988], Min et al. [1998] vermitteln einen Literaturüberblick und entwickeln ein Klassifikationsschema.
[358] Vgl. beispielsweise Jacobsen und Madsen [1980], Madsen [1981], Perl und Daskin [1985], Srivastava [1993], Hansen et al. [1994] und Laporte und Dejax [1998].
[359] Vgl. Love et al. [1988], S. 146, und Amin et al. [1994], S. 121.

der Halbleiterfertigung oder der Platinenbestückung[360]. Darüber hinaus kann die Lösung eines kontinuierlichen Modells als Näherungslösung für diskrete Problemstellungen verwendet werden. Durch die zunehmende Internationalisierung von Distributionsstrukturen sind zudem Probleminstanzen vorstellbar, die sehr große Gebiete abdecken.[361] Dies kann unter Umständen zu einer großen Anzahl potentieller Standorte führen, die durch die Ergebnisse kontinuierlicher Modelle vermindert werden kann.[362] Die so vorgenommene Einschränkung der Menge potentieller Standorte kann zu einer Verminderung des notwendigen Speicher- und Rechenzeitbedarfs gemischt-ganzzahliger Optimierungsverfahren für diskrete Optimierungsprobleme beitragen und den Aufwand zur Ermittlung notwendiger Daten bezüglich der Standorte verringern.[363]

ENGELE entwickelt 1980 mehrstufige Modelle für kontinuierliche Problemstellungen mit einem und mehreren Depots[364]. Im Folgenden wird ein einstufiges Modell vorgestellt, das durch Modifikation der von ENGELE formulierten Zielfunktion abgeleitet werden kann[365]. Dieses Problem besteht in der Bestimmung des Standortes eines kontinuierlich in der Ebene platzierbaren, unkapazitierten Depots und der simultanen Bestimmung der Touren für eine zu bestimmende Anzahl kapazitierter Fahrzeuge, deren Touren im Depot beginnen und enden. Bei der Formulierung wird unterstellt, dass die Kantenbewertungen den euklidischen Entfernungen zwischen den Knoten entsprechen, es handelt sich also um eine symmetrische Kostenmatrix $\boldsymbol{C}(c_{mn})$. Das Problem wird daher als kontinuierliches euklidisches Location-Routing Problem (CELRP) bezeichnet. Eine Modellformulierung für das Problem wird in Tabelle 4.2 angegeben, die Definitionen der im Modell verwendeten Symbole können Tabelle 4.1 entnommen werden.

[360] Vgl. Fujimura et al. [2001].
[361] Vgl. Lozano et al. [1998], S. 107.
[362] Kontinuierliche Modelle werden häufig eingesetzt, um mögliche Standort-Kandidaten für diskrete Modelle zu identifizieren, vgl. Love et al. [1988], S. 143 ff., und Lozano et al. [1998], S. 107.
[363] Vgl. Vogel [1975], S. 14 f.
[364] Vgl. Engele [1981], S. 90ff. und S. 117 ff.
[365] Vgl. Engele [1981], S. 91. Der Autor berücksichtigt in der Zielfunktion zusätzlich einen Term, der die gewichtete euklidische Entfernung zwischen einem bereits existierenden Produktionsstandort und dem zu planenden Depotstandort beschreibt.

Symbol	Interpretation
N	Anzahl der Knoten (Kunden)
K	Anzahl der Fahrzeuge
m	Index der Knoten
n	Index der Knoten
k	Index der Fahrzeuge
0	Index des Depotknotens
c_{mn}	Bewertung der Kante zwischen Knoten m und Knoten n
Q_k	Kapazität des Fahrzeuges k
B_n	Nachfrage des Knotens n
x_{mnk}	$x_{mnk} = \begin{cases} 1, & wenn\ Fahrzeug\ \ k\ Knoten\ \ n\ im\ Anschluss\ \ an\ Knoten\ \ m\ besucht \\ 0, & sonst \end{cases}$
x_{0nk}	$x_{0nk} = \begin{cases} 1, & wenn\ Knoten\ \ n\ der\ erste\ von\ Fahrzeug\ \ k\ besuchte\ \ Knoten\ \ ist \\ 0, & sonst \end{cases}$
x_{m0k}	$x_{m0k} = \begin{cases} 1, & wenn\ Knoten\ \ m\ der\ letzte\ von\ Fahrzeug\ \ k\ besuchte\ \ Knoten\ \ ist \\ 0, & sonst \end{cases}$
y_{nk}	$y_{nk} = \begin{cases} 1, & wenn\ Knoten\ \ n\ Fahrzeug\ \ k\ zugeordnet\ \ wird \\ 0, & sonst \end{cases}$
V	Menge der Knoten einschließlich des Depotknotens
Q	Teilmenge von V
l_{0y}	y-Koordinate des Depot-Standorts
l_{0x}	x-Koordinate des Depot-Standorts

Tabelle 4.1 : Symbole der Modellformulierung des CELRP

		Formale Beschreibung		Interpretation
Minimiere	*(4.4)*	$\sum_{m=1}^{N}\sum_{n=1}^{N} c_{mn}\sum_{k=1}^{K} x_{mnk} + \sum_{n=1}^{N}\left(c_{0n}\left(l_{0x}, l_{0y}\right)\sum_{k=1}^{K} x_{0nk}\right) + \sum_{m=1}^{N}\left(c_{m0}\left(l_{0x}, l_{0y}\right)\sum_{k=1}^{K} x_{m0k}\right)$		Die Zielfunktion minimiert die Gesamtlänge der Touren. Die Bewertung der Kanten, die das Depot mit den Knoten verbinden, hängt von der Position des Standortes (l_{0x}, l_{0y}) ab.
u.B.v.:	*(4.5)*	$\sum_{n=1}^{N} B_n y_{nk} \le Q_k \;, k = 1(1)K$		Die aggregierte Nachfrage der einem Fahrzeug zugeordneten Knoten darf die Fahrzeugkapazität nicht überschreiten.
	(4.6)	$\sum_{k=1}^{K} y_{nk} = 1 \;,\; n = 1(1)N$		Jeder Knoten muss genau einem Fahrzeug zugeordnet werden.
	(4.7)	$y_{nk} \in \{0,1\},\; n = 1(1)N,\; k = 1(1)K$		Binärbedingung für die Zuordnungsvariablen.
	(4.8)	$\sum_{m=0}^{N} x_{mnk} = y_{nk} \;,\; n = 0(1)N$	$k = 1(1)K$	Wenn der Knoten *n* dem Fahrzeug *k* zugeordnet wurde, muss das Fahrzeug den Kunden genau einmal anfahren. Wenn der Knoten *n* dem Fahrzeug nicht zugeordnet wurde, darf das Fahrzeug den Knoten nicht anfahren.
	(4.9)	$\sum_{n=0}^{N} x_{mnk} = y_{mk} \;,\; m = 0(1)N$		Wenn der Knoten *m* dem Fahrzeug *k* zugeordnet wurde, muss das Fahrzeug den Knoten genau einmal verlassen. Wenn der Knoten *m* dem Fahrzeug nicht zugeordnet wurde, darf das Fahrzeug den Knoten nicht verlassen.
	(4.10)	$\sum_{m\in Q}\sum_{n\in Q} x_{mnk} \le \lvert Q\rvert - 1 \;,\; \forall Q \subseteq V - \{0\},\quad 2 \le \lvert Q\rvert \le N - 1$		Bedingung zur Verhinderung von Kurzzyklen zwischen den Kunden.
	(4.11)	$x_{mnk} \in \{0, 1\},\; m, n = 0(1)N$		Binärbedingungen.
	(4.12)	$y_{0k} = 1$		Das Depot wird jedem Fahrzeug zugeordnet.

Tabelle 4.2 : Mögliche Modellformulierung für das CELRP

Die hier vorgenommene Modellformulierung weist starke Ähnlichkeit zum ECVRP auf, die in der Übereinstimmung der Restriktionen zu erkennen ist[366]. Das unterscheidende Merkmal stellt die Zielfunktion *(4.4)* dar, welche um die Terme $\sum_{n=1}^{N}\left(c_{0n}\left(l_{0x}, l_{0y}\right)\sum_{k=1}^{K} x_{0nk}\right)+\sum_{m=1}^{N}\left(c_{m0}\left(l_{0x}, l_{0y}\right)\sum_{k=1}^{K} x_{m0k}\right)$ erweitert wurde. Der erste Term beschreibt die Summe der euklidischen Entfernungen vom Depot zum jeweils ersten Kunden der Tour eines Fahrzeugs. Der zweite Term beschreibt die Summe der euklidischen Entfernungen vom jeweils letzten Kunden einer Tour zum Depot. Die euklidischen Entfernungen $c_{0n}\left(l_{0x}, l_{0y}\right)$ und $c_{m0}\left(l_{0x}, l_{0y}\right)$ hängen dabei von der Wahl der Standortkoordinaten $\left(l_{0x}, l_{0y}\right)$ in der Ebene ab. Das Modell benötigt unter Vernachlässigung der Binärbedingungen insgesamt $K+N+K\cdot\left(2\cdot(N+1)+1+\sum_{|Q|=2}^{N-1}\binom{N}{|Q|}\right)$ Nebenbedingungen.

Die Anzahl der Variablen beträgt $\left((N+1)^2-N\right)\cdot K+N\cdot K+2$, von denen $\left((N+1)^2-N\right)\cdot K+N\cdot K$ als Binärvariablen definiert werden. Verwendet man das von MIN ET AL.[367] eingeführte Klassifikationsschema, kann das Problem folgendermaßen eingeordnet werden:

Merkmal	**Ausprägung**
Anzahl Stufen	1
Form der Nachfrage / des Angebots	deterministisch
Anzahl Depots	1
Größe der Fahrzeugflotte	*K* heterogene Fahrzeuge
Fahrzeugkapazität	beschränkt
Kapazität der Standorte	unbeschränkt
Stufe der Standorte[368]	2
Anzahl Planungsperioden	1
Zeitfenster	keine
Zielfunktion	eine Zielgröße
Testdaten	Hypothetisch / Real

Tabelle 4.3 : Einordnung des CELRP

[366] Vgl. Domschke [1997], S. 215 f., und S. 71.
[367] Vgl. Min et al. [1998], S. 3.
[368] Die Einordnung wird auf Basis der in Min et al. [1998] gegebenen inhaltlichen Definitionen für mehrstufige Probleme vorgenommen. Die Stufe 2 beschreibt dort u.a. Distributionszentren und Depots, vgl. Min et al. [1998], S. 4.

Bisher existieren keine Ansätze zur Lösung von Problemen der simultanen Standort- und Tourenplanung mit Hilfe von Self-Organizing Maps. Im nächsten Abschnitt soll deswegen dargestellt werden, wie eine Erweiterung des Algorithmus vorgenommen werden kann, um das formulierte Problem zu lösen.

4.2 Erweiterung der Self-Organizing Map

Zur Ableitung einer Lösung wird eine Variante der SOM entwickelt, die verschiedene Aspekte der in der Literatur vorgefundenen Ansätze integriert. So werden zur Repräsentation der Touren der eingesetzten Fahrzeuge K Neuronenringe verwendet, die in einem speziellen Neuron, das den Standort des Depots darstellt, verbunden werden. Während bei der Anwendung der SOM auf Tourenplanungsprobleme oder EMTSPe jeweils nur die Neuronen des Neuronenringes adaptiert werden, der das Zentrum der jeweiligen Iteration enthält, können in dieser Variante allerdings Neuronen mehrerer Ringe adaptiert werden. Abbildung 4.3 verdeutlicht die Nachbarschaftsstruktur für ein Netz mit zwei Ringen. Bei hinreichend geringer Entfernung des Zentrums zum Standort-Neuron werden auch die im zweiten Neuronenring befindlichen Neuronen, die dem Standort-Neuron benachbart sind, adaptiert. Durch dieses Vorgehen soll die benachbarte Lage der Tour-Neuronen zum Standort erhalten und so die Auswirkung der Veränderung der Standortentscheidung in die Tourenplanung integriert werden. Ist die Entfernung des Zentrums einer Iteration vom Standort hingegen groß und wird dessen Lage nicht verändert, so werden nur Neuronen des Rings, der das Zentrum enthält, adaptiert. Die Stärke der Anpassung der Gewichtsvektoren hängt auch bei diesem Ansatz von der Entfernung zwischen Zentrum und betrachtetem Neuron im Neuronengitter ab.

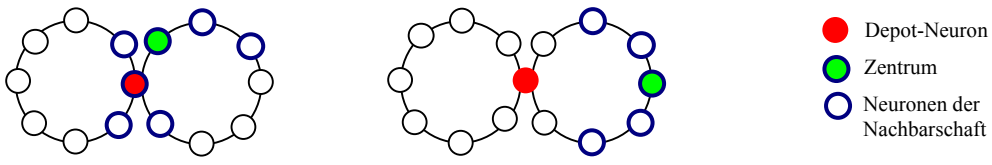


Abbildung 4.3 : Nachbarschaftsstruktur der verwendeten SOM-Variante, CELRP-Beispiel

Im Folgenden werden weitere wesentliche Merkmale des Algorithmus unter Bezugnahme auf die in Tabelle 4.2 angegebene Modellformulierung erläutert.

Eine ermittelte Lösung wird aufgrund der Struktur der Ausgabeschicht immer die Restriktionen *(4.6)* - *(4.12)* erfüllen: Die Verwendung von im Standortneuron verbundenen Neuronenringen, welche die Touren der Fahrzeuge repräsentieren, garantiert, dass alle Touren im Depot starten und enden. Die Zuordnung eines Knotens zu einem Neuron eines Ringes garantiert in Verbindung mit der Modellierung der Touren als Neuronenring, dass jeder Knoten genau einmal besucht und genau einmal verlassen wird.
Um die Kapazitätsrestriktionen *(4.5)* berücksichtigen zu können, wird folgende Vorgehensweise verfolgt, solange $t < t_{max}$:
Bei der Berechnung der Distanz eines Neurons zum Eingabevektor wird die gewichtete Auslastung des zum Neuron zugeordneten Fahrzeugs zur euklidischen Distanz addiert. Dieser Mechanismus, der die Entstehung von unzulässigen Lösungen verhindern soll, führt meistens zu einer Erhöhung der Zielfunktionswerte der ermittelten zulässigen Lösungen. Daher wird die gewichtete Fahrzeugauslastung im vorgeschlagenen Algorithmus nur in der ersten Hälfte der durchgeführten Präsentationszyklen in der Distanzfunktion berücksichtigt. Um trotzdem eine hohe Lösungswahrscheinlichkeit zu erreichen, wurde zusätzlich ein Mechanismus integriert, der die Wiederherstellung der Zulässigkeit unzulässiger Lösungen unterstützt: Wenn die einem Fahrzeug zugeordnete Kapazität die Kapazitätsrestriktion des Fahrzeuges verletzt, werden sukzessiv Knoten aus der Tour des Fahrzeugs entfernt, bis dessen Kapazitätsrestriktion wieder erfüllt wird. Dabei wird jeweils der Knoten aus der Tour ausgeschlossen, der die geringste Entfernung zu einem anderen Neuronenring aufweist. Dem die Tour repräsentierenden Neuronenring, aus dem Knoten entfernt wurden, wird ein sogenannter Tabu-Zähler zugewiesen, der beschreibt, wie oft die ausgeschlossenen Knoten dem Netz präsentiert werden müssen, bevor diese erneut der entsprechenden Tour zugewiesen werden dürfen. Die Distanzen für die Neuronen der Tour werden hierzu bei der Präsentation der Knoten auf eine hinreichend große Zahl gesetzt. Durch dieses Vorgehen wird erreicht, dass die ausgeschlossenen Knoten anderen Touren zugeordnet werden und dadurch ausgelöste Adaptionsprozesse zu einer Annäherung der zugehörigen Neuronen führen. Um zu verhindern, dass die Knoten bei einer Verletzung der Kapzitätsrestriktionen ebenfalls aus diesen Touren entfernt werden, wird in der ersten Hälfte der durchzuführenden Präsentationszyklen zusätzlich ein sogenanntes Ausschlussverbot ausgesprochen. Dieses Verbot verhindert, dass aus einer Tour ausgeschlossene Knoten aus der Tour eines anderen Fahrzeuges entfernt werden, solange der Tabu-Zähler einen von Null verschiedenen Wert aufweist. Der Tabu-Zähler wird im Anschluss an die Präsentation eines ausge-

schlossenen Knotens jeweils um Eins reduziert, bis der Zähler den Wert Null annimmt und der Knoten der Tour, aus der er entfernt wurde, erneut zugewiesen werden darf.
In der Iteration $t=t_{max}$ wird der beschriebene Mechanismus übersprungen. Dadurch wird sichergestellt, dass jeder Knoten in der endgültigen Lösung genau einer Tour zugewiesen wird. Es ist zu erwähnen, dass dieses Vorgehen zu unzulässigen Lösungen führen kann. Ist der Anteil der ermittelten zulässigen Lösungen zu gering, kann der Maximalwert des Tabu-Zählers und / oder die Gewichtung der Kapazitätsauslastung in der Distanzfunktion erhöht werden[369]. Zusätzlich kann die Berücksichtigung der Fahrzeugauslastung in der Distanzfunktion und die Gültigkeit des Ausschlussverbotes auf einen größeren Anteil an Präsentationszyklen ausgedehnt werden.
Die mit einer ermittelten Lösung verbundenen Touren können aus den Verbindungen zwischen den Eingabevektoren (Knoten) und den Neuronen der Neuronenringe abgelesen werden, die gesuchten Koordinaten des Standortes können dem Gewichtsvektor des Standortneurons entnommen werden.
Abbildung 4.4 fasst die einzelnen Schritte des Algorithmus, der im Folgenden mit dem Kürzel *LRPSOM* bezeichnet wird, in einem Flussdiagramm zusammen.

[369] Zusätzlich wäre eine dynamische Anpassung der Auslastungsgewichtung im Verlauf des Verfahrens denkbar, wie sie beispielsweise von Retzko [1996], S. 146, vorgeschlagen wird.

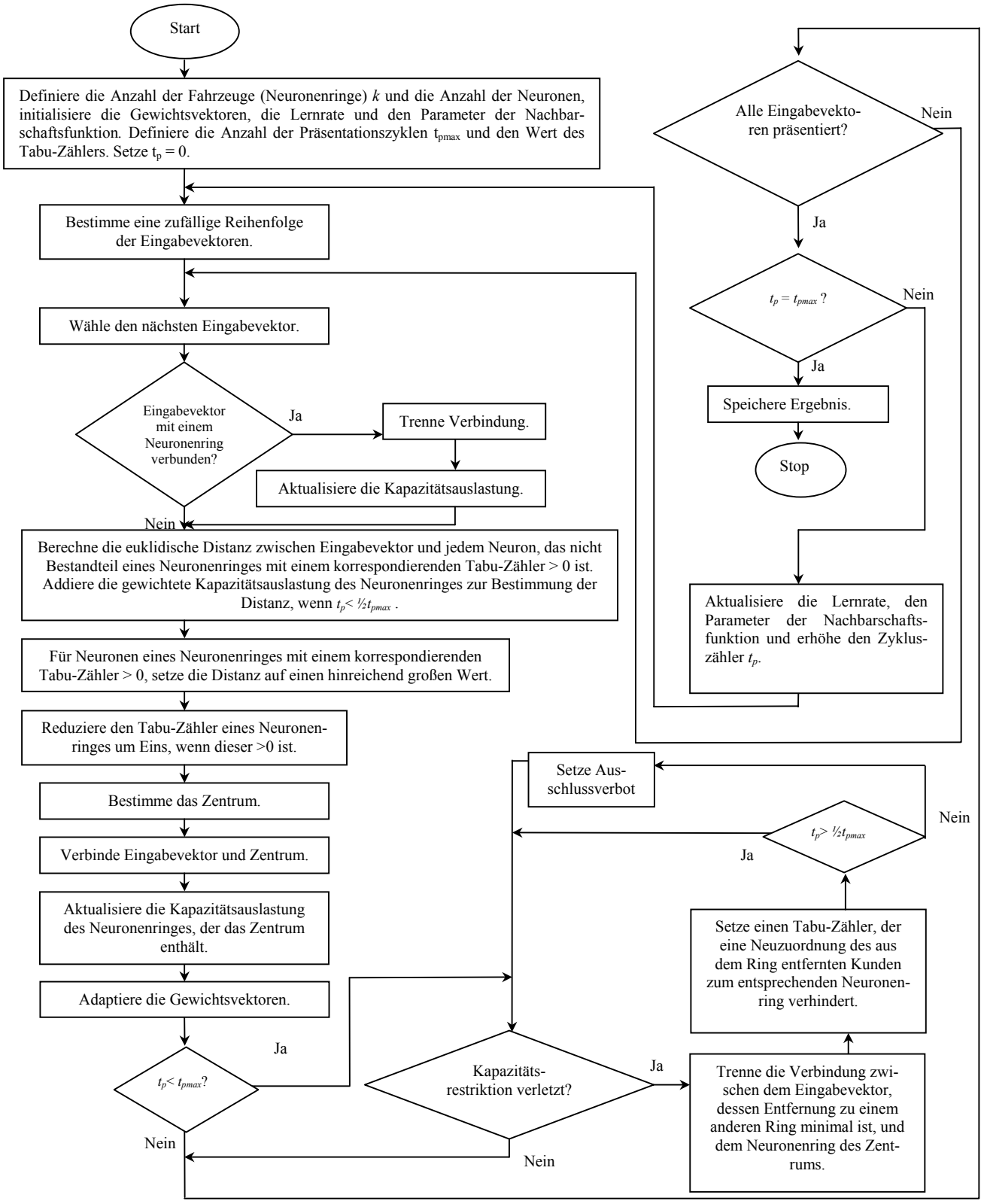


Abbildung 4.4 : Ablaufdiagramm des vorgeschlagenen Algorithmus

Der für eine gegebene Lösung des im formulierten Modell enthaltenen Tourenplanungsproblems beste Standort stimmt mit einer Lösung des Weber-Problems überein, wenn die Knoten, die innerhalb der Touren jeweils die ersten beziehungsweise die letzten bedienten Knoten darstellen, als die Punkte P_n angesehen werden. Die bereits beschriebene Lösungsprozedur für das Weber-Problem kann somit verwendet werden, um die ermittelte Lösung für die Standortkoordinaten zu verbessern. Im Folgenden wird für diese Kombination das Kürzel *LRPSOMW* verwendet.

4.3 Anwendungsbeispiel

Die Anwendung der entwickelten SOM-Variante zur Lösung des CELRP soll anhand eines Beispiels vorgestellt werden. Tabelle 4.4 gibt einen Überblick über die wesentlichen Merkmale des Anwendungsbeispiels.

Merkmal	**Anzahl / Ausprägung**	**Interpretation / Effekt**
Anzahl Knoten	83	Eingabevektoren.
Anzahl Neuronen der Eingabeschicht = Anzahl Komponenten des Eingabevektors	2	Die Eingabevektoren stellen die Ortsvektoren der Knoten dar.
Anzahl der Neuronen der Ausgabeschicht	301	101 Neuronen je Neuronenring, Ringe im Standortneuron verbunden.
Anzahl Fahrzeuge	3	Annahme.
Fahrzeugkapazität	300 Kapazitätseinheiten.	Homogene Fahrzeugflotte. Heterogene Fahrzeugflotte möglich.
Gesamtnachfrage	830 Kapazitätseinheiten	Nachfrageverteilung heterogen.
Kapazitätsüberschuss	8,4 %	-
Präsentationszyklen	$t_{p\max} = 400$ Zyklen	-
Anzahl präsentierter Eingabevektoren	33.200	$t_{\max} = t_{p\max} \cdot 83$
Topologie der Ausgabeschicht	Im Depot-Neuron verbundene Neuronenringe.	Nachbarschaftsstruktur s. Abbildung 4.3.
Initialisierungswert des Lernparameters $\sigma(t)$	$\sigma_{t_p=0} = 7{,}0$	-
Verlauf des Parameters $\sigma(t)$	$\sigma(t) = \sigma_{t_p=0} \cdot \left(\frac{\sigma_{t_{p\max}}}{\sigma_{t_p=0}} \right)^{\frac{t_p}{t_{p\max}}}$	Steuert die Breite der Nachbarschaftsfunktion.
Verlauf der Lernrate	$\alpha(t) = \max\left\{1 - \frac{t_p}{t_{p\max}},\ 0{,}001\right\}$	Steuert die Stärke der Adaption der Gewichtsvektoren.
Initialisierung der Gewichtsvektoren	Ellipsenförmige Neuronenringe.	-
Gewichtsvektoren der Neuronen der Ausgabeschicht	2	Koordinaten der Neuronen.
Präsentation der Eingabevektoren	Zufällig.	Präsentationszyklen mit zufälliger Reihenfolge.
Distanzfunktion	$\left\|\mathbf{x}(t) - \mathbf{w}_j^k(t)\right\|^2 + \gamma \cdot \frac{q_k}{Q_k}$	Die quadrierte euklidische Distanz wird um die gewichtete Auslastung erhöht.
Auslastungsgewichtung	$\gamma = 10$	-
Bestimmung des Zentrums	Neuron, dessen Gewichtsvektor die geringste Distanz zum Eingabevektor aufweist.	-
Nachbarschaftsfunktion	$h_{cj}(t) = \alpha(t) \cdot e^{\left(\frac{\Delta_{cj}^2}{2 \cdot \sigma^2(t)} \right)}$	Verlauf s. Abbildung (4.6)
Lernregel	$\mathbf{w}_j(t+1) = \left(1 - h_{cj}(t)\right) \cdot \mathbf{w}_j(t) + h_{cj}(t) \cdot \mathbf{x}(t)$	-
Maximaler Wert des Tabu-Zählers	2	-

Tabelle 4.4 : Merkmale des Anwendungsbeispiels für das CELRP

Abbildung 4.5 zeigt die Anordnung der Ausgabeschicht der SOM zu Beginn des Algorithmus, nach einem Viertel beziehungsweise der Hälfte der maximalen Iterationszahl t_{max} und schließlich die resultierende Lösung. Der Vergleich der Initialisierungsanordnung und der Anordnungen der Zwischenlösungen lässt erkennen, dass der Standort des rot markierten Depots durch die veränderte Nachbarschaftsstruktur innerhalb der Ausgabeschicht vom jeweiligen Verlauf der Touren der drei eingesetzten Fahrzeuge abhängt.

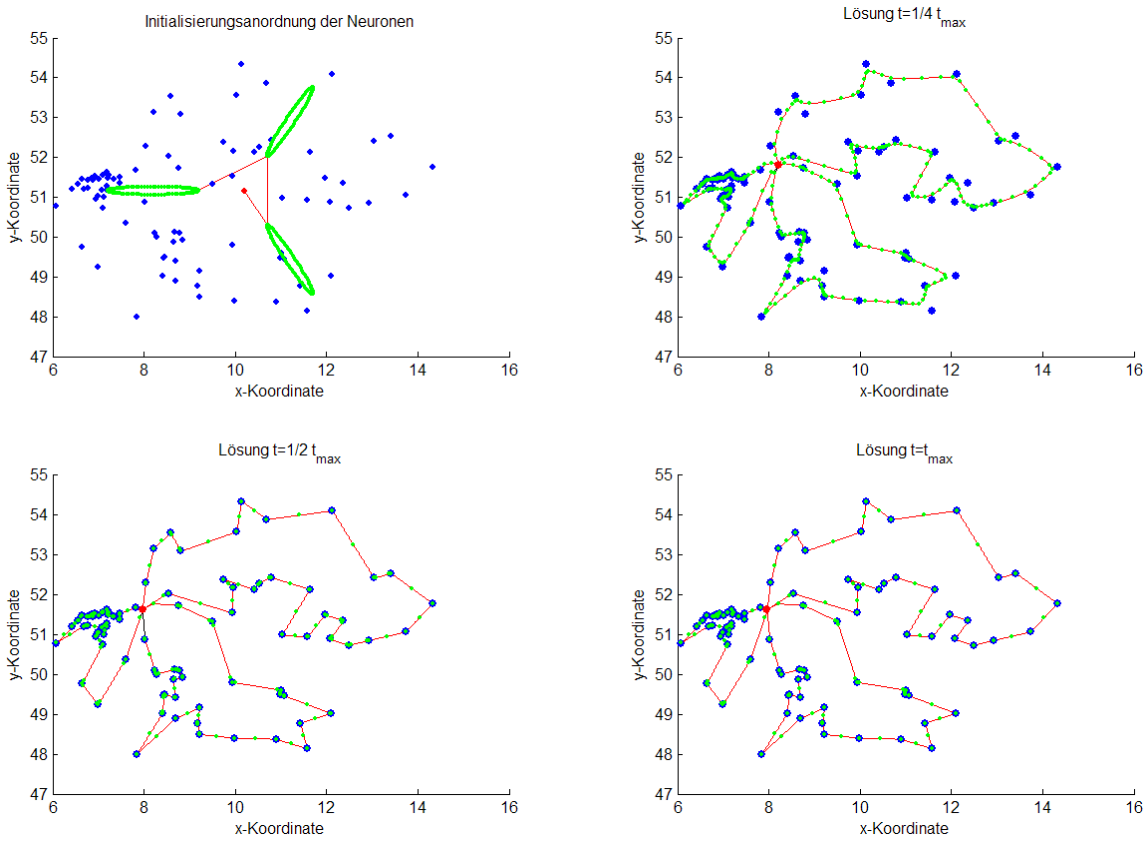


Abbildung 4.5 : Entfaltung der Self-Organizing Map, CELRP-Beispiel

Abbildung 4.6 zeigt den Verlauf der Parameter in Abhängigkeit von der Anzahl durchgeführter Präsentationszyklen t_p und die Adaption des Standortneurons in den ersten und letzten 83 der insgesamt 33.200 Iterationen. Am Beispiel des Standortneurons ist gut zu erkennen, dass aufgrund der Verringerung des Linearkombinationsparameters die Neuronen der Ausgabeschicht „einfrieren“. Durch Definition einer Untergrenze für den Lernparameter wird allerdings sichergestellt, dass zumindest das Zentrum auch zum Ende des Verfahrens noch geringfügig in Richtung des präsentierten Eingabevektors verschoben wird.

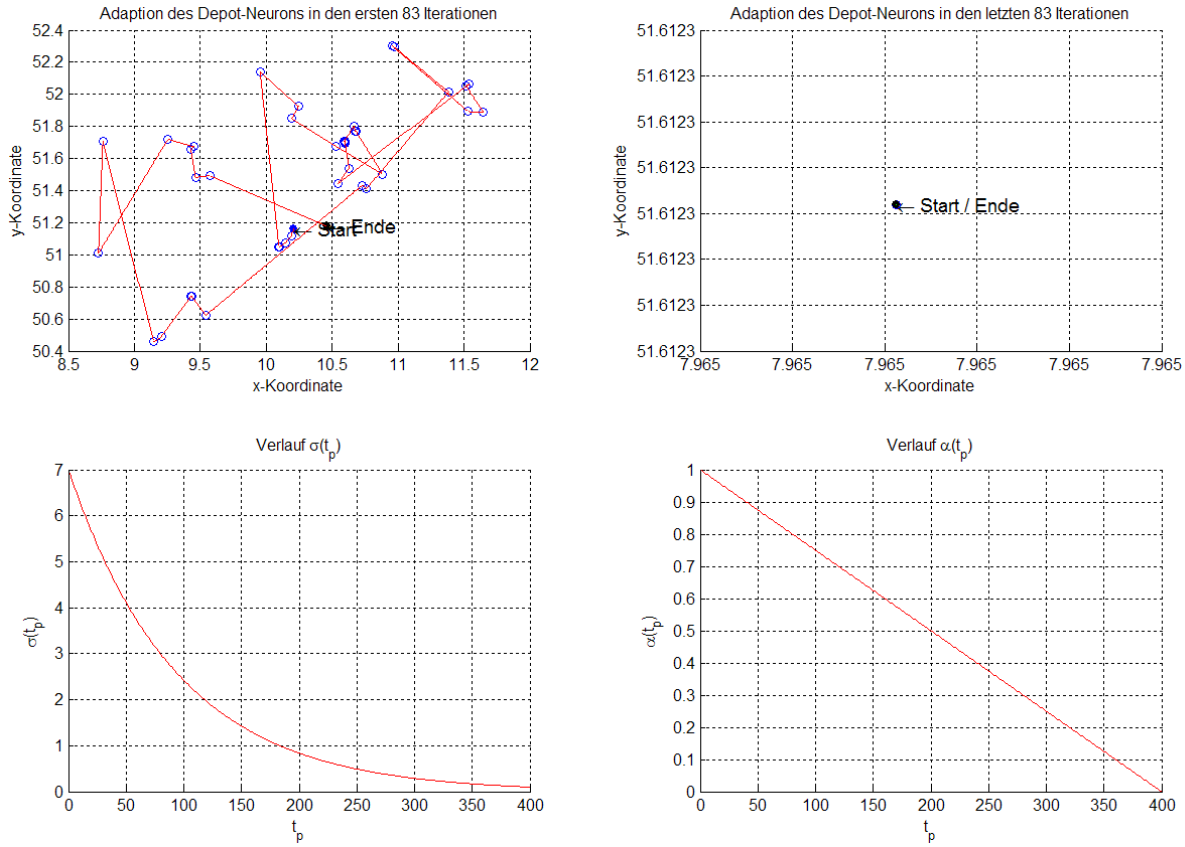


Abbildung 4.6 : Adaption des Standortneurons und Verlauf der Parameter, CELRP-Beispiel

4.4 Test des Ansatzes

Um die Qualität des vorgeschlagenen Algorithmus einschätzen zu können, wurden verschiedene Testinstanzen für euklidische Tourenplanungsprobleme aus der *TSPLIB-Bibliothek* gewählt[370], auf die der Algorithmus angewendet wurde. Die ursprünglichen Standortkoordinaten werden in der folgenden Beschreibung mit angegeben, jedoch bei der Anwendung des Verfahrens ignoriert. Tabelle 4.5 gibt einen Überblick über die Merkmale der Testinstanzen. Sie enthält die Anzahl der Knoten, die Art der Verteilung der Knoten, die ursprünglichen Standortkoordinaten und deren Lage in bezug auf die Lage der Knoten der Testinstanz, die Fahrzeugkapazität und den Gesamtkapazitätsüberschuss, wenn die in Klammern angegebene minimale Anzahl an Fahrzeugen verwendet wird.

Instanz	Anzahl Knoten	Verteilung Knoten	Koordinaten des Depots	Lage des Depots	Fahrzeugkapazität	Kapazitätsüberschuss %
C1	50	U	(30/40)	Z	160	3,0 (5)
C2	75	U	(40/40)	Z	140	2,6 (10)
C3	100	U	(35/35)	Z	200	9,7 (8)
C4	100	C	(40/50)	Z	200	10,5 (10)
C6	120	C	(10/45)	D	200	1,8 (7)
C8	150	U	(35/35)	Z	200	7,4 (12)
C10	199	U	(35/35)	Z	200	0,5 (16)
eil22	21	S	(145/215)	Z	6000	6,7 (4)
eil23	22	S	(266/235)	Z	4500	32,5 (3)
eil30	29	S	(162/354)	Z	4500	5,9 (3)
eil33	32	S	(292/495)	D	8000	9,0 (4)
F1	44	S (R)	(0/0)	Z	2010	11,3 (4)
F2	71	C (R)	(0/0)	D	30.000	4,5 (4)
F3	134	S (R)	(-6/15)	D	2210	5,8 (7)
GJ1	249	U (R)	(0/0)	Z	500	3,3 (25)

C: geclustert, D: dezentral, (R): Daten realer Probleme, S: unregelmäßig verteilt, U: gleichverteilt, Z: zentral

Tabelle 4.5 : Testinstanzen für das CELRP

Tabelle 4.6[371] vergleicht die in 100 Durchläufen ermittelten Ergebnisse des vorgeschlagenen Algorithmus *LRPSOM* mit denen verschiedener, sequentiell arbeitender Lösungsverfahren, die ebenfalls zur Lösung des CELRP vom Verfasser implementiert

[370] Die Testinstanzen können unter der Internet-Adresse http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/ abgerufen werden. Eine Beschreibung der Bibliothek findet sich in Reinelt [1991]. Die meisten Instanzen können auch Christofides et al. [1979], Christofides und Eilon [1969] und Fisher [1994] entnommen werden. Anhang B stellt Abbildungen der Testinstanzen zu Verfügung.

[371] Vgl. S. 129.

wurden. Diese Vergleichsverfahren bestimmen in Anlehnung an ein von ENGELE vorgeschlagenes Verbesserungsverfahren für ein mehrstufiges Problem[372] in einem ersten Schritt die Standortkoordinaten durch Lösung des Weber-Problems, dabei werden die Koordinaten der Knoten als Koordinaten der Punkte P_n verwendet. Zur Bestimmung eines zulässigen Tourenplans wird im zweiten Schritt jeweils das Savings-Verfahren[373] *(SV)*, zwei unterschiedliche Nearest Neighbor-Verfahren[374] (*NNA* und *NNB*) sowie das parametrisierte Savings-Verfahren[375] *(PSV)* verwendet. Die Variante *NNB* des Nearest Neighbor-Verfahrens bestimmt zunächst eine „Giant-Route" durch alle Knoten, die anschließend aufgebrochen wird, um die Kapazitätsbedingungen zu erfüllen. Dabei startet die erste Tour mit dem Knoten mit minimaler euklidischer Entfernung zum Depot, als Folgeknoten wird jeweils der Knoten mit der geringsten Entfernung zum zuletzt eingefügten Knoten aufgenommen. Die Variante *NNA* bildet keine Giant-Route, sondern beginnt die erste Tour mit dem Knoten, der die geringste euklidische Distanz zum Depot aufweist. Es werden dann sukzessiv Knoten mit der geringsten Distanz zum Vorgänger eingefügt, bis die Kapazitätsrestriktion greift. Weitere Touren werden durch identische Vorgehensweise erzeugt.

Bei Anwendung des parametrisierten Savings-Verfahrens werden die von PAESSENS vorgeschlagenen Parameterbereiche $0 < g \leq 3$ und $0 < f \leq 1$ für die modifizierte Savings-Funktion

$$(4.13) \quad S_{mn} = dp_m + dp_n - g \cdot d_{mn} + f \cdot (dp_m - dp_n)$$

untersucht.[376] Bei konstantem Parameter g wird der Parameter f von 0,1 ausgehend jeweils in Zehntel-Schritten bis zur Obergrenze erhöht, anschließend wird der Parameter g (ebenfalls vom Startwert 0,1 ausgehend) um ein Zehntel erhöht, insgesamt werden also 300 Parameterkombinationen überprüft. Die für eine Testinstanz jeweils beste Lösung wird gemeinsam mit den zugehörigen Parameterwerten (in Klammern) in den Tabellen angegeben.

Für jedes Verfahren werden zwei unterschiedliche Ansätze bei der Bestimmung des Standortes untersucht. Der erste Ansatz verwendet jeweils die mit den Nachfragemengen der Knoten gewichteten euklidischen Distanzen *(WGED)*, der zweite Ansatz verwendet die euklidischen Distanzen *(WUED)*. Die in der Tabelle angegebenen Lösungs-

[372] Vgl. Engele [1981], S. 98.
[373] Vgl. Clarke und Wright [1964].
[374] Vgl. hierzu beispielsweise Johnson und McGeoch [1997], S. 219 ff.
[375] Vgl. hierzu Paessens [1988].
[376] Vgl. Paessens [1988], S. 339.

wahrscheinlichkeiten geben die Anteile ermittelter zulässiger Lösungen an, die sich in den Testreihen für die SOM-Varianten ergaben. Die Anteile gelten entsprechend für die Tabellen 4.7 und 4.8. Die letzte Spalte der Tabellen 4.6 und 4.7 gibt jeweils die prozentuale Abweichung zwischen der besten Lösung der sequentiellen Ansätze und der geringsten, der durchschnittlichen und der maximalen Gesamtlänge der Tourenpläne an, die mit Hilfe der SOM-Varianten *LRPSOM* bzw. *LRPSOMW* ermittelt wurden.

Bei allen Testläufen wurden Ausgabeschichten verwendet, deren Neuronenzahl dem Vierfachen der Knotenzahl der jeweiligen Testinstanz entspricht. Eine detaillierte Übersicht der in den Testläufen verwendeten Parameter kann Anhang C entnommen werden.

Instanz	WGED, SV	WUED, SV	WGED, PSV	WUED, PSV	WGED, NNA	WUED, NNA	WGED, NNB	WUED, NNB	Bestes/∅-/schlechtestes Ergebnis LRPSOM	Lösungs-wahrscheinlichkeit (%)	DEV LRPSOM (%)		
C1	573,9	567,5	558,2 (g=1,6, f=0,1)	561,7 (g=1,6, f=0,1)	739,6	742,3	731,1	732,2	**530,0**/561,1/608,9	100	-5,1	0,5	9,1
C2	894,5	906,2	883,3 (g=1,4, f=0,1)	890,0 (g=1,3, f=0,1)	1037,6	1044,3	1033,5	1037,0	**881,0**/929,4/1032,1	98	-0,3	5,2	16,8
C3	875,1	874,2	866,2 (g=1,6, f=0,1)	873,0 (g=1,3, f=0,1)	1093,4	1092,0	1172,6	1172,6	**836,3/859,5**//997,7	96	-3,5	-0,8	15,2
C4	832,9	830,3	826,5 (g=1,0, f=0,1)	**826,2** (g=1,3, f=0,1)	1127,1	1128,1	1130,8	1131,6	829,7/835,3/912,0	100	0,4	1,1	10,4
C6	953,1	978,9	964,0 (g=0,8, f=0,1)	938,1 (g=1,4, f=0,1)	1154,5	1139,5	1146,0	1144,2	**911,7**/946,7/1215,8	100	-2,8	0,9	29,6
C8	1150,7	1140,6	1113,2 (g=1,9, f=0,3)	1118,2 (g=1,3, f=0,1)	1314,7	1311,4	1339,9	1398,8	**1061,3/1086,5**/1136,8	100	-4,7	-2,4	2,1
C10	1386,5	1390,0	1381,3 (g=1,2, f=0,1)	1377,5 (g=1,1, f=0,1)	1611,1	1610,7	1705,0	1704,5	**1359,4**/1416,8/1695,2	96	-1,3	2,9	23,1
eil22	388,3	423,1	**387,0** (g=0,8, f=0,1)	393,9 (g=0,8, f=0,3)	561,2	474,0	525,1	493,2	390,4/398,7/448,6	95	0,9	3,0	15,9
eil23	**501,1**	551,7	503,2 (g=0,7, f=0,1)	544,9 (g=1,2, f=0,2)	639,9	723,6	778,6	732,9	518,3/581,0/711,1	100	3,4	15,9	41,9
eil30	520,9	532,1	508,8 (g=1,3, f=0,3)	**504,1** (g=1,2, f=0,2)	742,8	615,2	598,0	656,8	512,5/555,9/685,4	98	1,7	10,3	36,0
eil33	488,4	503,1	490,7 (g=1,0, f=0,1)	497,8 (g=0,1, f=0,7)	564,2	676,0	629,5	652,6	**486,6**/524,5/587,5	100	-0,4	8,2	20,3
F1	739,2	744,2	**729,0** (g=0,8, f=0,1)	749,7 (g=1,0, f=0,2)	930,3	954,6	955,4	894,7	736,7/800,0/934,0	98	1,1	9,7	28,1
F2	**224,1**	244,4	231,9 (g=1,1, f=0,1)	239,2 (g=1,3, f=0,2)	303,5	306,2	278,6	298,9	236,7/264,4/306,7	98	5,6	18,0	36,9
F3	**1194,1**	1224,3	1199,4 (g=1,3, f=0,1)	1202,5 (g=1,0, f=0,1)	1546,2	1438,9	1459,4	1527,5	1238,5/1357,9/1520,2	99	3,7	13,7	27,3
GJ1	5490,0	5482,3	5471,9 (g=1,0, f=0,1)	**5454,2** (g=1,0, f=0,1)	6357,9	6359,6	6481,0	6485,7	5607,9/5868,9/6501,2	93	2,8	7,6	19,2
∅	-	-	-	-	-	-	-	-	-	-	**0,1**	**6,3**	**22,1**
Anzahl bester Lösungen	3	0	2	3	0	0	0	0	7/2/0	-	-	-	-

Tabelle 4.6 : Vergleich der sequentiellen Ansätze ohne Verbesserung des Standortes und der vorgeschlagenen SOM-Variante LRPSOM bei Anwendung auf das CELRP[377]

[377] Fett markierte Einträge zeigen an, dass mit dem entsprechenden Verfahren die beste Lösung ermittelt wurde. Fett markierte Einträge in der Spalte *Bestes / ∅- / schlechtestes Ergebnis LRPSOM* zeigen an, dass das Verfahren auch ohne Verbesserung der Standortkoordinaten bessere Ergebnisse als die sequentiellen Ansätze ermitteln konnte.

Tabelle 4.7[378] vergleicht die Ergebnisse der Vergleichsverfahren mit den in 100 Durchläufen ermittelten Ergebnissen der SOM-Variante *LRPSOMW*, wenn jeweils im Anschluss an die Tourenbildung erneut der Algorithmus zur näherungsweisen Lösung des Weber-Problems angewendet wird. Dabei werden jeweils die ersten und letzten Kunden auf einer Tour zu der Menge der Punkte P_n zusammengefasst. Die Gewichtung der euklidischen Entfernung beträgt Eins, wenn ein Knoten erster beziehungsweise letzter Knoten in der Besuchsreihenfolge eines Fahrzeuges ist, für auf Stichfahrten belieferte Knoten wird die Gewichtung auf den Wert Zwei gesetzt. Abbildung 4.8 fasst die Vorgehensweise zusammen.

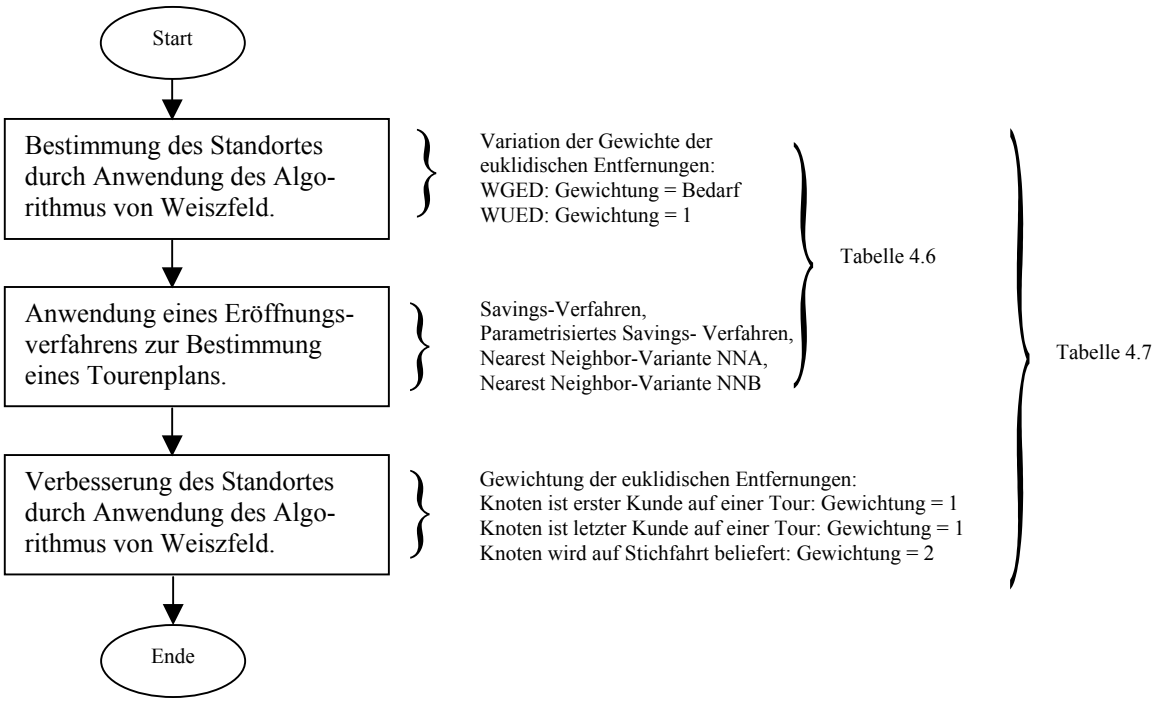


Abbildung 4.7: Vergleichsverfahren

[378] Vgl. S. 131.

Instanz	WGED, SV	WUED, SV	WGED, PSV	WUED, PSV	WGED, NNA	WUED, NNA	WGED, NNB	WUED, NNB	Bestes/∅-/schlechtestes Ergebnis LRPSOMW	DEV LRPSOMW (%)		
C1	527,7	565,4	552,0 (g=1,6, f=0,1)	554,9 (g=1,6, f=0,1)	736,4	736,4	731,0	731,0	**522,5**/552,7/587,1	-1,0	4,7	11,3
C2	893,6	905,4	881,9 (g=1,4, f=0,1)	888,5 (g=1,3, f=0,1)	1031,8	1031,8	1033,3	1033,3	**879,0**/920,1/988,3	-0,3	4,3	12,1
C3	871,3	871,3	864,5 (g=1,6, f=0,1)	872,6 (g=1,3, f=0,1)	1079,1	1079,1	1172,5	1172,5	**835,6/848,3**/876,4	-3,3	-1,9	1,4
C4	832,6	830,3	**826,0** (g=1, f=0,1)	**826,0** (g=1,0, f=0,1)	1123,8	1123,8	1124,5	1124,5	828,9/829,9/832,5	0,4	0,5	0,8
C6	946,1	973,5	933,0 (g=0,7, f=0,2)	931,8 (g=0,8, f=0,1)	1133,4	1132,4	1143,8	1143,8	**905,7/930,5**/1080,4	-2,8	-0,1	15,9
C8	1147,8	1140,3	1110,9 (g=1,9, f=0,3)	1117,9 (g=1,3, f=0,1)	1312,2	1307,8	1335,8	1398,6	**1058,4/1075,6/1095,5**	-4,7	-3,2	-1,4
C10	1385,3	1389,8	1380,8 (g=1,2, f=0,1)	1377,1 (g=1,1, f=0,1)	1609,9	1609,9	1700,3	1700,3	**1344,0/1369,7**/1391,7	-2,4	-0,5	1,1
eil22	386,9	412,0	384,9 (g=0,8, f=0,1)	384,9 (g=0,8, f=0,3)	559,3	454,4	520,7	488,4	**374,7/377,1**/400,5	-2,7	-2,0	4,1
eil23	**501,1**	523,2	503,2 (g=0,7, f=0,1)	523,2 (g=1,0, f=0,1)	633,3	707,2	745,5	691,4	503,1/549,7/666,2	0,4	9,7	32,9
eil30	516,3	527,4	508,6 (g=1,3, f=0,3)	**502,3** (g=1,5, f=0,2)	709,9	612,8	596,0	653,9	512,3/528,5/615,2	2,0	5,2	22,5
eil33	485,9	497,8	499,3 (g=1,0, f=0,3)	487,9 (g=0,7, f=0,1)	559,7	670,0	625,6	636,0	**480,3**/490,4/521,9	-1,2	0,9	7,4
F1	738,9	744,0	**728,5** (g=0,8, f=0,1)	749,2 (g=1,0, f=0,2)	922,4	953,7	931,5	875,4	733,6/788,2/933,7	0,7	8,2	28,2
F2	**223,8**	240,5	231,2 (g=1,4, f=0,1)	234,5 (g=1,5, f=0,2)	301,7	290,3	269,7	278,9	232,5/257,5/297,5	3,9	15,1	32,9
F3	**1192,1**	1209,9	1194,9 (g=1,3, f=0,1)	**1192,1** (g=1,1, f=0,1)	1539,4	1425,5	1455,2	1501,8	1216,4/1334,9/1505,7	2,0	12,0	26,3
GJ1	5490,0	5480,6	5467,5 (g=1,0, f=0,1)	**5454,0** (g=1,0, f=0,1)	6356,0	6356,0	6478,6	6478,6	5585,5/5762,4/6019,2	2,4	5,7	10,4
∅	-	-	-	-	-	-	-	-	-	**-0,6**	**5,3**	**18,7**
Anzahl bester Lösungen	3	0	2	4	0	0	0	0	8/5/1	-	-	-

Tabelle 4.7 : Vergleich der sequentiellen Ansätze mit Verbesserung des Standortes und der vorgeschlagenen SOM-Variante LRPSOMW bei Anwendung auf das CELRP[379]

[379] Fett markierte Einträge zeigen an, dass mit dem entsprechenden Verfahren die beste Lösung ermittelt wurde.

Der Algorithmus wird anschließend zur Ableitung einer Lösung für ECVRP-Instanzen verwendet. Hierzu wird unter Vernachlässigung des tatsächlichen Depot-Standortes durch Anwendung des Algorithmus zunächst eine Lösung für das simultane Touren- und Standortplanungsproblem bestimmt. Um eine Lösung für das ECVRP zu erhalten, wird der Depot-Standort schließlich auf die tatsächlichen Koordinaten des Depots verschoben. Da der Algorithmus Lösungen generiert, die eine zentrale Lage des Depots aufweisen, ist eine sinnvolle Anwendung auf ECVRP-Instanzen beschränkt, deren Depot eine zentrale Lage aufweist. Bei dezentraler Lage des Depots kann der Algorithmus gegebenenfalls verwendet werden, um mögliche Verbesserungen des Zielfunktionswertes bei Veränderung des Depot-Standortes abzuschätzen. Tabelle 4.8[380] gibt eine Übersicht über die Ergebnisse, die in jeweils 100 Durchläufen ermittelt wurden. Die Anzahl der eingesetzten Fahrzeuge wird in Klammern angegeben. Die Spalten der Tabelle 4.8 enthalten folgende Einträge: *Bestes/durchschnittliches(∅-)/schlechtestes Ergebnis LRPSOM* zeigt das beste, durchschnittliche und das schlechteste Ergebnis der zulässigen Lösungen, die mit der Variante *LRPSOM* ermittelt wurden. Die Spalte *DEV1* beschreibt die zugehörige Abweichung von der in Spalte 9 angegebenen besten bekannten Lösung *(BBL)* für die Probleminstanz. *Bestes/∅-/schlechtestes Ergebnis LRPSOMW* fasst die entsprechenden Ergebnisse bei Verwendung der *LRPSOMW*-Variante zusammen. Die Spalte *DEV2* enthält die Abweichung des resultierenden Zielfunktionswertes von der besten bekannten Lösung, die Spalte *DEV3* zeigt die Verbesserung des Zielfunktionswertes gegenüber der besten Lösung, die mit Hilfe der *LRPSOM*-Variante abgeleitet wurde. Die Spalte *Bestes/∅-/schlechtestes Ergebnis LRPSOMD* enthält jeweils das beste, durchschnittliche und schlechteste Ergebnis, wenn das Depot in den ursprünglichen Koordinaten errichtet wird, in der Spalte *SV* werden die mit Hilfe des Savings-Verfahrens ermittelten Zielfunktionswerte für die Instanz angegeben. Testergebnisse alternativer SOM-Varianten für Probleminstanzen der TSPLIB wurden den Untersuchungen von TORKI ET AL. und RETZKO entnommen.[381] Die Spalte *DEV4* beschreibt schließlich die Abweichung zwischen dem besten, durchschnittlichen und dem schlechtesten Ergebnis für *LRPSOMD* und der besten bekannten Lösung. [382]

[380] Vgl. S. 133.

[381] Die Lösungswahrscheinlichkeit der von Retzko verwendeten Variante ist mit der Lösungswahrscheinlichkeit der vorgeschlagenen Variante vergleichbar, vgl. Retzko [1996], S. 131. Torki et al. stellen keine Informationen bzgl. entsprechender Wahrscheinlichkeiten zur Verfügung.

[382] $DEV4 = \frac{Ergebnis_{LRPSOMD} - BBL}{BBL} \cdot 100$, vgl. hierzu bspw. Silver [2004], S. 951.

Instanz	Bestes/∅-/schlechtestes Ergebnis LRPSOM (Anzahl Fahrzeuge)	DEV1 (%)	Bestes[383]/∅-/schlechtestes Ergebnis LRPSOMW (Anzahl Fahrzeuge)	DEV2 (%)	DEV3 (%)	Bestes[384]/∅-/schlechtestes Ergebnis LRPSOMD (Anzahl Fahrzeuge)	SV[385]	TORKI ET AL.[386]	RETZKO[387]	BBL[388]		DEV4 (%)	
C1	530,0/561,1/608,9 (5)	1,0	**522,5**/552,7/587,1 (5)	-0,4	-1,4	528,2/566,6/610,7 (5)	578,6	537	524,9/537,4/579,1	***524,6* (5)**	0,7	8,0	16,4
C2	881,0/929,4/1032,1 (10)	5,5	879,0/920,1/988,3 (10)	5,2	-0,2	885,3/927,6/994,3 (10)	888,0	876	853,9/896,7/981,7	**835,3 (10)**	6,0	11,0	19,0
C3	836,3/859,5//997,7 (8)	1,2	835,6/848,3/876,4 (8)	1,1	-0,1	836,7/851,8/884,2 (8)	878,7	863	840,6/867,4/906,2	**826,1 (8)**	1,3	3,1	7,0
C4	829,7/835,3/912,0 (10)	1,2	828,9/829,9/832,5 (10)	1,1	-0,1	829,7/830,7/833,7 (10)	824,4	1082	829,7/831/834,5	***819,6* (10)**	1,2	1,4	1,7
C6 (D)	911,7/946,7/1215,8 (7)	-12,5	**905,7**/930,5/1080,4 (7)	-13,1	-0,7	1102,8/1113,5/1304,7 (7)	1048,5	-	1046/1076/1116	**1042,1 (7)**	-	-	-
C8	1061,3/1086,5/1136,8 (12)	3,2	1058,4/1075,6/1095,5 (12)	2,9	-0,3	1062,1/1080,2/1108,8 (12)	1140	-	1080/1105/1160	**1028,4 (12)**	3,3	5,0	7,8
C10	1359,4/1416,8/1695,2 (17)	5,3	1344,0/1369,7/1391,7 (17)	4,1	-1,1	1349,5/1374,4/1397,7 (17)	1396	-	-	**1291,5 (16)**	4,5	6,4	8,2
eil22	390,4/398,7/448,6 (4)	4,1	**374,7**/377,1/400,5 (4)	-0,1	-4,2	**375,3**/377,7/420,2 (4)	388,8	390	-	***375* (4)**	0,1	0,6	7,4
eil23	518,3/581,0/711,1 (3)	-8,9	**503,1**/549,7/666,2 (3)	-11,6	-3,0	569,9/588,7/703,0 (3)	621,1	585	-	***569* (3)**	0,2	3,5	23,6
eil30	512,5/555,9/685,4 (3)	-4,0	**512,3**/528,5/615,2 (3)	-4,1	0,0	542,7/560,2/796,2 (3)	534,4	557	-	***534* (3)**	1,6	4,9	49,1
eil33 (D)	486,6/524,5/587,5 (4)	-41,7	**480,3**/490,4/521,9 (4)	-42,5	-1,3	902,8/953,4/1007,8 (4)	843,1	889	-	***835* (4)**	-	-	-
F1	736,7/800,0/934,0 (4)	1,8	733,6/788,2/933,7 (4)	1,4	-0,4	739,5/829,8/958,6 (4)	739,0	-	-	***723,5* (4)**	2,2	14,7	32,5
F2 (D)	236,7/264,4/306,7 (4)	-2,2	**232,5**/257,5/297,5 (4)	-3,9	-1,8	265,5/298,7/365,0 (4)	256,2	-	-	***242* (4)**	-	-	-
F3 (D)	1238,5/1357,9/1520,2 (7)	6,6	1216,4/1334,9/1505,7 (7)	4,7	-1,8	1252,7/1378,1/1573,2 (7)	1219,3	-	-	***1162* (7)**	-	-	-
GJ1[389]	5607,9/5868,9/6501,2 (25)	1,0	5585,5/5762,4/6019,2 (25)	0,6	-0,4	5587,8/5801,5/6058,1 (25)	5510,2	-	-	**5552 (26)**	-	-	-
∅	-	-	-	-	**-1,1**	-	-	-	-	-	**2,1**[390] **(3,7)**	**5,9 (8,7)**	**17,3 (21,8)**

Tabelle 4.8: Ergebnisübersicht für das CELRP und das ECVRP

[383] Fett markierte Einträge zeigen an, dass geeignetere Koordinaten als die im ECVRP vorgegebenen Koordinaten für die Standorte gefunden wurden.
[384] Fett markierte Einträge zeigen an, dass die beste bekannte Lösung gefunden wurde.
[385] Vgl. Cordeau et al. [2002], S. 515, und Paessens [1988], S. 338. Werte für die Instanzen eil22 – GJ1 entstammen eigenen Berechnungen.
[386] Vgl. Torki et al. [1997], S. 476. Identische Ergebnisse werden in Modares et al. [1999], S. 605, veröffentlicht. Dort wird ebenfalls nur die beste ermittelte Lösung angegeben.
[387] Vgl. Retzko [1996], S. 129 ff.
[388] Die Ergebnisse wurden Cordeau et al. [2002], S. 515, Fisher [1994], S. 633, Paessens [1988], S. 338 f., http://www.branchandcut.org und http://neo.lcc.uma.es/radi-aeb/WebVRP entnommen. Kursive Einträge stellen Optimallösungen dar.
[389] Ergebnisse der Verfahren bei Vernachlässigung der Restriktionen, die die maximale Länge einer Tour beschränken. Der Wert *BBL* ist daher zum Vergleich nicht geeignet.
[390] Ergebnisse für Instanzen mit zentralem Depot (ohne Instanz GJ1), Werte in Klammern berücksichtigen auch Instanzen mit dezentralem Depot.

Abbildung 4.8 zeigt den Anstieg der benötigten Rechenzeiten[391] der Variante *LRPSOMW* und den übrigen eingesetzten Verfahren für die Testinstanzen. Es wurden jeweils 400 Präsentationszyklen durchgeführt, die Anzahl der Neuronen der Ausgabeschicht entspricht jeweils dem Vierfachen der Anzahl an Knoten der Testinstanz.

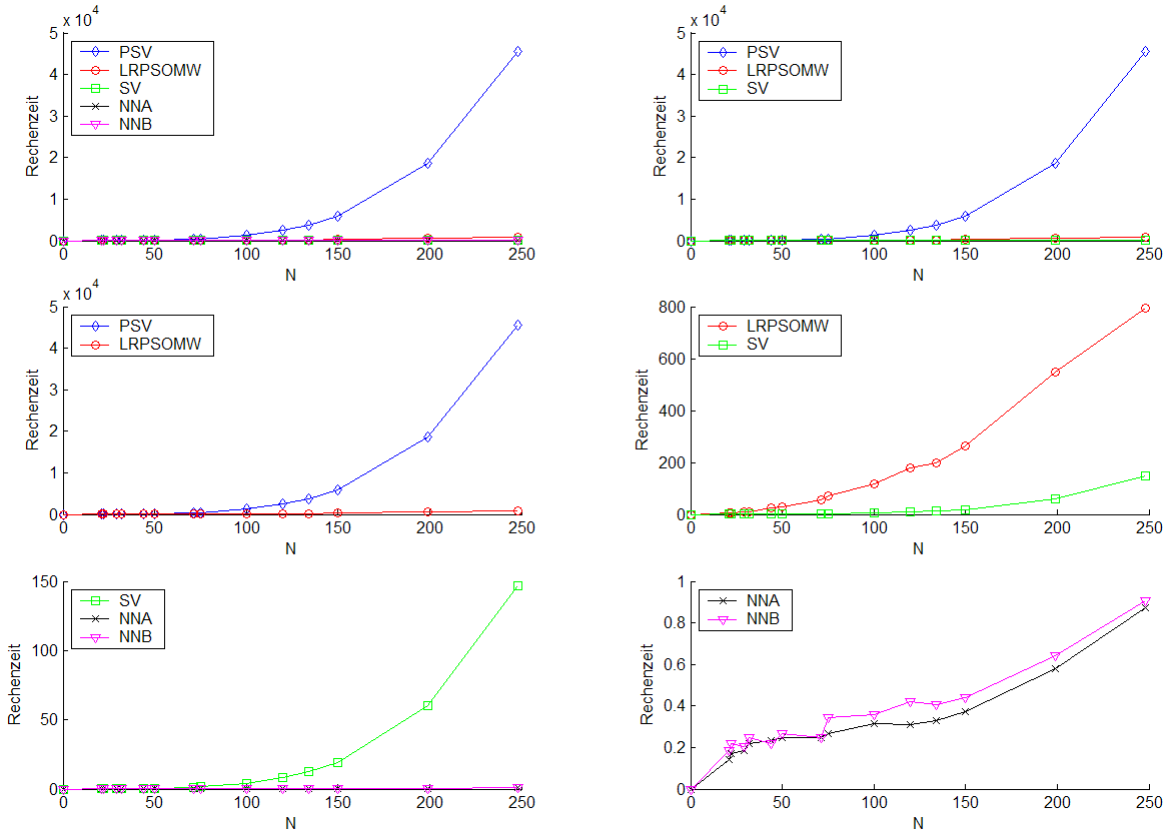


Abbildung 4.8: Benötigte Rechenzeiten für die Testinstanzen (Angaben in [s])

Aufgrund der hohen Anzahl untersuchter Parameterkombinationen weist das parametrisierte Savings-Verfahren (*PSV*) den größten Rechenzeitbedarf auf, der mit steigender Problemgröße deutlich zunimmt. Die SOM-Variante *LRPSOMW* benötigt im Vergleich zum *PSV* für die kleinsten untersuchten Testinstanzen etwas weniger als die Hälfte der benötigten Rechenzeit, mit zunehmender Problemgröße sinkt dieser Anteil auf ca. $\frac{1}{57}$ ab. Das Savings-Verfahren (*SV*) weist geringere Rechenzeiten als die verwendete SOM auf. Allerdings sinkt das Verhältnis zwischen der Rechenzeit der Variante *LRPSOMW* und der Rechenzeit des *SV* vom ca. 30-fachen mit zunehmender Problemgröße auf das ca. Fünffache ab. Es ist zu vermuten, dass sich die benötigten Rechenzeiten für größere Probleme weiter annähern. Die Nearest Neighbor-Varianten *NNA* und *NNB* sind wie-

[391] Alle Algorithmen wurden vom Verfasser in Matlab implementiert. Die Rechenzeiten können bei Verwendung anderer Programmiersprachen vermindert werden. Testläufe wurden auf einem Standard-PC (1,4 GHz Athlon-Prozessor, 512 MB RAM) durchgeführt.

derum deutlich schneller als das *SV*, beispielsweise wird für die größte untersuchte Testinstanz etwa nur $\frac{1}{150}$ der Rechenzeit benötigt.

4.5 Diskussion der Ergebnisse

Im Direktvergleich zu den einzelnen sequentiellen Verfahren zur Lösung des CELRP zeigt sich, dass die SOM-Varianten häufiger die Lösung mit dem geringsten Zielfunktionswert ermitteln. Während das beste Vergleichsverfahren insgesamt in drei beziehungsweise vier Fällen bessere Lösungen bestimmt, können die Varianten der SOM sieben beziehungsweise acht Mal die beste Lösung ableiten. Darüber hinaus sind für *LRPSOM* und *LRPSOMW* die durchschnittlich ermittelten Ergebnisse in zwei beziehungsweise fünf Fällen den Vergleichsverfahren überlegen, in einem Fall weist sogar die schlechteste Lösung der *LRPSOMW*-Prozedur einen geringeren Zielfunktionswert als die beste Lösung der Vergleichsverfahren auf.

Der Gesamtvergleich mit den sequentiellen Verfahren zeigt, dass die KNN sieben beziehungsweise acht Mal die beste Lösungsqualität aufweisen, während die Vergleichsverfahren entsprechend acht beziehungsweise sieben Mal bessere Ergebnisse erzielen.

Die prozentualen Abweichungen der mit Hilfe der *LRPSOMW*-Prozedur ermittelten besten Lösung von der besten Lösung aller Vergleichsverfahren mit Verbesserung des Standortes schwanken zwischen –4,7% und 3,9%, im Durchschnitt ist die ermittelte Gesamtlänge der Tourenpläne um 0,6% kürzer. Die entsprechenden Abweichungen bei Vergleich der *LRPSOM*-Prozedur mit den Vergleichsverfahren ohne Verbesserung des Standortes schwanken zwischen –5,1% und +5,6%, die ermittelte Gesamtlänge ist durchschnittlich um 0,1% länger.

Die Ergebnisse der *LRPSOM*-Variante können durch Anschluss des Algorithmus von WEISZFELD durchschnittlich um 1,1% verbessert werden. Eine Abhängigkeit der Lösungsgüte der Verfahren von der Verteilung der Knoten lässt sich nicht erkennen. Sowohl für geclusterte als auch für gleichmäßig und unregelmäßig verteilte Knoten bestimmen die SOM-Algorithmen teilweise bessere, teilweise schlechtere Ergebnisse als die herangezogenen Vergleichsverfahren. Die Mittelwerte der Gesamtlängen sind durchschnittlich um 5,3% bzw. 6,3% länger als die beste Lösung aller Vergleichsverfahren. Betrachtet man die schlechtesten Lösungen, ergeben sich Abweichungen von 18,7% bzw. 22,1%. Um einen Eindruck über eine typische Verteilung der Zielfunktionswerte ermittelter Lösungen zu geben, werden in der Abbildung 4.9 stellvertretend

die mit Hilfe der Variante *LRPSOM* bestimmten Ergebnisse für die Instanz C8 dargestellt[392].

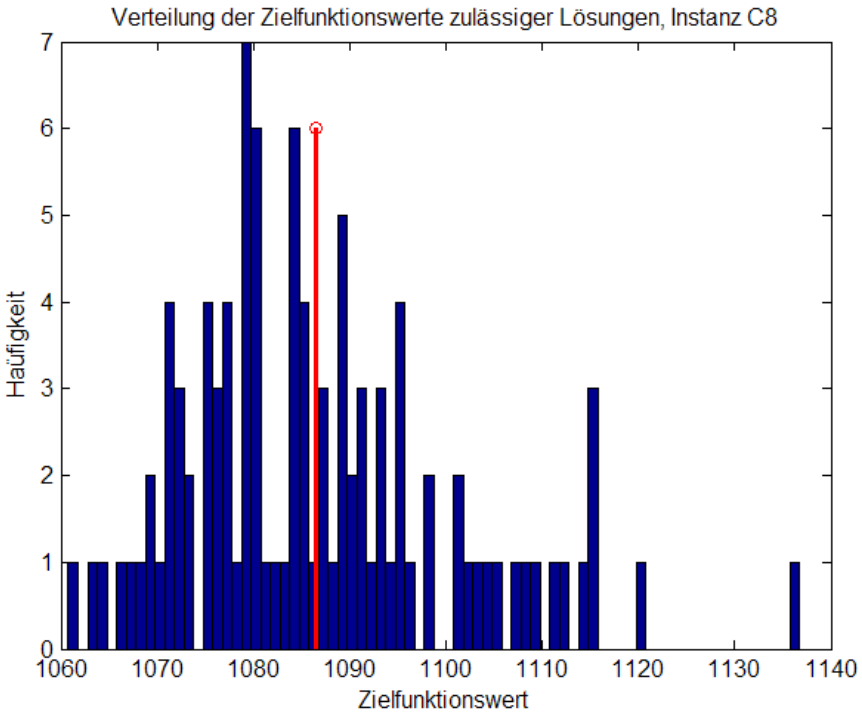


Abbildung 4.9 : Verteilung ermittelter Zielfunktionswerte für die Testinstanz C8, Variante LRPSOM

Die Ergebnisse der Nearest Neighbor-Verfahren sind aufgrund der Kurzsichtigkeit der Verfahren bezüglich der Gesamtlängen nicht konkurrenzfähig, allerdings ist der Rechenaufwand geringer als bei den übrigen Verfahren. Da die Qualität einer Eröffnungslösung Auswirkungen auf den Erfolg eines angeschlossenen Verbesserungsverfahrens haben kann, sollten die Nearest Neighbor-Varianten nur eingesetzt werden, wenn die Lösung möglichst schnell bestimmt werden muss.[393]

Die Verfahren, die das parametrisierte Savings-Verfahren zur Bestimmung der Tourenpläne einsetzen, stellen unter den sequentiellen Verfahren erwartungsgemäß die leistungsfähigsten Verfahren bezüglich der Lösungsqualität dar. Es ist allerdings darauf hinzuweisen, dass durch die Variation der Parameter deutlich erhöhte Rechenzeiten in Kauf genommen werden müssen.

Zusätzlich ist zu erwähnen, dass andere Ansätze zur Ermittlung einer zulässigen Lösung für das Tourenplanungsproblem zu besseren Ergebnissen für die sequentiellen Verfahren führen könnten. Außerdem könnten an alle Algorithmen Verbesserungsverfahren

[392] Der in der Abbildung rot markierte Mittelwert beträgt 1086,5, die Standardabweichung weist für den Testlauf einen Wert von 14,1 auf.

[393] In diesem Fall könnte auch der Einsatz einer anderen Heuristik mit geringer Zeitkomplexität in Betracht gezogen werden.

angeschlossen werden, um eine Verringerung der erzielten Zielfunktionswerte zu erreichen.

Bei Anwendung auf ECVRP-Instanzen mit einem zentralen Depot generiert der vorgeschlagene Algorithmus *LRPSOMD* zufriedenstellende Ergebnisse, die durchschnittlich um 2,1% von der besten bekannten Lösung abweichen. In einem Fall gelingt es, die besten bekannte Lösung mit Hilfe der *LRPSOMD*-Variante zu bestimmen, in sieben von fünfzehn Fällen können bessere Lösungen als mit dem Savings-Verfahren ermittelt werden. Berücksichtigt man, dass vier Instanzen, in denen das Savings-Verfahren bessere Ergebnisse erzielt, eine dezentrale Lage des Depots aufweisen, so sind auch diese Ergebnisse als zufriedenstellend zu bezeichnen. Ein Vergleich der Variante *LRPSOMD* mit den Ergebnissen von TORKI ET AL.[394] erscheint problematisch, da nur die besten Ergebnisse angegeben werden und aufgrund der Ganzzahligkeit der präsentierten Ergebnisse davon auszugehen ist, dass die Autoren mit gerundeten euklidischen Distanzen arbeiten. Darüber hinaus wird keine Angabe zur Anzahl durchgeführter Testläufe zur Verfügung gestellt. Führt man dennoch einen Vergleich durch, so zeigt sich, dass die Variante von TORKI ET AL. nur in zwei von acht Fällen bessere Ergebnisse generiert.

RETZKO führt zur Ermittlung des besten, durchschnittlichen und des schlechtesten Ergebnisses jeweils 500 Testläufe durch[395]. Aufgrund der präsentierten Ergebnisse ist davon auszugehen, dass - wie auch bei den Varianten *LRPSOM(W/D)* - keine Rundung der euklidischen Distanzen auf ganzzahlige Werte vorgenommen wurde, so dass bei einem Vergleich lediglich die unterschiedliche Anzahl an Testläufen zu kritisieren ist. In zwei von sechs Fällen weisen die mit der Variante *LRPSOMD* ermittelten besten Lösungen geringere Zielfunktionswerte als die Variante von RETZKO auf, in einem Fall stimmen die besten ermittelten Lösungen überein. Für drei von sechs Instanzen sind die durchschnittlichen und schlechtesten Gesamtlängen kürzer, in den übrigen Fällen länger. Insgesamt können die Ergebnisse der Variante *LRPSOMD* im Vergleich zu den herangezogenen Vergleichsverfahren und den besten bekannten Lösungen für Instanzen mit zentralem Depot als gut bezeichnet werden. Die hohe Abweichung von der besten bekannten Lösung für die Testinstanz *eil33* lässt sich durch die dezentrale Lage des Depots erklären.

Nachteilig stellten sich folgende Eigenschaften der KNN-Ansätze dar: Zum einen müssen die verwendeten Parameter an die jeweilige Instanz angepasst werden, zum anderen

[394] Vgl. Torki et al. [1997], S. 476.
[395] Vgl. Retzko [1996], S. 131ff.

treten Probleme auf, wenn scharfe Kapazitätsrestriktionen vorliegen. In diesem Fall kann die Gewichtung der Fahrzeugauslastung in der Distanzfunktion erhöht werden, damit der Anteil ermittelter zulässiger Lösungen zunimmt. Dieses Vorgehen führt jedoch dazu, dass die Wahl des Zentrums in geringerem Maße von der euklidischen Distanz abhängt, was zu Lösungen mit einer höheren Gesamtlänge führen kann.[396] Einen Überblick über diese Problematik gibt die Tabelle 4.9, in der Ergebnisse von jeweils 100 Durchläufen für die Instanz F1 zusammengestellt wurden.

Bestes/Ø-/schlechtestes Ergebnis LRPSOM	Bestes/Ø-/schlechtestes Ergebnis LRPSOMW	Bestes/Ø- /schlechtestes Ergebnis LRPSOMD	Gewicht der Fahrzeugauslastung/Tabu-Zähler/ Ausschlussverbot	Anteil unzulässiger Lösungen (%)
740,1/781,2/905,8	736,6/774,2/850,5	748,0/800,6/962,4	65/0/Nein	0
-	-	-	-	-
741,7/774,9/847,7	740,1/768,2/816,9	750,9/788,4/899,1	50/0/Nein	7
744,7/786,8/877,3	741,2/779,2/853,1	752,7/813,5/977,2	50/1/Ja	0
735,3/767,0/**804,0**	734,1/763,5/789,0	745,5/779,2/853,0	45/0/ Nein	10
735,3/787,3/882,1	734,3/780,3/860,5	743,4/817,1/972,4	45/1/Ja	1
735,0/770,3/814,4	734,0/762,5/803,9	**739,0**/781,2/882,2	40/0/ Nein	18
735,3/791,6/915,0	734,2/784,1/905,3	740,5/821,5/931,5	40/1/Ja	0
735,0/764,5/804,8	734,0/759,5/788,9	740,3/776,6/882,1	35/0/ Nein	42
735,1/804,2/959,1	734,1/792,5/958,7	745,6/804,2/1031,6	35/1/Ja	1
735,0/**761,0**/821,7	734,0/**755,8**/**784,2**	741,7/**774,2**/**848,2**	30/0/ Nein	80
735,0/791,0/979,1	734,0/804,7/889,5	745,4/834,3/936,6	30/1/Ja	1
-	-	-	20/0/ Nein	100
730,9/808,3/946,2	**729,6**/792,2/887,1	741,1/830,2/984,4	20/1/Ja	3
-	-	-	10/0/ Nein	100
765,3/837,2/992,3	751,5/822,8/959,6	758,0/852,6/979,0	10/1/Ja	47

Tabelle 4.9: Ergebnisübersicht für das CELRP bei Variation des Auslastungsgewichtes, des Tabu-Zählers und des Ausschlussverbotes, Instanz F1[397]

Die gewichtete Fahrzeugauslastung wurde dabei nur in der ersten Hälfte der Präsentationszyklen berücksichtigt. Für ein bestimmtes Gewicht der Fahrzeugauslastung wurden in der Tabelle jeweils die resultierenden Zielfunktionswerte und der Anteil unzulässiger Lösungen bei Variation des Tabu-Zählers und des Ausschlussverbotes angegeben. Wird beispielsweise das Gewicht der Fahrzeugauslastung auf den Wert 30 gesetzt (s. Spalte „Gewicht der Fahrzeugauslastung/Tabu-Zähler/Ausschlussverbot“), so ergibt sich ohne Verwendung des Tabu-Zählers und des Ausschlussverbotes der Anteil unzulässiger Lösungen zu 80% (s. zugehöriger Eintrag in der Spalte „Anteil unzulässiger Lösungen“). Bei Aktivierung des Ausschlussverbots und des Tabu-Zählers (die erneute Zuordnung

[396] Vgl. hierzu auch die Ergebnisse von Retzko [1996], S. 140 f.
[397] Fett markierte Werte stellen die besten Ergebnisse einer Kategorie dar.

eines ausgeschlossenen Kunden zur Tour, aus der er entfernt wurde, wird hier für einen Präsentationszyklus untersagt) kann der Anteil unzulässiger Lösungen auf nur 1% gesenkt werden. Desweiteren ist zu erkennen, dass sich die Zielfunktionswerte der besten Lösungen nicht wesentlich voneinander unterscheiden, während für die Mittelwerte und schlechtesten Lösungen jeweils höhere Werte bei Verwendung des Tabu-Zählers und des Ausschlussverbotes resultieren. Eine mögliche Erklärung hierfür ist, dass für viele unzulässige Lösungen im Verlaufe des Verfahrens nur unter Erhöhung des Zielfunktionswertes die Zulässigkeit wiederhergestellt werden kann.

Die Tabelle 4.10 gibt die Ergebnisse für jeweils 1000 Durchläufe für die Instanz *eil30* wieder, wenn kein Ausschlussverbot ausgesprochen und nur der Wert des Tabu-Zählers variiert wird. Die gewichtete Fahrzeugauslastung wurde über die gesamte Laufzeit des Algorithmus in der Distanzfunktion berücksichtigt. Es ist zu erkennen, dass hieraus eine Erhöhung der Lösungswahrscheinlichkeit resultiert, ohne dass eine wesentliche Veränderung der Ergebnisqualität stattfindet. Offenbar weist ein größerer Anteil der zulässigen Lösungen eine günstigere Zuordnung der Kunden zu Touren auf, wenn das Ausschlussverbot nicht aktiviert wird. Im Gegenzug fällt die Verminderung des Anteils unzulässiger Lösungen allerdings sehr viel geringer aus als bei zusätzlicher Verwendung des Ausschlussverbots.

Tabu-Zähler	Anteil unzulässiger Lösungen (%)	Bestes/Ø-/schlechtestes Ergebnis LRPSOM	Bestes/Ø-/schlechtestes Ergebnis LRPSOMW	Bestes/Ø- /schlechtestes Ergebnis LRPSOMD	Auslastungsgewicht
0	32,8	507,3/**533,5**/661,3	**506,8**/**516,1**/**604,8**	**537,1**/**546,4**/**609,4**	130
1	31,9	512,0/575,2/**654,4**	**506,8**/567,4/632,0	**537,1**/596,7/672,5	130
2	26,5	507,6/534,3/685,0	**506,8**/516,9/616,5	**537,1**/546,7/620,4	130
3	25,2	**506,9**/535,4/**654,4**	**506,8**/516,3/612,0	**537,1**/546,7/614,1	130

Tabelle 4.10: Ergebnisübersicht für das CELRP bei Variation des Tabu-Zählers, Instanz eil30[398]

Sehr geringe Kapazitätsüberschüsse bei Verwendung der mindestens benötigten Fahrzeuganzahl[399] können dazu führen, dass der vorgeschlagene Algorithmus keine zulässige Lösung ermitteln kann. Ähnliche Probleme können auch bei anderen Autoren beobachtet werden[400], so dass vermutet werden kann, dass solche scharfen Restriktionen ein noch nicht gelöstes Problem für Self-Organizing Maps darstellen. Ein möglicher Ansatz

[398] Fett markierte Werte stellen die besten Ergebnisse einer Kategorie dar.
[399] Instanz C10 weist beispielsweise einen Kapazitätsüberschuss von nur ca. 1% bei Einsatz von 16 Fahrzeugen auf, vgl. Tabelle 3.19 und Tabelle 3.22.
[400] Vgl. beispielsweise Retzko [1996], S. 131.

zur Lösung stellt in diesem Fall die Erhöhung des Exponenten der Fahrzeugauslastung in der Distanzfunktion dar. Erste Testläufe ergaben hier, dass auch für sehr scharfe Restriktionen unter zum Teil starker Erhöhung der Gesamtlänge des Tourenplanes eine zulässige Lösung bestimmt werden kann.

Es ist zu erwähnen, dass mehrere Läufe mit unterschiedlicher Anzahl an Neuronenringen durchgeführt werden können, um eine hinsichtlich der Zielfunktion geeignete Zahl einzusetzender Fahrzeuge zu bestimmen.

5. Zusammenfassung und Ausblick

Die wesentlichen Ergebnisse der vorliegenden Untersuchung können folgendermaßen zusammengefasst werden:

Die vorgenommene Literaturauswertung ermöglicht neben der Vermittlung eines Überblicks über die Anwendungsmöglichkeiten und der Einschätzung der Lösungsqualität der unterschiedlichen SOM-Varianten auch die Identifikation besonders erfolgsversprechender Erweiterungen.
Bisher wurde der Algorithmus überwiegend zur Lösung euklidischer Traveling Salesman Probleme, Multiple Traveling Salesman Probleme und Tourenplanungsprobleme eingesetzt. Anhand der in der Literatur vorgefundenen Ergebnisse für die unterschiedlichen Problemtypen lässt sich folgern, dass die SOM im Vergleich zu anderen Eröffnungsverfahren durchaus konkurrenzfähig ist. So dominieren weder das Elastic Net, das Savings-Verfahren, noch Nearest Neighbor-Verfahren oder der Spacefilling Curve-Ansatz das neuronale Netz hinsichtlich der Lösungsqualität. Bezüglich der Rechenzeiten relativiert sich die Leistungsfähigkeit des KNN, so weisen beispielsweise der Spacefilling Curve-Ansatz oder Nearest Neighbor-Verfahren eine geringere Zeitkomplexität als die meisten vorgeschlagenen Varianten der SOM auf. Ansätze zur Beschleunigung des SOM-Algorithmus sind verfügbar, dennoch verzichtet der überwiegende Teil der vorgefundenen Untersuchungen auf entsprechende Modifikationen und verweist auf die mögliche Beschleunigung durch Parallelisierung des Verfahrens.
In der Literatur vorgenommene Vergleiche mit verschiedenen Verbesserungsverfahren zeigen, dass die SOM meistens hinsichtlich der Lösungsqualität (gilt beispielsweise für Tabu Search-Prozeduren), zum Teil auch hinsichtlich der benötigten Rechenzeiten unterliegt (gilt beispielsweise für 2-Opt-Verfahren). Untersuchungen verschiedener Autoren zeigen allerdings, dass sich durch eine Kombination der SOM mit 2-Opt-Verfahren im Vergleich zu anderen Verbesserungsverfahren ebenfalls gute Lösungen mit relativ geringem Gesamtzeitaufwand ermitteln lassen. Ein umfassender Vergleich unterschiedlicher hybrider SOM-Varianten, die um verschiedene Verbesserungsverfahren erweitert wurden, und ein Vergleich dieser mit den besten bekannten Verbesserungsverfahren existiert bisher allerdings für keine der vorgestellten Problemstellungen.

In der Literatur wurden bisher keine Erweiterungen des SOM-Algorithmus vorgeschlagen, die eine Anwendung des Verfahrens zur Lösung simultaner Standort- und Tourenplanungsprobleme ermöglicht.

Deshalb wurde auf der Basis der von ENGELE vorgeschlagenen Modellformulierung für ein mehrstufiges Problem der simultanen Standort- und Tourenplanung in der Ebene[401] zunächst ein einstufiges Problem (CELRP) entwickelt. Das resultierende Modell kann - wie alle kontinuierlichen Standortmodelle - zu Ergebnissen führen, die für die Errichtung eines Depots nicht zulässig sind. Das Modell kann dennoch eingesetzt werden, um eine Näherungslösung für den Standort zu generieren und / oder die Anzahl potentieller Standorte einzugrenzen. Darüber hinaus sind unterschiedliche Problemstellungen vorstellbar, die eine kontinuierliche Modellierung mit euklidischen Entfernungen erfordern.

Zur Lösung des formulierten Problems wurde die SOM adaptiert, indem eine Veränderung der Struktur des Neuronengitters der Ausgabeschicht und eine Anpassung der Definition der Nachbarschaftsbeziehungen innerhalb des Neuronengitters vorgenommen wurde. Zur Einschätzung der Lösungsqualität des Eröffnungsverfahrens wurden verschiedene alternative Verfahren implementiert. Während bei diesen alternativen Verfahren Heuristiken kombiniert wurden, um sequentiell die Teilprobleme des CELRP zu lösen, sind die entwickelten Self-Organizing Map-Varianten *LRPSOM* und *LRPSOMW* in der Lage, Aspekte dieser Teilprobleme *simultan* zu berücksichtigen. Anhand des Vergleichs der Ergebnisse der auch von der Wahl des Standortes abhängenden Gesamtlängen der resultierenden Tourenpläne konnte für eine Reihe von Testinstanzen gezeigt werden, dass die Algorithmen *LRPSOM* und *LRPSOMW* im Direktvergleich mit dem besten sequentiellen Verfahren – eine Kombination aus dem von WEISZFELD zur Lösung des Weber-Problems entwickelten iterativen Näherungsverfahrens und dem von PAESSENS vorgeschlagenen parametrisierten Savings-Verfahren – für die untersuchten Testinstanzen konkurrenzfähige Ergebnisse bei geringerem Rechenzeitbedarf erzielen. Auch der Gesamtvergleich zu den übrigen herangezogenen Verfahren zeigt, dass die SOM-Varianten gute Ergebnisse für die Testinstanzen ermitteln.

Die entwickelten Algorithmen sind zur Erweiterung auf den Mehr-Depot-Fall geeignet, die Überprüfung der Lösungsqualität der SOM für solche Problemstellungen stellt einen möglichen Ansatzpunkt für weiterführenden Forschungsarbeiten dar. Darüber hinaus sind die Algorithmen auf Tourenplanungsprobleme mit zentralem Depot anwendbar. Zur Einschätzung der Lösungsqualität wurden verschiedene Testinstanzen herangezo-

[401] Vgl. Engele [1981], S. 91.

gen. Auch hier konnten mit den vorgeschlagenen SOM-Ansätzen gute Ergebnisse erzielt werden.

Verschiedene Problemstellungen aus dem Bereich der Routenplanung weisen Restriktionen auf, deren Einhaltung nicht durch eine Anpassung des Neuronengitters oder der Distanzfunktionen garantiert werden kann, so dass die SOM im Verlauf des Verfahrens unter Umständen einen Zustand annimmt, der eine unzulässige Lösung repräsentiert. Bisher vorgeschlagene Erweiterungen der Distanzfunktion zur Berücksichtigung entsprechender Restriktionen führen neben einer Erhöhung der Lösungswahrscheinlichkeit meist auch zu einer Verschlechterung der Zielfunktionswerte der ermittelten Lösungen. Als vorteilhaft erweist sich die in dieser Arbeit vorgenommene Erweiterung der SOM durch einen Mechanismus, der die Wiederherstellung der Zulässigkeit im Verlauf des Verfahrens unterstützt. Es konnte gezeigt werden, dass die Lösungswahrscheinlichkeit des Algorithmus durch Einsatz des entwickelten Mechanismus, dessen grundsätzliche Struktur an unterschiedliche Restriktionstypen angepasst werden kann, gesteigert werden kann, ohne dass eine wesentliche Verschlechterung der Lösungsqualität auftritt.

Abschließend kann zusammengefasst werden, dass die Self-Organizing Map als Eröffnungsverfahren zur Bestimmung von zulässigen Lösungen für verschiedene euklidische Routenplanungsprobleme geeignet ist, aber kein dominierendes Verfahren darstellt.

Anhang

A. Koordinaten der Knoten der in den Beispielen verwendeten Instanz.

x-Koordinate	y-Koordinate
13,409	52,521
10,033	53,567
11,573	48,138
6,957	50,942
8,684	50,117
7,014	51,457
7,465	51,513
9,179	48,775
6,782	51,224
8,808	53,076
9,739	52,374
6,772	51,438
12,375	51,341
11,079	49,452
13,738	51,053
7,217	51,479
7,199	51,266
8,533	52,021
8,462	49,483
7,097	50,732
8,404	49,009
7,100	51,549
8,240	50,097
8,860	49,925
6,434	51,197
10,897	48,371
12,926	50,839
6,084	50,775
10,525	52,264
11,970	51,483
6,561	51,337
10,139	54,324
11,635	52,122
6,864	51,479
10,685	53,868
7,850	47,995
7,464	51,361
11,029	50,976
12,129	54,089
9,497	51,319
8,271	49,997
7,815	51,675
6,998	49,236
7,223	51,541
6,879	51,430
7,085	51,171

8,042	52,278
8,443	49,465
6,993	51,029
8,213	53,142
6,683	51,199
8,690	49,403
8,757	51,718
8,651	49,873
13,044	52,402
9,932	49,794
12,094	49,016
7,198	51,617
9,934	51,533
10,786	52,422
6,929	51,525
9,218	49,151
7,184	51,185
8,580	53,544
8,774	50,099
9,985	48,398
8,699	48,890
11,427	48,766
10,415	52,116
9,215	48,492
10,987	49,478
12,084	50,876
8,024	50,875
7,594	50,356
6,637	51,454
14,331	51,755
7,130	50,991
9,952	52,153
7,338	51,440
11,005	49,596
12,495	50,718
11,589	50,929
6,644	49,757

Tabelle A.1:
Koordinaten der Knoten der in den Beispielen verwendeten Instanz

B. Abbildungen der Testinstanzen.

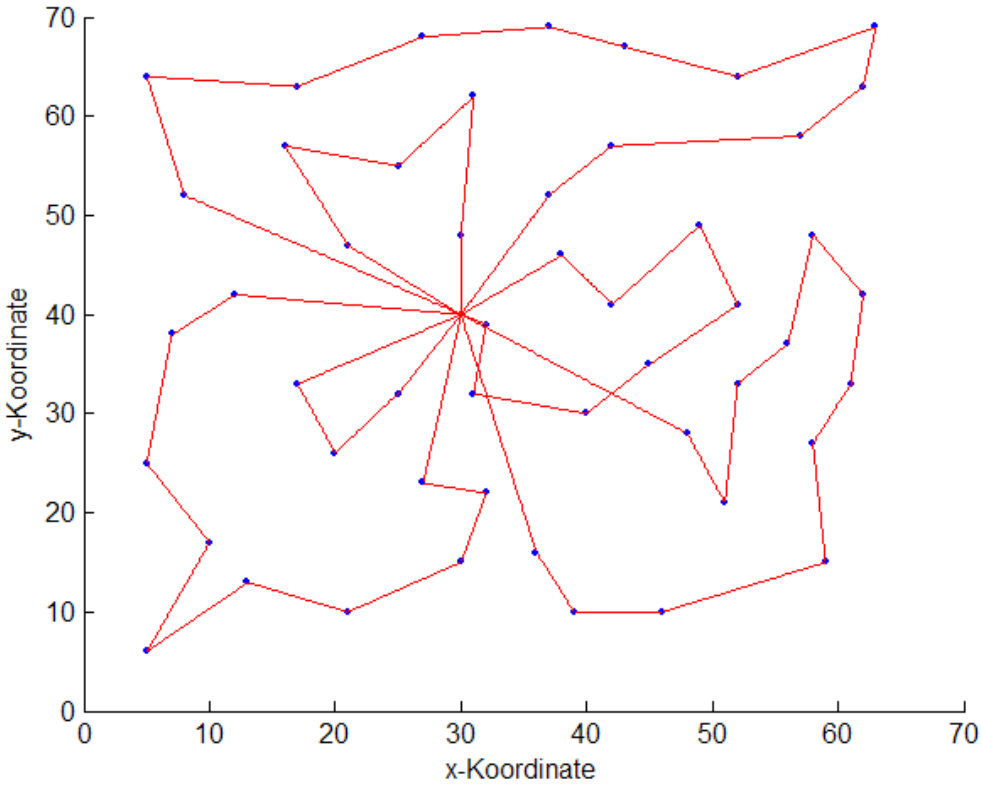


Abbildung A.1: Testinstanz C1

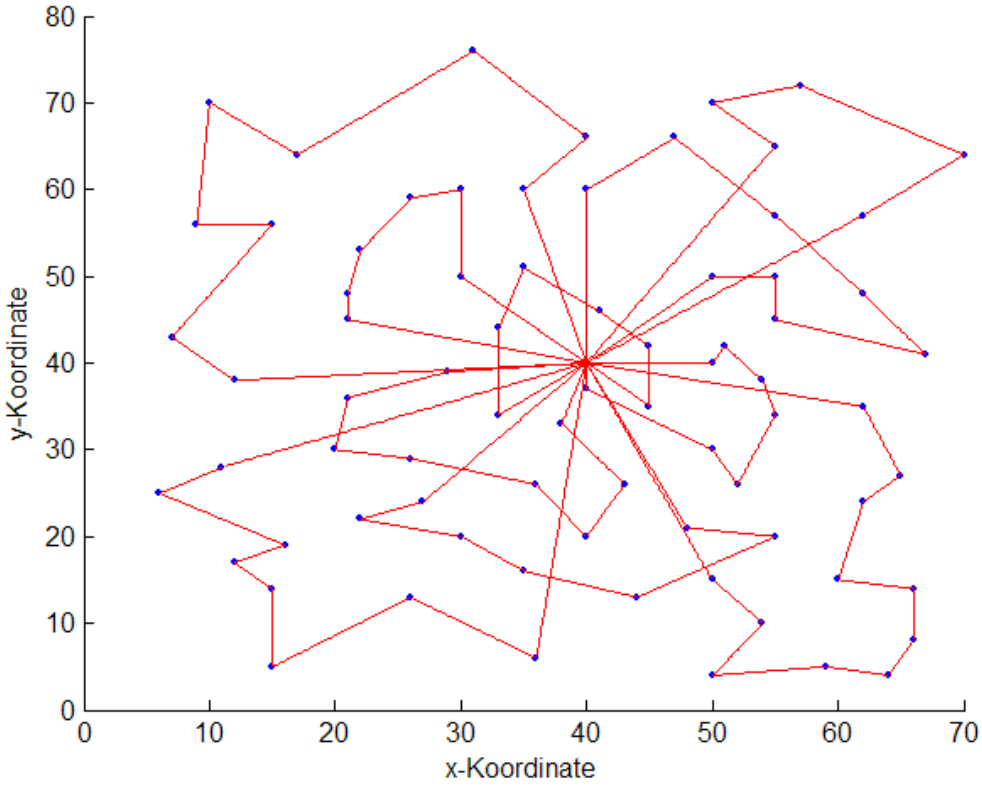


Abbildung A.2: Testinstanz C2

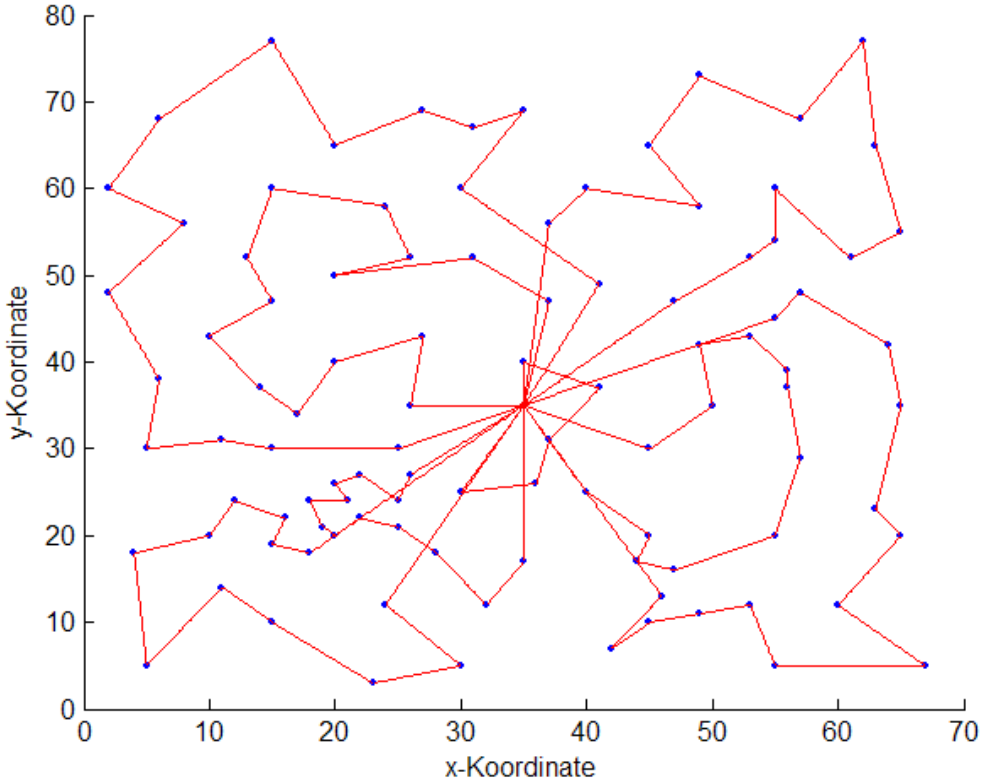


Abbildung A.3: Testinstanz C3

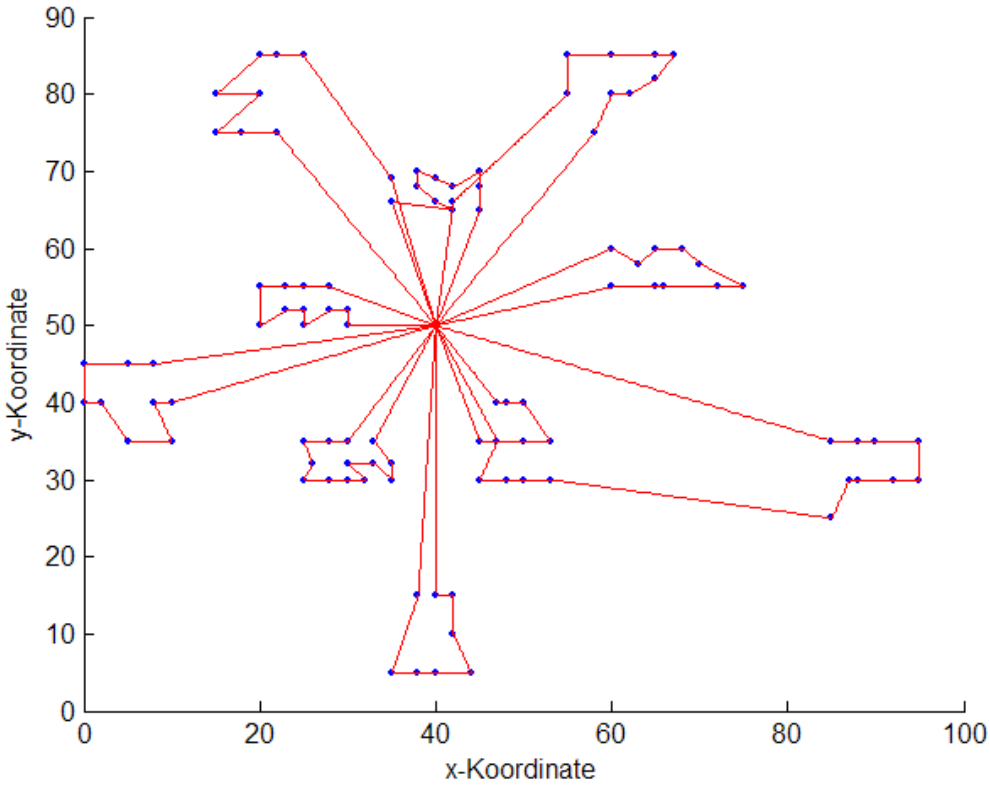


Abbildung A.4: Testinstanz C4

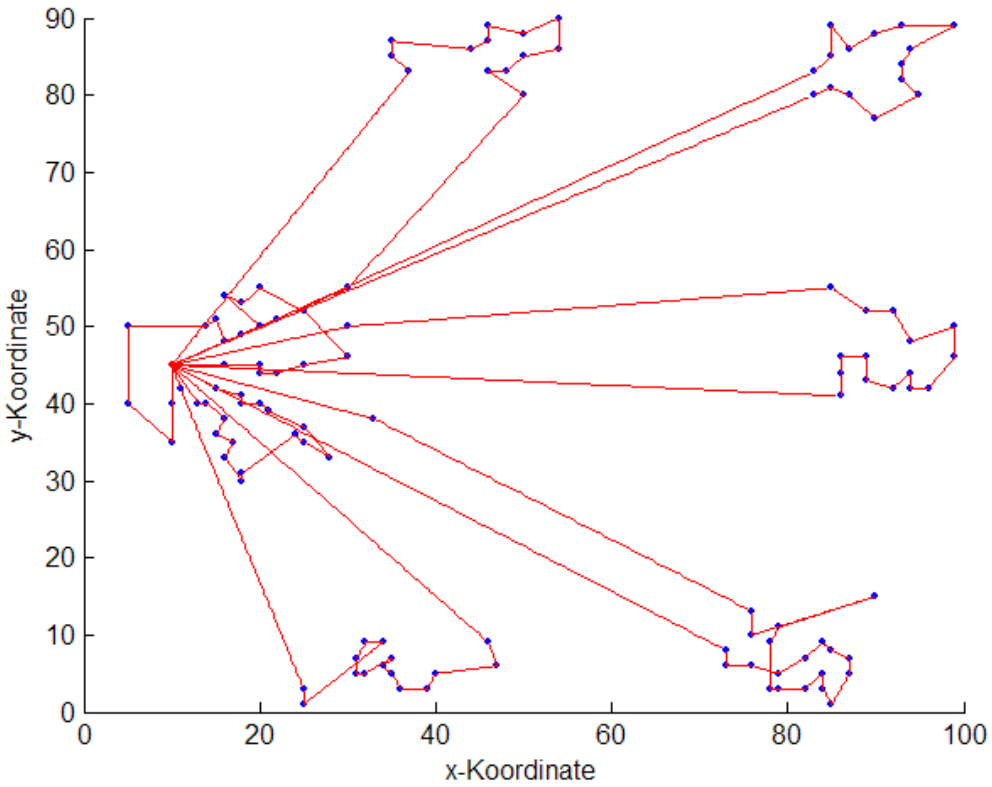


Abbildung A.5: Testinstanz C6

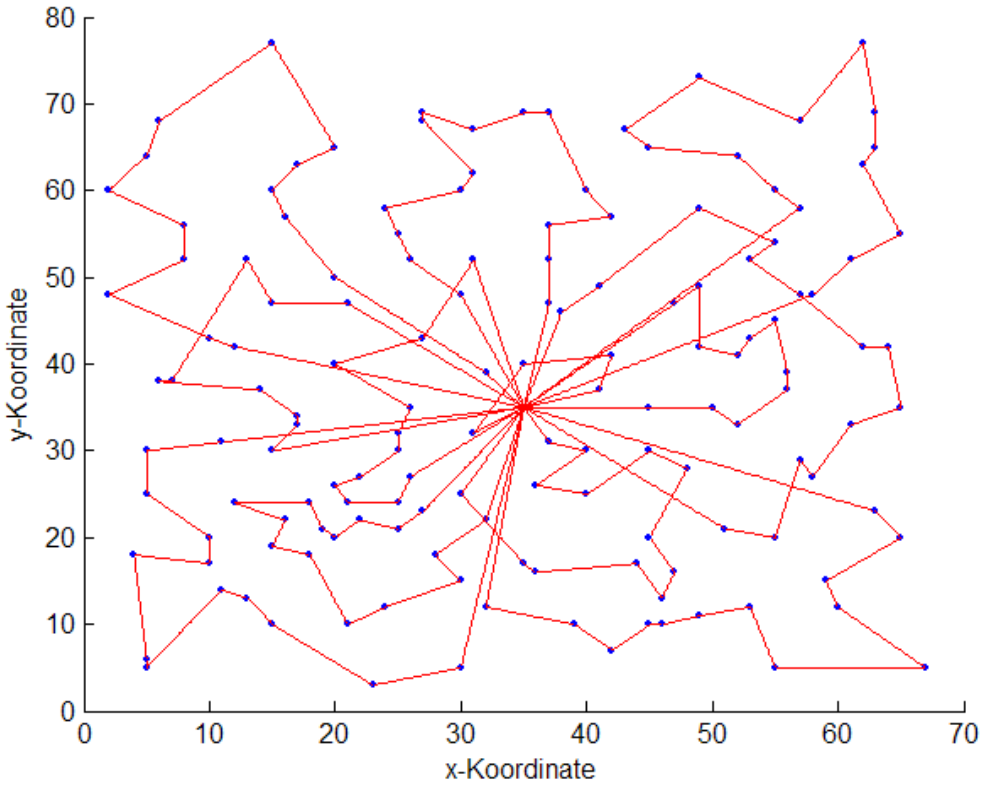


Abbildung A.6: Testinstanz C8

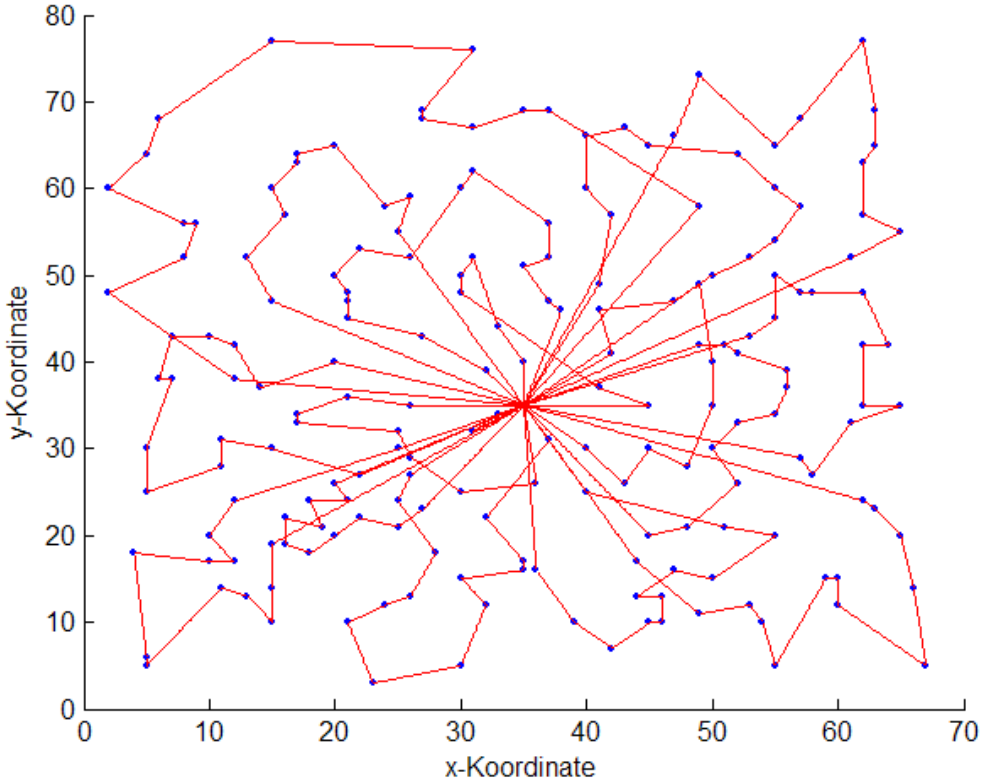


Abbildung A.7: Testinstanz C10

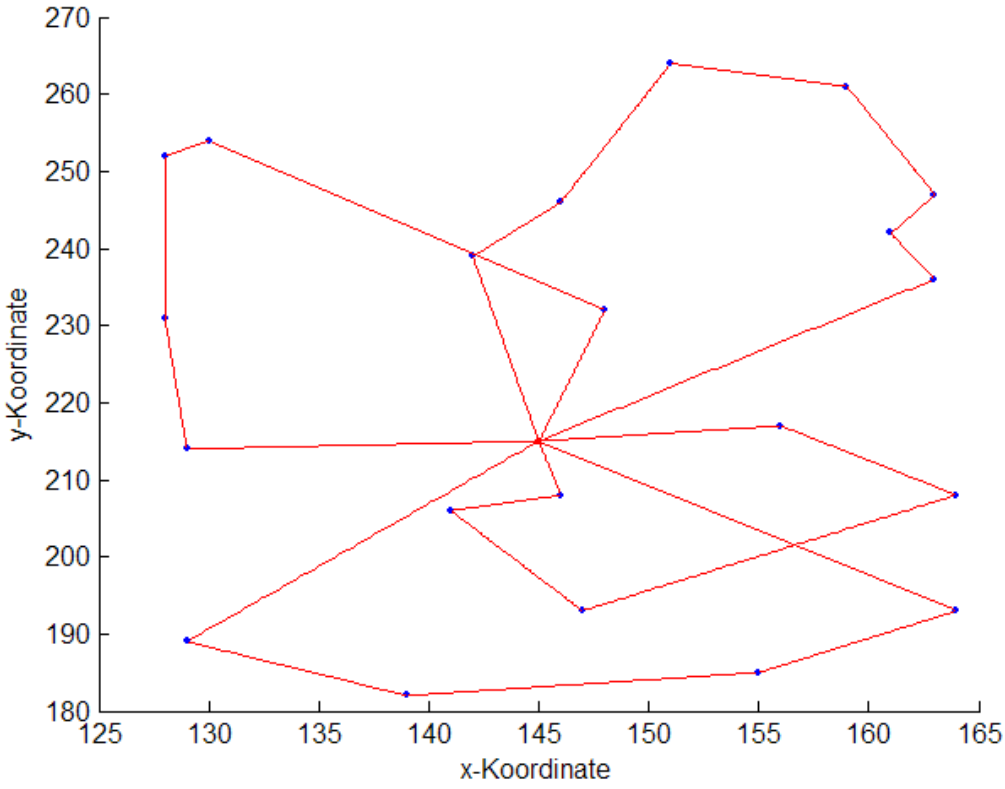


Abbildung A.8: Testinstanz eil22

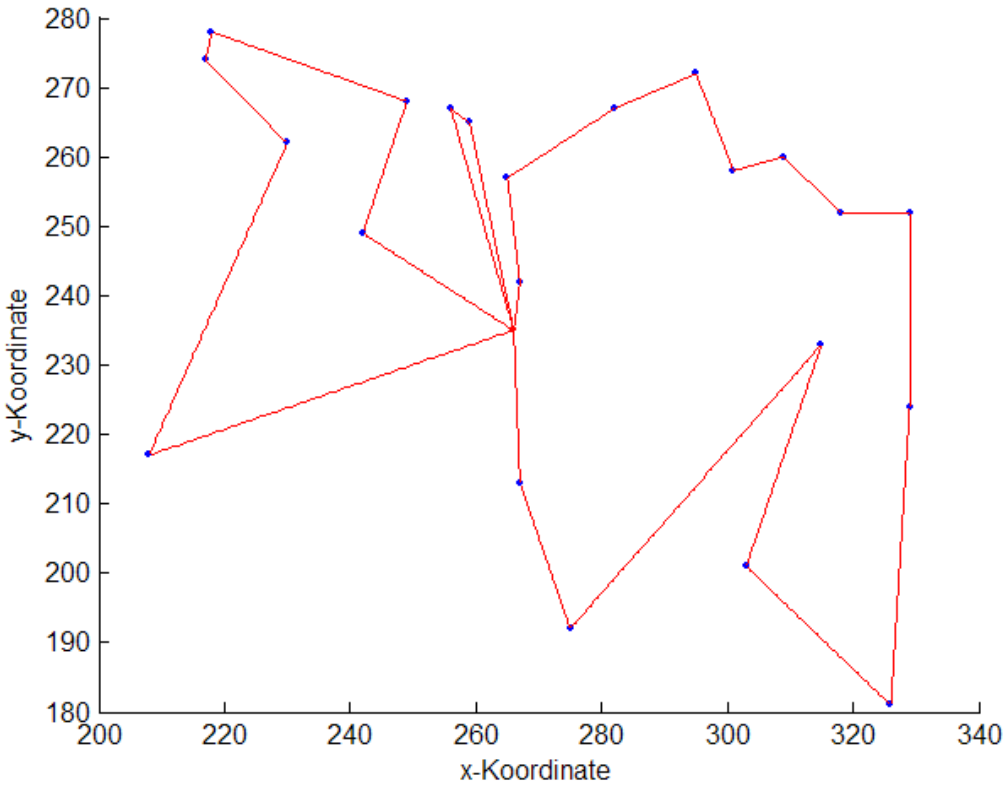


Abbildung A.9: Testinstanz eil23

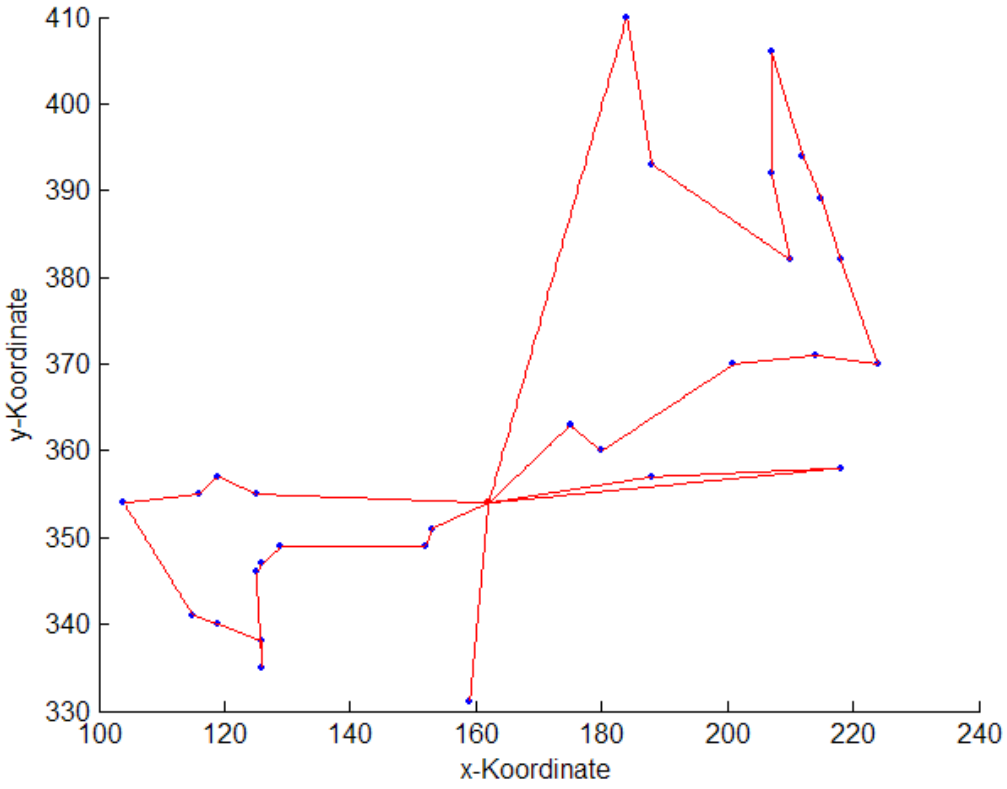


Abbildung A.10: Testinstanz eil30

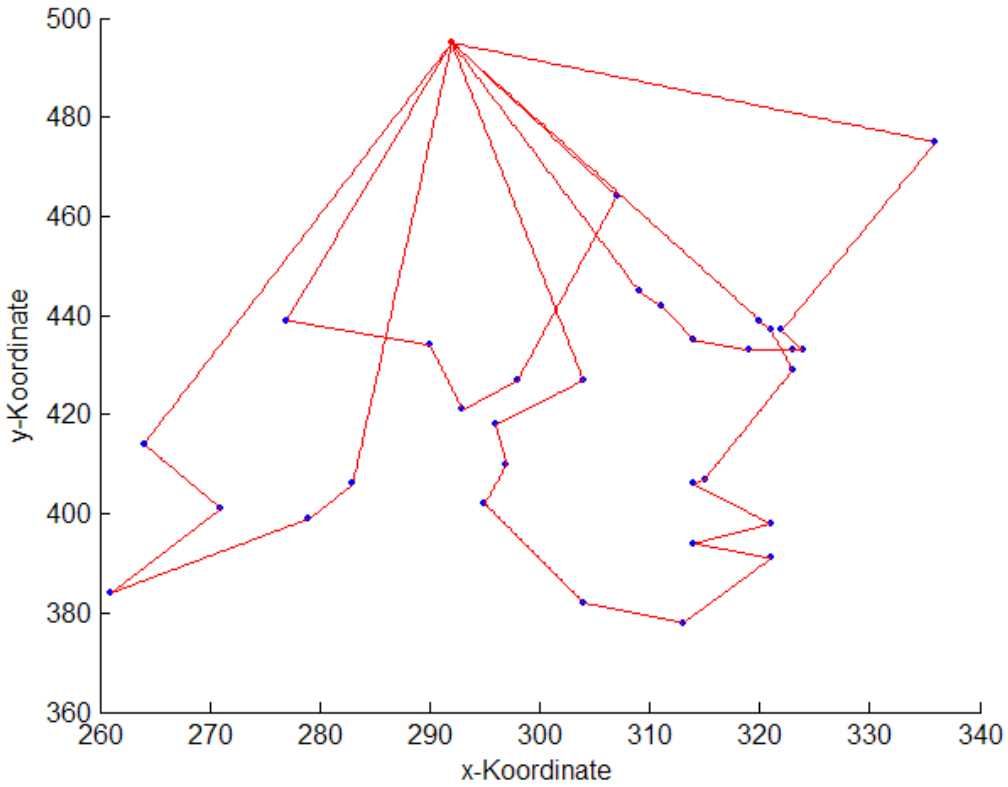


Abbildung A.11: Testinstanz eil33

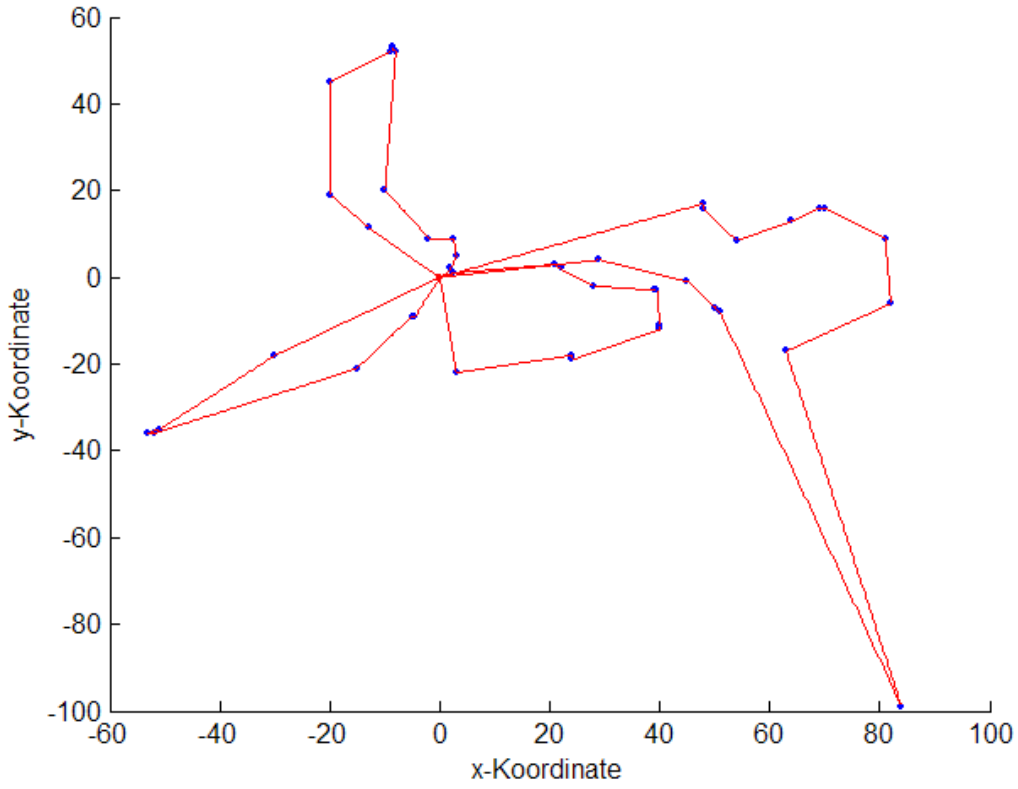


Abbildung A.12: Testinstanz F1

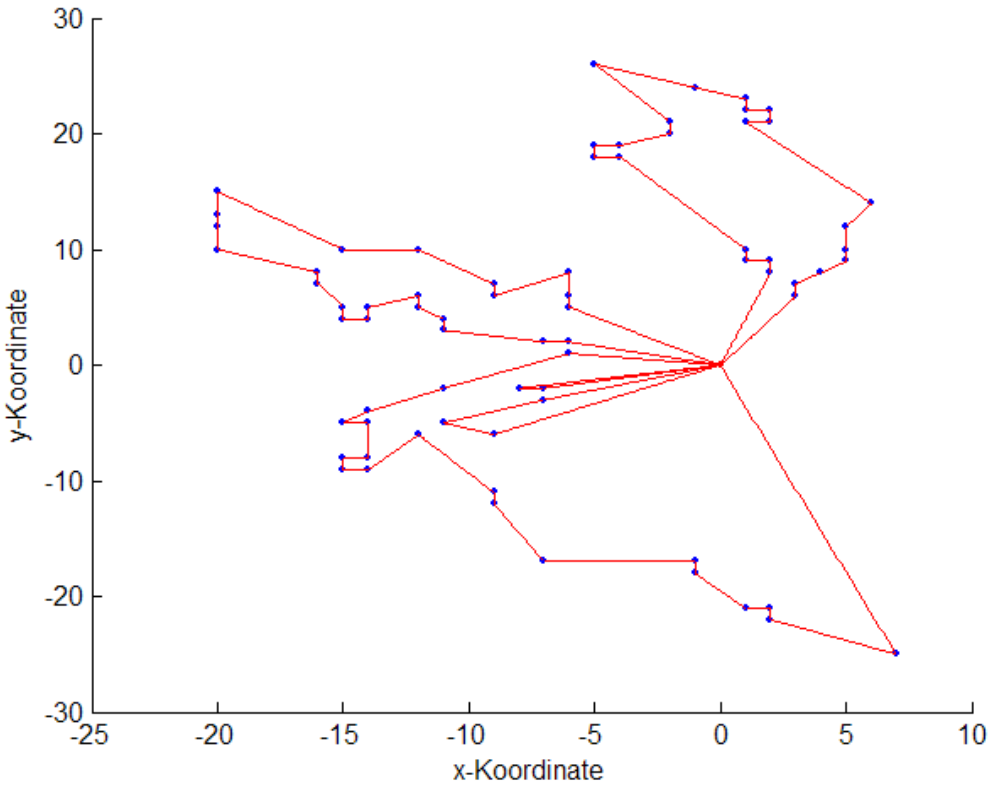


Abbildung A.13: Testinstanz F2

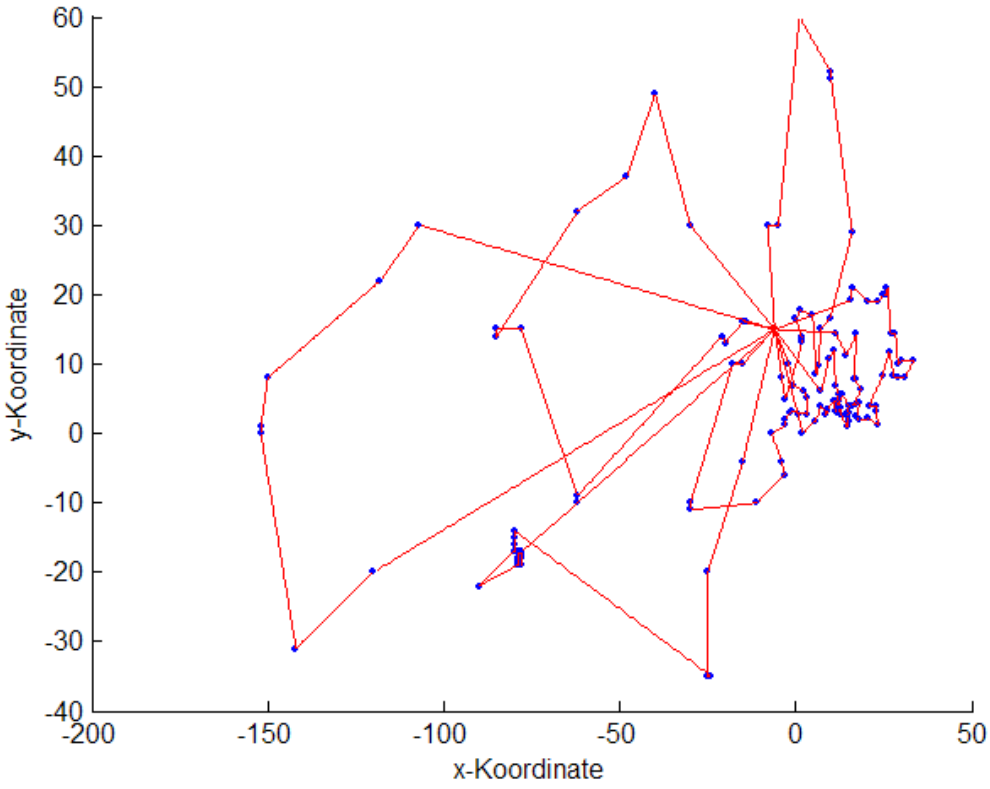


Abbildung A.14: Testinstanz F3

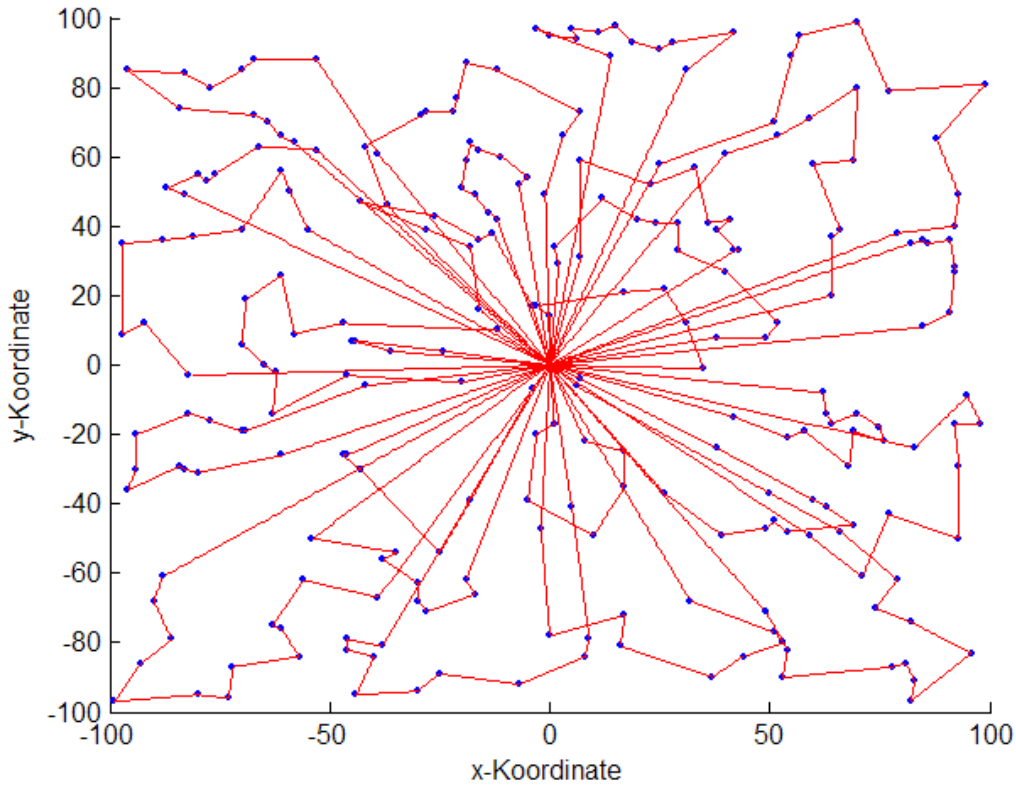


Abbildung A.15: Testinstanz GJ1

C. Parameter der Testläufe.

Instanz	Auslastungsgewicht	Tabu-Zähler	$\sigma_{t_p=0}$	Präsentationszyklen
C1	15	2	7	400
C2	75	3	5	400
C3	15	2	5	400
C4	15	2	6	400
C6	120	1	7	400
C8	30	1	7	400
C10	25	1	8	400
eil22	45	1	4	400
eil23	45	1	4	400
eil30	200	2	6	400
eil33	50	1	4	400
F1	40	1	7	400
F2	25	1	7	400
F3	200	2	10	400
GJ1	250	1	6	400

Tabelle A.2: Parameter der Testläufe

D. Programmcodes.

```
%Call-Skript LRPSOM(W/D)=========================================================================================================
clear; %Löschen aller Variablen aus dem Workspace
for counterRun=1:100 %Anzahl Durchläufe festlegen
   tic; %Stoppuhr starten
   schaltermovie=0; %Film aufnehmen? 0:Nein 1:Ja
   schalterfiguren=0; %Figuren zeichnen? 0:Nein 1:Ja
   AnzIter=400; %Anzahl Präsentationszyklen festlegen
   sigmastart=10; %Nachbarschaftsweite festlegen
   LPbstart=1; %Lernparameter festlegen
   phistart=0;%Ausgangswinkel für Drehung der Ellipsen bestimmen
   Auslastungsgewicht=200; %Auslastungsgewichtung
   Tabulength=2; %Dauer des Verbotes einer Zuordnung eines Bedarfortes zu einer Tour
   AnzNeur=537; %Anzahl Neuronen festlegen
   AnzTouren=7; %Anzahl der Fahrzeuge festlegen
   FZKap=2210; %Berechnen der Fahrzeugkapazität
   OrgSta=[-6 15]; %Original-Standort Depot

[Gesamtlaen-
ge,w1,w2,UsedCap,Auslastungsgrad,ProzentKapazitaetsueberschuss,EDStandort,WeberX,WeberY,WeberDist,VWNN,GesamtlaengeW,GesamtlaengeOrgStandort,x,w]=lrperm
(AnzTouren,Auslastungsgewicht,Tabulength,FZKap,AnzNeur,AnzIter,sigmastart,schaltermovie,schalterfiguren,OrgSta,LPbstart,phistart); %Aufruf der Funktion LRP

   Auswertung(counterRun,19)=toc; %Rechenzeit speichern
   Auswertung(counterRun,1)=Gesamtlaenge; %Gesamtlänge nach NN
   Auswertung(counterRun,2)=w1; %Standortkoordinate X nach NN
   Auswertung(counterRun,3)=w2; %Standortkoordinate Y nach NN
   Auswertung(counterRun,4)=ProzentKapazitaetsueberschuss; %Kapazitätsüberschuss der Gesamtfahrzeugkapazität
   Auswertung(counterRun,5)=EDStandort; %Entfernung vom NN-Standort zu Randkunden
   Auswertung(counterRun,6)=WeberX; %Standortkoordinate X nach Weber
   Auswertung(counterRun,7)=WeberY; %Standortkoordinate Y nach Weber
   Auswertung(counterRun,8)=WeberDist; %Distanz Weber-Punkt zu Randkunden
   Auswertung(counterRun,9)=VWNN; %Verhältnis der Entfernungen Weber/NN vom Standort zu Randkunden der Touren
   for counterAnzTouren=1:AnzTouren %Kapazitätverbrauch der FZ und Auslastungsgrad dokumentieren
      KapVerbrauchVerlauf(counterRun,counterAnzTouren)=UsedCap(counterAnzTouren);
      AuslastungsgradVerlauf(counterRun,counterAnzTouren)=Auslastungsgrad(counterAnzTouren);
   end
   %[Gesamtdistanz]=Savings(WeberX,WeberY,FZKap,OrgSta);%Aufruf der Funktion Savings, optional; das Skript wird im Anhang nicht zur Verfügung gestellt.
   %Auswertung(counterRun,10)=Gesamtdistanz; %Savings-Distanz
   Auswertung(counterRun,11)=GesamtlaengeW; %Distanz NN mit Weber-Standort
   %Auswertung(counterRun,12)=Gesamtdistanz/Gesamtlaenge; %Verhältnis zwischen Savings- und NN-Lösung
   %Auswertung(counterRun,13)=Gesamtdistanz/GesamtlaengeW; %Verhältnis zwischen Savings- und NNWeber-Lösung
   Auswertung(counterRun,14)=Gesamtlaenge/GesamtlaengeW; %Verhältnis zwischen NN- und NNWeber-Lösung
   Auswertung(counterRun,15)=GesamtlaengeOrgStandort; %Distanz mit Original-Standort
   Auswertung(counterRun,16)=sigmastart; %Parameter der Nachbarschaftsfunktion speichern
   Auswertung(counterRun,17)=Tabulength; %Tabu-Counter speichern
   Auswertung(counterRun,18)=Auslastungsgewicht; %Auslastungsgewicht speichern
   Auswertung(counterRun,20)=phistart; %Winkel speichern
   Touren1(:,counterRun)=x(:,4); %Tourenzuordnung speichern
   Touren2(:,counterRun)=x(:,5); %Neuronenzuordnung speichern
end
%Ende Call-Skript LRPSOM(W/D)=====================================================================================================

% LRPSOM(W/D)-Function=============================================================================================================
function [Ge-
samtlaenge,w1,w2,UsedCap,Auslastungsgrad,ProzentKapazitaetsueberschuss,EDStandort,WeberX,WeberY,WeberDist,VWNN,GesamtlaengeW,GesamtlaengeOrgStandort,x,w
]=lrperm(AnzTouren,Auslastungsgewicht,Tabulength,FZKap,AnzNeur,AnzIter,sigmastart,schaltermovie,schalterfiguren,OrgSta,LPbstart,phistart)
%Daten für Testläufe
%====================
testlauf=0;
if testlauf==1
  AnzTouren=7
  FZKap=2210
  OrgSta=[-6 15]
  AnzNeur=537
  AnzIter=400
  Auslastungsgewicht=50
  Tabulength=1
  LPbstart=1
  schalterfiguren=1
  schaltermovie=1
  sigmastart=8
  phistart=0
  Iter=0; %Setze den Zähler Iteration auf den Wert Null
  IterPattern=0; %Zähler auf Null setzen
  Gesamtdistanz=0; %Wert der Gesamtdistanz auf Null setzen
end
Iter=0; %Setze den Zähler Iteration auf den Wert Null
IterPattern=0; %Zähler auf Null setzen
Gesamtdistanz=0; %Wert der Gesamtdistanz auf Null setzen
DistFolgeort=0;
OrgSt=OrgSta;
BestOrgSta=100000;
%==========Daten einlesen ========================================================================
%[x] = xlsread('C1.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%[x] = xlsread('C2.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%[x] = xlsread('C3.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%[x] = xlsread('C4.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%[x] = xlsread('C6.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%[x] = xlsread('C8.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%[x] = xlsread('C10.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%[x] = xlsread('F1.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%[x] = xlsread('F2.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
[x] = xlsread('F3.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%[x] = xlsread('GJ1.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%[x] = xlsread('eil22.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%[x] = xlsread('eil23.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%[x] = xlsread('eil30.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%[x] = xlsread('eil33.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen

%===== Daten verarbeiten ========
Capacity=zeros(1,AnzTouren);
```

```
Capacity(1,:)=FZKap; %Kapazitätsvektor der Touren (Position 1= Kapazität Fahrzeug der Tour 1, Position 2= Kapazität des FZ der Tour 2....)
UsedCap=zeros(1,AnzTouren); %Vektor zur Erfassung der benutzten Kapazität eines Fahrzeuges definieren
b=ones(1,AnzTouren); %Bias der Neuronen wird jeweils auf 1 festgelegt
w=zeros(AnzNeur,2); %Vektor der Verbindungsgewichte für die Neuronen der Ausgabeschicht (Input: 2, Kohonen Map: AnzNeur Neuronen)
[clusters, dimension]=size(w); %Dimension der Matrix auslesen
[inputs, patterns]= size(x'); %Dimension von x ausgelesen
MSEW1=zeros(AnzIter,1); %Vektor zur Erfassung des MSE nach jeder vollen Iteration definieren
%== Figuren definieren ============
if schalterfiguren==1;
  figure(1);
  figure(2);
  figure(3);
  figure(4);
  figure(5);
  %Hilfsmittel zur graph. Ausgabe einrichten
  Tourpkt=1:100; %dient zur Zeichnung der Neuronenverbindung, Matrix mit den Eintragungen 1 bis 100
  Tourpkt=0.01*Tourpkt; %Skalierung auf Werte zwischen 0,01 und 1
end
%==== Initialisierung des Gewichtes des Standort-Neurons und Initialisieren der Parameter zur Gewichtsbestimmung ==============
xmax=max(x); %Bestimmen der Maximalwerte der Koordinaten
xmin=min(x); %Bestimmen der Minimalwerte der Koordinaten
xcenter(1)=(xmax(1)+xmin(1))/2; %Bestimmen des Schwerpunktes der Koordinaten
xcenter(2)=(xmax(2)+xmin(2))/2;
alpha1=0; %Bestimmen des Ausgangswinkels alpha
phi=phistart; %Bestimmen des Ausgangswinkels phi
alpha2=0;%Bestimmen des Ausgangswinkels für die Mittelpunkte der Ellipsen
Winkelschritt=(2*pi)/(round(((AnzNeur-1)/AnzTouren))); %Bestimmen der Winkelschritte zwischen den einzelnen Touren
Winkelschritt2=2*pi/AnzTouren; %Bestimmen der Winkelschritte zwischen den einzelnen Touren
Winkelschrittphi=2*pi/AnzTouren; %Bestimmen der Winkelschritte zwischen Neuronen
w(1,1)=xcenter(1); %der Standort wird im Zentrum positioniert
w(1,2)=xcenter(2) ; %s.o.
w1(1,1)=xcenter(1); %Positionierung des Standortes im Schwerpunkt
w1(1,2)=xcenter(2) ; %s.o.
%Initialisierung der Gewichtsneuronen entlang von Ellipsen / Festlegung der Gewichtsvektoren durch Koordinatentransformationen
for counterIni=1:AnzTouren
   WiniUG=2+((counterIni-1)*round((AnzNeur-1)/AnzTouren)); %Untergrenze der Neuronen einer Tour ermitteln
   WiniOG=1+(counterIni)*round((AnzNeur-1)/AnzTouren); %Obergrenze der Neuronen der Tour ermitteln
   if WiniOG>AnzNeur
      WiniOG=AnzNeur; %Obergrenze ist maximal die Anzahl der Neuronen
   end
   phi=phi+Winkelschrittphi; %Erhöhen des Winkels phi
   for counterWini=WiniUG:WiniOG
      w1(counterWini,1)=(xcenter(1)+.5*cos(alpha1)-1); %die x-Koordinate wird entlang einer Ellipse angeordnet
      w1(counterWini,2)=(xcenter(2)+.1*sin(alpha1)); %die y-Koordinate wird entlang einer Ellipse angeordnet
      %Drehung um den Winkel Phi
      w(counterWini,1)=xcenter(1)+10*(w1(counterWini,1)-xcenter(1))*cos(phi)-(w1(counterWini,2)-xcenter(2))*sin(phi); %die x-Koordinate
      w(counterWini,2)=xcenter(2)+10*(w1(counterWini,1)-xcenter(1))*sin(phi)+(w1(counterWini,2)-xcenter(2))*cos(phi); %die y-Koordinate wird verschoben
      alpha1=alpha1+Winkelschritt; %Anpassen des Winkels
   end
   %Markieren der Neuronen
   if schalterfiguren==1;
      figure(2);
      subplot(2,2,1);
      hold on;
      plot(w(WiniUG,1),w(WiniUG,2),'r.','markersize',6);
      plot(w(WiniOG,1),w(WiniOG,2),'r.','markersize',6);
   end
   alpha1=0; %Rücksetzen des Winkelschrittes auf 1
end
showw=w; %Optional: Anzeige der Gewichtsinitialisierung
% == Ausgabe ======================
if schalterfiguren==1;
  plot(w(:,1),w(:,2),'r-');
  plot(x(:,1),x(:,2),'.b','markersize',12);
  plot(w(:,1),w(:,2),'.g','markersize',10);
  plot(w(1,1),w(1,2),'.r','markersize',12);
  xlabel('x-Koordinate');
  ylabel('y-Koordinate');
  %legend('Route','Bedarfsorte','Neuronen','Standort'); %Optional:Legendenbeschriftung der Figur
  title('Initialisierungsanordnung der Neuronen');
  %Ausgabe der Gauss-Glocke zum Zeitpunkt t=0
  r=zeros(100,2);
  gaussbild1=-2;
  for counter10=1:100;
      r(counter10,1)=gaussbild1+1;
      %r(counter10,2)=(1-(r(counter10,1)/52))^11;
      %r(counter10,2)=1*a^((r(counter10,1)^2/(2*(sigmastart^2))));
      r(counter10,2)=LPbstart*2.71^(-(r(counter10,1)^2/(2*(sigmastart^2))));
      gaussbild1=gaussbild1+1;
  end
  figure(1)
  subplot(2,2,1);
  axis ([0 50 0 1]);
  hold on;
  plot(r(:,1),r(:,2),'r-');
  xlabel('\Delta_{cj}');
  ylabel('h_{cj}(t)');
   title('Nachbarschaftsfunktion t=0');
end
%========================== Festlegen der Neuronenzugehörigkeiten zu den Touren ================================================
TourNeuronen=zeros(1,AnzNeur); %Vektor mit Nullen auffüllen
TourNeuronen(1,1)=AnzTouren+1; %Erste Eintragung im Vektor ist das Standortneuron, dem eine eigene Tour zugewiesen wird
for counter20=1:AnzTouren %für die Anzahl der Touren
   if (1+(counter20)*round((AnzNeur-1)/AnzTouren))>AnzNeur %wenn die Summe (1+TourNr*Runde((AnzNeur-1)/AnzTouren)) größer als die Anzahl der Neuronen ist
      Grenze(counter20)=AnzNeur; %Grenze ist Anzahl der Neuronen, d.h. der letzten Tour werden u.U. weniger Orte zugeteilt
   else
      Grenze(counter20)=(1+(counter20)*round((AnzNeur-1)/AnzTouren));
   end
   for counter21=((counter20-1)*round((AnzNeur-1)/AnzTouren)+2):Grenze(counter20) %von Anfang der Tour in der Neuronenkette bis zur Grenze:
      TourNeuronen(1,counter21)=counter20; %Zuweisung der Tourennr. zu den einzelnen Neuronen
   end
end
NZNeuronenTour=find(TourNeuronen==0); %Finde nicht zugeordnete Neuronen (befinden sich nur am Ende des Vektors)
```

```
TourNeuronen(NZNeuronenTour)=AnzTouren; %weise die verbleibenden nicht zugeordneten Neuronen der letzten Tour zu
showTourNeuronen=TourNeuronen; %Ausgabe der Matrix
%Film erstellen
if schaltermovie==1
   mov = avifile('Ausbreitung.avi','fps',60); %Film erstellen
   mov.Quality=100; %maximale Qualität einstellen
end
%=============================================================Beginn Ausgabe=============================================
for counter4=1:(AnzIter); %Anzahl Präsentationszyklen definieren
   Ausgabe=Iter/AnzIter; %Wert der Ausgabe ermitteln
   if Ausgabe<0.5
      Auslgewicht=Auslastungsgewicht;
   else
      Auslgewicht=0;
   end
   sigma=sigmastart*((.1/sigmastart)^(Iter/AnzIter)); %Festlegen des aktuellen Wertes der "Breite" der Gauss-Funktion
   LPb=LPbstart-LPbstart*(Iter/AnzIter); %Maximalwert der Gauss-Funktion in Abh. vom Iterationsfortschritt festlegen
   if LPb<0.0001
      LPb=0.0001;
   end
  if schaltermovie==1
      figure(5); %Wahl der Figur 5
      cla; %Löschen aller Einträge in Figur 5
      hold on; %Halten der eingezeichneten Objekte
      for counter1=2:(AnzNeur-1); %von Neuron 1 bis AnzNeur-1
         if TourNeuronen(1,counter1)==TourNeuronen(1,counter1+1) %wenn zwei aufeinanderfolgende Neuronen zu einer Tour gehören:
            for counter18=1:100 %Ermitteln von 100 Punkten auf der Verbindungsstrecke zwischen direkt benachbarten Neuronen
               xkoord(counter18)=((1-Tourpkt(counter18))*w(counter1,1)+Tourpkt(counter18)*w((counter1+1),1)); %Speichern der 100 x-Koordinaten
               ykoord(counter18)=((1-Tourpkt(counter18))*w(counter1,2)+Tourpkt(counter18)*w((counter1+1),2)); %                y-Koordinaten
            end
               plot(xkoord,ykoord,'r-');%Einzeichnen der Verbindungsstrecke zwischen den benachbarten Neuronen
         else
            for counter18=1:100 %Ermitteln von 100 Punkten auf der Verbindungsstrecke zwischen Standort und Neuron counter+1
               xkoord(counter18)=((1-Tourpkt(counter18))*w(1,1)+Tourpkt(counter18)*w((counter1+1),1)); %Speichern der 100 Koordinaten der Linearkombinationen
               ykoord(counter18)=((1-Tourpkt(counter18))*w(1,2)+Tourpkt(counter18)*w((counter1+1),2));
            end
            plot(xkoord,ykoord,'r-');%Einzeichnen der Verbindungsstrecke zwischen Standort und Neuron counter
            if counter1~=1
               for counter18=1:100 %Ermitteln von 100 Punkten auf der Verbindungsstrecke zwischen Standort und Neuron counter
                  xkoord(counter18)=((1-Tourpkt(counter18))*w(1,1)+Tourpkt(counter18)*w((counter1),1)); %Speichern der 100 Koordinaten der Linearkombinationen
                  ykoord(counter18)=((1-Tourpkt(counter18))*w(1,2)+Tourpkt(counter18)*w((counter1),2));
               end
               plot(xkoord,ykoord,'r-'); %Einzeichnen der Verbindungsstrecke zwischen Standort und Neuron Counter
            end
         end
      end
      for counter19=1:100; %Verbinden des ersten Neurons der ersten Tour mit dem Standort
         xkoord(counter19)=((1-Tourpkt(counter19))*w(1,1)+Tourpkt(counter19)*w(2,1));
         ykoord(counter19)=((1-Tourpkt(counter19))*w(1,2)+Tourpkt(counter19)*w(2,2));
   end
   plot(xkoord,ykoord,'k-');
   for counter30=1:100; %Verbinden des letzten Neurons der letzten Tour mit dem Standort
         xkoord(counter30)=((1-Tourpkt(counter30))*w(1,1)+Tourpkt(counter30)*w(AnzNeur,1));
         ykoord(counter30)=((1-Tourpkt(counter30))*w(1,2)+Tourpkt(counter30)*w(AnzNeur,2));
   end
   plot(xkoord,ykoord,'k-');
   plot(x(:,1),x(:,2),'.b','markersize',15); %Einzeichnen der Orte
   plot(w(:,1),w(:,2),'.g','markersize',10); %Einzeichnen der Neuronengewichte
   plot(w(1,1),w(1,2),'.r','markersize',15); %Einzeichnen der Neuronengewichte des Standortneurons
   xlabel('x-Koordinate');
   ylabel('y-Koordinate');
   title('Momentane Lösung');
   F = getframe; %Frame für den Film grabben
   mov = addframe(mov,F); %Frame hinzufügen
 end
 if schalterfiguren==1;
      switch Ausgabe %mit Hilfe der Switch-Anweisung werden Zwischenergebnisse ausgegeben
         case(1/4)
            figure(2); %Wahl der Figur Loesung
            subplot(2,2,2); %Wahl des Subplots 2 in der Figur
            hold on; %Halten der eingezeichneten Objekte
            for counter1=2:(AnzNeur-1); %von Neuron 1 bis AnzNeur-1
               if TourNeuronen(1,counter1)==TourNeuronen(1,counter1+1) %wenn zwei aufeinanderfolgende Neuronen zu einer Tour gehören:
                  for counter18=1:100 %Ermitteln von 100 Punkten auf der Verbindungsstrecke zwischen direkt benachbarten Neuronen
                     xkoord(counter18)=((1-Tourpkt(counter18))*w(counter1,1)+Tourpkt(counter18)*w((counter1+1),1)); %Speichern der 100 x-Koordinaten
                     ykoord(counter18)=((1-Tourpkt(counter18))*w(counter1,2)+Tourpkt(counter18)*w((counter1+1),2)); %                y-Koordinaten
                  end
                     plot(xkoord,ykoord,'r-'); %Einzeichnen der Verbindungsstrecke zwischen den benachbarten Neuronen
               else
                  for counter18=1:100 %Ermitteln von 100 Punkten auf der Verbindungsstrecke zwischen Standort und Neuron counter+1
                     xkoord(counter18)=((1-Tourpkt(counter18))*w(1,1)+Tourpkt(counter18)*w((counter1+1),1)); %Speichern der 100 Koordinaten der Linearkombinationen
                     ykoord(counter18)=((1-Tourpkt(counter18))*w(1,2)+Tourpkt(counter18)*w((counter1+1),2));
                  end
                  plot(xkoord,ykoord,'r-'); %Einzeichnen der Verbindungsstrecke zwischen Standort und Neuron counter
                  if counter1~=1
                     for counter18=1:100 %Ermitteln von 100 Punkten auf der Verbindungsstrecke zwischen Standort und Neuron Counter
                        xkoord(counter18)=((1-Tourpkt(counter18))*w(1,1)+Tourpkt(counter18)*w((counter1),1)); %Speichern der 100 Koordinaten der Linearkombinationen
                        ykoord(counter18)=((1-Tourpkt(counter18))*w(1,2)+Tourpkt(counter18)*w((counter1),2));
                     end
                     plot(xkoord,ykoord,'r-'); %Einzeichnen der Verbindungsstrecke zwischen Standort und Neuron Counter
                  end
               end
            end
            %
            for counter19=1:100; %Verbinden des ersten Neurons der ersten Tour mit dem Standort
               xkoord(counter19)=((1-Tourpkt(counter19))*w(1,1)+Tourpkt(counter19)*w(2,1));
               ykoord(counter19)=((1-Tourpkt(counter19))*w(1,2)+Tourpkt(counter19)*w(2,2));
            end
            plot(xkoord,ykoord,'k-');

            for counter30=1:100;
               xkoord(counter30)=((1-Tourpkt(counter30))*w(1,1)+Tourpkt(counter30)*w(AnzNeur,1));
               ykoord(counter30)=((1-Tourpkt(counter30))*w(1,2)+Tourpkt(counter30)*w(AnzNeur,2));
```

```
        end
        plot(xkoord,ykoord,'k-');
        plot(x(:,1),x(:,2),'.b','markersize',15); %Einzeichnen der Orte
        plot(w(:,1),w(:,2),'.g','markersize',10); %Einzeichnen der Neuronengewichte
        plot(w(1,1),w(1,2),'.r','markersize',15); %Einzeichnen der Neuronengewichte des Standortneurons
        xlabel('x-Koordinate');
        ylabel('y-Koordinate');
        title('Lösung t=1/4 t_{max}');
        r=zeros(52,2);
        gaussbild1=-2;
        for counter10=1:52;
          r(counter10,1)=gaussbild1+1;
          r(counter10,2)=LPb*2.71^(-(r(counter10,1)^2/(2*(sigma^2))));
          gaussbild1=gaussbild1+1;
        end;
        figure(1)
        subplot (2,2,2)
        axis ([0 50 0 1]);
        hold on;
        plot(r(:,1),r(:,2),'r-');
       xlabel('\Delta_{cj}');
       ylabel('h_{cj}(t)');
       title('Nachbarschaftsfunktion t= 1/4 t_{max}');
    case(.5)
        figure(2); %Wahl der Figur Loesung
        subplot(2,2,3); %Wahl des Subplots 3 in der Figur
        hold on; %Halten der eingezeichneten Objekte
        for counter1=2:(AnzNeur-1); %von Neuron 1 bis AnzNeur-1
          if TourNeuronen(1,counter1)==TourNeuronen(1,counter1+1) %wenn zwei aufeinanderfolgende Neuronen zu einer Tour gehören:
            for counter18=1:100 %Ermitteln von 100 Punkten auf der Verbindungsstrecke zwischen direkt benachbarten Neuronen
              xkoord(counter18)=((1-Tourpkt(counter18))*w(counter1,1)+Tourpkt(counter18)*w((counter1+1),1)); %Speichern der 100 x-Koordinaten
              ykoord(counter18)=((1-Tourpkt(counter18))*w(counter1,2)+Tourpkt(counter18)*w((counter1+1),2)); %                y-Koordinaten
            end
              plot(xkoord,ykoord,'r-'); %Einzeichnen der Verbindungsstrecke zwischen den benachbarten Neuronen
          else
            for counter18=1:100 %Ermitteln von 100 Punkten auf der Verbindungsstrecke zwischen Standort und Neuron counter+1
              xkoord(counter18)=((1-Tourpkt(counter18))*w(1,1)+Tourpkt(counter18)*w((counter1+1),1)); %Speichern der 100 x-Koordinaten der Linearkombinationen
              ykoord(counter18)=((1-Tourpkt(counter18))*w(1,2)+Tourpkt(counter18)*w((counter1+1),2));
            end
            plot(xkoord,ykoord,'r-'); %Einzeichnen der Verbindungsstrecke zwischen Standort und Neuron counter
            if counter1~=1
              for counter18=1:100 %Ermitteln von 100 Punkten auf der Verbindungsstrecke zwischen Standort und Neuron Counter
                xkoord(counter18)=((1-Tourpkt(counter18))*w(1,1)+Tourpkt(counter18)*w((counter1),1)); %Speichern der 100 Koordinaten der Linearkombinationen
                ykoord(counter18)=((1-Tourpkt(counter18))*w(1,2)+Tourpkt(counter18)*w((counter1),2));
              end
              plot(xkoord,ykoord,'r-'); %Einzeichnen der Verbindungsstrecke zwischen Standort und Neuron Counter
            end
          end
        end
        for counter19=1:100; %Verbinden des ersten Neurons der ersten Tour und dem Standort
          xkoord(counter19)=((1-Tourpkt(counter19))*w(1,1)+Tourpkt(counter19)*w(2,1));
          ykoord(counter19)=((1-Tourpkt(counter19))*w(1,2)+Tourpkt(counter19)*w(2,2));
        end
        plot(xkoord,ykoord,'k-');
        for counter30=1:100; %Verbinden des letzten  Neurons der letzten Tour und dem Standort
          xkoord(counter30)=((1-Tourpkt(counter30))*w(1,1)+Tourpkt(counter30)*w(AnzNeur,1));
          ykoord(counter30)=((1-Tourpkt(counter30))*w(1,2)+Tourpkt(counter30)*w(AnzNeur,2));
        end
        plot(xkoord,ykoord,'k-');
        plot(x(:,1),x(:,2),'.b','markersize',15); %Einzeichnen der Orte
        plot(w(:,1),w(:,2),'.g','markersize',10); %Einzeichnen der Neuronengewichte
        plot(w(1,1),w(1,2),'.r','markersize',15); %Einzeichnen der Neuronengewichte des Standortneurons
        xlabel('x-Koordinate');
        ylabel('y-Koordinate');
        title('Lösung t=1/2 t_{max}');
        r=zeros(52,2);
        gaussbild1=-2;
        for counter10=1:52;
          r(counter10,1)=gaussbild1+1;
          r(counter10,2)=LPb*2.71^(-(r(counter10,1)^2/(2*(sigma^2))));
          gaussbild1=gaussbild1+1;
        end;
        figure(1)
        subplot (2,2,3)
        axis ([0 50 0 1]);
        hold on;
        plot(r(:,1),r(:,2),'r-');
        title('Nachbarschaftsfunktion, t=1/2 t_{max}');
        xlabel('\Delta_{cj}');
        ylabel('h_{cj}(t)');
    end %Ende der Switch-Anweisung
  end %Ende der Ausgabe-Abfrage
%==============Beginn Iterationen=============================================================================================
Perm=randperm(patterns); %Zufällige Reihenfolge für Eingabevektoren in den Präsentationszyklen ermitteln
Zaehle=0;
for counter5=1:patterns; %Präsentation der Eingabedaten in einem Präsentationszyklus
  y=zeros(size(b)); %die Elemente des Vektors y werden auf Null gesetzt
  ChosenKoord(1,1)=x(Perm(counter5),1); %hier wir die x-Koordinate des Bedarfsortes gewählt
  ChosenKoord(1,2)=x(Perm(counter5),2); %hier wird die y-Koordinate des Bedarfsortes gewählt
  if x(Perm(counter5),4)~=0 %wenn das Ortsneuron bereits einer Tour zugewiesen ist, dann
    Zaehle=Zaehle+1;
    usCapalt2=UsedCap;
    UsedCap(1,x(Perm(counter5),4))=UsedCap(1,x(Perm(counter5),4))-x(Perm(counter5),3); %verringere die Auslastung der FZ-Kapazität um den Bedarf
    x(Perm(counter5),4)=0; %und hebe die Zuordnung auf
    x(Perm(counter5),5)=0; %und setze den Index des zugeordneten Neurons auf 0
  end
  UsedCaptotal=0; %Variable benutzter Kapazität auf Null setzen
  for countertotcap=1:AnzTouren
    UsedCaptotal=UsedCaptotal+UsedCap(1,countertotcap); %Berechnung der verwendeten Kapazität aller Touren
  end
  for counterfraction=1:AnzTouren
    %b(counterfraction)=UsedCap(1,counterfraction)/(UsedCaptotal+.0001); %Optional: Berechnung der Anteile der Fahrzeuge an der verwendeten Gesamtkapazität
    b(counterfraction)=UsedCap(1,counterfraction)/(FZKap); %
```

```
   %b(counterfraction)=1-((UsedCap(1,counterfraction)-((UsedCaptotal+.0001)/AnzTouren))/((UsedCaptotal+.0001)/AnzTouren));%Optional
end
for counter6=2:clusters; %für jedes Neuron (außer Standort)
                dist(counter6)=((ChosenKoord(1,1)-w(counter6,1))^2+(ChosenKoord(1,2)-w(counter6,2))^2)^.5;% wird die euklidische Distanz berechnet
    %if b(1,TourNeuronen(1,counter6))>1 %Optionale Version: dynamischer Exponent
      % exponent=b(1,TourNeuronen(1,counter6))*10;
    %else
      % exponent=0;
    %end
   dist(counter6)=dist(counter6)+Auslgewicht*b(1,TourNeuronen(1,counter6)); %Erhöhung der Distanz um den gewichteten Auslastungsgrad
   if x(Perm(counter5),6)~=0&TourNeuronen(counter6)==x(Perm(counter5),6) %wenn der Ort aufgrund einer Kapazitätsbeschränkung zur Tabu-Liste gehört
     dist(counter6)=dist(counter6)+10000; %erhöhe die Distanz um einen großen Wert, sodaß die Tourneuronen nicht gewinnen können.
   end
end
if x(Perm(counter5),6)~=0&x(Perm(counter5),7)==0 %wenn der Ort präsentiert wurde, dann hebe das Tourenzuordnungsverbot auf, wenn es besteht und die Anzahl der
                                                  %Tabu-Iterationen ereicht wurde
  x(Perm(counter5),6)=0;
elseif x(Perm(counter5),6)~=0&x(Perm(counter5),7)>0 %wenn ein Tourenzuordnungsverbot besteht und die Anzahl an Tabu-Iterationen noch nicht erreicht wurde, dann
  x(Perm(counter5),7)=x(Perm(counter5),7)-1; %Vermindere den Zähler der Tabu-Iterationen um 1
end
dist(1)=10000000; %setzen der Distanz des Standortes auf eine große Zahl M=10000
indwinner=find(dist==min(dist)); %Ermittlung des Index des Gewinner-Neurons
ind=indwinner(1);              %bei identischer Distanz verschiedener Neuronen wird das erste Neuron in der Liste gewählt
UsedCap(1,TourNeuronen(1,ind))=UsedCap(1,TourNeuronen(1,ind))+x(Perm(counter5),3); %Erhöhung der Tourenauslastung der Tour des Gewinnerneurons
x(Perm(counter5),4)=TourNeuronen(1,ind); %Zuordnung des präsentierten Ortes zum Gewinner-Fahrzeug
x(Perm(counter5),5)=ind; %ordne den Index des Gewinner-Neurons zu
%========================= Anpassung der Gewichtsvektoren ==========================================================================
%====Nachbarschaftsbestimmung der Tour des Gewinners (Ringanordnung der Tourneuronen) ====================
 Nachbarn=find(TourNeuronen==TourNeuronen(1,ind)); %Bestimmen der Tourteilnehmer des Gewinners (ohne Standort)
 Tour=zeros(length(Nachbarn')+1,1); %Hinzufügen eines Eintrages des Standortes zur Tour des Gewinners
 Tour(1)=1; %Der Standort wird in die ersten Zelle geschrieben
 for counter21=1:length(Nachbarn') %von 1 bis Anzahl Nachbarn
    Tour(counter21+1)=Nachbarn(counter21); %Schreibe die Neuronennr. in den Vektor Tour
 end
 %Tour=Nachbarn; %Optional: nur bei Ausschluss des Standortes zu verwenden
 showtour=Tour; %Anzeigen der Tourteilnehmer
 TourPos=find(Tour==ind); %Finden der Position des Gewinners in der Tour
 for counter12=1:length(Tour); %für die Anzahl der Neuronen der Nachbarschaft
      if TourPos>counter12 %wenn die Position des Gewinners im Ring größer als die Position des betrachteten Neurons ist
        Entfernung1=(TourPos-counter12); %berechne die Entfernung zwischen Gewinner und betr. Neuron im math. negativen Drehsinn
        Entfernung2=(counter12+(length(Tour)-TourPos)); %und im math. positiven Drehsinn
      else
        Entfernung1=abs(TourPos-counter12); %s.o.
        Entfernung2=(TourPos+(length(Tour)-counter12)); %s.o.
      end
      if Entfernung1<Entfernung2 %wähle die kleinste Entfernung
        DistG=Entfernung1;
      else
        DistG=Entfernung2;
      end
      %Optional: Berechnung der Entfernung zum Standort und Neuronenring in der Nähe des Standortes versteifen
      %Center1=counter12-1; %Entfernung zum Standort entlang Tourneuronen, math. pos. Richtung
      %Center2=length(Tour)-counter12;%Entfernung zum Standort entlang Tourneuronen, math. neg. Richtung
      %if Center1<Center2 %Zuweisen der geringeren Entfernung
       %  DistCenter=Center1;
       %else
      %   DistCenter=Center2;
      %end
      %Stiff=.1+(0.9/(ceil(length(Tour)/2)))*DistCenter; %Berechnung der Verminderung der Anpassung an den präsentierten Vektor
      %Berechnung der Stärke der Anpassung
      Gauss=LPb*(2.71^(((-DistG^2)/(2*(sigma^2)))));
      %Verschiebung der Neuronengewichte durch Linearkombination
      w(Tour(counter12),:)=((1-Gauss)*w(Tour(counter12),:)+Gauss*ChosenKoord(1,:)); %Gewichtsadaption durch Linearkombination
 end
%=========================================Anpassung der Neuronen der übrigen Touren=====================================
 schalter=1; %An/Ausschalten der Anpassung der Neuronen der übrigen Touren
 if schalter==1
   %Berechnung der Distanz zum Standort
   Stadist1=TourPos-1; %Entfernung zwischen Standort und TourPos
   Stadist2=(length(Tour)-TourPos)+1; %Entfernung zwischen TourPos und Standort
   if Stadist1<Stadist2 %wenn die Distanz 1 kleiner Distanz 2 ist
      Stadist=Stadist1; %dann wähle die kleinere Distanz 1
   else
      Stadist=Stadist2; %sonst wähle die kleinere (oder gleiche Distanz) Distanz2
   end
   %Anpassung der Nachbarschaft der übrigen Touren
   for counter33=1:AnzTouren %für alle Touren
      if counter33~=TourNeuronen(1,ind) %nur nicht für die GewinnerTour
         NachbarTour=find(TourNeuronen==counter33); %Bestimmen der Tourteilnehmer der NachbarTour (ohne Standort)
         showNachbarTour=NachbarTour; %Anzeigen der Tourteilnehmer
         for counter34=1:round(length(NachbarTour)/2) %Für die erste Hälfte der Tourteilnehmer
            DistGau=Stadist+(counter34); %Distanz zum Gewinnerneuron= Distanz des Gewinners zum Standort + Entfernung des Neurons zum Standort
            Gauss=LPb*(2.71^((-(DistGau^2)/(2*(sigma^2))))); %Ermittlung des Parameters der Linearkombination gemäß Gauss-Funktion
            w(NachbarTour(counter34),:)=((1-Gauss)*w(NachbarTour(counter34),:)+Gauss*ChosenKoord(1,:)); %Gewichtsadaption durch Linearkombination
         end
         for counter35=1:round(length(NachbarTour)/2) %für die 2te Hälfte der Ringneuronen der Nachbartour
            DistGau=Stadist+(counter35); %Distanz zum Gewinnerneuron= Distanz des Gewinners zum Standort + Entfernung des Neurons zum Standort
            Gauss=LPb*(2.71^(((-DistGau^2)/(2*(sigma^2))))); %Ermittlung des Parameters der Linearkombination gemäß der Gauss-Funktion
            w(NachbarTour(length(NachbarTour)-counter35+1),:)=((1-Gauss)*w(NachbarTour(length(NachbarTour)-counter35+1),:)+Gauss*ChosenKoord(1,:));
         end
      end
   end
end
% ========================================Überprüfen der Kapazitätsbedingungen=========================================
if Tabulength>0
   if Iter<AnzIter/2
      schalterTabuC=1; %An/Ausschalten Tabu-Mechanismus
      schalterAussverbot=1;
   else
      schalterTabuC=1; %An/Ausschalten Tabu-Mechanismus
      schalterAussverbot=0;
   end
   if Iter<(AnzIter-1)
```

```
            for countercapcheck=1:AnzTouren
              if UsedCap(countercapcheck)>Capacity(countercapcheck)
                Tsort=find(x(:,4)==countercapcheck); %finden der Ortsindizes, die dem Fahrzeug mit verletzter Kapazitäts-Restriktion zugeordnet sind
                Tsort2=x(Tsort,:); %Eintragen der Ortsinformationen in die Matrix Tsort2
                [tmp,idx]=sort(Tsort2(:,5)); %Sortieren der Neuronenindex-Spalte
                Tsort3=Tsort2(idx,:); %Sortieren der übrigen Spalten in der Reihenfolge der neuen Neuronen-Index-Spalte
                [AnzzugOrte, Eintrage]=size(Tsort3); %Ermitteln der Anzahl zugeordneten Orte
                NZNeuronen=find(TourNeuronen(1,:)~=countercapcheck&TourNeuronen(1,:)~=(AnzTouren+1)); %finden der nicht der Tour zugeordneten Neuronen
                [RowAnzNZN, AnzNZN]=size(NZNeuronen);
                for counterKapDist=1:AnzzugOrte
                  for counterNZNeuronen=1:AnzNZN
                    DistanzNZN(counterKapDist,counterNZNeuronen)=((w(NZNeuronen(counterNZNeuronen),1)-
                          Tsort3(counterKapDist,1))^2+(w(NZNeuronen(counterNZNeuronen),2)-Tsort3(counterKapDist,2))^2)^.5;%Distanzen berechnen
                    if schalterAussverbot==1;
                      if Tsort3(counterKapDist,6)>0
                        DistanzNZN(counterKapDist,counterNZNeuronen)=100000*Tsort3(counterKapDist,6); %wenn für den Ort ein Zuordnungsverbot besteht, dann setze
                                                                                                      %die Distanz auf einen großen Wert, um erneuten Ausschluß zu verhindern
                      end
                    end
                  end
                end
                while UsedCap(countercapcheck)>Capacity(countercapcheck) %wenn die Kapazitätsbeschränkung verletzt wurde
                  [mind,minr1]=min(DistanzNZN(:)); %Finden der minimalen Distanz in der Matrix DistanzNZN
                  mincol2=rem(minr1,AnzzugOrte); %Berechnen, welcher Rest bei der Division des GewinnerIndex durch die Anzahl der Orte übrigbleibt
                  if (mincol2==0) %Wenn das Ergebnis der Division eine ganze Zahl ist, dann ist es immer der letzte Ort
                    minr=AnzzugOrte; %Zeile ist der letzte Ort
                  else
                    minr=mincol2; %sonst ist der zugeordnete Ort gleich dem Restwert aus der Division
                  end
                  DistanzNZNAlt2=DistanzNZN;
                  DistanzNZN(minr,:)=1000000; %Eintragung in DistanzNZN auf hohen Wert setzen, um erneute Wahl des Ortes auszuschließen
                  BedarfKickOrtIndex1=find(x(:,1)==Tsort3(minr,1)&x(:,2)==Tsort3(minr,2)); %ermitteln des Index des Ortes, der aus der Tour entfernt wird
                  if isempty(BedarfKickOrtIndex)==1
                     Tsort4=Tsort3;
                     [tmp2,idx2]=sort(Tsort4(:,1));
                     Tsort3=Tsort4(idx2,:);
                     minr=1;
                     BedarfKickOrtIndex1=find(x(:,1)==Tsort3(minr,1)&x(:,2)==Tsort3(minr,2));
                  else
                     BedarfKickOrtIndex=BedarfKickOrtIndex1(1);%Wahl des ersten Eintrages der Matrix BedarfKickOrtIndex1
                  end
                  Tsort3(minr,:)=1000;
                  uscapalt=UsedCap;
                  UsedCap(countercapcheck)=UsedCap(countercapcheck)-x(BedarfKickOrtIndex,3); %Verringern der Kapazitätsbelastung des FZ
                  x(BedarfKickOrtIndex,4)=0; %bisherige Zuordnung Tour aufheben
                  x(BedarfKickOrtIndex,5)=0; %bisherige Zuordnung Neuron aufheben
                  x(BedarfKickOrtIndex,6)=countercapcheck; %Sperrvermerk eintragen (Nummer der Tour, die Kapazitätsverletzung aufwies);
                  x(BedarfKickOrtIndex,7)=Tabulength; %Tabu-Länge einstellen (kann Tabulength Iterationen lang nicht mehr Gewinner bei Präsentation des Ortes werden)
                end
                DistanzNZNalt=DistanzNZN;
                clear DistanzNZN;
              end
            end
          end
        end %Ende der Kapazitätsüberprüfung
        IterPattern=IterPattern+1; %Zählen der präsentierten Eingabevektoren
        %Speichern der Gewichtskoordinaten für Standort 1 nach jeder Iteration
        if schalterfiguren==1
          W1History(counter5+(Iter*patterns),1)=w(1,1);
          W1History(counter5+(Iter*patterns),2)=w(1,2);
        end
      end; %Ende des Präsentationszyklus
      if schalterfiguren==1
        SE=0; %Nullsetzen des quadrierten Abstandes des Standortneurons zu den Bedarfsorten
        for counterSE=1:patterns %für alle Bedarfsorte
          DistSE=((w(1,1)-x(counterSE,1))^2+(w(1,2)-x(counterSE,2))^2)^.5; %Ermitteln des Abstandes des Standortes zum Bedarfsort counterSE
          SE=SE+DistSE;
        end
        MSE(counter4,1)=SE/patterns;
      end
      Iter=Iter+1; %Zählen der durchgeführten Präsentationen
    end;%Ende des Zyklus
    %=======================================Graphische Ausgabe=============================================
    if schalterfiguren==1;
      %Lösung ausgeben
      figure(2); %Wahl der Figur Loesung
      subplot(2,2,4); %Wahl des Subplots 4 in der Figur
      hold on; %Halten der eingezeichneten Objekte
      for counter1=2:(AnzNeur-1); %von Neuron 1 bis AnzNeur-1
        if TourNeuronen(1,counter1)==TourNeuronen(1,counter1+1) %wenn zwei aufeinanderfolgende Neuronen zu einer Tour gehören
          for counter18=1:100 %Ermitteln von 100 Punkten auf der Verbindungsstrecke zwischen direkt benachbarten Neuronen
            xkoord(counter18)=((1-Tourpkt(counter18))*w(counter1,1)+Tourpkt(counter18)*w((counter1+1),1)); %Speichern der 100 x-Koordinaten der Linearkombinationen
            ykoord(counter18)=((1-Tourpkt(counter18))*w(counter1,2)+Tourpkt(counter18)*w((counter1+1),2)); %              y-Koordinaten
          end
            plot(xkoord,ykoord,'r-'); %Einzeichnen der Verbindungsstrecke zwischen den benachbarten Neuronen
        else
          for counter18=1:100 %Ermitteln von 100 Punkten auf der Verbindungsstrecke zwischen Standort und Neuron counter+1
            xkoord(counter18)=((1-Tourpkt(counter18))*w(1,1)+Tourpkt(counter18)*w((counter1+1),1)); %Speichern der 100 Koordinaten der Linearkombinationen
            ykoord(counter18)=((1-Tourpkt(counter18))*w(1,2)+Tourpkt(counter18)*w((counter1+1),2));
          end
          plot(xkoord,ykoord,'r-'); %Einzeichnen der Verbindungsstrecke zwischen Standort und Neuron counter
          if counter1~=1
            for counter18=1:100 %Ermitteln von 100 Punkten auf der Verbindungsstrecke zwischen Standort und Neuron Counter
              xkoord(counter18)=((1-Tourpkt(counter18))*w(1,1)+Tourpkt(counter18)*w((counter1),1)); %Speichern der 100 x-Koordinaten der Linearkombinationen
              ykoord(counter18)=((1-Tourpkt(counter18))*w(1,2)+Tourpkt(counter18)*w((counter1),2));
            end
            plot(xkoord,ykoord,'r-'); %Einzeichnen der Verbindungsstrecke zwischen Standort und Neuron Counter
          end
        end
      end
      for counter19=1:100; %Verbinden des ersten Neurons der ersten Tour und dem Standort
        xkoord(counter19)=((1-Tourpkt(counter19))*w(1,1)+Tourpkt(counter19)*w(2,1));
        ykoord(counter19)=((1-Tourpkt(counter19))*w(1,2)+Tourpkt(counter19)*w(2,2));
```

```
    end
    plot(xkoord,ykoord,'r-');
    for counter30=1:100; %Verbinden des ersten Neurons der ersten Tour und dem Standort
      xkoord(counter30)=((1-Tourpkt(counter30))*w(1,1)+Tourpkt(counter30)*w(AnzNeur,1));
      ykoord(counter30)=((1-Tourpkt(counter30))*w(1,2)+Tourpkt(counter30)*w(AnzNeur,2));
    end
    plot(xkoord,ykoord,'r-');
    plot(x(:,1),x(:,2),'.b','markersize',15); %Einzeichnen der Orte
    plot(w(:,1),w(:,2),'.g','markersize',10); %Einzeichnen der Neuronengewichte
    plot(w(1,1),w(1,2),'.r','markersize',15); %Einzeichnen der Neuronengewichte des Standortneurons
    xlabel('x-Koordinate');
    ylabel('y-Koordinate');
    title('Lösung t=t_{max}');
    figure(6); %Wahl der Figur Loesung
    hold on; %Halten der eingezeichneten Objekte
    for counter1=2:(AnzNeur-1); %von Neuron 1 bis AnzNeur-1
      if TourNeuronen(1,counter1)==TourNeuronen(1,counter1+1) %wenn zwei aufeinanderfolgende Neuronen zu einer Tour gehören
        for counter18=1:100 %Ermitteln von 100 Punkten auf der Verbindungsstrecke zwischen direkt benachbarten Neuronen
          xkoord(counter18)=((1-Tourpkt(counter18))*w(counter1,1)+Tourpkt(counter18)*w((counter1+1),1)); %Speichern der 100 x-Koordinaten der Linearkombinationen
          ykoord(counter18)=((1-Tourpkt(counter18))*w(counter1,2)+Tourpkt(counter18)*w((counter1+1),2)); %                y-Koordinaten
        end
        plot(xkoord,ykoord,'r-'); %Einzeichnen der Verbindungsstrecke zwischen den benachbarten Neuronen
      else
        for counter18=1:100 %Ermitteln von 100 Punkten auf der Verbindungsstrecke zwischen Standort und Neuron counter+1
          xkoord(counter18)=((1-Tourpkt(counter18))*w(1,1)+Tourpkt(counter18)*w((counter1+1),1)); %Speichern der 100 Koordinaten der Linearkombinationen
          ykoord(counter18)=((1-Tourpkt(counter18))*w(1,2)+Tourpkt(counter18)*w((counter1+1),2));
        end
        plot(xkoord,ykoord,'r-'); %Einzeichnen der Verbindungsstrecke zwischen Standort und Neuron counter
        if counter1~=1
          for counter18=1:100 %Ermitteln von 100 Punkten auf der Verbindungsstrecke zwischen Standort und Neuron Counter
            xkoord(counter18)=((1-Tourpkt(counter18))*w(1,1)+Tourpkt(counter18)*w((counter1),1)); %Speichern der 100 Koordinaten der Linearkombinationen
            ykoord(counter18)=((1-Tourpkt(counter18))*w(1,2)+Tourpkt(counter18)*w((counter1),2));
          end
          plot(xkoord,ykoord,'r-'); %Einzeichnen der Verbindungsstrecke zwischen Standort und Neuron Counter
        end
      end
    end
    for counter19=1:100; %Verbinden des ersten Neurons der ersten Tour und dem Standort
      xkoord(counter19)=((1-Tourpkt(counter19))*w(1,1)+Tourpkt(counter19)*w(2,1));
      ykoord(counter19)=((1-Tourpkt(counter19))*w(1,2)+Tourpkt(counter19)*w(2,2));
    end
    plot(xkoord,ykoord,'r-');
    for counter30=1:100;
      xkoord(counter30)=((1-Tourpkt(counter30))*w(1,1)+Tourpkt(counter30)*w(AnzNeur,1));
      ykoord(counter30)=((1-Tourpkt(counter30))*w(1,2)+Tourpkt(counter30)*w(AnzNeur,2));
    end
  plot(xkoord,ykoord,'r-');
  plot(x(:,1),x(:,2),'.b','markersize',15); %Einzeichnen der Orte
  plot(w(:,1),w(:,2),'.g','markersize',10); %Einzeichnen der Neuronengewichte
  plot(w(1,1),w(1,2),'.r','markersize',15); %Einzeichnen der Neuronengewichte des Standortneurons
  xlabel('x-Koordinate');
  ylabel('y-Koordinate');
  title('Lösung t=t_{max}');
%============= Ausgabe der Gauss-Glocke T=Tmax=========================================
  r=zeros(52,2);
  gaussbild1=-2;
  for counter10=1:52;
      r(counter10,1)=gaussbild1+1;
      r(counter10,2)=LPb*2.71^(-(r(counter10,1)^2/(2*(sigma^2))));
      gaussbild1=gaussbild1+1;
  end;
  figure(1)
  subplot (2,2,4)
  axis ([0 50 0 1]);
  hold on;
  plot(r(:,1),r(:,2),'r-');
  title('Nachbarschaftsfunktion, t=t_{max}');
  xlabel('\Delta_{cj}');
  ylabel('h_{cj}(t)');
%=============Ausgabe des Verlaufs des MSE und Auffinden des minimalen MSE==============
  MSEwinner=find(MSE==min(MSE));
  figure(3);
  subplot(2,1,1);
  hold on;
  ZaehlerIter=1:counter4;
  plot(ZaehlerIter,MSE,'r-');
  plot(MSEwinner,MSE(MSEwinner,1),'bo');
  xlabel('t');
  ylabel('MSE(t)');
  title('Verlauf MSE');
%=============Neuronenweg des Standortes 1 in der ersten Iteration zeichnen=============
  schalterNW1=1; %Ein- Ausschalten der NW-Zeichnung
  if schalterNW1==1
      figure(4); %Wahl der Figur 4
      subplot(2,2,1); %Subplot 1 wählen
      grid on; %Gitter an
      hold on; %Halten eingezeichneter Objekte an
      %axis ([5 15 45 55]);
      for counterHist=1:patterns %Iternummer*patterns:(Iternummer+1)*patterns
        if counterHist==1
          plot(W1History(counterHist,1),W1History(counterHist,2),'b.','markersize',15);
          text(W1History(counterHist,1),W1History(counterHist,2),'\leftarrow Start','FontSize',12);
        elseif counterHist==patterns
          plot(W1History(counterHist,1),W1History(counterHist,2),'k.','markersize',15);
          text(W1History(counterHist,1),W1History(counterHist,2),'\leftarrow Ende','FontSize',12);
        else
          plot(W1History(counterHist,1),W1History(counterHist,2),'bo');
        end
      end
      for counterNW1=1:patterns-1; %Iternummer*patterns:(Iternummer+1)*patterns-1 :für alle aufgez. Werte, außer für den letzten
        for counter18=1:100 %Ermitteln von 100 Punkten auf der Verbindungsstrecke zwischen direkt benachbarten Neuronen
          xkoord(counter18)=((1-Tourpkt(counter18))*W1History(counterNW1,1)+Tourpkt(counter18)*W1History(counterNW1+1,1)); %Speichern der 100 x-Koordinaten
          ykoord(counter18)=((1-Tourpkt(counter18))*W1History(counterNW1,2)+Tourpkt(counter18)*W1History(counterNW1+1,2)); %                y-Koordinaten
```

```
            end
            plot(xkoord,ykoord,'r-'); %Einzeichnen der Verbindungsstrecke zwischen den benachbarten Neuronen
        end
        title('Adaption des Depot-Neurons in den ersten 83 Iterationen');
        xlabel('x-Koordinate');
        ylabel('y-Koordinate');
end
%=============Neuronenweg des Standortes 1 im letzten Zyklus zeichnen=======================
 if schalterNW1==1
        figure(4); %Wahl der Figur 4
        subplot(2,2,2); %Subplot 2 wählen
        grid on; %Gitter an
        hold on; %Halten eingezeichneter Objekte
         %axis ([5 15 45 55]);
        for counterHist=1:patterns %Iternummer*patterns:(Iternummer+1)*patterns
            if counterHist==1
               plot(W1History((AnzIter-1)*patterns+counterHist,1),W1History((AnzIter-1)*patterns+counterHist,2),'b.','markersize',15);
               text(W1History((AnzIter-1)*patterns+counterHist,1),W1History((AnzIter-1)*patterns+counterHist,2),'\leftarrow Start / Ende','FontSize',12);
            elseif counterHist==patterns
               plot(W1History((AnzIter-1)*patterns+counterHist,1),W1History((AnzIter-1)*patterns+counterHist,2),'k.','markersize',15);
            else
               plot(W1History((AnzIter-1)*patterns+counterHist,1),W1History((AnzIter-1)*patterns+counterHist,2),'bo');
            end
         end
        for counterNW2=1:patterns-1; %Iternummer*patterns:(Iternummer+1)*patterns-1 :für alle aufgez. Werte, außer für den letzten
            for counter18=1:100 %Ermitteln von 100 Punkten auf der Verbindungsstrecke zwischen direkt benachbarten Neuronen
                xkoord(counter18)=((1-Tourpkt(counter18))*W1History((AnzIter-1)*patterns+counterNW2,1)+Tourpkt(counter18)*W1History((AnzIter-
                    1)*patterns+counterNW2+1,1));%Speichern der 100 x-Koordinaten der Linearkombinationen
                ykoord(counter18)=((1-Tourpkt(counter18))*W1History((AnzIter-1)*patterns+counterNW2,2)+Tourpkt(counter18)*W1History((AnzIter-
                    1)*patterns+counterNW2+1,2));%
            end
            plot(xkoord,ykoord,'g-'); %Einzeichnen der Verbindungsstrecke zwischen den benachbarten Neuronen
        end
        title('Adaption des Depot-Neurons in den letzten 83 Iterationen');
        xlabel('x-Koordinate');
        ylabel('y-Koordinate');
  end
   %============Ausgabe Verlauf Sigma=============================================
    sigmaVerlauf=zeros(1,AnzIter*patterns);
    Iterpatt=1:AnzIter*patterns;
    for countersVer=1:AnzIter*patterns
        sigmaVerlauf(1,countersVer)=sigmastart*((1/sigmastart)^(countersVer/(AnzIter*patterns)))-.9;%Festlegen des aktuellen Wertes der Breite der Gauss-Funktion
    end
    figure(4); %Wahl der Figur 4
    subplot(2,2,3); %Subplot 3 wählen
    grid on; %Gitter an
    hold on; %Halten eingezeichneter Objekte an
    plot(Iterpatt,sigmaVerlauf,'r-');
    xlabel('t');
    ylabel('\sigma(t)');
    title('Verlauf \sigma(t)');
    %Ausgabe Verlauf Lernparameter
    LPbVerlauf=zeros(1,AnzIter*patterns);
    for counterLVer=1:AnzIter*patterns
        LPbVerlauf(1,counterLVer)=1-(counterLVer/(AnzIter*patterns));
        if LPbVerlauf(1,counterLVer)<0.005
            LPbVerlauf(1,counterLVer)=.005;
        end
    end
    figure(4); %Wahl der Figur 4
    subplot(2,2,4); %Subplot 4 wählen
    grid on; %Gitter an
    hold on; %Halten eingezeichneter Objekte an
    plot(Iterpatt,LPbVerlauf,'r-');
    xlabel('t');
    ylabel('\alpha(t)');
    title('Verlauf \alpha(t)');
end
%=================Berechnung der Distanz=============================================================================================
for counterDistcheck=1:AnzTouren
    Tsort=find(x(:,4)==counterDistcheck); %finden der Ortsindizes, die dem Fahrzeug counterDistcheck zugeordnet sind
    Tsort2=x(Tsort,:); %Eintragen der Ortsinformationen in die Matrix Tsort2
    Tsort2(:,7)=Tsort(:,1);
    [tmp,idx]=sort(Tsort2(:,5)); %Sortieren der Neuronenindex-Spalte
    Tsort3=Tsort2(idx,:); %Sortieren der übrigen Spalten in der Reihenfolge der neuen Neuronen-Index-Spalte
    [AnzzugOrte2, Eintrage]=size(Tsort3); %Ermitteln der Anzahl zugeordneten Orte
    for countermaxsav=1:AnzzugOrte2
        if countermaxsav==1
            DistanzA=((w(1,1)-Tsort3(countermaxsav,1))^2+(w(1,2)-Tsort3(countermaxsav,2))^2)^.5; %Distanz zum Standort
            DistanzC(counterDistcheck,countermaxsav)=DistanzA; %Eintragen der Entfernung zum Vorgänger
            EDistStandort(counterDistcheck,countermaxsav)=DistanzA; %Ermitteln der Entfernung von Standort zum ersten Kunden einer Tour
            if AnzzugOrte2==1
              DistanzC2(counterDistcheck,countermaxsav)=2*DistanzA; %Eintragen der Entfernung zum Vorgänger
            end
        elseif countermaxsav==AnzzugOrte2
            DistanzA=((w(1,1)-Tsort3(countermaxsav,1))^2+(w(1,2)-Tsort3(countermaxsav,2))^2)^.5; %Distanz zum Standort,1
            DistanzB=((Tsort3(countermaxsav,1)-Tsort3(countermaxsav-1,1))^2+(Tsort3(countermaxsav,2)-Tsort3(countermaxsav-1,2))^2)^.5; %Distanz zum Vorgänger
            DistanzC(counterDistcheck,countermaxsav)=DistanzA+DistanzB; %Summieren und Eintragung der Entfernungen
            EDistStandort(counterDistcheck,countermaxsav)=DistanzA; %Ermitteln der Entfernung des Standortes vom letzten Kunden einer Tour
            if AnzzugOrte2==1
              DistanzC2(counterDistcheck,countermaxsav)=2*DistanzA; %Eintragen der Entfernung zum Vorgänger
            end
        else
            DistanzA=((Tsort3(countermaxsav,1)-Tsort3(countermaxsav-1,1))^2+(Tsort3(countermaxsav,2)-Tsort3(countermaxsav-1,2))^2)^.5; %Distanz zum Vorgänger
            DistanzC(counterDistcheck,countermaxsav)=DistanzA; %Eintragen der Entfernung zum Vorgänger
        end
    end
    StartortX(1,counterDistcheck)=Tsort3(1,1); %Eintragen der x-Koordinate des ersten Ortes in einer Tour
    StartortY(1,counterDistcheck)=Tsort3(1,2); %Eintragen der x-Koordinate des letzten Ortes in einer Tour
    EndortX(1,counterDistcheck)=Tsort3(AnzzugOrte2,1); %Eintragen der y-Koordinate des ersten Ortes in einer Tour
    EndortY(1,counterDistcheck)=Tsort3(AnzzugOrte2,2); %Eintragen der y-Koordinate des letzten Ortes in einer Tour
end
for counterDistTour=1:AnzTouren
```

```
    DistTour(counterDistTour,1)=sum(DistanzC(counterDistTour,:)); %Ermitteln der Länge der einzelnen Touren
end
for counterGesamtlaenge=1:AnzTouren
    Gesamtlaenge=sum(DistTour(:,1)); %Ermitteln der Gesamtlänge des Tourenplanes
end
w1=w(1,1) %Ausgabe Standortkoordinaten
w2=w(1,2)
Gesamtlaenge=Gesamtlaenge %Ausgabe der Gesamtlänge
UsedCap=UsedCap %Ausgabe der Kapazitätsbelastung
Capacity=Capacity %Ausgabe der Fahrzeugkapazitäten
Auslastungsgrad=UsedCap./Capacity %Ausgabe der Auslastungsgrade
ProzentKapazitaetsueberschuss=(sum(Capacity)/sum(UsedCap)-1) %Verhältnis zwischen Kapazitätsangebot und Kapazitätsnachfrage-1=Kapazitätsüberschuß
EDStandort=sum(EDistStandort(:)) %Ausgabe der Entfernung der ersten und letzten Kunden aller Touren zum Standort
% Aufbereiten der Koordinatenmatrix für Weiszfeld-Ansatz
KoordinatenOrte1(1,:)=StartortX(1,:);
KoordinatenOrte1(2,:)=StartortY(1,:);
KoordinatenOrte2(1,:)=EndortX(1,:);
KoordinatenOrte2(2,:)=EndortY(1,:);
KoordinatenOrte=[KoordinatenOrte1,KoordinatenOrte2];
%=============Aufrufen der Funktion WeiszfeldWeber==================================================
[WeberX,WeberY,WeberDist]=WeiszfeldWeberLRP(KoordinatenOrte);
WeberDist=WeberDist %Ausgabe der Distanz des Weber-Punktes
VWNN=EDStandort/WeberDist %Ausgabe der Abweichung der SOM-Standort-Lösung von der Weiszfeld-Lösung
%=============Distanz mit Weber-Standort berechnen=================================================================
for counterDistcheck2=1:AnzTouren
    Tsort=find(x(:,4)==counterDistcheck2); %finden der Ortsindizes, die dem Fahrzeug counterDistcheck zugeordnet sind
    Tsort2=x(Tsort,:); %Eintragen der Ortsinformationen in die Matrix Tsort2
    Tsort2(:,7)=Tsort(:,1);
    [tmp,idx]=sort(Tsort2(:,5)); %Sortieren der Neuronenindex-Spalte
    Tsort3=Tsort2(idx,:); %Sortieren der übrigen Spalten in der Reihenfolge der neuen Neuronen-Index-Spalte
    [AnzzugOrte2, Eintrage]=size(Tsort3); %Ermitteln der Anzahl zugeordneter Orte
    for countermaxsav2=1:AnzzugOrte2
        if countermaxsav2==1
            DistanzA2=((WeberX-Tsort3(countermaxsav2,1))^2+(WeberY-Tsort3(countermaxsav2,2))^2)^.5; %Distanz zum Standort
            DistanzC2(counterDistcheck2,countermaxsav2)=DistanzA2; %Eintragen der Entfernung zum Vorgänger
            EDistStandort2(counterDistcheck2,countermaxsav2)=DistanzA2; %Ermitteln der Entfernung von Standort zum ersten Kunden einer Tour
            if AnzzugOrte2==1
              DistanzC2(counterDistcheck2,countermaxsav2)=2*DistanzA2; %Eintragen der Entfernung zum Vorgänger
            end
        elseif countermaxsav2==AnzzugOrte2
            DistanzA2=((WeberX-Tsort3(countermaxsav2,1))^2+(WeberY-Tsort3(countermaxsav2,2))^2)^.5; %Distanz zum Standort
            DistanzB2=((Tsort3(countermaxsav2,1)-Tsort3(countermaxsav2-1,1))^2+(Tsort3(countermaxsav2,2)-Tsort3(countermaxsav2-1,2))^2)^.5; %Distanz zum Vorgänger
            DistanzC2(counterDistcheck2,countermaxsav2)=DistanzA2+DistanzB2; %Summieren und Eintragung der Entfernungen
            EDistStandort2(counterDistcheck2,countermaxsav2)=DistanzA2; %Ermitteln der Entfernung des Standortes vom letzten Kunden einer Tour
            if AnzzugOrte2==1
              DistanzC2(counterDistcheck2,countermaxsav2)=2*DistanzA2; %Eintragen der Entfernung zum Vorgänger
            end
        else
            DistanzA2=((Tsort3(countermaxsav2,1)-Tsort3(countermaxsav2-1,1))^2+(Tsort3(countermaxsav2,2)-Tsort3(countermaxsav2-1,2))^2)^.5; %Distanz zum Vorgänger
            DistanzC2(counterDistcheck2,countermaxsav2)=DistanzA2; %Eintragen der Entfernung zum Vorgänger
        end
    end
    StartortX2(1,counterDistcheck2)=Tsort3(1,1); %Eintragen der x-Koordinate des ersten Ortes in einer Tour
    StartortY2(1,counterDistcheck2)=Tsort3(1,2); %Eintragen der x-Koordinate des letzten Ortes in einer Tour
    EndortX2(1,counterDistcheck2)=Tsort3(AnzzugOrte2,1); %Eintragen der y-Koordinate des ersten Ortes in einer Tour
    EndortY2(1,counterDistcheck2)=Tsort3(AnzzugOrte2,2); %Eintragen der y-Koordinate des letzten Ortes in einer Tour
end
for counterDistTour2=1:AnzTouren
   DistTour2(counterDistTour2,1)=sum(DistanzC2(counterDistTour2,:)); %Ermitteln der Länge der einzelnen Touren
end
for counterGesamtlaenge2=1:AnzTouren
   GesamtlaengeW=sum(DistTour2(:,1)); %Ermitteln der Gesamtlänge des Tourenplanes
end
GesamtlaengeW=GesamtlaengeW %Ausgabe Gesamtlänge mit Weber-Standort
%=============Distanz mit Orginal-Standort berechnen=================================================================
for counterDistcheck2=1:AnzTouren
    Tsort=find(x(:,4)==counterDistcheck2); %finden der Ortsindizes, die dem Fahrzeug counterDistcheck zugeordnet sind
    Tsort2=x(Tsort,:); %Eintragen der Ortsinformationen in die Matrix Tsort2
    Tsort2(:,7)=Tsort(:,1);
    [tmp,idx]=sort(Tsort2(:,5)); %Sortieren der Neuronenindex-Spalte
    Tsort3=Tsort2(idx,:); %Sortieren der übrigen Spalten in der Reihenfolge der neuen Neuronen-Index-Spalte
    [AnzzugOrte2, Eintrage]=size(Tsort3); %Ermitteln der Anzahl zugeordneten Orte
    for countermaxsav2=1:AnzzugOrte2
        if countermaxsav2==1
            DistanzA2=((OrgSt(1,1)-Tsort3(countermaxsav2,1))^2+(OrgSt(1,2)-Tsort3(countermaxsav2,2))^2)^.5; %Distanz zum Standort
            DistanzC2(counterDistcheck2,countermaxsav2)=DistanzA2; %Eintragen der Entfernung
            EDistStandort2(counterDistcheck2,countermaxsav2)=DistanzA2; %Ermitteln der Entfernung von Standort zum ersten Kunden einer Tour
            if AnzzugOrte2==1
              DistanzC2(counterDistcheck2,countermaxsav2)=2*DistanzA2; %Eintragen der Entfernung
            end
        elseif countermaxsav2==AnzzugOrte2
            DistanzA2=((OrgSt(1,1)-Tsort3(countermaxsav2,1))^2+(OrgSt(1,2)-Tsort3(countermaxsav2,2))^2)^.5; %Distanz zum Standort,1
            DistanzB2=((Tsort3(countermaxsav2,1)-Tsort3(countermaxsav2-1,1))^2+(Tsort3(countermaxsav2,2)-Tsort3(countermaxsav2-1,2))^2)^.5; %Distanz zum Vorgänger
            DistanzC2(counterDistcheck2,countermaxsav2)=DistanzA2+DistanzB2; %Summieren und Eintragung der Entfernungen
            EDistStandort2(counterDistcheck2,countermaxsav2)=DistanzA2; %Ermitteln der Entfernung des Standortes vom letzten Kunden einer Tour
            if AnzzugOrte2==1
              DistanzC2(counterDistcheck2,countermaxsav2)=2*DistanzA2; %Eintragen der Entfernung
            end
        else
            DistanzA2=((Tsort3(countermaxsav2,1)-Tsort3(countermaxsav2-1,1))^2+(Tsort3(countermaxsav2,2)-Tsort3(countermaxsav2-1,2))^2)^.5; %Distanz zum Vorgänger
            DistanzC2(counterDistcheck2,countermaxsav2)=DistanzA2; %Eintragen der Entfernung
        end
    end
    StartortX2(1,counterDistcheck2)=Tsort3(1,1); %Eintragen der x-Koordinate des ersten Ortes in einer Tour
    StartortY2(1,counterDistcheck2)=Tsort3(1,2); %Eintragen der y-Koordinate des letzten Ortes in einer Tour
    EndortX2(1,counterDistcheck2)=Tsort3(AnzzugOrte2,1); %Eintragen der xKoordinate des ersten Ortes in einer Tour
    EndortY2(1,counterDistcheck2)=Tsort3(AnzzugOrte2,2); %Eintragen der y-Koordinate des letzten Ortes in einer Tour
end
for counterDistTour2=1:AnzTouren
   DistTour2(counterDistTour2,1)=sum(DistanzC2(counterDistTour2,:)); %Ermitteln der Länge der einzelnen Touren
end
GesamtlaengeW2=0;
for counterGesamtlaenge2=1:AnzTouren
```

```
   GesamtlaengeW2=sum(DistTour2(:,1)); %Ermitteln der Gesamtlänge des Tourenplanes
end
GesamtlaengeOrgStandort=GesamtlaengeW2 %Ausgabe Gesamtlänge mit Weber-Standort
if schaltermovie==1
   mov = close(mov)%Movie schließen
end
% Ende LRPSOM(W/D)-Function=========================================================================================================

% WeiszfeldWeber -Function=========================================================================================
function [WeberX,WeberY,WeberDist]=WeiszfeldWeberLRP(KoordinatenOrte)
% ==========Testlauf=============
skript=0;
if skript==1
   clear;
   KoordinatenOrte=[49      40          21          42          31          36          42          38          32          32          25
                    48; 49  30          47          41          32          16          57          46          22          39          32
                    28];
   KoordinatenOrte=KoordinatenOrte';
   [Koordinaten AnzahlOrte]=size(KoordinatenOrte');
   KoordinatenOrte=KoordinatenOrte';
else
   [Koordinaten AnzahlOrte]=size(KoordinatenOrte);
end
%================================
b=ones(1,AnzahlOrte); % Gewichtung der eukl. Entfernung=1
x(1,1)=45; %beliebiger Startort x-Koordinate
y(1,1)=45; %beliebiger Startort y-Koordinate
for counterIterationen=1:100   %Anzahl durchzuführender Iterationen
   SummeX=0; %Nullsetzen der Summen
   SummeY=0;
   SummeX2=0;
   SummeY2=0;
   for i=1:AnzahlOrte %Iteration nach Weiszfeld
      if counterIterationen==1
         SummeX=SummeX+((b(1,i)*KoordinatenOrte(1,i)/((x(1,counterIterationen)-KoordinatenOrte(1,i))^2+(y(1,counterIterationen)-KoordinatenOrte(2,i)+10^(-10))^2)^.5));
         SummeX2=SummeX2+(b(1,i)/(((x(1,counterIterationen)-KoordinatenOrte(1,i))^2+(y(1,counterIterationen)-KoordinatenOrte(2,i)+10^(-10))^2)^.5));
         SummeY=SummeY+((b(1,i)*KoordinatenOrte(2,i)/((x(1,counterIterationen)-KoordinatenOrte(1,i))^2+(y(1,counterIterationen)-KoordinatenOrte(2,i)+10^(-10))^2)^.5));
         SummeY2=SummeY2+(b(1,i)/(((x(1,counterIterationen)-KoordinatenOrte(1,i))^2+(y(1,counterIterationen)-KoordinatenOrte(2,i)+10^(-10))^2)^.5));
      else
         SummeX=SummeX+((b(1,i)*KoordinatenOrte(1,i)/((x(1,counterIterationen-1)-KoordinatenOrte(1,i))^2+(y(1,counterIterationen-1)-KoordinatenOrte(2,i)+10^(-
                10))^2)^.5));
         SummeX2=SummeX2+((b(1,i)/((x(1,counterIterationen-1)-KoordinatenOrte(1,i))^2+(y(1,counterIterationen-1)-KoordinatenOrte(2,i)+10^(-10))^2)^.5));
         SummeY=SummeY+((b(1,i)*KoordinatenOrte(2,i)/((x(1,counterIterationen-1)-KoordinatenOrte(1,i))^2+(y(1,counterIterationen-1)-KoordinatenOrte(2,i)+10^(-
                10))^2)^.5));
         SummeY2=SummeY2+((b(1,i)/((x(1,counterIterationen-1)-KoordinatenOrte(1,i))^2+(y(1,counterIterationen-1)-KoordinatenOrte(2,i)+10^(-10))^2)^.5));
      end
   end
   x(1,counterIterationen)=SummeX/SummeX2; %Eintragen des Verlaufes der Annäherung
   y(1,counterIterationen)=SummeY/SummeY2;
end
WeberX=x(1,counterIterationen) %Ausgabe der Endergebnisse
WeberY=y(1,counterIterationen)
WeberDist=0;
for counterWeberDist=1:AnzahlOrte %Berechnen der euklidischen Distanz des Steiner-Weber-Punktes zu den Anfangs- und Endkunden
   WeberDist=WeberDist+((WeberX-KoordinatenOrte(1,counterWeberDist))^2+(WeberY-KoordinatenOrte(2,counterWeberDist))^2)^.5;
end
%==============Graphische Ausgabe================================
schalterweberfig=0;
if schalterweberfig==1
   WeberFig=figure;
   hold on;
   plot(KoordinatenOrte(1,:),KoordinatenOrte(2,:),'ro');
   plot(WeberX,WeberY,'bo');
end
%Ende WeiszfeldWeber -Function ====================================================================================================

%NNA-Skript=======================================================================================================
clear *;%lösche Variablen
%Einlesen Koordinaten
%KoordinatenOrte1 = xlsread('C1.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1 = xlsread('C2.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1= xlsread('C3.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1 = xlsread('C4.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1 = xlsread('C6.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1 = xlsread('C8.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1 = xlsread('C10.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1 = xlsread('F1.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1 = xlsread('F2.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1 = xlsread('F3.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1 = xlsread('ATT48.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1 = xlsread('GJ1.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1 = xlsread('eil22.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1 = xlsread('eil23.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1 = xlsread('eil30.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
KoordinatenOrte1 = xlsread('eil33.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%Eingelesene Koordinaten verarbeiten
KoordinatenOrte2(:,1)=KoordinatenOrte1(:,1);
KoordinatenOrte2(:,2)=KoordinatenOrte1(:,2);
KoordinatenOrte2(:,3)=KoordinatenOrte1(:,3);
KoordinatenOrte=KoordinatenOrte2';
[Koordinaten AnzahlOrte]=size(KoordinatenOrte);
[inputs, patterns]= size(KoordinatenOrte)
%Fahrzeugkapazität und Original-Standort setzen
FZKap=8000;% %Setzen der Fahrzeugkapazität
%OrgSta=[30 40];%Original-Standort Depot
%=====================================================Steiner-Weber-Problem, Weiszfeld-Algorithmus=================================
%Gewichtung der Distanzen bestimmen
Gewichtung=1;
if Gewichtung==0
   b=ones(1,AnzahlOrte); %Gewichtung der eukl. Entfernung=1
else
   b1(:,1)=KoordinatenOrte1(:,3); %Bei Ermittlung des Weber Punktes für alle Eingabeorte der Testinstanz mit Gewichtung
   b=b1'; %Matrix transponieren
```

```
end
bold=b;% Matrix speichern
x(1,1)=8.5; %beliebiger Startort x-Koordinate
y(1,1)=51;  %beliebiger Startort y-Koordinate
for counterIterationen=1:100  %Anzahl durchzuführender Iterationen
   SummeX=0; %Nullsetzen der Summen
   SummeY=0;
   SummeX2=0;
   SummeY2=0;
   for i=1:AnzahlOrte %Eigentliche Iteration nach Weiszfeld
      if counterIterationen==1
         SummeX=SummeX+((b(1,i)*KoordinatenOrte(1,i)/((x(1,counterIterationen)-KoordinatenOrte(1,i))^2+(y(1,counterIterationen)-KoordinatenOrte(2,i)+10^(-10))^2)^.5));
         SummeX2=SummeX2+(b(1,i)/(((x(1,counterIterationen)-KoordinatenOrte(1,i))^2+(y(1,counterIterationen)-KoordinatenOrte(2,i)+10^(-10))^2)^.5));
         SummeY=SummeY+((b(1,i)*KoordinatenOrte(2,i)/((x(1,counterIterationen)-KoordinatenOrte(1,i))^2+(y(1,counterIterationen)-KoordinatenOrte(2,i)+10^(-10))^2)^.5));
         SummeY2=SummeY2+(b(1,i)/(((x(1,counterIterationen)-KoordinatenOrte(1,i))^2+(y(1,counterIterationen)-KoordinatenOrte(2,i)+10^(-10))^2)^.5));
      else
         SummeX=SummeX+((b(1,i)*KoordinatenOrte(1,i)/((x(1,counterIterationen-1)-KoordinatenOrte(1,i))^2+(y(1,counterIterationen-1)-KoordinatenOrte(2,i)+10^(-
               10))^2)^.5));
         SummeX2=SummeX2+((b(1,i)/((x(1,counterIterationen-1)-KoordinatenOrte(1,i))^2+(y(1,counterIterationen-1)-KoordinatenOrte(2,i)+10^(-10))^2)^.5));
         SummeY=SummeY+((b(1,i)*KoordinatenOrte(2,i)/((x(1,counterIterationen-1)-KoordinatenOrte(1,i))^2+(y(1,counterIterationen-1)-KoordinatenOrte(2,i)+10^(-
               10))^2)^.5));
         SummeY2=SummeY2+((b(1,i)/((x(1,counterIterationen-1)-KoordinatenOrte(1,i))^2+(y(1,counterIterationen-1)-KoordinatenOrte(2,i)+10^(-10))^2)^.5));
      end
   end
   x(1,counterIterationen)=SummeX/SummeX2; %Eintragen des Verlaufes der Annäherung
   y(1,counterIterationen)=SummeY/SummeY2;
end
WeberX=x(1,counterIterationen) %Ausgabe der Endergebnisse
WeberY=y(1,counterIterationen)
WeberDist=0;
for counterWeberDist=1:AnzahlOrte %Berechnen der euklidischen Distanz des Steiner-Weber-Punktes zu den Anfangs- und Endkunden
   WeberDist=WeberDist+((WeberX-KoordinatenOrte(1,counterWeberDist))^2+(WeberY-KoordinatenOrte(2,counterWeberDist))^2)^.5;
end
WeberDistold=WeberDist;
%=============================================
testlauf=0;
  if testlauf==1
  FZKap=10;
  x=[1 2 3;3 4 3;5 6 3;7 8 5];
  Depot=[0 0];
  [inputs, patterns]= size(x'); %hier wird die Dimension von x ausgelesen
end
Depot=[WeberX WeberY]
Depotold=[WeberX WeberY];
Touren=zeros(patterns,patterns);
TAnzZug=zeros(patterns,1);
Kap=zeros(patterns);
%Distanzen zwischen Orten berechnen
for i=1:patterns
   for j=1:patterns
      DistanzO(i,j)=((KoordinatenOrte2(i,1)-KoordinatenOrte2(j,1))^2+(KoordinatenOrte2(i,2)-KoordinatenOrte2(j,2))^2)^.5;
      if i==j
         DistanzO(i,j)=100000;
      end
   end
end
DistanzOORG=DistanzO;
%Distanzen zwischen Orten und Depot berechnen
for k=1:patterns
   DistanzD(1,k)=((KoordinatenOrte2(k,1)-Depot(1,1))^2+(KoordinatenOrte2(k,2)-Depot(1,2))^2)^.5;
end
DistanzDORG=DistanzD;
%Algorithmus
%=========================================================================
%Eröffnen der ersten Tour
Erster=find(DistanzD==min(DistanzD));     %Ermittlung des Index des Gewinner-Neurons
LetzterOrt=Erster;
t=1; %Tour 1 wählen
Touren(t,1)=LetzterOrt;
Kap(t,1)=KoordinatenOrte2(LetzterOrt,3);%zugewiesene Kapazität erhöhen
AnzZugO=1;
TAnzZug(1)=1;%Zählen der Tour 1 zugeordneten Orte
DistanzD(1,LetzterOrt)=200000;%Distanz auf hohen Wert setzen
DistanzO(:,LetzterOrt)=200000;%Einträge in der Matrix Distanz auf hohen Wert setzen
%Übrige Orte zuweisen
while AnzZugO<patterns
    DistanzZO=DistanzO(LetzterOrt,:);
    [Zeile NextOrt]=find(DistanzZO==min(DistanzZO));
     if  Kap(t,1)+KoordinatenOrte2(NextOrt,3)<=FZKap
        Kap(t,1)=Kap(t,1)+KoordinatenOrte2(NextOrt(1,1),3);
        AnzZugO=AnzZugO+1;
        TAnzZug(t)=TAnzZug(t)+1;%Erhöhen der der Tour zugeordneten Orte
        Touren(t,TAnzZug(t))=NextOrt(1,1);%Einfügen in Tour
        DistanzO(LetzterOrt,:)=200000;%Erhöhen der Einträge in der Matrix
        DistanzO(:,NextOrt(1,1))=200000;
        DistanzD(NextOrt(1,1))=200000;
        LetzterOrt=NextOrt(1,1);
     else
         t=t+1;%Eröffnung neuer Tour
         [Zeile NextOrt]=find(DistanzD==min(DistanzD));
         TAnzZug(t)=TAnzZug(t)+1;%Erhöhen der der Tour zugeordneten Orte
         Touren(t,TAnzZug(t))=NextOrt(1,1);%Einfügen in Tour
         Kap(t,1)=Kap(t,1)+KoordinatenOrte2(NextOrt(1,1),3);
         AnzZugO=AnzZugO+1;
         DistanzD(1,NextOrt(1,1))=200000;
         DistanzO(:,NextOrt(1,1))=200000;
         LetzterOrt=NextOrt(1,1);
     end
end
%=============================================
b=1; %alten Gewichtungsvektor eliminieren
%Koordinaten der ersten und letzten Orte bestimmen
for h=1:t
```

```
    if h==1
       KoordinatenOrteN=KoordinatenOrte(:,Touren(h,1));
       b(1,h)=1; %Gewichtungsfaktor bestimmen
    else
       KoordinatenOrteN=[KoordinatenOrteN(:,:) KoordinatenOrte(:,Touren(h,1))]; %hier werden die Daten der ersten Orte einer Tour eingetragen
       b=[b(:,:) 1];
    end
    if TAnzZug(h,1)>1
      KoordinatenOrteN=[KoordinatenOrteN(:,:) KoordinatenOrte(:,Touren(h,TAnzZug(h,1)))] ; % hier werden die Daten des jeweils letzten Ortes einer Tour ausgelesen
      b=[b(:,:) 1]; %Gewichtungsfaktor für Ort eintragen
    else
      [AnzGew GEW]=size(b);
      b(1,GEW)=2; %Gewichtungsfaktor für Stichfahrten auf den Wert 2 setzen;
    end
end
 [KoordinatenT AnzahlOrteN]=size(KoordinatenOrteN);
x(1,1)=8.5; %beliebiger Startort x-Koordinate
y(1,1)=51;  %beliebiger Startort y-Koordinate
for counterIterationen=1:100  %Anzahl durchzuführender Iterationen
    SummeX=0; %Nullsetzen der Summen
    SummeY=0;
    SummeX2=0;
    SummeY2=0;
    for i=1:AnzahlOrteN %Eigentliche Iteration nach Weiszfeld
       if counterIterationen==1
          SummeX=SummeX+((b(1,i)*KoordinatenOrteN(1,i)/((x(1,counterIterationen)-KoordinatenOrteN(1,i))^2+(y(1,counterIterationen)-KoordinatenOrteN(2,i)+10^(-
                10))^2)^.5));
          SummeX2=SummeX2+(b(1,i)/(((x(1,counterIterationen)-KoordinatenOrteN(1,i))^2+(y(1,counterIterationen)-KoordinatenOrteN(2,i)+10^(-10))^2)^.5));
          SummeY=SummeY+((b(1,i)*KoordinatenOrteN(2,i)/((x(1,counterIterationen)-KoordinatenOrteN(1,i))^2+(y(1,counterIterationen)-KoordinatenOrteN(2,i)+10^(-
                10))^2)^.5));
          SummeY2=SummeY2+(b(1,i)/(((x(1,counterIterationen)-KoordinatenOrteN(1,i))^2+(y(1,counterIterationen)-KoordinatenOrteN(2,i)+10^(-10))^2)^.5));
       else
          SummeX=SummeX+((b(1,i)*KoordinatenOrteN(1,i)/((x(1,counterIterationen-1)-KoordinatenOrteN(1,i))^2+(y(1,counterIterationen-1)-KoordinatenOrteN(2,i)+10^(-
                10))^2)^.5));
          SummeX2=SummeX2+((b(1,i)/((x(1,counterIterationen-1)-KoordinatenOrteN(1,i))^2+(y(1,counterIterationen-1)-KoordinatenOrteN(2,i)+10^(-10))^2)^.5));
          SummeY=SummeY+((b(1,i)*KoordinatenOrteN(2,i)/((x(1,counterIterationen-1)-KoordinatenOrteN(1,i))^2+(y(1,counterIterationen-1)-KoordinatenOrteN(2,i)+10^(-
                10))^2)^.5));
          SummeY2=SummeY2+((b(1,i)/((x(1,counterIterationen-1)-KoordinatenOrteN(1,i))^2+(y(1,counterIterationen-1)-KoordinatenOrteN(2,i)+10^(-10))^2)^.5));
       end
    end
    x(1,counterIterationen)=SummeX/SummeX2; %Eintragen des Verlaufes der Annäherung
    y(1,counterIterationen)=SummeY/SummeY2;
end
WeberX=x(1,counterIterationen) %Ausgabe der Endergebnisse
WeberY=y(1,counterIterationen)
WeberDist=0;
for counterWeberDist=1:AnzahlOrteN %Berechnen der euklidischen Distanz des Steiner-Weber-Punktes zu den Anfangs- und Endkunden
    WeberDist=WeberDist+((WeberX-KoordinatenOrteN(1,counterWeberDist))^2+(WeberY-KoordinatenOrteN(2,counterWeberDist))^2)^.5;
end
%=========================================
Depot=[WeberX WeberY]
%Distanzen zwischen Orten und neuem Depot-Standort berechnen
for k=1:patterns
    DistanzDN(1,k)=((KoordinatenOrte2(k,1)-Depot(1,1))^2+(KoordinatenOrte2(k,2)-Depot(1,2))^2)^.5;
end
DistanzDORG=DistanzDN;
%Distanz berechnen
DistA=zeros(1);
DistB=zeros(1);
DistC=zeros(1);
DistA2=0;
DistB2=0;
for g=1:t
    %Entfernungen vom Depot zum Ort1 jeder Tour
     DistA(1,g)=DistanzDORG(Touren(g,1));
    %Entfernung vom letzten Ort zum Depot
     DistB(1,g)=DistanzDORG(Touren(g,TAnzZug(g)));
    %Entfernungen zwischen den Orten einer Tour
    for f=1:TAnzZug(g)-1
       DistC(g,f)=DistanzOORG(Touren(g,f),Touren(g,f+1));
    end
end
DistanzGesamt1=0;
for d=1:t
    DistA2=DistA2+DistA(d);
    DistB2=DistB2+DistB(d);
end
DistanzGesamt1=DistA2+DistB2;
DistanzGesamt2=0;
for s=1:t
    for a=1:TAnzZug(s)-1
       DistanzGesamt2=DistanzGesamt2+DistC(s,a);
    end
end
Distanzgesamt=DistanzGesamt2+DistanzGesamt1;
Distanz=Distanzgesamt %Ausgeben Ergebnis
Tourenplan=Touren; %AusgebenTouren
Kapazitaetsbelastung=Kap; %Ausgeben zugeordneter Kapazität
%Ende NNA-Skript ==========================================================================================================

%NNB- Skript ==========================================================================================================
clear *;%lösche Variablen
testlauf=0;
if testlauf==0;
%Einlesen Koordinaten
%KoordinatenOrte1 = xlsread('C1.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1 = xlsread('C2.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1= xlsread('C3.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1 = xlsread('C4.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1 = xlsread('C6.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1 = xlsread('C8.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1 = xlsread('C10.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1 = xlsread('F1.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
```

```
%KoordinatenOrte1 = xlsread('F2.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1 = xlsread('F3.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1 = xlsread('ATT48.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1 = xlsread('GJ1.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1 = xlsread('eil22.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1 = xlsread('eil23.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1 = xlsread('eil30.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
KoordinatenOrte1 = xlsread('eil33.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%Eingelesene Koordinaten verarbeiten
KoordinatenOrte2(:,1)=KoordinatenOrte1(:,1);
KoordinatenOrte2(:,2)=KoordinatenOrte1(:,2);
KoordinatenOrte2(:,3)=KoordinatenOrte1(:,3);
KoordinatenOrte=KoordinatenOrte2';
[Koordinaten AnzahlOrte]=size(KoordinatenOrte);
[inputs, patterns]= size(KoordinatenOrte)
%Fahrzeugkapazität und Original-Standort setzen
FZKap=8000;% %Setzen der Fahrzeugkapazität
%OrgSta=[30 40];%Original-Standort Depot
%=========================================================Steiner-Weber-Problem, Weiszfeld-Algorithmus=========================================
%Gewichtung der Distanzen bestimmen
Gewichtung=1;
if Gewichtung==0
  b=ones(1,AnzahlOrte); %keine Gewichtung der eukl. Entfernung
else
  b1(:,1)=KoordinatenOrte1(:,3); %Bei Ermittlung des Weber Punktes für alle Eingabeorte der Testinstanz mit Gewichtung
  b=b1'; %Matrix transponieren
end
bold=b;
x(1,1)=8.5; %beliebiger Startort x-koordinate
y(1,1)=51;  %beliebiger Startort y-koordinate
for counterIterationen=1:100   %Anzahl durchzuführender Iterationen
      SummeX=0; %Nullsetzen der Summen
      SummeY=0;
      SummeX2=0;
      SummeY2=0;
      for i=1:AnzahlOrte %Eigentliche Iteration nach Weiszfeld
        if counterIterationen==1
          SummeX=SummeX+((b(1,i)*KoordinatenOrte(1,i)/((x(1,counterIterationen)-KoordinatenOrte(1,i))^2+(y(1,counterIterationen)-KoordinatenOrte(2,i)+10^(-
              10))^2)^.5));
          SummeX2=SummeX2+(b(1,i)/(((x(1,counterIterationen)-KoordinatenOrte(1,i))^2+(y(1,counterIterationen)-KoordinatenOrte(2,i)+10^(-10))^2)^.5));
          SummeY=SummeY+((b(1,i)*KoordinatenOrte(2,i)/((x(1,counterIterationen)-KoordinatenOrte(1,i))^2+(y(1,counterIterationen)-KoordinatenOrte(2,i)+10^(-
              10))^2)^.5));
          SummeY2=SummeY2+(b(1,i)/(((x(1,counterIterationen)-KoordinatenOrte(1,i))^2+(y(1,counterIterationen)-KoordinatenOrte(2,i)+10^(-10))^2)^.5));
        else
          SummeX=SummeX+((b(1,i)*KoordinatenOrte(1,i)/((x(1,counterIterationen-1)-KoordinatenOrte(1,i))^2+(y(1,counterIterationen-1)-KoordinatenOrte(2,i)+10^(-
              10))^2)^.5));
          SummeX2=SummeX2+((b(1,i)/((x(1,counterIterationen-1)-KoordinatenOrte(1,i))^2+(y(1,counterIterationen-1)-KoordinatenOrte(2,i)+10^(-10))^2)^.5));
          SummeY=SummeY+((b(1,i)*KoordinatenOrte(2,i)/((x(1,counterIterationen-1)-KoordinatenOrte(1,i))^2+(y(1,counterIterationen-1)-KoordinatenOrte(2,i)+10^(-
              10))^2)^.5));
          SummeY2=SummeY2+((b(1,i)/((x(1,counterIterationen-1)-KoordinatenOrte(1,i))^2+(y(1,counterIterationen-1)-KoordinatenOrte(2,i)+10^(-10))^2)^.5));
        end
      end
      x(1,counterIterationen)=SummeX/SummeX2; %Eintragen des Verlaufes der Annäherung
      y(1,counterIterationen)=SummeY/SummeY2;
end
WeberX=x(1,counterIterationen) %Ausgabe der Endergebnisse
WeberY=y(1,counterIterationen)
WeberDist=0;
for counterWeberDist=1:AnzahlOrte %Berechnen der euklidischen Distanz des Steiner-Weber-Punktes zu den Anfangs- und Endkunden
      WeberDist=WeberDist+((WeberX-KoordinatenOrte(1,counterWeberDist))^2+(WeberY-KoordinatenOrte(2,counterWeberDist))^2)^.5;
end
WeberDistold=WeberDist;
%=============================================================================================================================================
end %Testlauf-Abfrage
if testlauf==1
FZKap=10;
KoordinatenOrte2=[1 2 5;3 4 5;5 6 5]
Depot=[0 0];
[inputs, patterns]= size(KoordinatenOrte2'); %hier wird die Dimension von x ausgelesen
else
   Depot=[WeberX WeberY]
   Depotold=Depot;
end
Touren=zeros(patterns,patterns);
Kap=zeros(patterns,1);
%Distanzen zwischen Orten berechnen
for i=1:patterns
   for j=1:patterns
      DistanzO(i,j)=((KoordinatenOrte2(i,1)-KoordinatenOrte2(j,1))^2+(KoordinatenOrte2(i,2)-KoordinatenOrte2(j,2))^2)^.5;
      if i==j
         DistanzO(i,j)=100000;
      end
   end
end
DistanzOORG=DistanzO;
%Distanzen zwischen Orten und Depot berechnen
for k=1:patterns
   DistanzD(1,k)=((KoordinatenOrte2(k,1)-Depot(1,1))^2+(KoordinatenOrte2(k,2)-Depot(1,2))^2)^.5;
end
DistanzDORG=DistanzD;
%=============================================================================
%Giant-Route konstruieren
[Zeile NextOrt]=find(DistanzD==min(DistanzD));
GiantAnzZug=1;%Erhöhen der der Giant-Tour zugeordneten Orte
Giant(1,GiantAnzZug)=NextOrt;%Einfügen in Tour
DistanzD(1,NextOrt(1,1))=200000;
DistanzO(:,NextOrt(1,1))=200000;
LetzterOrt=NextOrt(1,1);
while GiantAnzZug<patterns
    DistanzZO=DistanzO(LetzterOrt,:);
   [Zeile NextOrt]=find(DistanzZO==min(DistanzZO));
    DistanzD(NextOrt(1,1))=200000; %könnte wegfallen
    DistanzO(:,NextOrt(1,1))=200000;
```

```
    GiantAnzZug=GiantAnzZug+1;%Erhöhen der der Tour zugeordneten Orte
    if GiantAnzZug>1
        DistanzO(LetzterOrt,:)=200000;
    end
    LetzterOrt=NextOrt(1,1);%Ort der verlassen wurde darf nicht mehr verlassen werden
    Giant(1,GiantAnzZug)=NextOrt(1,1);%Einfügen in Tour
end
%Aufspalten der Giant-Route
t=1;
ZaehZugO=zeros(patterns,1);%einer tour zugeordneten orte
for m=1:patterns
    if Kap(t,1)+KoordinatenOrte2(Giant(1,m),3)<=FZKap
        Kap(t,1)=Kap(t,1)+KoordinatenOrte2(Giant(1,m),3);
        ZaehZugO(t)=ZaehZugO(t)+1;%Anzahl zugeordneter Orte für Tour t erhöhen
        Touren(t,ZaehZugO(t))=Giant(1,m);
    else
        t=t+1;%Eröffnung neuer Tour
        Kap(t,1)=Kap(t,1)+KoordinatenOrte2(Giant(1,m),3);
        ZaehZugO(t)=ZaehZugO(t)+1;%Anzahl zugeordneter Orte für Tour t erhöhen
        Touren(t,ZaehZugO(t))=Giant(1,m);
    end
end
%=============================================
verbesserung=1;
if verbesserung==1
   % Weiszfeld-Weber für berechneten Tourenplan durchführen
   b=1; %alten Gewichtungsvektor eliminieren
   %Koordinaten der ersten und letzten Orte bestimmen
   for h=1:t
      if h==1
         KoordinatenOrteN=KoordinatenOrte(:,Touren(h,1));
         b(1,h)=1; %Gewichtungsfaktor bestimmen
      else
         KoordinatenOrteN=[KoordinatenOrteN(:,:) KoordinatenOrte(:,Touren(h,1))]; %hier werden die Daten der ersten Orte einer Tour eingetragen
         b=[b(:,:) 1];
      end
      if ZaehZugO(h,1)>1
         KoordinatenOrteN=[KoordinatenOrteN(:,:) KoordinatenOrte(:,Touren(h,ZaehZugO(h,1)))] ; % hier werden die Daten des jeweils letzten Ortes einer Tour ausgelesen
         b=[b(:,:) 1]; %Gewichtungsfaktor für Ort eintragen
      else
          [AnzGew GEW]=size(b);
          b(1,GEW)=2; %Gewichtungsfaktor für Stichfahrten auf den Wert 2 setzen;
      end
end
[KoordinatenT AnzahlOrteN]=size(KoordinatenOrteN);
x(1,1)=8.5; %beliebiger Startort x-Koordinate
y(1,1)=51;  %beliebiger Startort y-Koordinate
for counterIterationen=1:1000  %Anzahl durchzuführender Iterationen
       SummeX=0; %Nullsetzen der Summen
       SummeY=0;
       SummeX2=0;
       SummeY2=0;
       for i=1:AnzahlOrteN %Eigentliche Iteration nach Weiszfeld
          if counterIterationen==1
             SummeX=SummeX+((b(1,i)*KoordinatenOrteN(1,i)/((x(1,counterIterationen)-KoordinatenOrteN(1,i))^2+(y(1,counterIterationen)-KoordinatenOrteN(2,i)+10^(-
                 10))^2)^.5));
             SummeX2=SummeX2+(b(1,i)/(((x(1,counterIterationen)-KoordinatenOrteN(1,i))^2+(y(1,counterIterationen)-KoordinatenOrteN(2,i)+10^(-10))^2)^.5));
             SummeY=SummeY+((b(1,i)*KoordinatenOrteN(2,i)/((x(1,counterIterationen)-KoordinatenOrteN(1,i))^2+(y(1,counterIterationen)-KoordinatenOrteN(2,i)+10^(-
                 10))^2)^.5));
             SummeY2=SummeY2+(b(1,i)/(((x(1,counterIterationen)-KoordinatenOrteN(1,i))^2+(y(1,counterIterationen)-KoordinatenOrteN(2,i)+10^(-10))^2)^.5));
          else
             SummeX=SummeX+((b(1,i)*KoordinatenOrteN(1,i)/((x(1,counterIterationen-1)-KoordinatenOrteN(1,i))^2+(y(1,counterIterationen-1)-KoordinatenOrteN(2,i)+10^(-
                 10))^2)^.5));
             SummeX2=SummeX2+((b(1,i)/((x(1,counterIterationen-1)-KoordinatenOrteN(1,i))^2+(y(1,counterIterationen-1)-KoordinatenOrteN(2,i)+10^(-10))^2)^.5));
             SummeY=SummeY+((b(1,i)*KoordinatenOrteN(2,i)/((x(1,counterIterationen-1)-KoordinatenOrteN(1,i))^2+(y(1,counterIterationen-1)-KoordinatenOrteN(2,i)+10^(-
                 10))^2)^.5));
             SummeY2=SummeY2+((b(1,i)/((x(1,counterIterationen-1)-KoordinatenOrteN(1,i))^2+(y(1,counterIterationen-1)-KoordinatenOrteN(2,i)+10^(-10))^2)^.5));
          end
       end
       x(1,counterIterationen)=SummeX/SummeX2; %Eintragen des Verlaufes der Annäherung
       y(1,counterIterationen)=SummeY/SummeY2;
end
WeberX=x(1,counterIterationen) %Ausgabe der Endergebnisse
WeberY=y(1,counterIterationen)
WeberDist=0;
for counterWeberDist=1:AnzahlOrteN %Berechnen der euklidischen Distanz des Steiner-Weber-Punktes zu den Anfangs- und Endkunden
    WeberDist=WeberDist+((WeberX-KoordinatenOrteN(1,counterWeberDist))^2+(WeberY-KoordinatenOrteN(2,counterWeberDist))^2)^.5;
end
%=============================================
Depot=[WeberX WeberY]
%Distanzen zwischen Orten und neuem Depot-Standort berechnen
for k=1:patterns
   DistanzDN(1,k)=((KoordinatenOrte2(k,1)-Depot(1,1))^2+(KoordinatenOrte2(k,2)-Depot(1,2))^2)^.5;
end
DistanzDORG=DistanzDN;
end
%Distanz berechnen
DistA=zeros(1);
DistB=zeros(1);
DistC=zeros(1);
DistA2=0;
DistB2=0;
for g=1:t
    %Entfernungen vom Depot zum Ort1 jeder Tour
     DistA(1,g)=DistanzDORG(Touren(g,1));
    %Entfernung vom letzten Ort zum Depot
     DistB(1,g)=DistanzDORG(Touren(g,ZaehZugO(g)));
    %Entfernungen zwischen den Orten einer Tour
    for f=1:ZaehZugO(g)-1
        DistC(g,f)=DistanzOORG(Touren(g,f),Touren(g,f+1));
    end
end
DistanzGesamt1=0;
```

```
for d=1:t
   DistA2=DistA2+DistA(d);
   DistB2=DistB2+DistB(d);
end
DistanzGesamt1=DistA2+DistB2;
DistanzGesamt2=0;
for s=1:t
   for a=1:ZaehZugO(s)-1
      DistanzGesamt2=DistanzGesamt2+DistC(s,a);
   end
end
Distanzgesamt=DistanzGesamt2+DistanzGesamt1;
Distanz=Distanzgesamt %Ausgeben Ergebnis
Tourenplan=Touren; %AusgebenTouren
Kapazitaetsbelastung=Kap; %Ausgeben zugeordneter Kapazität
%Ende NNB- Skript ==========
```

```
%Call-Skript PSV==========
clear;
BestGesamtdistanz=10000;
for counterpsv1=1:30
   for counterpsv2=1:10
      [Gesamtdistanz]=wwpsvalter(counterpsv1,counterpsv2);
      if Gesamtdistanz<BestGesamtdistanz
         BestGesamtdistanz=Gesamtdistanz;
         BestGewichtKK=0.1*counterpsv1;
         BestGewichtDK=0.1*counterpsv2;
      end
      GesamtdistanzPSV(counterpsv1,counterpsv2)=Gesamtdistanz;
   end
end
showBestGesamtdistanz=BestGesamtdistanz
SBestGewichtKK=BestGewichtKK
SBestGewichtDK=BestGewichtDK
%Ende Call-Skript PSV==========
```

```
%PSV- Skript ==========
function [Gesamtdistanz]=wwpsvalter(counterpsv1,counterpsv2)
%KoordinatenOrte1 = xlsread('C1.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1 = xlsread('C2.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1= xlsread('C3.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1 = xlsread('C4.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1 = xlsread('C6.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1 = xlsread('C8.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1 = xlsread('C10.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1 = xlsread('F1.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1 = xlsread('F2.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
KoordinatenOrte1 = xlsread('F3.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1 = xlsread('GJ1.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1 = xlsread('eil22.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1 = xlsread('eil23.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1 = xlsread('eil30.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
%KoordinatenOrte1 = xlsread('eil33.xls');%liest aus einem Excel-Spreadsheet die geograph. Positionen
FZKap=2210;% %Berechnen der Fahrzeugkapazität
GewichtKK=0.1*counterpsv1;% .8, 0.1, 2.0
GewichtDK=0.1*counterpsv2;% 0.0, 0.1, 1.0
%OrgSta=[30 40];%Original-Standort Depot
KoordinatenOrte2(:,1)=KoordinatenOrte1(:,1);
KoordinatenOrte2(:,2)=KoordinatenOrte1(:,2);
KoordinatenOrte=KoordinatenOrte2';
[Koordinaten AnzahlOrte]=size(KoordinatenOrte);
schalterweber=1;
if schalterweber==1
  Gewichtung=1;
  if Gewichtung==0
    b=ones(1,AnzahlOrte); % Gewichtung der eukl. Entfernung=1
  else
    b1(:,1)=KoordinatenOrte1(:,3); %Bei Ermittlung des Weber Punktes für alle Eingabeorte der Testinstanz mit Gewichtung
    b=b1'; %Matrix transponieren
  end
  bold=b;
  x(1,1)=8.5; %beliebiger Startort x-Koordinate
  y(1,1)=51;  %beliebiger Startort y-Koordinate
  for counterIterationen=1:100   %Anzahl durchzuführender Iterationen
      SummeX=0; %Nullsetzen der Summen
      SummeY=0;
      SummeX2=0;
      SummeY2=0;
      for i=1:AnzahlOrte %Eigentliche Iteration nach Weiszfeld
        if counterIterationen==1
          SummeX=SummeX+((b(1,i)*KoordinatenOrte(1,i)/(((x(1,counterIterationen)-KoordinatenOrte(1,i))^2+(y(1,counterIterationen)-KoordinatenOrte(2,i)+10^(-
              10))^2)^.5));
          SummeX2=SummeX2+(b(1,i)/(((x(1,counterIterationen)-KoordinatenOrte(1,i))^2+(y(1,counterIterationen)-KoordinatenOrte(2,i)+10^(-10))^2)^.5));
          SummeY=SummeY+((b(1,i)*KoordinatenOrte(2,i)/(((x(1,counterIterationen)-KoordinatenOrte(1,i))^2+(y(1,counterIterationen)-KoordinatenOrte(2,i)+10^(-
              10))^2)^.5));
          SummeY2=SummeY2+(b(1,i)/(((x(1,counterIterationen)-KoordinatenOrte(1,i))^2+(y(1,counterIterationen)-KoordinatenOrte(2,i)+10^(-10))^2)^.5));
        else
          SummeX=SummeX+((b(1,i)*KoordinatenOrte(1,i)/(((x(1,counterIterationen-1)-KoordinatenOrte(1,i))^2+(y(1,counterIterationen-1)-KoordinatenOrte(2,i)+10^(-
              10))^2)^.5));
          SummeX2=SummeX2+((b(1,i)/(((x(1,counterIterationen-1)-KoordinatenOrte(1,i))^2+(y(1,counterIterationen-1)-KoordinatenOrte(2,i)+10^(-10))^2)^.5));
          SummeY=SummeY+((b(1,i)*KoordinatenOrte(2,i)/(((x(1,counterIterationen-1)-KoordinatenOrte(1,i))^2+(y(1,counterIterationen-1)-KoordinatenOrte(2,i)+10^(-
              10))^2)^.5));
          SummeY2=SummeY2+((b(1,i)/(((x(1,counterIterationen-1)-KoordinatenOrte(1,i))^2+(y(1,counterIterationen-1)-KoordinatenOrte(2,i)+10^(-10))^2)^.5));
        end
      end
       x(1,counterIterationen)=SummeX/SummeX2; %Eintragen des Verlaufes der Annäherung
      y(1,counterIterationen)=SummeY/SummeY2;
    end
    WeberX=x(1,counterIterationen); %Ausgabe der Endergebnisse
    WeberY=y(1,counterIterationen);
    WeberDist=0;
    for counterWeberDist=1:AnzahlOrte %Berechnen der euklidischen Distanz des Steiner-Weber-Punktes zu den Anfangs- und Endkunden
      WeberDist=WeberDist+((WeberX-KoordinatenOrte(1,counterWeberDist))^2+(WeberY-KoordinatenOrte(2,counterWeberDist))^2)^.5;
```

```
    end
    WeberDistold=WeberDist;
end
%====================================%Savings-Verfahren mit Kapazitätsbeschränkungen=============
Iter=0; %Setze den Zähler Iteration auf den Wert Null
Gesamtdistanz=0;%Wert der Gesamtdistanz auf Null setzen
[Nachfrage] = KoordinatenOrte1(:,3);
xmedian=[WeberX WeberY];%Ermittlung der Weber-Lösung für die Eingabedaten
xmedianold=xmedian;%Speichern des ersten Standortes für das Depot
[inputs, patterns]= size(KoordinatenOrte1'); %hier wird die Dimension von x ausgelesen
DKVerbindungen=2*ones(patterns,1);%Matrix zur Erfassung der bestehenden Verbindungen vom Depot zu den Kunden
%Matrix zur Erfassung der benutzten Verbindungen vom Kunden zum Depot
KKVerbindungen=zeros(patterns,patterns);%Matrix zur Erfassung der benutzten Verbindungen von Kundenorten zu Kundenorten
OrtTour=1:patterns; %jeder Ort hat zu Beginn eine eigene Tour (Stichfahrt)
Kapazitaet=zeros(1,patterns);
Kapazitaet(1,:)=FZKap;
%Ermitteln der Distanzen vom Depot zu jedem Kunden
for counter1=1:patterns;
    DistDepot(counter1)=((KoordinatenOrte1(counter1,1)-xmedian(1,1))^2+(KoordinatenOrte1(counter1,2)-xmedian(1,2))^2)^.5;%Eukl. Distanz zw. Depot und Kunden
end;
%Berechnen der Distanzen zwischen den Kunden
for counter2=1:patterns;
    for counter3=1:patterns;
        DistKunden(counter2,counter3)=((KoordinatenOrte1(counter2,1)-KoordinatenOrte1(counter3,1))^2+(KoordinatenOrte1(counter2,2)-
            KoordinatenOrte1(counter3,2))^2)^.5;%Berechnung der euklidischen Distanz zwischen allen Kundenorten
        if counter2==counter3;
          DistKunden(counter2,counter3)=-1;%Die Distanz von Ort i zu Ort i wird auf -1 gesetzt
        end;
    end;
end;
%Berechnung der Savings
for counter4=1:patterns;
    for counter5=1:patterns;
        if counter4<counter5
          Savings(counter4,counter5)=DistDepot(counter4)+DistDepot(counter5)-GewichtKK*DistKunden(counter4,counter5)+GewichtDK*(DistDepot(counter4)-
              DistDepot(counter5));%Berechnung der Savings
        else
           Savings(counter4,counter5)=-1;%Da die Distanzen symmetrisch sind, muss nur eine Hälfte der Matrix verwendet werden
        end
   end;
end;
%Zusammenlegung der Touren
for counter6=1:(round((patterns^2-patterns)/2));%diese Schleife wird für die maximale Anzahl durchlaufen
        [mx,r]=max(Savings); %Ermittlung des aktuell maximalen Savings
        mmx=max(mx);
        if mmx>0 ;%wenn der maximale Saving >0 ist
          [r,c]=find(Savings==mmx);%finde Reihe und Spalte
          i=r(1); %i=Reihe
          j=c(1); %j=Spalte, d.h. wenn j in die Tour von i aufgenommen wird ergibt sich der Savings-Wert (i muss aufgrund der Matrix Savings hier kleiner j sein!)

          if  DKVerbindungen(i)==1 | DKVerbindungen(i)==2;%Überprüfen, ob i ein Endkunde einer Tour darstellt
              if  DKVerbindungen(j)==1 | DKVerbindungen(j)==2;%Überprüfen, ob j ein Endkunde einer Tour darstellt
                  if  OrtTour(j)~=OrtTour(i);   %i und j müssen Endkunden zweier unterschiedlicher Touren sein.
                  %Zuweisung des Ortes j zur Tour des Ortes i
                     y=find(OrtTour==OrtTour(j));% Ermitteln, welche Orte in der Tour des Ortes j enthalten sind
                     KapazitaetTourj=0;
                     for counter7=1:length(y);
                        KapazitaetTourj=KapazitaetTourj+Nachfrage(y(counter7));%Kapazitätsverbrauch der Orte in Tour des Ortes j
                     end
                     z=find(OrtTour==OrtTour(i));%Ermitteln, welche Orte in der Tour des Ortes i enthalten sind
                     KapazitaetTouri=0;
                     for counter8=1:length(z)
                        KapazitaetTouri=KapazitaetTouri+Nachfrage(z(counter8));%Kapazitätsverbrauch der Orte in Tour des Ortes i
                     end
                     if (Kapazitaet(OrtTour(i))-KapazitaetTourj-KapazitaetTouri)<0 ;%Die Kapazität für eine Tour darf nicht überschritten werden
                        Savings(i,j)=-1;%Savings in der Matrix auf -1 setzen, da Verbindung der Touren aufgrund der Kapazitätsbedingungen nicht in Frage kommt
                     else
                        u=find(OrtTour==OrtTour(j));
                        for counter14=1:length(u);
                           OrtTour(u(counter14))=OrtTour(i); %nehme alle Orte der Tour des Ortes j in die Tour von Ort i auf
                     end
                      DKVerbindungen(j)=DKVerbindungen(j)-1;%Verminderung der Anzahl der Verbindungen zum Depot von Ort j
                      DKVerbindungen(i)=DKVerbindungen(i)-1;%Verminderung der Anzahl der Verbindungen vom Depot zum Ort i
                      KKVerbindungen(i,j)=KKVerbindungen(i,j)+1;%Setzen der Verbindung zwischen i und j
                      KKVerbindungen(j,i)=KKVerbindungen(j,i)+1;%Setzen der Verbindung zwischen j und 1
                      Savings(i,j)=-1;%Savings in der Matrix auf -1 setzen
                  end
              else
                  Savings(i,j)=-1;%Savings auf -1 setzen, da die Orte nicht direkt verbunden werden dürfen
               end
          else
              Savings(:,j)=-1;%alle Savings in der Spalte j der Matrix auf -1 setzen, da j kein Endknoten darstellt
          end
    else
        Savings(i,:)=-1;%alle Savings in der Reihe i der Matrix auf -1 setzen, da i kein Endknoten darstellt
     end
  end
end
Iter=counter6;
%=========================Verbesserung des Standortes für den berechneten Tourenplan durch WW-Routine=================
verbesserung=1;
if verbesserung==1
KoordinatenOrteN=[0;0];
p=1;
b=0;
for v=1:AnzahlOrte
    if DKVerbindungen(v,1)==1;
      KoordinatenOrteN(:,p)=KoordinatenOrte(:,v);
      b(1,p)=1;
      p=p+1;
    elseif DKVerbindungen(v,1)==2;
       KoordinatenOrteN(:,p)=KoordinatenOrte(:,v);
       b(1,p)=2;
```

```
        p=p+1;
    end
end
[KoordinatenN AnzahlOrteN]=size(KoordinatenOrteN);
x(1,1)=8.5; %beliebiger Startort x-Koordinate
y(1,1)=51;  %beliebiger Startort y-Koordinate
for counterIterationen=1:100   %Anzahl durchzuführender Iterationen
    SummeX=0; %Nullsetzen der Summen
    SummeY=0;
    SummeX2=0;
    SummeY2=0;
    for i=1:AnzahlOrteN %Eigentliche Iteration nach Weiszfeld
        if counterIterationen==1
            SummeX=SummeX+((b(1,i)*KoordinatenOrteN(1,i)/((x(1,counterIterationen)-KoordinatenOrteN(1,i))^2+(y(1,counterIterationen)-KoordinatenOrteN(2,i)+10^(-
                10))^2)^.5));
            SummeX2=SummeX2+(b(1,i)/(((x(1,counterIterationen)-KoordinatenOrteN(1,i))^2+(y(1,counterIterationen)-KoordinatenOrteN(2,i)+10^(-10))^2)^.5));
            SummeY=SummeY+((b(1,i)*KoordinatenOrteN(2,i)/((x(1,counterIterationen)-KoordinatenOrteN(1,i))^2+(y(1,counterIterationen)-KoordinatenOrteN(2,i)+10^(-
                10))^2)^.5));
            SummeY2=SummeY2+(b(1,i)/(((x(1,counterIterationen)-KoordinatenOrteN(1,i))^2+(y(1,counterIterationen)-KoordinatenOrteN(2,i)+10^(-10))^2)^.5));
        else
            SummeX=SummeX+((b(1,i)*KoordinatenOrteN(1,i)/((x(1,counterIterationen-1)-KoordinatenOrteN(1,i))^2+(y(1,counterIterationen-1)-KoordinatenOrteN(2,i)+10^(-
                10))^2)^.5));
            SummeX2=SummeX2+((b(1,i)/((x(1,counterIterationen-1)-KoordinatenOrteN(1,i))^2+(y(1,counterIterationen-1)-KoordinatenOrteN(2,i)+10^(-10))^2)^.5));
            SummeY=SummeY+((b(1,i)*KoordinatenOrteN(2,i)/((x(1,counterIterationen-1)-KoordinatenOrteN(1,i))^2+(y(1,counterIterationen-1)-KoordinatenOrteN(2,i)+10^(-
                10))^2)^.5));
            SummeY2=SummeY2+((b(1,i)/((x(1,counterIterationen-1)-KoordinatenOrteN(1,i))^2+(y(1,counterIterationen-1)-KoordinatenOrteN(2,i)+10^(-10))^2)^.5));
        end
    end
    x(1,counterIterationen)=SummeX/SummeX2; %Eintragen des Verlaufes der Annäherung
    y(1,counterIterationen)=SummeY/SummeY2;
  end
  WeberX=x(1,counterIterationen); %Ausgabe der Endergebnisse
  WeberY=y(1,counterIterationen);
  WeberDist=0;
  for counterWeberDist=1:AnzahlOrteN %Berechnen der euklidischen Distanz des Steiner-Weber-Punktes zu den Anfangs- und Endkunden
    WeberDist=WeberDist+((WeberX-KoordinatenOrteN(1,counterWeberDist))^2+(WeberY-KoordinatenOrteN(2,counterWeberDist))^2)^.5;
  end
  xmedian=[WeberX WeberY];%Festlegung des Standortes auf neue Koordinaten
end %Ende der Abfrage, ob Verbesserung stattfinden soll
%================================Graphische Ausgabe der Lösung=============================================================
schaltersavplot=0;
if schaltersavplot==1
  LoesungSav=figure;%erstellen einer Figur
  LoesungSav=newplot;%Wahl der Figur Loesung
  hold on;%Halten der eingezeichneten Objekte
  plot(KoordinatenOrte1(:,1),KoordinatenOrte1(:,2),'.b','markersize',20); %Einzeichnen der Bedarfsorte
  plot(xmedian(1,1),xmedian(1,2),'.r','markersize',10); %Einzeichnen des Depots
  Tourpkt=1:100; %dient zur Zeichnung der Verbindungen, Matrix mit den Eintragungen 1 bis 100
  Tourpkt=0.01*Tourpkt; %Skalierung auf Werte zwischen 0,01 und 1
end
%Kundenverbindungen zeichnen
for counter9=1:patterns
  for counter10=1:patterns
    if counter9<counter10
      if KKVerbindungen(counter9,counter10)==1
        Gesamtdistanz=Gesamtdistanz+((KoordinatenOrte1(counter9,1)-KoordinatenOrte1(counter10,1))^2+(KoordinatenOrte1(counter9,2)-
            KoordinatenOrte1(counter10,2))^2)^.5;%Erhöhung der euklidischen Gesamtdistanz
        if schaltersavplot==1
          for counter11=1:100 % Ermitteln von 100 Punkten auf der Verbindungsstrecke zwischen verbundenen Kundenorten
            xkoord(counter11)=((1-Tourpkt(counter11))*KoordinatenOrte1(counter9,1)+Tourpkt(counter11)*KoordinatenOrte1((counter10),1));
            ykoord(counter11)=((1-Tourpkt(counter11))*KoordinatenOrte1(counter9,2)+Tourpkt(counter11)*KoordinatenOrte1((counter10),2));
          end
        LoesungSav=newplot;%Wahl der Figur Loesung
        hold on
        plot(xkoord,ykoord,'r-');%Einzeichnen der Verbindungsstrecke zwischen den Nachfrageorten
        end
      end
    end
  end
end
%Verbindungen Depot-Kunden zeichnen
for counter12=1:patterns
  if DKVerbindungen(counter12)==1|DKVerbindungen(counter12)==2
    Gesamtdistanz=Gesamtdistanz+DKVerbindungen(counter12)*((KoordinatenOrte1(counter12,1)-xmedian(1,1))^2+(KoordinatenOrte1(counter12,2)-
            xmedian(1,2))^2)^.5;%Erhöhung der euklidischen Gesamtdistanz um das 1 oder 2 fache der eukl.Entfernung
    if schaltersavplot==1
      for counter13=1:100 % Ermitteln von 100 Punkten auf der Verbindungsstrecke zwischen verbundenen Kundenorten
        xkoordDK(counter13)=((1-Tourpkt(counter13))*KoordinatenOrte1(counter12,1)+Tourpkt(counter13)*xmedian(1,1));
        ykoordDK(counter13)=((1-Tourpkt(counter13))*KoordinatenOrte1(counter12,2)+Tourpkt(counter13)*xmedian(1,2));%
      end
      LoesungSav=newplot;%Wahl der Figur Loesung
      hold on
      plot(xkoordDK,ykoordDK,'r-');%Einzeichnen der Verbindungsstrecke zwischen Depot/Nachfrageort
    end
  end
end
Gesamtdistanz=Gesamtdistanz;%Ausgabe Gesamtdistanz
% Ende PSV- Skript ==========================================================================================================
```

E. Kurzzyklenbedingungen für das MTSP

Die von SVESTKA und HUCKFELDT [1973] vorgeschlagenen Bedingungen zur Verhinderung von Kurzzyklen für die zu besuchenden Knoten (Kunden) lauten[402]:

$$\pi_m - \pi_n + \left\lfloor \frac{N}{R} \right\rfloor \cdot x_{mn} \leq \left\lfloor \frac{N}{R} \right\rfloor + R - 2 \qquad \forall m \neq n \wedge m,n \notin \Lambda_0 = \{1,2,...,R\}.$$

Gegeben sei nun die in Abbildung A.16 abgebildete unzulässige Lösung, in der die Knoten 1 und 2 das Depot repräsentieren. Die Knoten 3 bis 6 stellen die von R=2 Handlungsreisenden zu besuchenden Knoten dar. Da ein künstliches Depot eingeführt wurde, beträgt die Anzahl der ursprünglich vorhandenen Knoten N=5.

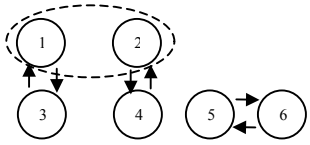


Abbildung A.16: Unzulässige Lösung für das MTSP

Für diese unzulässige Lösung gilt $x_{13} = x_{31} = x_{24} = x_{42} = x_{56} = x_{65} = 1$, die übrigen Variablen nehmen den Wert Null an. Während die Kurzzyklen zwischen den Knoten 3 und 4 isoliert betrachtet zulässige Kurzzyklen für das MTSP darstellen, gilt dies nicht für den Kurzzyklus zwischen den Knoten 5 und 6. Somit ist die skizzierte Lösung für das MTSP unzulässig und müsste durch die Kurzzyklenbedingungen ausgeschlossen werden, da die übrigen Nebenbedingungen des Modells erfüllt werden: Jeder Knoten wird genau einmal angefahren und genau einmal verlassen, die Binärbedingungen für die Variablen werden ebenfalls erfüllt.

Insgesamt ergeben sich für die von SVESTKA und HUCKFELDT vorgeschlagene Formulierung $5^2 - 3 \cdot 5 + 2 = 12$ Kurzzyklenbedingungen:

m=3,	n=4	$\pi_3 - \pi_4 + \left\lfloor \frac{5}{2} \right\rfloor \cdot x_{34} \leq \left\lfloor \frac{5}{2} \right\rfloor + 2 - 2$	$\pi_3 - \pi_4 \leq 2$
	n=5	$\pi_3 - \pi_5 + 2 \cdot x_{35} \leq 2$	$\pi_3 - \pi_5 \leq 2$
	n=6	$\pi_3 - \pi_6 + 2 \cdot x_{36} \leq 2$	$\pi_3 - \pi_6 \leq 2$
m=4,	n=3	$\pi_4 - \pi_3 + 2 \cdot x_{43} \leq 2$	$\pi_4 - \pi_3 \leq 2$
	n=5	$\pi_4 - \pi_5 + 2 \cdot x_{45} \leq 2$	$\pi_4 - \pi_5 \leq 2$
	n=6	$\pi_4 - \pi_6 + 2 \cdot x_{46} \leq 2$	$\pi_4 - \pi_6 \leq 2$

[402] Vgl. Svestka und Huckfeldt [1973], S. 791.

m=5,	n=3	$\pi_5 - \pi_3 + 2 \cdot x_{53} \leq 2$	$\pi_5 - \pi_3 \leq 2$
	n=4	$\pi_5 - \pi_4 + 2 \cdot x_{54} \leq 2$	$\pi_5 - \pi_4 \leq 2$
	n=6	$\pi_5 - \pi_6 + 2 \cdot x_{56} \leq 2$	$\pi_5 - \pi_6 \leq 0$
m=6,	n=3	$\pi_6 - \pi_3 + 2 \cdot x_{63} \leq 2$	$\pi_6 - \pi_3 \leq 2$
	n=4	$\pi_6 - \pi_4 + 2 \cdot x_{64} \leq 2$	$\pi_6 - \pi_4 \leq 2$
	n=5	$\pi_6 - \pi_5 + 2 \cdot x_{65} \leq 2$	$\pi_6 - \pi_5 \leq 0$

Damit ergeben sich für die Differenzen zwischen den Hilfsvariablen die Forderungen

$$-2 \leq \pi_3 - \pi_4 \leq 2 ,$$

$$-2 \leq \pi_3 - \pi_5 \leq 2 ,$$

$$-2 \leq \pi_3 - \pi_6 \leq 2 ,$$

$$-2 \leq \pi_4 - \pi_5 \leq 2 ,$$

$$-2 \leq \pi_4 - \pi_6 \leq 2 ,$$

$$0 \leq \pi_5 - \pi_6 \leq 0 .$$

Nur für $\pi_5 = \pi_6$ wird die letzte Forderung erfüllt. Setzt man in den übrigen Gleichung $\pi_5 = \pi_6$, so folgt

$$-2 \leq \pi_3 - \pi_4 \leq 2 ,$$

$$-2 \leq \pi_3 - \pi_6 \leq 2 ,$$

$$\left(-2 \leq \pi_3 - \pi_6 \leq 2\right),$$

$$-2 \leq \pi_4 - \pi_6 \leq 2 ,$$

$$\left(-2 \leq \pi_4 - \pi_6 \leq 2\right).$$

Es verbleiben als Anforderungen an die Differenzen zwischen den Hilfsvariablen

$$-2 \leq \pi_3 - \pi_4 \leq 2 ,$$

$$-2 \leq \pi_3 - \pi_6 \leq 2 ,$$

$$-2 \leq \pi_4 - \pi_6 \leq 2 ,$$

die bspw. für $\pi_6 = 1$, $\pi_4 = 2$ und $\pi_3 = 3$ erfüllt werden. Damit wird die unzulässige Lösung durch die Kurzzyklenbedingungen nicht ausgeschlossen.

Verwendet man hingegen die in dieser Arbeit vorgeschlagenen Kurzzyklenbedingungen

$$\pi_m - \pi_n + N \cdot x_{mn} \leq N - 1 \qquad \forall m \neq n \wedge m,n \notin \Lambda_0 = \{1,2,...,R\},$$

die sich an den Formulierungen für Tourenplanungsprobleme orientieren, so ergeben sich für das Beispiel die folgenden Bedingungen:

m=3,	n=4	$\pi_3 - \pi_4 + 5 \cdot x_{34} \leq 5 - 1$	$\pi_3 - \pi_4 \leq 4$
	n=5	$\pi_3 - \pi_5 + 5 \cdot x_{35} \leq 4$	$\pi_3 - \pi_5 \leq 4$
	n=6	$\pi_3 - \pi_6 + 5 \cdot x_{36} \leq 4$	$\pi_3 - \pi_6 \leq 4$
m=4,	n=3	$\pi_4 - \pi_3 + 5 \cdot x_{43} \leq 4$	$\pi_4 - \pi_3 \leq 4$
	n=5	$\pi_4 - \pi_5 + 5 \cdot x_{45} \leq 4$	$\pi_4 - \pi_5 \leq 4$
	n=6	$\pi_4 - \pi_6 + 5 \cdot x_{46} \leq 4$	$\pi_4 - \pi_6 \leq 4$
m=5,	n=3	$\pi_5 - \pi_3 + 5 \cdot x_{53} \leq 4$	$\pi_5 - \pi_3 \leq 4$
	n=4	$\pi_5 - \pi_4 + 5 \cdot x_{54} \leq 4$	$\pi_5 - \pi_4 \leq 4$
	n=6	$\pi_5 - \pi_6 + 5 \cdot x_{56} \leq 4$	$\pi_5 - \pi_6 \leq -1$
m=6,	n=3	$\pi_6 - \pi_3 + 5 \cdot x_{63} \leq 4$	$\pi_6 - \pi_3 \leq 4$
	n=4	$\pi_6 - \pi_4 + 5 \cdot x_{64} \leq 4$	$\pi_6 - \pi_4 \leq 4$
	n=5	$\pi_6 - \pi_5 + 5 \cdot x_{65} \leq 4$	$\pi_6 - \pi_5 \leq -1$

Damit ergeben sich für die Differenzen zwischen den Hilfsvariablen die Forderungen

$$-4 \leq \pi_3 - \pi_4 \leq 4\,,$$

$$-4 \leq \pi_3 - \pi_5 \leq 4\,,$$

$$-4 \leq \pi_3 - \pi_6 \leq 4\,,$$

$$-4 \leq \pi_4 - \pi_5 \leq 4\,,$$

$$-4 \leq \pi_4 - \pi_6 \leq 4\,,$$

$$1 \leq \pi_5 - \pi_6 \leq -1\,.$$

Die Forderung $1 \leq \pi_5 - \pi_6 \leq -1$ schließt den unzulässigen Kurzzyklus somit wie gewünscht aus.

Literaturverzeichnis

Aarts et al. [1997] Aarts, E. H. L., Korst, J. H. M., van Laarhoven, P. J. M., "Simulated annealing", in: Aarts, E. H. L., Lenstra, J. K. (Hrsg.), *Local Search in Combinatorial Optimization*, John Wiley and Sons, Chichester, S. 91-120.

Aarts und Lenstra [1997] Aarts, E. H. L., Lenstra, J. K., *Local Search in Combinatorial Optimization*, John Wiley and Sons, Chichester.

Allinson et al. [2001] Allinson, N., Yin, H., Allinson, L., Slack, J., *Advances in Self-Organising Maps*, Springer-Verlag, London.

Altinel et al. [2000] Altinel, İ. K., Aras, N., Oommen, B. J. , "Fast, efficient and accurate solutions to the Hamiltonian path problem using neural approaches", *Computers & Operations Research* **27**, S. 461-494.

Amin [1994] Amin, S., "A Self-Organised Travelling Salesman", *Neural Computing and Applications* **2**, S. 129-133.

Amin et al. [1994] Amin, S., Fernández-Villacañas, J.-L., Cochrane, P., "A Natural Solution to the Travelling Salesman Problem", *British Telecommunications Engineering* **13**, S. 117-122.

Ang et al. [2002] Ang, T. L., Tarui, Y., Sakusabe, T., Takahashi, T., Schibuya, N., "A Hybrid Force-Directed Self-Organizing Neural Network Approach to Automatic Printed Circuit Board Component Placement with EMC Consideration", *IEICE Transactions on Communications* **E85 B**, S. 1797-1805.

Angeniol et al. [1988] Angéniol, B., De La Croix Vaubois, G., Le Texier, J. Y., "Self-Organizing Feature Maps and the Travelling Salesman Problem", *Neural Networks* **1**, S. 289-293.

Aras et al. [1999] Aras, N., Oommen, B.J., Altinel, İ. K., "The Kohonen network incorporating explicit statistics and its application to the travelling salesman problem", *Neural Networks* **12**, S. 1273-1284.

Assad [1988] Assad, A. A., "Modeling and Implementation Issues in Vehicle Routing" in: Assad, A. A., Golden, B .L. (Hrsg.), *Vehicle Routing: Methods and Studies.* Elsevier Science Publishers B.V., Amsterdam.

Avella et al. [1998] Avella, P. et al.[403], "Some personal views on the current state and the future of Locational Analysis", *European Journal of Operational Research* **104**, S. 269-287.

Bartholdi und Platzman [1982] Bartholdi, J. J., Platzman, L. K., "An O(N log N) Planar Travelling Salesman Heuristic Based on Spacefilling Curves", *Operations Research Letters* **1**, S. 121-125.

Berens und Körling [1983] Berens, W., Körling, F.-J., "Das Schätzen von realen Entfernungen bei der Warenverteilungsplanung mit gebietspaarspezifischen Umwegfaktoren", *OR Spektrum* **5**, S. 67-75.

Budinich [1996] Budinich, M., "A Self-Organizing Neural Network for the Traveling Salesman Problem That Is Competitive with Simulated Annealing", *Neural Computation* **8**, S. 416-424.

[403] Die komplette Liste der insgesamt 20 Autoren wird nicht angegeben.

Budinich [1997] Budinich, M., “Neural Networks for NP-Complete Problems”, *Nonlinear Analysis, Theory, Methods & Applications* **30**, S. 1617-1624.

Burke [1994] Burke, L. I., “Neural Methods for the Traveling Salesman Problem: Insights From Operations Research”, *Neural Networks* **7,** S. 681-690.

Burke [1996] Burke. L. I., ““Conscientious” Neural Nets for Tour Construction in the Traveling Salesman Problem: The Vigilant Net”, *Computers & Operations Research* **23**, S. 121-129.

Burke und Damany [1992] Burke, L. I., Damany, P., “The Guilty Net for the Traveling Salesman Problem”, *Computers & Operations Research* **19**, S. 255-265.

Burke und Ignizio [1992] Burke, L. I., Ignizio, J. P., “Neural Networks and Operations Research: An Overview”, *Computers & Operations Research* **19**, S. 179-189.

Christofides und Eilon [1969] Christofides, N., Eilon, S., “An Algorithm for the Vehicle-dispatching Problem”, *Operational Research Quarterly* **20**, S. 309-318.

Christofides et al. [1979] Christofides, N., Mingozzi, A., Toth, P., “The Vehicle Routing Problem”, in: Christofides, N., Mingozzi, A., Toth, P., Sandi, C. (Hrsg.), *Combinatorial Optimization,* John Wiley & Sons Ltd., Chichester, S. 315-338.

Clarke und Wright [1964] Clarke, G., Wright, J. W., “Scheduling of vehicles from a central depot to a number of delivery points”, *Operations Research* **12**, S. 568-581.

Cochrane und Beasley [2003] Cochrane, E. M., Beasley, J. E., “The co-adaptive neural network approach to the Euclidean Traveling Salesman Problem”, *Neural Networks* **16**, S. 1499-1525.

Cooper [1963] Cooper, L., “Location-Allocation Problems”, *Operations Research* **11**, S. 331-343.

Cooper [1972] Cooper, L., “The Transportation-Location Problem”, *Operations Research* **20**, S. 94-108.

Cordeau et al. [2002] Cordeau, J. F., Gendreau, M., Laporte, G., Potvin, J. Y., Semet, F., “A guide to vehicle routing heuristics”, *Journal of the Operational Research Society* **53**, S. 512-522.

Corsten und May [1996] Corsten, H., May, C., “Anwendungsfelder Neuronaler Netze und ihre Umsetzung“, in: Corsten, H., May, C. (Hrsg.), *Neuronale Netze in der Betriebswirtschaft*, Gabler Verlag, Wiesbaden, S. 1-13.

Croes [1958] Croes, G. A., “A method for solving traveling salesman problems”, *Operations Research* **6** (1958), S. 791-812.

Dantzig et al. [1954] Dantzig, G. B., Fulkerson, D. R., Johnson, S. M., “Solution of a Large-Scale Traveling Salesman Problem”, *Operations Research* **2**, S. 393-410.

Dantzig und Ramser [1959] Dantzig, G. B., Ramser, J. H., „The Truck Dispatching Problem“, *Management Science* **6**, S. 80-91.

Dethloff [1994] Dethloff, J., *Verallgemeinerte Tourenplanungsprobleme*, Vandenhoeck und Ruprecht, Göttingen.

DeSieno [1988] DeSieno, D., “Adding a Conscience to Competitive Learning”, *IEEE International Conference on Neural Networks*, S. 117-124.

Domschke [1995] Domschke, W., *Logistik: Transport*, R. Oldenbourg Verlag, München.

Domschke [1997] Domschke, W., *Logistik: Rundreisen und Touren*, R. Oldenbourg Verlag, München.

Domschke und Drexl [1996] Domschke, W., Drexl, A., *Logistik: Standorte*, R. Oldenbourg Verlag, München.

Domschke und Drexl [2004] Domschke, W., Drexl, A., *Einführung in Operations Research*, Springer-Verlag, Berlin.

Drezner et al. [2002] Drezner, Z., Klamroth, K., Schöbl, A., Wesolowsky, G. O., "The Weber Problem", in: Drezner, Z., Hamacher, H. W. (Hrsg.), *Facility Location,* Springer-Verlag, Berlin, S. 1-36.

Drezner und Hamacher [2002] Drezner, Z., Hamacher, H. W., *Facility location,* Springer-Verlag, Berlin.

Durbin und Willshaw [1987] Durbin, R., Willshaw, D., "An analogue approach to the travelling salesman problem using an elastic net method", *Nature* **326**, S. 689-691.

Durbin et al. [2001] Durbin, R., Szeliski, R., Yuille, A., "An Analysis of the Elastic Net Approach to the Traveling Salesman Problem", in: Obermayer, K., Sejnowski, T. J. (Hrsg.), *Self-Organizing Map Formation,* The MIT Press, Cambridge, Massachusetts, S. 407-417.

Engele [1981] Engele, G., *Simultane Standort- und Tourenplanung*, Heymanns, Köln.

Favata und Walker [1991] Favata, F., Walker, R., "A study of the application of Kohonen-type neural networks to the Travelling Salesman Problem", *Biological Cybernetics* **64**, S. 463-468.

Fischer und Kruschwitz [1980] Fischer, J., Kruschwitz, L.: *Methodische Probleme bei der Evaluation heuristischer Lösungsverfahren*, Wirtschaftswissenschaftliche Dokumentation der Technischen Universität Berlin, Berlin.

Fisher [1994] Fisher, M. L., "Optimal Solution of Vehicle Routing Problems Using Minimum K-Trees", *Operations Research* **42**, S. 626-642.

Fort [1988] Fort, J. C., "Solving a Combinatorial Problem via Self-Organizing Process: An Application of the Kohonen Algorithm to the Traveling Salesman Problem", *Biological Cybernetics* **59**, S. 33-40.

Foulds [1983] Foulds, L. R., "The Heuristic Problem-Solving Approach", *Journal of the Operational Research Society* **34**, S. 927-934.

Fritzke und Wilke [1991] Fritzke, B., Wilke, P., "FLEXMAP – A Neural Network For The Traveling Salesman Problem With Linear Time And Space Complexity", *IEEE International Conference on Neural Networks*, S. 929-934.

Fujimura et al. [2001] Fujimura, K., Fujiwaki, S. I., Kwaw, O. C., Tokutaka, H., "Optimization of Electronic Parts Mounting Machines using SOM-TSP Method with 5 Dimensional Data", in: Allinson, N., Hujun, Y., Allinson, L., Slack, J. (Hrsg.), *Advances in Self-Organising Maps*, Springer-Verlag, London, S. 246-252.

Fuller [1978] Fuller, J. A.: "Optimal Solutions Versus "Good" Solutions: An Analysis of Heuristic Decision Making", *OMEGA* **6**, S. 479-484.

Garey und Johnson [1979] Garey, M. R., Johnson, D. S., *Computers and Intractability*, Freeman and Company, San Francisco.

Gendreau et al. [1997] Gendreau, M., Laporte, G., Potvin, J. Y., "Vehicle routing: modern heuristics", in: Aarts, E. H. L., Lenstra, J. K. (Hrsg.), *Local Search in Combinatorial Optimization*, John Wiley and Sons, Chichester, S. 311-336.

Ghaziri [1991] Ghaziri, H. , "Solving Routing Problems by a Self-Organizing Map", in: Kohonen, T., Mäkisara, K., Simula, O., Kangas, J. (Hrsg.), *Artificial Neural Networks*, Elsevier Science Publishers, Amsterdam, S. 829-834.

Ghaziri [1996] Ghaziri, H., "Supervision in the Self-Organizing Feature Map: Application to the Vehicle Routing Problem", in: Osman, I. H., Kelly, J. P. (Hrsg.), *Meta-Heuristics: Theory & Applications*, Kluwer Academic Publishers, Boston, S. 651-660.

Ghaziri und Osman [2003] Ghaziri, H., Osman, I. H., "A neural network algorithm for the traveling salesman problem with backhauls", *Computers & Industrial Engineering* **44**, S. 267-281.

Gillet und Miller [1974] Gillet, B. E., Miller, L. R., "A Heuristic Algorithm for the Vehicle-Dispatch Problem", *Operations Research* **22**, S. 340-349.

Glover und Laguna [1997] Glover, F., Laguna, M., *Tabu Search*, Kluwer Academic Publishers, Boston.

Golden und Stewart [1985] Golden, B. L., Stewart, W. R., "Empirical Analysis of heuristics", in: Lawler, E. L., Lenstra J. K., Rinnooy Kan, A. H. G., Shmoys, D.B. (Hrsg.), *The Traveling Salesman Problem*, John Wiley and Sons Ltd., Chichester, S. 207-249.

Goldstein [1990] Goldstein, M., "Self-organizing Feature Maps for the Multiple Travelling Salesmen Problem (MTSP)", *IEEE International Conference on Neural Networks*, S. 258-261.

Grauel [1992] Grauel, A., *Neuronale Netze*, BI-Wissenschaftsverlag, Mannheim.

Gutenberg [1967] Gutenberg, E., *Betriebswirtschaftslehre als Wissenschaft*, 3. Auflage, Scherpe Verlag, Krefeld.

Gutenberg [1983] Gutenberg, E., *Grundlagen der Betriebswirtschaftslehre, 1. Band: Die Produktion*, 24. Auflage, Springer-Verlag, Berlin.

Hansen et al. [1994] Hansen, P. H., Hegedahl, B., Hjortkjaer, S., Obel, B., "A heuristic solution to the warehouse location routing problem", *European Journal of Operational Research* **76**, S. 111-127.

Hansmann [1974] Hansmann, K.-W., *Entscheidungsmodelle zur Standortplanung der Industrieunternehmen*, Gabler Verlag, Wiesbaden.

Hansmann [2001] Hansmann, K.-W., *Industrielles Management*, 7. Auflage, R. Oldenbourg Verlag, München.

Hämäläinen [2002] Hämäläinen, T. D., "Parallel Implementations of Self-Organizing Maps" in: Seiffert, U., Lakhmi, C. J. (Hrsg.)*, Self-Organizing Neural Networks,* Physica-Verlag, Heidelberg, S. 245-278.

Hao und Lai [1996] Hao, G., Lai, K. K., "Solving the AGV Problem via a Self-Organizing Neural Network", *Journal of the Operational Research Society* **47**, S. 1477-1493.

Hueter [1988] Hueter, G. J., "Solution of the Traveling Salesman Problem with an Adaptive Ring", *Proceedings of the International Conference on Neural Networks*, **I**, S. 85-92.

Hoffman und Wolfe [1985]	Hoffman, A. J., Wolfe, P., “History“, in: Lawler, E. L., Lenstra J. K., Rinnooy Kan, A. H. G., Shmoys, D.B. (Hrsg.), *The Traveling Salesman Problem*, John Wiley and Sons Ltd., Chichester, S. 1-15.
Hopfield und Tank [1985]	Hopfield, J. J., Tank, D. W., “”Neural” Computation of Decisions in Optimization Problems”, *Biological Cybernetics* **52**, S. 141-152.
Jacobsen und Madsen [1980]	Jacobsen, S. K., Madsen, O. B. G., “A comparative study of heuristics for a two-level routing-location problem”, *European Journal of Operational Research* **5**, S. 378-387.
Johnson und McGeoch [1997]	Johnson, D. S., McGeoch, L. A., “The travelling salesman problem: a case study”, in: Aarts, E. H. L., Lenstra, J. K. (Hrsg.), *Local Search in Combinatorial Optimization*, John Wiley and Sons Ltd., Chichester, S. 215-310.
Johnson und Papadimitriou [1985]	Johnson, D. S., Papadimitriou, C. H., “Computational complexity”, in: Lawler, E. L., Lenstra J. K., Rinnooy Kan, A. H. G., Shmoys, D.B. (Hrsg.), *The Traveling Salesman Problem*, John Wiley and Sons Ltd., Chichester, S. 37- 85.
Kasabov [1996]	Kasabov, N. K., *Foundations of Neural Networks, Fuzzy Systems, and Knowledge Engineering*, The MIT Press, Cambridge, Massachusetts.
Kinnebrock [1992]	Kinnebrock, W., *Neuronale Netze*, R. Oldenbourg Verlag, München.
Kirkpatrick et al. [1983]	Kirkpatrick, S., Gelatt, C. D., Vecchi, M.P., “Optimization by Simulated Annealing”, *Science* **220**, S. 671-680.
Kohonen [1982a]	Kohonen, T., “Self-Organized Formation of Topologically Correct Feature Maps“, *Biological Cybernetics* **43**, S. 59-69.
Kohonen [1982b]	Kohonen, T., “Analysis of a Simple Self-Organizing Process“, *Biological Cybernetics* **44**, S. 135-140.
Kohonen [1990]	Kohonen, T., “The Self-Organizing Map”, *Proceedings of the IEEE,* **78,** S. 1464-1480.
Kohonen [1991]	Kohonen, T., “Self-Organizing Maps: Optimization Approaches” , in: Kohonen, T., Mäkisara, K., Simula, O., Kangas, J. (Hrsg.), *Artificial Neural Networks*, Elsevier Science Publishers, Amsterdam, S. 981-990.
Kohonen [1993]	Kohonen, T., “Things You Haven’t Heard about the Self-Organizing Map”, *IEEE International Conference on Neural Networks,* **III**, S. 1147-1156.
Kohonen [2001]	Kohonen, T., *Self-Organizing Maps*, 3. Auflage, Springer-Verlag, Berlin.
Kohonen [2002]	Kohonen, T., “Overture“ in: Seiffert, U., Lakhmi, C. J. (Hrsg.)*, Self-Organizing Neural Networks,* Physica-Verlag, Heidelberg, S. 1-12.
Kratzer [1993]	Kratzer, K. P., *Neuronale Netze*, 2. Auflage, Carl Hanser Verlag, München.
Kuhn [1973]	Kuhn, H.W., “A note on Fermat’s problem”, Mathematical Programming **4**, S. 98-107.
Kwok und Smith [2004]	Kwok, T., Smith, K. A., “A Noisy Self-Organizing Neural Network With Bifurcation Dynamics for Combinatorial Optimization”, *IEEE Transactions on Neural Networks* **15**, S. 84-98.

Laporte [1988] — Laporte, G., "Location Routing Problems" in: Golden, B. L., Assad, A. A. (Hrsg.), *Vehicle Routing: Methods and Studies,* Elsevier Science Publishers, Amsterdam, S. 163-197.

Laporte [1992] — Laporte, G., "The Traveling Salesman Problem: An overview of exact and approximate algorithms", *European Journal of Operational Research* **59**, S. 231-247.

Laporte und Dejax [1989] — Laporte, G., Dejax, P. J., "Dynamic Location-routeing Problems", *Journal of the Operational Research Society* **40**, S. 471-482.

Laporte und Nobert [1980] — Laporte, G., Nobert, Y., "A Cutting Planes Algorithm for the m-Salesmen Problem", *Journal of the Operational Research Society* **31**, S. 1017-1023.

Lohrbach [1994] — Lohrbach, T., *Einsatz von Künstlichen Neuronalen Netzen für ausgewählte betriebswirtschaftliche Aufgabenstellungen und Vergleich mit konventionellen Lösungsverfahren*, Unitext-Verlag, Göttingen.

Love et al. [1988] — Love, R. F., Morris, J. G., Wesolowsky, G. O., *Facilities Location*, Elsevier Science Publishers, Amsterdam.

Lozano et al. [1998] — Lozano, S., Guerrero, F., Onieva, L., Larrañeta, J., "Kohonen maps for solving a class of location-allocation problems", *European Journal of Operational Research* **108**, S. 106-117.

Madsen [1983] — Madsen, O. B. G., "Methods for solving combined two level location-routing problems of realistic dimension", *European Journal of Operational Research* **12**, S. 295-301.

Maffioli [1979] — Maffioli, F., "The Complexity of Combinatorial Optimization Algorithms and the Challenge of Heuristics", in: Christofides, N., Mingozzi, A., Toth, P., Sandi, C. (Hrsg.), *Combinatorial Optimization,* John Wiley & Sons Ltd., Chichester, S. 107-129.

Matsuyama [1991] — Matsuyama, Y., "Self-Organization via Competition, Cooperation and Categorization Applied to Extended Vehicle Routing Problems", IEEE *International Conference on Neural Networks,* S. 385-390.

Matsuyama [1992] — Matsuyama, Y., "Self-Organizing Neural Networks and Various Euclidean Traveling Salesman Problems", *Systems and Computers in Japan* **23**, S. 101-112.

Matsuyama [1995] — Matsuyama, Y., "Competitive Self-Organization and Combinatorial Optimization: Applications to the Traveling Salesman Problem", *IEEE International Conference on Neural Networks Proceedings* **III**, S. 819-824.

McCulloch und Pitts [1943] — McCulloch, W. S., Pitts, W., "A Logical Calculus of the Ideas immanent in Nervous Activity", *Bulletin of Mathematical Biophysics* **5**, S. 115-133.

Miehle [1958] — Miehle, W., "Link-Length Minimization in Networks", *Operations Research* **6**, S. 232-243.

Miller et al. [1960] — Miller, C.E., Tucker, A.W., Zemlin, R.A., "Integer Programming formulation of Travelling Salesman Problems", *Journal of the ACM* **7**, S. 326 – 329.

Min et al. [1998] — Min, H., Jayaraman, V., Srivastava, R., "Combined location-routing problems: A synthesis and future research directions", *European Journal of Operational Research* **108**, S. 1-15.

Modares et al. [1999]	Modares, A., Somhom, S., Enkawa, T., "A self-organizing neural network approach for multiple traveling salesman and vehicle routing problems", *International Transactions in Operational Research* **6**, S. 591-606.
Mole und Jameson [1976]	Mole, R. H., Jameson, S. R., "A Sequential Route-building Algorithm Employing a Generalised Savings Criterion", *Operational Research Quarterly* **27**, S. 503-511.
Nour und Madey [1996]	Nour, M. A., Madey, G. R., "Heuristic and optimization approaches to extending the Kohonen self organizing algorithm", *European Journal of Operational Research* **93**, S. 428 - 448.
Oja und Kaski [1999]	Oja, E., Kaski, S., *Kohonen Maps*, Elsevier Science B.V., Amsterdam.
Obermayer und Sejnowski [2001]	Obermayer, K., Sejnowski, T. J., *Self Organizing Map Formation*, The MIT Press, Cambridge, Massachusetts.
Paessens [1988]	Paessens, H., "The savings algorithm for the vehicle routing problem", *European Journal of Operational Research* **34**, S. 336-344.
Papadimitriou [1995]	Papadimitriou, C. H., *Computational complexity*, Addison-Wesley Publishing Company, Reading.
Perl und Daskin [1985]	Perl, J., Daskin, M. S., "A Warehouse Location-Routing Problem", *Transportation Research* **19B**, S. 381-396.
Pietsch und Teubner [1996]	Pietsch, W., Teubner, A., "Konnektionistische Tourenplanung für die dezentrale Vertriebsaußendienststeuerung", in: Corsten, H., May, C. (Hrsg.), *Neuronale Netze in der Betriebswirtschaft*, Gabler Verlag, Wiesbaden, S. 225-234.
Platzman und Bartholdi [1989]	Platzman, L. K., Bartholdi, J.J., "Spacefilling Curves and the Planar Travelling Salesman Problems", *Journal of the Association for Computing Machinery* **36**, S. 719-737.
Plebe und Anile [2001]	Plebe, A., Anile, A. M., "A Neural-Network-Based Approach to the Double Traveling Salesman Problem", *Neural Computation* **14**, S. 437-471.
Potvin [1993]	Potvin, J. Y., "The Traveling Salesman Problem: A Neural Network Perspective", *ORSA Journal on Computing* **3**, S. 328-348.
Potvin und Robillard [1995]	Potvin, J. Y., Robillard, C., "Clustering for vehicle routing with a competitive neural network", *Neurocomputing* **8**, S. 125-139.
Reinelt [1991]	Reinelt, G., "TSPLIB - A Traveling Salesman Problem Library", *ORSA Journal on Computing* **3**, S. 376-384.
Retzko [1996]	Retzko, R., *Flexible Tourenplanung mit selbstorganisierenden Netzen*, Unitext-Verlag, Bovenden.
Ritter und Schulten [1988a]	Ritter, H., Schulten, K., "Kohonens Self-Organizing Maps: Exploring their Computational Capabilities", *IEEE International Conference on Neural Networks* **I**, S. 109 –116.
Ritter und Schulten [1988b]	Ritter, H., Schulten, K., "Convergence Properties of Kohonen's Topology Conserving Maps: Fluctuations, Stability, and Dimension Selection", *Biological Cybernetics* **60**, S. 59 –71.

Ritter et al. [1994] Ritter, H., Martinetz, T., Schulten, K., *Neuronale Netze: Eine Einführung in die Neuroinformatik selbstorganisierender Netzwerke,* Addison-Wesley Publishing Company, Reading.

Rumelhart und Zipser [1985] Rumelhart, D.E., Zipser, D., "Feature Discovery by Competitive Learning", *Cognitive Science* **9**, S. 75-112.

Salhi und Rand [1989] Salhi, S., Rand, G. K., "The effect of ignoring routes when locating depots", *European Journal of Operational Research* **39**, S. 150-156.

Schmitter [1991] Schmitter, E.D., *Neuronale Netze*, Hofacker-Verlag, Holzkirchen.

Seiffert und Lakhmi [2002] Seiffert, U., Lakhmi, C. J., *Self Organizing Neural Networks*, Physica-Verlag, Heidelberg.

Silver [2004] Silver, E. A., "An overview of heuristic solution methods", *Journal of the Operational Research Society* **55**, S. 936-956.

Smith [1995] Smith, K. A., "Solving the Generalised Quadratic Assignment Problem using a Self-Organising Process", *IEEE International Conference on Neural Networks* **4**, S. 1876-1879.

Smith [1999] Smith, K. A., "Neural Networks for Combinatorial Optimization: A Review of More Than a Decade of Research", *INFORMS Journal on Computing* **11**, S. 15-34.

Smith und Gupta [2000] Smith, K. A., Gupta, J. N. D., " Neural Networks in business: techniques and applications for the operations researcher", C*omputers & Operations Research* **27**, S. 1023-1044.

Somhom et al. [1997] Somhom, S., Modares, A., Enkawa, T., "A self-organising model for the travelling salesman problem", *Journal of the Operational Research Society* **48**, S. 919-928.

Somhom et al. [1999] Somhom, S., Modares, A., Enkawa, T., "Competition-based neural network for the multiple travelling salesmen problem with minimax objective", *Computers & Operations Research* **26**, S. 395-407.

Soylu et al. [2000] Soylu, M., Özdemirel, N. E., Kayligil, S., "A self-organizing neural network approach for the single AGV routing problem", *European Journal of Operational Research* **121**, S. 124-137.

Srivastava [1993] Srivastava, R., "Alternate Solution Procedures for the Location-Routing Problem", *OMEGA International Journal of Management Science 21***,** S. 497-506.

Stein [1977] Stein, D., *Scheduling dial-a-ride transportation systems: an asymptotic approach*, Ph.D. thesis, Harvard University, Cambridge.

Svestka und Huckfeldt [1973] Svestka, J. A., Huckfeldt, V. E., "Computational Experience with an M-Salesman Traveling Salesman Algorithm", *Management Science* **19,** S. 790-799.

Tokutaka und Fujimura [1999] Tokutaka, H., Fujimura, K., "SOM-TSP: An approach to optimize surface component mounting on a printed circuit board", in: Oja, E., Kaski, S., *Kohonen Maps*, Elsevier Science B.V., Amsterdam, S. 219-230.

Torki et al. [1997] Torki, A., Somhom, S., Enkawa, T., "A Competitive Neural Network Algorithm for Solving vehicle Routing Problem", *Computers & Industrial Engineering* **33**, S. 473-476.

Vahrenkamp [2003]	Vahrenkamp, R., *Quantitative Logistik für das Supply Chain Management*, R. Oldenbourg Verlag, München.
Vakhutinsky und Golden [1994]	Vakhutinsky, A. I., Golden, B., "Solving vehicle routing problems using elastic net", *IEEE International Conference on Neural Networks*, S. 4535-4540.
Vakhutinsky und Golden [1995]	Vakhutinsky, A. I., Golden, B., "A Hierarchical Strategy for Solving Traveling Salesman Problems Using Elastic Net", *Journal of Heuristics* **1**, S. 67-76.
Vogel [1975]	Vogel, K. H., *Das Steiner-Weber-Problem und Erweiterungen des Problems bei der Bestimmung optimaler Standorte,* Dissertation, Hamburg.
Wacholder et al. [1989]	Wacholder, E., Han, J., Mann, R. C., "A Neural Network Algorithm for the MTSP", *Biological Cybernetics* **61**, S. 11-19.
Weber [1909]	Weber, A., *Über den Standort der Industrien, 1.Teil: Reine Theorie der Standorte*, J.C.B. Mohr, Tübingen.
Weissermel [1999]	Weissermel, M., *Tourenplanungsprobleme mit Zeitfensterrestriktionen*, Vandenhoeck und Ruprecht, Göttingen.
Weiszfeld [1937]	Weiszfeld, E., "Sur le point pour lequel la somme des distances de n points donnes est minimum", *Tôhoku Mathematical Journal* **43**, S. 355-386.
Willshaw und von der Malsburg [1976]	Willshaw, D. J., von der Malsburg, C., "How patterned neural connections can be set up by self-organization", *Proceeding of the Royal Society of London* **194**, S. 431-445.
Wong et al. [1997]	Wong, B. K., Bodnovich, T. A., Selvi, Y., "Neural network applications in business: A review and analysis of the literature (1988-95)", *Decision Support Systems* **19**, S. 301-320.
Wong et al. [2000]	Wong, B. K., Vincent, S. L., Lam, J., "A bibliography of neural network business applications research: 1994-1998", *Computers & Operations Research* **27**, S. 1045-1076.
Xu und Tsai [1991]	Xu, X., Tsai, W. T., "Effective Neural Algorithms for the Traveling Salesman Problem", *Neural Networks* **4**, S. 193-205.
Zell [1994]	Zell, A., *Simulation Neuronaler Netze*, Addison-Wesley Publishing Company, Bonn.