Universität Heidelberg
Interdisziplinäres Zentrum für Wissenschaftliches Rechnen
Seminar Optimierung im Sommersemester 2000
Leitung: Prof. Dr. G. Reinelt
Referent: Wolfgang Stille

Moderne Heuristiken zur Lösung des Vehicle Routing Problems (VRP)

Inhaltsverzeichnis

1 Einleitung

Das *Vehicle Routing Problem (VRP)* nimmt eine zentrale Stellung im Logistikmanagement ein. Es definiert sich dadurch, unter gewissen Nebenbedingungen optimale Strecken durch eine Menge von Knoten zur Lieferung oder Abholung von Gegenständen zu finden. Formal wird das VRP auf einem gerichteten Graphen $G = (V, A)$ definiert, wobei $V = \{v_0, \ldots, v_n\}$ eine Menge von Knoten und $A = \{(v_i, v_j) : v_i, v_j \in V, i \neq j\}$ eine Menge von Kanten ist. Der Knoten v_0 stellt das Depot dar, in dem m identische Fahrzeuge stationiert sind. Der Wert von m ist entweder a priori festgelegt oder frei wählbar, jedoch im allgemeinen nach oben hin durch eine Konstante $\bar{m} \leq n$ beschränkt.
Alle anderen Knoten stellen Städte oder Kunden dar, die beliefert werden sollen. Auf A ist eine nichtnegative Distanzenmatrix $D = (d_{ij})$ definiert. Je nach Problem können die d_{ij} als Reisekosten oder Reisezeit interpretiert werden.
Man sagt, das Problem sei *symmetrisch*, falls $d_{ij} = d_{ji} \quad \forall \quad (v_i, v_j) \in A$.
Das VRP besteht nun darin, m Routen mit minimalen Kosten zu finden, die am Depot starten und enden und jeden Kunden genau einmal besuchen.

Dazu können noch folgende Nebenbedingungen kommen:

- im *Capacitated Vehicle Routing Problem (CVRP)* wird jeder Knoten mit einem nichtnegativen Gewicht q_i versehen. Das Gesamtgewicht einer jeden Route darf die Fahrzeugkapazität Q nicht überschreiten.

- im *weg- oder zeitbeschränkten VRP* stellen die d_{ij} Reisezeiten dar. Zusätzlich hat jeder Kunde eine nichtnegative Servicezeit δ_i zum Be- oder Entladen des Fahrzeuges. Insgesamt darf die Gesamtzeit einer Route eine Konstante L nicht überschreiten.

- im *VRP mit Zeitfenstern (VRPTW)* muß jede Stadt v_i innerhalb eines vorgegebenen Zeitintervalls $[a_i, b_i]$ besucht werden. Den Fahrzeugen ist es dabei nicht erlaubt, an einem Ort zu warten, falls sie vor dem Beginn des Zeitfensters dort eintreffen.

Die *Zielfunktion* kann beüglich der Reisekosten oder -zeiten aufgestellt werden oder aber darin bestehen, in der Lösung mit möglichst wenigen Fahrzeugen auszukommen.

Das VRP ist ein $\mathcal{NP}$-schweres kombinatorische Optimierungsproblem. Verschiedene Optimierungsalgorithmen wurden beispielsweise von Cornuejols und Harche [1993], Christofides, Hadjiconstantinou und Mingozzi [1993], Araque et al. [1994] oder Fisher [1995] vorgeschlagen, sind jedoch oft nicht in der Lage, Instanzen jenseits von 50 Knoten zu lösen.
Deshalb konzentrierte sich der Großteil der Forschung darauf, approximative Algorithmen zu entwickeln. Viele Lösungsansätze aus heutiger Zeit wurden im Bereich *lokaler Suchalgorithmen* veröffentlicht, zu denen *Simulated Annealing, Tabu Search, Genetische Algorithmen und Neuronale Netze* zählen. Diese Approximationsalgorithmen durchsuchen den Lösungsraum mit dem Ziel, eine möglichst gute Lösung zu finden.
Simulated Annealing und *Tabu Search* gehen von einer Anfangslösung x_1 aus und bewegen sich in jeder Iteration t von x_t zu einer Lösung x_{t+1} in der Nachbarschaft $N(x_t)$ von x_t bis eine Abbruchbedingung zutrifft. Wenn $f(x)$ die Kosten von x bezeichnen, so ist $f(x_{t+1})$ nicht notwendigerweise geringer als $f(x_t)$. Demnach müssen unbedingt sich wiederholende Zirkeln vermieden werden, damit der Algorithmus terminiert.

Genetische Algorithmen dursuchen in jedem Schritt eine Population von Lösungen; jede Population erhält man aus dem vorangegangenen Schritt, indem man dessen beste Individuen kombiniert und die schlechtesten aussondert.
Neuronale Netze beinhalten eine Art lernfähige Heuristik, die eine Menge von Gewichten schrittweise verändert bis eine akzeptable Lösung erreicht ist.
Die Regeln und Parameter, die die Suche steuern, variieren in allen Fällen und müssen daher auf die Art des Problem zugeschnitten sein. Auch sind eine gute Portion Kreativität und Experimentiergeist gefragt.

2 Simulated Annealing

In Iteration t des Simulated Annealing Algorithmus wird eine Lösung x zufällig aus $N(x_t)$ ausgewählt.
Falls $f(x) \leq f(x_t)$, so sei $x_{t+1} = x$; andernfalls:

$$x_{t+1} = \begin{cases} x & \text{mit Wahrscheinlichkeit } p_t; \\ x_t & \text{mit Wahrscheinlichkeit } 1 - p_t, \end{cases}$$

wobei p_t gewöhnlich eine fallende Funktion von t und $f(x) - f(x_t)$ ist. Im allgemeinen wird p_t als

$$p_t = exp\big(-\frac{f(x) - f(x_t)}{c_t}\big)$$

definiert, wobei c_t die *Temperatur* zur Iteration t darstellt und nach einem *Abkühlungsschema* berechnet wird. Typischerweise ist c_t eine schrittweise fallende Funktion: zu Beginn wird c_t ein Wert $c_1 > 0$ zugewiesen und nach jeweils T Iterationen mit einem Faktor $\alpha \quad (0 < \alpha < 1)$ multipliziert, so daß die Wahrscheinlichkeit, daß eine schlechtere Lösung akzeptiert wird, im Laufe der Zeit sinkt.

Es gibt drei gebräuchliche Abbruchbedingungen:

(1) der Wert $f^\star$ der Optimallösung $x^\star$ ist über einen Zeitraum von mindestens k_1 aufeinanderfolgenden Zirkeln von T Iterationen nicht um mindestens $\pi_1\%$ kleiner geworden.

(2) die Anzahl der akzeptierten Veränderungen über einen Zeitraum von k_2 aufeinanderfolgenden Zirkeln von T Iterationen betrug weniger als $\pi_2\%$ von T.

(3) k_3 Zirkel von T Iterationen wurden ausgeführt.

2.1 Der Algorithmus von Alfa, Heragu und Chen

Bei dieser *route first - cluster second* Methode von Alfa, Heragu und Chen [1991] wird zuerst eine große Tour konstruiert, ohne Knotengewichte und Fahrzeugkapazitäten zu berücksichtigen. Diese wird anschließend in Segmente aufeinanderfolgender Knoten zerteilt, deren Gesamtgewicht das Limit Q nicht überschreitet. Alfa, Heragu und Chen betrachten symmetrische Probleme und nehmen implizit an, daß die Anzahl an Fahrzeugen nicht von vornherein bekannt

ist. Sie beginnen mit der Anfangstour $(v_0, v_1, \ldots, v_n, v_0)$ und wählen in jeder Iteration t drei Kanten durch Zufall aus, die sie aus der Tour entfernen. Die Tour wird anschließend auf acht verschiedene Weisen wieder zusammengebaut wie im klassischen $3 - OPT$ Kantentauschverfahren von Lin [1965]. Diese acht Touren spiegeln die Nachbarschaft der derzeitigen Lösung x_t wieder. Die optimalste Tour davon wird ausgewählt. Als Abbruchbedingung verwenden die Autoren Regel (3) von oben.
Der Algorithmus wurde an 3 Instanzen aus der Literatur getestet ($n = 30, 50, 75$), die Ergebnisse waren alles andere als zufriedenstellend. In jedem Fall war die vom Algorithmus erzeugte Optimallösung weitaus schlechter als die derzeit als optimal bekannte. Dies ist weitgehend der *route first - cluster second* Stategie zuzuschreiben: obwohl davon erwartet wird, daß sie sich in asymptotischem Sinne gut verhält [Haimovich und Rinnooy Kan, 1985], konnte nie gezeigt werden, daß sie bei Problemen kleiner oder mittlerer Größe konkurrenzfähig gegenüber anderen Verfahren ist.

2.2 Osman's Simulated Annealing Algorithmus

In vielen Aspekten ist die Simulated Annealing Implementation von Osman [1993] wesentlich fortgeschrittener:

(1) sie benutzt eine bessere Startlösung.

(2) einige Parameter des Algorithmus werden in einer Testphase angepaßt.

(3) reichhaltigere Nachbarschaften von Lösungen werden untersucht.

(4) das Abkühlungsschema ist ausgereifter.

Der Algorithmus macht sich einen sogenannten λ-*interchange generation mechanism* zueigen: Zwei Wege p und q werden ausgewählt, gleichzeitig zwei Knotenteilmengen S_p von p und S_q von q, die den Bedingungen $|S_p| \leq \lambda$ und $|S_q| \leq \lambda$ genügen. Nun werden die Knoten aus S_p mit denen aus S_q getauscht, solange dies zulässig ist. Da die Anzahl an Kombinationen in der Regel groß ist, wählt man $\lambda = 1$ oder 2. In der effizientesten Fassung des Algorithmus, endet die Suche, sobald eine Lösungsverbesserung erzielt wurde. Falls diese nicht auftritt, muß die gesamte Nachbarschaft durchsucht werden.
Der Algorithmus wurde an symmetrischen Instanzen mit frei wählbarem m getestet:

Phase 1: Abstieg

- erzeuge eine Startlösung mittels des *Clarke-Wright-Algorithmus* [1964]
- durchsuche den Lösungsraum mittels des λ-*Austausch-Schemas*. Jede Verbesserung wird sofort übernommen. Stop, falls eine Suche über die gesamte Nachbarschaft erfolglos bleibt.

Phase 2: Simulated Annealing

(1) *Startlösung:* als Startlösung dient die oben erzeugte oder eine Clarke-Wright-Lösung

(2) *Parameterinitialisierung:* führe mittels des λ-*Austausch-Schemas* eine Suche über die gesamte Nachbarschaft durch, ohne Verbesserungen zu übernehmen. Zeichne die extremsten

Veränderungen in der Zielfunktion Δ_{min} und Δ_{max} auf und berechne die Anzahl β der zulässigen Lösungen. Setze $c_1 := \Delta_{max}, \delta := 0, k := 1, k_3 := 3, t := 1, t^\star := 1$ ($t^\star$ bezeichnet diejenige Iteration, in der die beste Lösung innerhalb des aktuellen Zirkels gefunden wurde). Sei x_1 die aktuelle Lösung und $x^\star := x_1$.

(3) *Iteration:* Durchsuche die Nachbarschaft von x_1 mittels des λ-*Austausch-Schemas*. Sobald eine Lösung x mit $f(x) < f(x_1)$ gefunden wurde, setze $x_{t+1} := x$; falls $f(x) < f(x^\star)$, setze $x^\star := x$ und $c^\star := c_k$. Falls eine Suche auf der kompletten Nachbarschaft keine bessere Lösung als x_1 erzielt, so sei x die beste Lösung in der Nachbarschaft von x_t und setze:

$$x_{t+1} = \begin{cases} x & \text{mit Wahrscheinlichkeit } p_t; \\ x_t & \text{mit Wahrscheinlichkeit } 1 - p_t, \end{cases}$$

wobei p_t wie oben definiert ist. Falls $x_{t+1} = x_t$, setze $\delta := 1$.

(4) *Temperaturupdate:*

- *normalerweise Senken:* falls $\delta = 0$, setze $c_{t+1} := \frac{c_t}{n(\beta+\sqrt{t})\Delta_{max}\Delta_{min}}$.
- *gelegentlich Erhöhen:* falls $\delta = 1$, setze $c_{t+1} := max\{\frac{c_t}{2}, c^\star\}, \delta := 0, k := k + 1$.

Setze $t = t + 1$.

(5) *Terminierungstest:* Falls $k = k_3$ STOP, andernfalls weiter bei Schritt (3).

Besonders zu erwähnen ist die Abkühlungsfunktion, die Osman verwendet. Während die Standardabkühlung nur ein kontinuierliches Senken der Temperatur vorsieht, setzt er die Temperatur zurück, falls keine Änderung in der Lösung mehr auftritt. Angewendet auf das *Capacitated Clustering Problem* und das *Generalized Assignment Problem* übertrifft er damit die Lösungsqualität erheblich im Vergleich zu herkömmlichen Abkühlungsschemata.
Der Algorithmus wurde mit $\lambda = 1$ implementiert und benutzt die beste Lösung aus Phase 1 als Start für Phase 2. Test wurden auf 26 Instanzen ausgeführt. Im folgenden sind die Werte aus den 14 Benchmarkproblemen von Christofides, Mingozzi und Toth [1979] aufgelistet.

Table 9.1 Computational results for Osman's algorithm

Problem number	n	Type[a]	f^*	Best known solution value	Time[b]
1	50	C	528	524.61[c,d,e]	167.4
2	75	C	838.62	835.26[c]	6434.3
3	100	C	829.18	826.14[c,d]	9334.0
4	150	C	1058	1028.42[c]	5012.3
5	199	C	1378	1298.79[c]	2318.1
6	50	C,D	555.43	555.43[c,d,e]	3410.2
7	75	C,D	909.68	909.68[c,d,e]	626.5
8	100	C,D	866.75	865.94[c,d]	957.2
9	150	C,D	1164.12	1162.55[c,d]	84301.2
10	199	C,D	1417.85	1397.94[c]	5708.0
11	120	C	1176	1042.11[c,d]	315.8
12	100	C	826	819.56[c,d]	632.0
13	120	C,D	1545.98	1541.14[c]	7622.5
14	100	C,D	890	866.37[c,d,e]	305.2

[a] C = capacity restrictions; D = distance restrictions.
[b] Seconds on a VAX 8600 computer.
[c] Taillard [1993a].
[d] Gendreau, Hertz & Laporte [1994].
[e] Osman [1993].

Tab1: Ergebnisse des Algorithmus von Osman

Die Implementation liefert gute Ergebnisse, jedoch nicht die besten Approximationen. Osman zeigt noch, daß sein Algorithmus nicht sehr robust ist, und in manchen Fällen (z.B. Problem 14) der Abstieg mit $\lambda = 2$ eine bessere Lösung erzielt.

3 Tabu Search

Wie im Simulated Annealing werden im Tabu Search Folgen von Lösungen untersucht, jedoch bewegt sich der nächste Schritt zum besten Nachbarn der aktuellen Lösung x_t. Um Zirkel zu vermeiden, sind Lösungen, die bereits untersucht wurden, in einer festgelegten Anzahl darauffolgender Iterationen verboten (tabu). Um Zeit und Speicherplatz zu sparen, ist es üblich, anstatt der Lösungen selbst Merkmale derer aufzuzeichnen.

3.1 Zwei frühe Tabu-Search-Algorithmen

Einer der ersten Versuche, Tabu Search auf das VRP anzuwenden, war der von Willard [1989], der die Lösung zuerst in eine große Tour überführt, indem er den Depotknoten repliziert. Nachbarschaften sind hier alle zulässigen Lösungen, die von der aktuellen aus durch $2-OPT$ und $3-OPT$ Kantentauschverfahren gewonnen werden können. Die nächste Lösung wird anhand des besten *Nontabu-Moves* erzielt. Der Autor präsentiert nur beschränkt Ergebnisse seines Verfahrens. Auf drei der o.g.Benchmarkprobleme angewandt, liefert das Verfahren keine konkurrenzfähigen Ergebnisse.
Pureza und Franca[1991] definieren die Nachbarn einer Lösung, indem sie einen Knoten einer anderen Tour hinzufügen oder zwei Knoten verschiedener Touren vertauschen, während die Lösungen zulässig bleiben müssen. Wie bei Willard wird der beste zulässige Nontabu-Move in jeder Iteration ausgewählt. Die Ergebnisse sind wie in der Tabelle zu sehen etwas besser als Willard's, jedoch immer noch nicht vergleichbar mit anderen Tabu-Search-Verfahren.

3.2 Osman's Tabu-Search-Algorithmus

Osman [1993] definiert Nachbarschaft wieder durch sein λ-Austauschverfahren mit $\lambda = 2$. Dies beinhaltet eine Variation von $2-OPTs$, Knotenhinzunahme und Knotenaustausch zwischen zwei Touren. In der einen Variante des Algorithmus *(best-admissible)* werden ganze Nachbarschaften durchsucht und der beste Nontabu-Move gewählt, in einer anderen Version *(first-best-admissible)* die erste Verbesserung, falls eine existiert, implementiert, ansonsten die best-admissible genommen. Die Ergebnisse in obiger Tabelle zeigen, daß diese beiden Tabu-Search-Verfahren exzellente Ergebnisse liefern, jedoch in den meisten Fällen immer noch verbessert werden können.

3.3 TABUROUTE

Im Vergleich zu den vorangegangenen Implementationen ist TABUROUTE von Gendreau, Hertz und Laporte [1994] recht forgeschritten und enthält einige innovative Merkmale. Die Nachbarschaftsrelation ist durch alle Lösungen definiert, die durch Wegnahme eines Knotens aus einer Tour und dessen Einfügen in eine andere Tour, die seine nächsten Nachbarn enthält, entstehen. Dies geschieht mittels der *GENI (generalized insertion)* Prozedur, die von Gendreau, Hertz und Laporte [1992] für das *Traveling Salesman Problem (TSP)* entwickelt wurde. Dies

mag natürlich eine bestehende Tour zerstören oder eine neue erzeugen. GENI besteht darin, einen Knoten zwischen zweien seiner p am nächsten benachbarten in eine Tour einzufügen und währenddessen die Tour lokal zu reoptimieren.
Diese kurze Beschreibung von GENI ist für symmetrische Probleme gültig:
Ausgehend von einer Tour $(v_0, \dots, v_n, v_0)$ definieren wir die p-Nachbarschaft des Knotens v wie folgt:

- falls v kein Bestandteil der Tour ist, sei $N_p(v)$ die Menge derjenigen p Knoten der Tour, die sich am nächsten an v befinden, falls $p \leq t-1$; oder aber sei es die Menge aller Knoten der Tour, falls $p \geq t$.

- falls v Bestandteil der Tour ist, sei $N_p(v)$ die Menge derjenigen p Knoten der Tour, die sich am nächsten an v befinden, falls $p \leq t$; oder aber sei es die Menge aller Knoten der Tour, falls $p \geq t+1$.

- P_{ij} sei die Menge der Knoten auf dem Weg von v_i nach v_j in einer vorgegebenen Richtung. Für jeden Knoten v, der sich nicht in der Tour befindet, werden zwei Einfügearten definiert:

  (1) Wähle $v_r, v_s \in N_p(v)$ und $v_k \in N_p(v_{r+1}) \cap P_{sr} \backslash \{v_r, v_s\}$. Entferne die Kanten (v_r, v_{r+1}), (v_s, v_{s+1}) und (v_k, v_{k+1}); füge die Kanten (v_r, v), (v, v_s), (v_{r+1}, v_k) und (v_{s+1}, v_{k+1}) ein. Die Wege $(v_{r+1}, \dots, v_s)$ und $(v_{s+1}, \dots, v_k)$ werden umgedreht.

  (2) Wähle $v_r, v_s \in N_p(v)$, $v_k \in N_p(v_{r+1}) \cap P_{sr} \backslash \{v_s, v_{s+1}\}$ und $v_l \in N_p(v_{s+1}) \cap P_{rs} \backslash \{v_r, v_{r+1}\}$. Entferne die Kanten (v_r, v_{r+1}), (v_s, v_{s+1}), (v_{l-1}, v_l) und (v_{k-1}, v_k); füge die Kanten (v_r, v), (v, v_s), (v_l, v_{s+1}), (v_{k-1}, v_{l-1}) und (v_{r+1}, v_k) ein. Die Wege $(v_{r+1}, \dots, v_{l-1})$ und $(v_l, \dots, v_s)$ werden umgedreht.

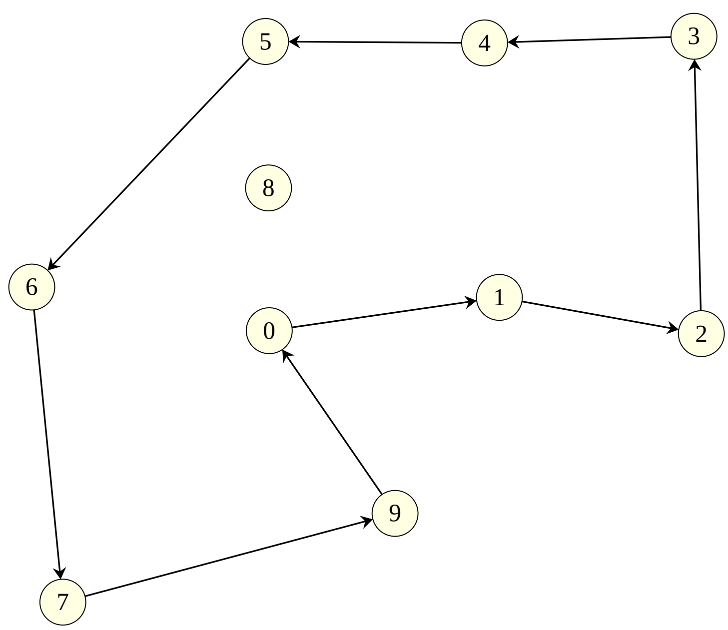

Abb1: Knoten 8 soll eingefügt werden, Knoten 0 und 5 sind seine nächsten Nachbarn

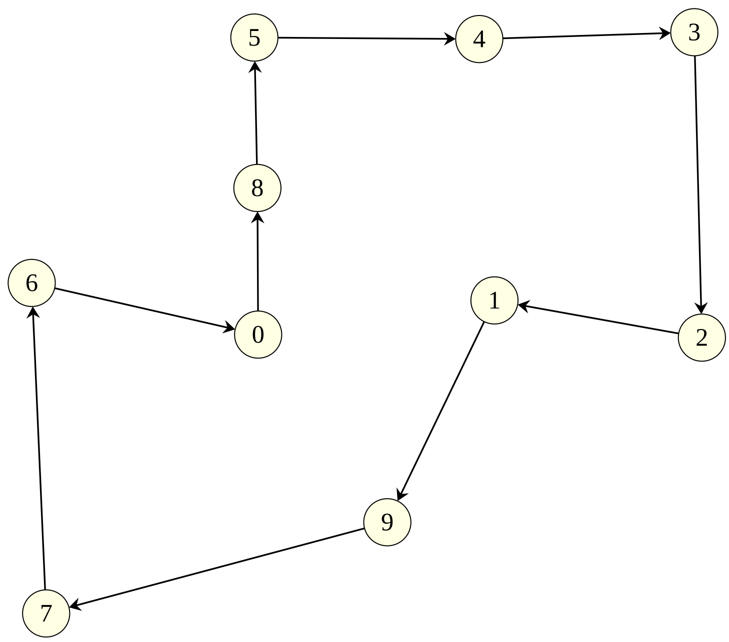

Abb2: nach den Einfügen mit GENI (1), Tour hat andere Orientierung

Um die beste Auswahl und Position eines Knotens festzustellen, ist es nun notwendig, die Kosten aller möglichen Einfügearten - beide Richtungen und jede mögliche Wahl von v_r, v_s, v_k und v_l berücksichtigend - zu berechnen. In TABUROUTE wird jede generalized insertion mit $p = 5$ ausgeführt.
Ein zweites wichtiges Merkmal von TABUROUTE ist die Untersuchung von Lösungen, die unzulässig im Bezug auf die Beschränkungen der Kapaztät oder Streckenlänge sind. Diese Unzulässigkeit wirkt sich durch Strafterme in der Zielfunktion aus, die von einem Parameter gewichtet werden, der alle 10 Iterationen halbiert wird, falls die letzten 10 Lösungen zulässig waren, ansonsten verdoppelt wird. Diese Vorgehensweise verringert die Gefahr, in lokalen Minima hängenzubleiben. Während des Suchprozesses wird die Tour, nachdem ein Knoten eingefügt wurde, mittels der *stringing and unstringing* TSP-Nachoptimierungsroutine (auch von Gendreau, Hertz und Laporte [1992]) verbessert.
TABUROUTE benutzt keine Tabuliste, sondern zufallsgesteuerte Tabuzeiten. Wenn ein Knoten von Tour r in Tour s in Iteration t verschoben wurde, ist sein Einfügen in Tour r bis zur Iteration $t + \theta$ verboten, wobei θ ein randomisierte ganze Zahl aus $[\theta_-, \theta^+]$, z.B. $[5, 10]$.
Weiterhin benutzt TABUROUTE eine Diversifikationsstrategie. Dies wird erreicht, indem Knoten, die oft hin- und herbewegt wurden, mit Strafpunkten belegt werden, um Knoten mit geringerer Auswahlwahrscheinlichkeit mehr in Betracht zu ziehen. Diese Strafe hat beim jeweilig zu betrachtenden Knoten Auswirkungen auf die Zielfunktion.
Letztlich macht sich TABUROUTE *false starts* zueigen: anfangs werden mehrere Lösungen generiert und eine eingeschränkte Suche auf jeder von ihnen ausgeführt. Die beste davon ist Ausgangspunkt für den eigentlichen Suchvorgang.

Im folgenden sei W die Knotenmenge, die zur Wiedereinfügung in eine andere Tour in jeder Iteration in Frage kommt, $q \leq |W|$ ist die Anzahl der Knoten, die im Moment probeweise wiedereingefügt wurden, und k ist die Anzahl der Iterationen ohne erzielte Verbesserung.

- Schritt 1, *Initialisierung*: erzeuge $\lfloor\sqrt{\frac{n}{2}}\rfloor$ Startlösungen und führe auf ihnen eine Tabusuche mit $W = V\backslash\{v_0\}, q = 5m$ und $k = 50$ durch. Der Wert von q sichert, daß die Wahrscheinlichkeit, einen Knoten von jeder Tour auszuwählen, mindestens 90% beträgt.

- Schritt 2, *Lösungsverbesserung:* führe ausgehend von der besten Lösung aus Schritt 1 eine Tabusuche mit $W = V\backslash\{v_0\}, q = 5m$ udn $k = 50n$ durch.

- Schritt 3, *Intensivierung*: führe ausgehend von der besten Lösung aus Schritt 2 eine Tabusuche mit $k = 50$ durch. W ist hierbei die Menge der $\lfloor\frac{|V|}{2}\rfloor$ Knoten, die am häufigsten in den Schritten 1 und 2 bewegt wurden, q ist $|W|$.

Im folgenden sieht man, daß TABUROUTE Ergebnisse höchster Qualität erzeugt und oft eine als optimalst bekannte Lösung erzielt.

3.4 Taillard's Algorithmus

Die Implementation enthält einige Merkmale von TABUROUTE, nämlich zufallsgesteuerte Tabuzeiten und Diversifikation. Nachbarschaften werden durch das λ-Austauschverfahren mit $\lambda = 1$ von Osman definiert. Der Algorithmus benutzt ein Standardeinfügeverfahren, was im Vergleich zu GENI weniger Zeit benötigt und nur zulässige Lösungen erzeugt. Nach jedem Einfügen eines Knotens werden die betreffenden Touren mittels des Optimierungsalgorithmus von Volgenant und Jonker [1983] reoptimiert.
Eine Neuerung ist die Zerlegung des Gesamtproblems in Teilprobleme. In planaren Problemen wird dies durch eine Partitionierung in Sektoren erreicht, die ihren Mittelpunkt im Depotknoten hat. Diese Sektoren werden wiederum in konzentrische Gebiete aufgegliedert. Die Teilprobleme können somit unabhängig voneinander gelöst werden, jedoch sind regelmässig Verlagerungen von Knoten in benachbarte Sektoren notwendig. Dieses Vorgehen macht Sinn, wenn das Depot in der Mitte gelegen ist, und die Knoten einheitlich darum in der Ebene verteilt sind. Für Probleme, die diese Eigenschaften nicht besitzen, und für nichtplanare Probleme schlägt der Autor andere Auteilungen vor, die auf einer minimal-spannenden Arboreszenz mit dem Depot als Wurzelknoten beruhen. Durch die Aufteilung kann das Problem gut parallelisiert werden.

3.5 Vergleich der Ergebnisse

Im folgenden werden die besten Ergebnisse präsentiert, die mit dem jeweiligen Algorithmus auf die 14 Benchmarkprobleme von Christofies, Mingozzi und Toth [1979] angewandt erzielt wurden. Dabei muß beachtet werden, daß der Algorithmus von Osman, der von Taillard und TABUROUTE mit Fließkomma-d_{ij}s arbeiten, während keine Regel für das Abschneiden oder Runden von Nachkommastellen bei Willard oder Pureza und Franca angegeben ist.
Ebenfalls sind bei den Ergebnissen von Willard und Taillard keine Rechenzeiten angegeben, so daß auch hier kein Vergleich möglich ist. Taillard benutzt unter anderem eine parallele Implementierung.
TABUROUTE-Best erreichte über mehrere Läufe mit verschiedenen Parametern das beste Resultat. Weiterhin ist aus untenstehender Tabelle ersichtlich, daß Taillard's Algorithmus immer Optimallösungen findet, dicht gefolgt von TABUROUTE.

			Willard	Pureza & França		Osman BA[c]		Osman FBA[c]		TABUROUTE Standard[f]		TABUROUTE Best[h]	Taillard
Problem	n	Type[a]	f^*	f^*	Time[b]	f^{*d}	Time[e]	f^{*d}	Time[e]	f^*	Time[g]	f^*	f^*
1	50	C	588	536	973	524.61*	67.2	524.61*	114.0	524.61*	360	524.61*	524.61*
2	75	C	893	842	155	844	70.8	844	50.3	835.77	3 228	835.32	835.26*
3	100	C	906	851	1 796	835	675.0	838	1 543.0	829.45	1 104	826.14*	826.14*
4	150	C	–	1 081	7 351	1 052	3 075.0	1 044.35	3 560.0	1 036.16	3 528	1 031.07	1 028.42*
5	199	C	–	–	–	1 354	1 972.7	1 334.55	3 246.0	1 322.65	5 454	1 311.35	1 298.79*
6	50	C,D	–	560	1 557	555.43*	140.2	555.43*	173.0	555.43*	810	555.43*	555.43*
7	75	C,D	–	929	2 772	913	203.0	911	1 056.7	913.23	3 276	909.68*	909.68*
8	100	C,D	–	887	7 245	866.75	1 200.0	878	2 998.0	865.94*	1 536	865.94*	865.94*
9	150	C,D	–	1 227	19 376	1 188	2 443.6	1 184	4 755.8	1 177.76	4 260	1 162.89	1 162.55*
10	199	C,D	–	–	–	1 422	3 310.1	1 441	4 561.0	1 418.51	5 988	1 040.75	1 397.94*
11	120	C	–	1 049	1 776	1 042.11*	1 398.4	1 043	1 445.4	1 073.47	1 332	1 042.11*	1 042.11*
12	100	C	–	829	3 527	819.59	407.5	819.59	892.2	819.56*	960	819.56*	819.56*
13	120	C,D	–	1 631	3 501	1 547	1 343.0	1 547	2 834.0	1 573.81	3 570	1 545.93	1 541.14*
14	100	C,D	–	866	6 166	866.37*	5 579.0	866.37*	1 175.9	866.37*	3 942	866.37	866.37*

[a] C = capacity restrictions; D = distance restrictions.
[b] Seconds on an MC68020 microcomputer.
[c] BA = best-admissible strategy; FBA = first-best admissible strategy.
[d] Values obtained with real distances; integer values correspond to a truncated f^*.
[e] Seconds on a VAX 8600 computer.
[f] Algorithm using a single set of parameters.
[g] Seconds on a Silicon Graphics workstation, 36 MHz, 5.7 Mflops.
[h] Best solution value over several runs of the algorithms, with various sets of parameters.

Tab3: Vergleich der Ergebnisse von Tabu Search

3.6 Tabu-Search-Algorithmen für das VRP mit Zeitfenstern (VRPTW) und Kapazitätsbeschränkungen (CVRP)

Potvin et al. [1996] beschreiben einen Tabu-Search-Algorithmus, der auf VRPTWs angewandt werden kann. Ihr erstes Ziel ist es, die Anzahl der Fahrzeuge zu minimieren, während sie in einer zweiten Phase versuchen, die Gesamtzeit zu minimieren. Die Nachbarschaftsrelation ist durch zwei Kantentauschoperatoren spezifiziert, $OR-OPT$ und $2-OPT^{\star}$. $OR-OPT$ besteht darin, eine Sequenz von ein, zwei oder drei aufeinanderfolgenden Knoten an eine andere Position innerhalb der Tour zu bewegen. $2-OPT^{\star}$ wird hauptsächlich in Problemen wie dem VRPTW verwendet, wo die Richtung der Tour eine entscheidende Rolle spielt. Ein $2-OPT^{\star}$ geht von zwei gerichteten Touren aus und löscht (v_i, v_j) von der ersten und (v_k, v_l) von der zweiten Tour. Die Kanten (v_i, v_l) und (v_k, v_j) werden anschließend hinzugefügt. Zumal die Anzahl der möglich $OR-OPT$ und $2-OPT^{\star}$ sehr groß sein kann, schränken die Autoren die Menge zulässiger Kantenaustausche wie folgt ein: Bei $OR-OPT$ sei v_i der erste Knoten der Sequenz, die verschoben wird, und v_j derjenige Knoten, nach dem v_i wieder eingefügt wird; daher muß v_i einer der p nächsten Nachbarn von $v_i^{\star}$ sein.
Ebenfalls muß in einem $2-OPT^{\star}$ v_l einer der q nächsten Nachbarn von v_i sein. p und q sind hier Eingabeparameter.

- Schritt 1, *Initialisierung:* erzeuge eine zulässige Startlösung mittels der $I1$ Einfügeheuristik von Solomon [1987].
- Schritt 2 *Verringerung der Fahrzeugzahl:* versuche, mittels $OR-OPT$ Touren mit drei oder weniger Knoten zu eliminieren, indem Sequenzen von höchstens drei Knoten in eine andere Tour eingefügt werden. Dabei muß diese zulässig bleiben und der Nachbarschaftsbedingung genügen. Falls eine Verbesserung erzielt wurde, wiederhole diesen Schritt, ansonsten weiter.
- Schritt 3, *Verringerung der Zeit:* wähle r Knoten v_i zufallsbedingt aus und teste alle $2-OPT^{\star}$s, die zulässig sind und der Nachbarschaftsbedingung genügen. Implementiere den

besten Nontabu-Tausch und erkläre den Tausch in die andere Richtung für 5 Iterationen für tabu. Wiederhole diesen Schritt, bis bis k_3 aufeinanderfolgende Iterationen keine Verbesserung erzielen.

- Schritt 4, *Verringerung der Fahrzeugzahl:* wie in Schritt 2
- Schritt 5, *Verringerung der Zeit:* wie in Schritt 3, jedoch mit $OR - OPT$ anstatt $2 - OPT^\star$. Eine Sequenz von maximal 3 Knoten kann innerhalb derselben Tour oder in eine andere Tour verschoben werden. Ende, falls k_4 aufeinanderfolgende Iterationen keine Verbesserung mehr erzielen.

Wiederhole die Schritte 2 bis 5, bis k_1 aufeinanderfolgende Iterationen ohne Verbesserung oder k_2 Iterationen insgesamt ausgeührt wurden.

Der Algorithmus wurde an Solomon's Problemen [1987] getestet. Die Probleme R1 und R2 sind zufällig in der Ebene gestreute Instanzen, C1 und C2 enthalten Gruppen von Knoten *(clusters)*, RC1 und RC2 einen Mix daraus. Jede Instanz beinhaltet 100 Knoten, die Distanzmatrix genügt der Dreicksungleichung.
Zusammenfassend kann man sagen, daß Tabusearch erfolgreich auf komplexe praktische VRPs angewandt werden konnte. Zum Beispiel lösten Semet und Taillard [1993] das Problem der Verteilung von Waren auf 45 Geschäfte in der Schweiz und erzielten bis zu 15% an Einsparungen. Rochat und Semet [1994] lösten ein Verteilungsproblem von Haustierfutter und Mehl zu zirka 100 Bauerhöfen und Läden. Der Knackpunkt daran war es, überhaupt zulässige Lösungen zu bekommen, was sich in der Praxis als überaus schwierig oder unmöglich herausstellte. Mittels Tabusearch erzeugten die Autoren in allen Fällen zulässige Lösungen und erreichten Verbesserungen um bis zu 16,6%.

		Solomon		Potvin et al.							
		I1		TABU1[c]			TABU2[d]			TABU-Best[e]	
Problem type	Number of instances	m^{a}	f^{*b}	m^{*a}	f^{*b}	Time[f]	m^{*c}	f^{*b}	Time[f]	m^{*a}	f^{*b}
*R*1	12	13.6	2 695.5	12.9	2 412.0	291	12.8	2 408.3	593	12.5	2 390.7
*R*2	11	3.3	2 578.1	3.2	2 213.4	346	3.2	2 196.6	671	3.1	2 185.7
*C*1	9	10.0	10 104.2	10.0	9 872.1	201	10.0	9 870.9	392	10.0	9 850.2
*C*2	8	3.1	9 921.4	3.0	9 615.8	200	3.0	9 611.2	441	3.0	9 594.6
*RC*1	8	13.5	2 775.0	12.8	2 519.4	221	12.8	2 513.3	478	12.6	2 507.6
*RC*2	8	3.9	2 955.4	3.5	2 516.4	305	3.5	2 498.1	660	3.4	2 459.8

[a] Best number of vehicles.
[b] Best route duration.
[c] Results with one tabu search application.
[d] Results with two tabu search applications.
[e] Best results with several tabu search applications, using various sets of parameters.
[f] Seconds on a SPARC 10 workstation.

Tab3: Ergebnisse von Tabu Search auf CVRPTW angewandt

4 Genetische Algorithmen

Evolutionäre Algorithmen waren ursprünglich zufallsgesteuerte Suchtechniken, die die natürliche Evolution von Populationen geschlechtloser Spezies nachahmten [Fogel, Owens und Walsh, 1966]. Holland [1975] ergänzte dieses Modell dadurch, indem er die Neukombination der Informationsteilstücke von zwei Eltern zuließ, um neuen Nachwuchs zu erzeugen. Dieses Verfahren nennt man *genetischen Algorithmus*, dessen wesentlich zum Erfolg beitragender Bestandteil der *Rekombinationsoperator* ist.
Genetische Algorithmen operieren auf einer *Population* von Lösungen, die als *Chromosomen*, im allgemeinen Bitstrings kodiert sind. Wir nehmen eine Population $X^t = \{x_1^t, \dots, x_N^t\}$ an, auf der eine *Fitnessfunktion* f definiert ist. Normalerweise wird f maximiert, in unserem Fall, wo es um kürzeste Touren geht, werden wir f minimieren. In der Iteration t wird X^t nach X^{t+1} überführt, dadurch werden neue Lösungen *(Nachwuchs)* aus alten *(Eltern)* erzeugt; der Nachwuchs ersetzt letztlich die Eltern. Dies wird dadurch bewerkstelligt, daß sich die Eigenschaften guter Lösungen einer Generation in der nächsten fortpflanzen, indem die Selektion in Richtung guter Lösungen ausgerichtet wird. Kleinere zufällige *Mutationen* werden auf den Nachwuchs angewendet, um eine gute Diversität von Lösungen in der Poulation zu gewährleisten. Man erhofft sich davon, daß sich der Prozeß in Richtung guter Lösungen bewegt.
Im folgenden sehen wir eine einfache Beschreibung eines genetischen Algorithmus, wobei k_1 die Anzahl der Generationen und $k_2 \leq \frac{N}{2}$ die Anzahl der Selektionen pro Generation darstellt.

In jeder Iteration $t = 1, \dots k_1$ wird k_2 mal Schritt 1 bis 3 ausgeführt, danach Schritt 4.

- Schritt 1, *Selektion*: wähle zufällig zwei Eltern aus X^t mit einer umgekehrt proportionalen Wahrscheinlichkeit zum Wert von f aus.
- Schritt 2 *One-Point Crossover:* erzeuge zwei Kindern von der Eltern, indem ein Bitstring ab einen zufällig gewählten Schnittpunkt vertauscht wird:

Parent 1	1	0	1	0	0
Parent 2	**0**	**0**	**1**	**1**	**1**
Offspring 1	1	0	**1**	**1**	**1**
Offspring 2	**0**	**0**	1	0	0

  Hier befindet der Schnittpunkt nach dem ersten Bit.

- Schritt 3, *Mutation*: Invertiere Bits in jedem Kind mit sehr geringer Wahrscheinlichkeit.
- Schritt 4, *neue Population*: X^{t+1} entsteht aus X^t durch Elimination der $2k_2$ schlechtesten Individuen, die durch die $2k_2$ generierten Kinder aus Schritt 3 ersetzt werden.

Zumal genetische Algorithmen ursprünglich in der linearen Optimierung angewandt wurden, ergeben sich im Falle VRP oder TSP einige Probleme:
Zum einen ist die Kodierung als Bitstring nicht wünschenswert. Es muß also ein ausgereifteres Kodierungsschema verwendet werden. Eine naturgemäße Darstellung wäre die der Pfadrepräsentation. Die zwei Chromosomen unten kodieren zwei verschiedene Touren durch die Knoten 1 bis 8:

Chromosome 1	2	3	5	6	4	1	7	8
Chromosome 2	1	4	2	3	6	5	8	7

Jedoch ist es unklar, ob einige theoretische Ergebnisse, die aus der Bitstringdarstellung gewonnen wurden, auf andere Darstellungen verallgemeinert werden können, und ob der genetische Algorithmus mit dieser Darstellung eine intelligente Suche auf dem Lösungraum vollführt.
Ein weitere Schwierigkeit ist die Wahl des Crossover-Operators. Beispielsweise erzeugt der One-Point-Crossover im folgenden Beispiel unzulässige Lösungen:

Parent 1	2	3	5	6	4	1	7	8
Parent 2	1	4	2	3	6	5	8	7
Offspring 1	2	3	2	3	6	5	8	7
Offspring 2	1	4	5	6	4	1	7	8

Die Tatsache, daß VRPs reine Abfolgeprobleme sind, d.h. alle Chromosomen die gleichen Werte tragen, sich nur in der Abfolge dieser Werte unterscheiden, macht spezielle Tauschoperatoren notwendig. Beispielsweise wurde für das TSP ein Cycle-Crossoveroperator [Oliver, Smith und Holland, 1987] entwickelt, der dadurch zulässigen Nachwuchs erzeugt, daß er eine Knotenmenge erkennt, die dieselbe Untermenge von Positionen in den beiden elterlichen Chromosomen belegt. Im folgenden Beispiel belegt die Knotenmenge $\{3, 4, 6\}$ jeweils die Menge der Positionen $\{2, 4, 5\}$. Daher können diese Knoten vertauscht werden, und man erhält zulässigen Nachwuchs.

Parent 1	2	3	5	6	4	1	7	8
Parent 2	1	4	2	3	6	5	8	7
Offspring 1	2	4	5	3	6	1	7	8
Offspring 2	1	3	2	6	4	5	8	7

Das VRP mit Kapazitäts- und Distanzbeschränkungen oder Zeitfenstern ist weitaus schwieriger in Chromosomen zu kodieren, als beispielsweise das TSP. Einerseits bestehen Lösungen aus mehreren Touren, andererseits müssen Nebenbedingungen erfüllt sein. Fitnessfunktionen mit zusätzlichen Termen, die unzulässige Lösungen bestrafen, und Reparaturoperatoren, die unzulässige wieder in zulässige Lösungen transformieren, wurden von Richardson et al. [1989], Siedlecki und Sklansky [1989] und Michalewicz und Janikow [1991] vorgeschlagen. Diese Veröffentlichungen betrachten jedoch spezielle Probleme, die dem VRP fern sind. Andere Arbeiten beschäftigen sich mit geeigneter Kodierung und Erfüllung von Nebenbedingungen. Die Literatur über genetische Algorithmen für das VRP fällt mager aus, die ersten Publikationen erschienen erst in 1991. Im folgenden werden drei bekannte Vorgehensweisen dargestellt.

4.1 Hybridisierung mit einer Greedy-Heuristik

Die Arbeit von Blanton und Wainwright [1991, 1992] ist vertretend für einen wohlbekannten Ansatz zur Lösung kombinatorischer Optimierungsprobleme. Sie beruht auf der Kombination

einer Greedy-Einfügeheuristik mit genetischen Algorithmen. Ein Chromosom ist hier eine Sequenz von Knoten, dessen Fitness ist der Zielfunktionswert der von der Greedy-Heuristik erzeugten Lösung. Die Greedy-Heuristik betrachtet die Knoten nacheinander und erzeugt eine zulässige Lösung durch iteratives Einfügen. Der genetische Algorithmus sucht lediglich gute Knotensequenzen; die Nebenbedingung obliegt der Greedy-Heuristik. Der verwendete genetische Algorithmus basiert auf dem GENITOR Paket von Whitley [1989], die Autoren führen jedoch zwei neue Permutationsoperatoren ein, deren Merkmal die Benutzung einer globalen Vorgängerrelation ist, um die Rekombinationsphase zu steuern.
Zumal das Problem Zeitfenster enthält, müssen die Knoten in der Einfügephase an dementsprechende Positionen gesetzt werden. So definieren die Autoren einen Präzedenzvektor, in dem die Knoten in aufsteigender Reihenfolge ihrer Zeitfenster sortiert sind. Dieser steuert den Crossover in der Form, daß er Knoten mit frühen Zeitfenstern im Nachwuchs nach vorn setzt. Die Autoren gehen nur näher auf den *Merge Cross (MX1)* Operator ein:

Precedence vector	**1**	**5**	**3**	**6**	**2**	**8**	**4**	**7**
Parent 1	2	3	5	6	4	1	7	8
Parent 2	1	4	2	3	6	5	8	7

Der Knoten 1 von Elternteil 2 hat Vorrang gegenüber Knoten 2 von Elternteil 1 (da er an früherer Position im Präzedenzvektor auftritt) und wird daher die erste Stelle im Nachwuchschromosom einnehmen. Weiterhin werden Knoten 1 und 2 von Elternteil 1 vertauscht, um zulässigen Nachwuchs zu garantieren:

Parent 1	1	3	5	6	4	2	7	8
Parent 2	1	4	2	3	6	5	8	7
Offspring	1	X	X	X	X	X	X	X

Nun werden die Knoten an zweiter Position verglichen: Knoten 3 an Elternteil 1 hat Vorrang gegenüber Knoten 4, und wird daher die zweite Stelle im Nachwuchschromosom einnehmen. Die Knoten 3 und 4 von Elternteil 2 werden dann vertauscht. Nach vollständiger Untersuchung aller Knoten, sieht der Nachwuchs wie folgt aus:

Offspring	1	3	5	6	4	2	8	7

Blanton und Wainwright [1993] beschrieben noch einen zweiten Operator MX2 als auch einen einfachen Mutationsoperator, der lediglich die Positionen zweier benachbarter Knoten in der Sequenz zufällig vertauscht. Die Fitness jedes Chromosoms wird von der Greedy-Heuristik bewertet, die die Abfolge des Chromosoms verwendet. Diese Heuristik nimmt die ersten m Knoten einer Sequenz, um m Touren zu ininitialisieren. Die übrigen Knoten werden dann in umgekehrter Reihernfolge sukzessive an einer zulässigen Position eingefügt, und zwar so, daß die zusätzliche Distanz minimiert wird. Da die Zahl der Touren festgelegt ist, wird es nicht möglich sein, jeden Knoten einzufügen. Daher werden Touren, die nicht alle Knoten enthalten, akzeptiert, und ihre korrespondierenden Sequenzen gehen in die Population ein. Natürlich

haben diese eine weitaus schlechtere Fitness als zulässige. Typischerweise läuft der genetische Algorithmus ein paar mal mit einer wachsenden Anzahl von Touren, bis überhaupt zulässige Lösungen entstehen.
Der Algorithmus wurde auf Instanzen von 15, 30, 75 und 99 Knoten mit zufallsgenerierten Zeitfenstern und Bedarfswerten angewendet. Vergleiche wurden mit MX1, MX2 dem *edge recombination crossover (ERX)* [Whitley, Starkweather und Fuquay, 1989], dem *cycle crossover (CX)* [Oliver, Smith und Holland, 1987] und dem *partially mapped crossover (PMX)* [Goldberg und Lingle, 1985] durchgeführt. Auf den größten Problemen fanden MX1 und MX2 zulässige Lösungen mit weniger Touren als die anderen Operatoren. Kein Vergleich wurde jedoch mit Approximationsalgorithmen aus der Operations Research Literatur vorgenommen.

4.2 GIDEON

GIDEON ist ein genetischer Algorithmus für CVRPTWs, der auf einer *cluster first - route second*-Strategie [Thangiah, Nygard und Juell, 1991; Thangiah, 1993; Thangiah und Gubbi, 1993] beruht. Der genetische Algorithmus wird nur in der Clustering-Phase bemüht: eine Routine namens 'genetische Sektorierung' teilt die Knoten in Sektoren oder Cluster auf, die am Depot zusammenlaufen.
Jedes Chromosom enthält eine Menge Binärzahlen, die die Winkel, die die Sektoren definieren, kodieren. Da die Kodierung also in Bitstrings erfolgt, können klassische one-point-Crossover- und Mutationsoperatoren angewandt werden, um neue Winkel zu generieren. Da sich die Population von Winkeln allmählich entwickelt, konvergiert der Algorithmus eventuell gegen eine Menge von Winkeln, die eine gute Partitionierung darstellen. Die Qualität eines Sektors (die Fitness eines Chromosomes) wird beurteilt, indem mittels *cheapest insertion* eine Tour durch die im Sektor befindlichen Knoten gelegt wird. Da der genetische Sektorierungsprozeß lediglich auf geographischen Vermutungen beruht, gibt es keinerlei Garantie, daß bezüglich der Kapazitätsbeschränkungen oder der Zeitfenster überhaupt zulässige Touren erzeugt werden können. Im Gegensatz zum Algorithmus von Blanton und Wainwright [1993] werden unzulässige Knoten in die Touren eingefügt, jedoch Strafen für Überladungen oder Zeitfensterverletzungen in der Zielfunktion eingeführt. Zum Schluß werden lokale Verbesserungsheuristiken auf die beste vom genetischen Algorithmus erzeugt Lösung angewendet. Die erste Verbesserungsheuristik bewegt Sequenzen von ein oder zwei Knoten von einer Tour in eine andere. Eine zweite vertauscht solche Sequenzen zwischen zwei verschiedenen Touren. Beide verwenden eine Fitnessfunktion mit Strafen, um eine gute zulässige Lösung zu erreichen.

4.3 GENEROUS

GENetic ROUting System von Potvin und Bengio [1996] für VRPTW vermeidet die schwierige Kodierung von Lösungen in Chromosomen, indem Crossover- und Mutationsoperatoren auf die Lösung als solche angewandt werden. Eine neue Lösung wird durch Verbindung der ersten Knoten der Tour aus Lösung 1 mit den letzten der Tour aus Lösung 2 erzeugt (wie in $2-OPT^{\star}$). Diese neue Tour ersetzt Lösung 1. Eine zweite Lösung kann generiert werden, indem man die Teile andersherum verkettet. Da der Crossoveroperator zwei Touren aus zwei verschiedenen Lösungen verbindet, ist die neue Lösung selten eine Permutation der Knoten. Einige Knoten werden dadurch verdoppelt, andere tauchen in der Lösung gar nicht auf. Man benötigt also einen *Reparaturoperator*, um eine zulässige Lösung zu bekommen. Zuerst werden Duplikate eliminiert; dies ist nicht schwer, zumal zulässige Touren zulässig bleiben, wenn Knoten entfernt

werden. Zweitens werden Knoten, die noch nicht in der Tour vorhanden sind, der neuen Lösung durch *cheapest insertion* hinzugefügt. Falls ein Knoten nicht zulässig eingefügt werden kann, wird der Nachwuchs verworfen. Es werden neue Elternlösungen für den Crossover ausgewählt, bis eine vollständige Population zulässiger Lösungen existiert. Bei den Testproblemen von Solomon war etwa 50% des Nachwuchses unzulässig. Folglich müssen etwa $2N$ Kinder erzeugt werden, um eine Population von N zu schaffen.
Obwohl dieses Vorgehen sehr teuer ist, werden auf Solomon's Problemen sehr gute Ergebnisse erzielt. Im folgenden werden diese mit den durch GIDEON erzielten Lösungen verglichen.

		GIDEON			GENEROUS			Best tabu search solution value[e]
Problem type	Number of instances	m^{*a}	$f^*(f^*)^b$	Time[c]	m^{*a}	$f^*(f^*)^b$	Time[d]	f^{*f}
*R*1	12	12.8	1 298.8 (2 528.5)	99.9	12.6	1 296.8 (2 407.4)	932.0	2 390.7
*R*2	11	3.2	1 124.7 (2 514.4)	183.2	3.1	1 167.1 (2 216.7)	1 431.6	2 185.7
*C*1	9	10.0	892.1 (9 972.4)	89.6	10.0	838.0 (9 838.0)	870.0	9 850.2
*C*2	8	3.0	749.1 (9 789.4)	142.3	3.0	596.9 (9 596.9)	1 371.6	9 594.6
*RC*1	8	12.5	1 472.2 (2 651.1)	110.0	12.1	1 446.2 (2 509.9)	937.0	2 507.6
*RC*2	8	3.4	1.411.0 (2 725.8)	159.4	3.4	1 360.6 (2 461.6)	1 456.4	2 459.8

[a] Best number of vehicles.
[b] Best distance (best route duration).
[c] Seconds on a SOLBOURNE 5/802 computer.
[d] Seconds on a Silicon Graphics workstation.
[e] Potvin et al. [1996].
[f] Best route duration.

Tab4: Ergebnisvergleich GIDEON, GENEROUS und TabuSearch

Das erste Ziel bei beiden Verfahren ist die Minimierung der Fahrzeuganzahl. Im zweiten Kriterium unterscheiden sie sich: GIDEON minimiert die Gesamtstrecke, während GENEROUS die Gesamtzeit minimiert. GENEROUS ist viel rechenintensiver, zumal es eine Lösungspopulation verwaltet, GIDEON jedoch nur eine Winkelpopulation. GENEROUS verwendet massiv lokale Verbesserungsheuristiken in Form von Mutation, während bei GIDEON nur die besten Touren mit lokalen Verbesserungsroutinen modifiziert werden. Dem Rechenaufwand von GENEROUS steht die Einsparung von sieben Touren bei Solomon's Testproblemen gegenüber.

5 Neuronale Netze

Neuronale Netze sind lernende Maschinen, die aus einfachen Prozeßeinheiten *(Neuronen)* bestehen, die wie Gehirnzellen reichhaltig miteinander verbunden sind. Die Verbindungen sind mit numerischen Werten gewichtet, die das Wissen des Netzwerkes kodieren. Ausgehend von Zufallswerten stellt ein neuronales Netz diese Werte sukzessive mittels eines Lernalgorithmus ein, um seine Fähigkeit, einen bestimmte Aufgabe zu erfüllen, zu verbessern. Verschiedene neuronale Netze wurden in letzter Zeit auf das TSP angewendet, zum Beispiel das Hopfield-Tank-Modell [Hopfield und Tank, 1985], das elastische Netz [Durbin und Willshaw, 1987] und die selbstorganisierende Karte [Kohonen, 1998a], waren jedoch nicht der Konkurrenz gewachsen [Potvin, 1993]. Die selbstorganisierende Karte kann räumliche Beziehungen zwischen Knoten verwalten, um gute Touren auf rein geographischen Problemen zu finden. Es ist nicht einfach, zusätzliche Nebenbedingungen in Betracht zu ziehen, auf VRPTW wurden neuronale Netze nach Wissen der Autoren bisher nicht angewandt.

5.1 Selbstorganisierende Karten

Selbstorganisierende Karten sind Instanzen sogenannter konkurrierender neuronaler Netzwerkmodelle [Kohonen, 1988a]. Sie organisieren sich selbständig, indem sie sukzessive die Gewichte auf ihren Verbindungen anpassen.

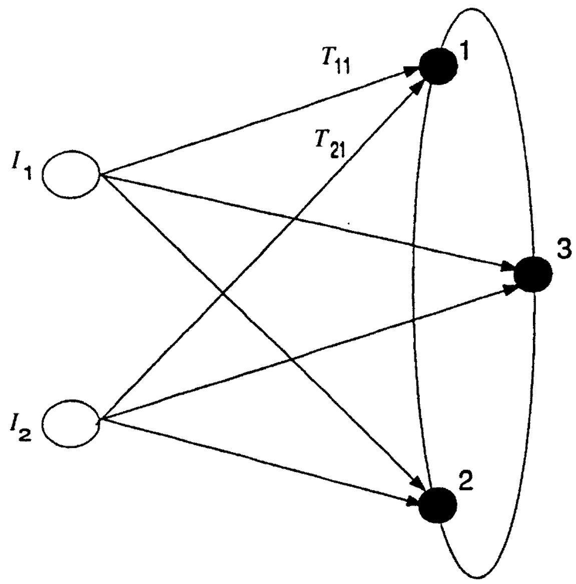

Abb3: Eine selbstorganisierende Karte mit 2 Eingabe- und einem Ring von 3 Ausgabeeinheiten

Topologische Beziehungen kommen im obigen Beispiel zwischen den Ausgabeeinheiten zustande, die Einheiten sind zu einem Ring verbunden. T_{11} und T_{21} stellen die Gewichte auf den Verbindungen zwischen den Eingabeeinheiten I_1 und I_2 zur Ausgabeeinheit 1 dar, und $T_1 = (T_{11}, T_{21})$ ist der Gewichtsvektor von Ausgabeeinheit 1.
Ausgehend von einer geordneten Menge von n Eingabevektoren in p Dimensionen und einer selbstorganisiernden Karte mit p Eingabeeinheiten und m Ausgabeeinheiten auf einem Ring, formuliert sich der Algorithmus zur Anpassung der Gewichte wie folgt:

- Schritt 1, *Initialisierung*: setze die Verbindungsgewichte auf beliebige Startwerte.
- Schritt 2, *Wettbewerb:* betrachte den nächsten Eingabevektor. Wenn alle Eingabevektoren betrachtet wurden, beginne erneut mit dem ersten. Berechne den Ausgabewert o_j jeder Ausgabeeinheit als die gewichtete Summe ihrer Eingaben:

$$o_j := \sum_{i=1}^{p} T_{ij} I_j \qquad (j = 1, \ldots, m).$$

  Der Gewinner ist die Ausgabeeinheit $j^\star$ mit maximaler Ausgabe,

- Schritt 3, *Gewichtsanpassung*: modifiziere die Verbindungsgewichtungen jeder Ausgabeeinheit wie folgt:

$$T_j := T_j + f(j, j^\star)(1 - T_j),$$

wobei $T_j = (T_{ij}), (i = 1, \ldots, p)$ der Gewichtsvektor von Einheit j und f eine Funktion von j und $j^\star$ ist.

Wiederhole die Schritte 2 und 3 bis die Gewichte der Vektoren sich stabilisieren. Zum Schluß wird jeder Eingabevektor der Ausgabeeinheit zugeordnet, die die gewichtete Summe aus Schritt zwei maximiert.

Dieses Gewichtungslernen benötigt noch zusätzliche Kommentare: In Schritt 3 bewegt sich die Funktion im Intervall $[0, 1]$ und ist typischerweise eine fallende Funktion der seitlichen Entfernung zwischen den Einheiten j und $j^\star$ auf dem Ring (die seitliche Distanz ist 1, falls die Einheiten adjazent sind, und $r + 1$, falls r Einheiten sich auf dem Ring dazwischen befinden). Der Gewichtsvektor der Einheit $j^\star$ und die Gewichtsvektoren der Einheiten, die sich nahe $j^\star$ auf dem Ring befinden, bewegen sich folglich in Richtung des Eingangsvektors I, jedoch ein geringeres Stück als $j^\star$. Oft wird f während des Lernprozesses dynamisch verändert, um die Bewegung der Gewichtsvektoren sukzessive zu verringern. Zu Beginn folgen die der gewinnenden Einheit benachbarten Einheiten stärker in deren Richtung, um Eingabevektoren in der Nachbarschaft des aktuellen Vektors zugeschrieben zu werden. Gegen Ende bewegt sich lediglich der Gewichtsvektor des gewinnenden Neurons in Richtung des aktuellen Eingabevektors, um die letzendliche Zuordnung zu fixieren.
Selbstorganisierende Karten werden benutzt, um topologische Zuordnungen von einem hochdimensionalen Eingaberaum in einen niederdimensionalen Ausgaberaum zu finden. In obigem Beispiel wird jeder zweidimensionale Eingabevektor einer eindimensionalen Ausgabe, nämlich der Position auf dem Ring zugeordnet. Die Transformation soll die topologischen Beziehungen der p Eingabevektoren erhalten, das bedeutet, Eingabevektoren, die im zweidimensionalen Eingaberaum nahe beieinander liegen, sollen Einheiten zugeordnet werden, die auf dem Ring nahe beisammen sind.
Im Kontext des VRP stellen die Eingabevektoren zweidimensionale Knotenkoordinaten dar. Es existieren mehrere verschiedene Ringe für verschiedene Touren. Schritt 2 wird dahingehend modifiziert, daß das Gewinnerneuron nun dasjenige ist, dessen Gewichtsvektor am meisten mit dem aktuellen Eingabevektor korreliert (Maximierung des inneren Produktes des Gewichtsvektors und des Eingabevektors). In euklidischen VRPs ist das Gewinnerneuron dasjenige, das den geringsten euklidischen Abstand zum derzeitigen Eingabevektor hat. Dies ist jedoch äquivalent zur Maximierung der inneren Norm, falls alle Vektoren dieselbe Norm haben.
Lediglich ein Neuron aus allen Ringen gewinnt den Wettbewerb. Die Gewichtsvektoren der Neuronen auf demselben Ring werden entgegen dem aktuellen Koordinatenvektor bewegt. Am Ende des Algorithmus wird jeder Knoten (Koordinatenvektor) der nächstgelegenen Ausgabeeinheit (Gewichtsvektor) zugewiesen. Durch diese Zuweisung definieren die Reihenfolgen der Neuronen auf den Ringen eine Knotensequenz in jeder Tour. Weiterhin werden dadurch kurze Touren vermieden, zumal benachbarte Neuronen auf einem Ring dazu neigen, in Richtung benachbarter Knoten in der euklidischen Ebene zu wandern.
Folgende Abbildung stellt die Entwicklung der Ringe in der zweidimensionalen euklidischen Ebene während des gewichtungslernenden Prozesses dar. Die Knoten sind durch weiße Kreise dargestellt, die Neuronen durch schwarze Punkte auf den Ringen. Die Koordinaten jedes Neurons sind korrespondent zu seinem aktuellen Gewichtungsvektor.

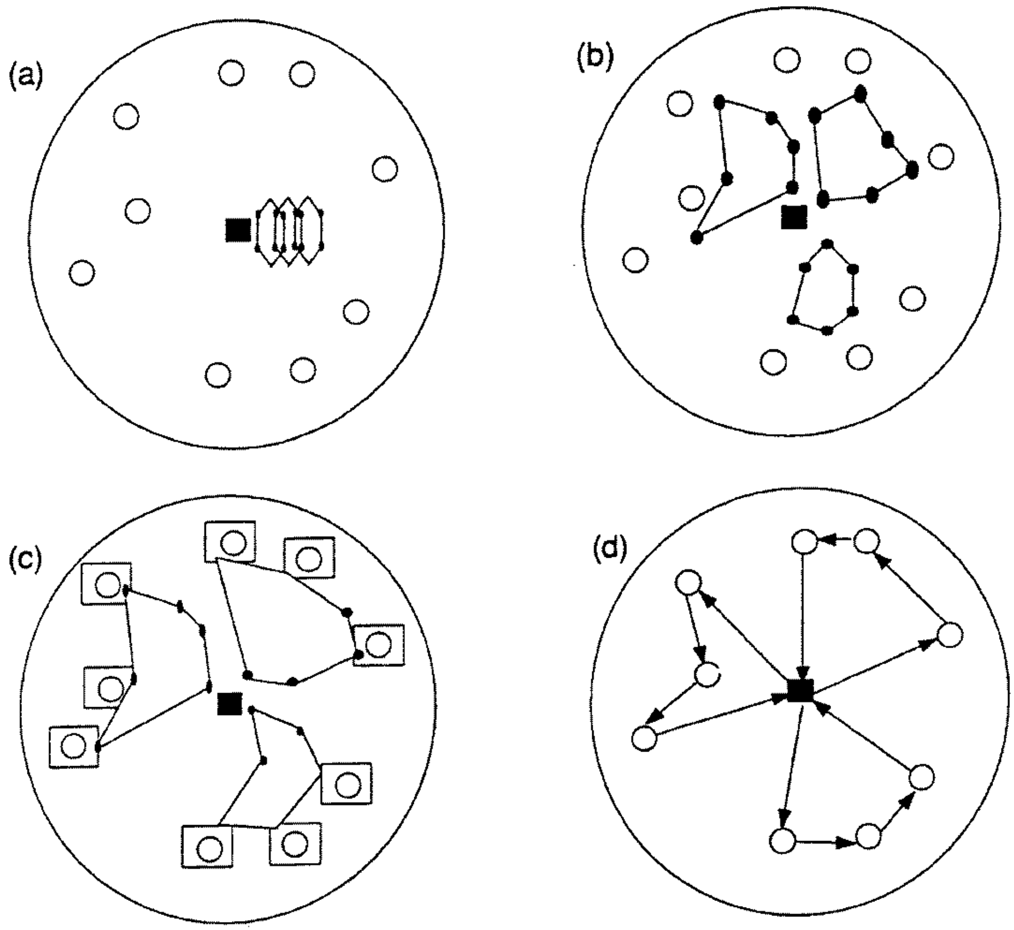

Abb4: (a,b,c) Entwicklung von drei Ringen; (d) endgültige Lösung

Ausgehend vom Initialzustand (a) werden die Vektoren sukzessive durch den Lernalgorithmus modifiziert, bis sich jeweils ein Gewichtsvektor nahe genug an jedem Knoten befindet. Die endgültige Zuordnung definiert eine Reihenfolge der Knoten in jeder Tour. Offensichtlich muß die Anzahl der Gewichtsvektoren oder Neuronen größer oder mindestens gleich der Zahl der Knoten sein. Die Abbildung zeigt weiterhin, daß die Suche nach einer Lösung in der euklidischen Ebene stattfindet. Zu Beginn, wenn die Gewichtsvektoren sehr weit von den Knoten entfernt sind, kann keine Lösung des Problems induziert werden. Sobald sich die Gewichtsvektoren den Knoten entgegen bewegen, zeichnet sich zunehmend eine Lösung ab. Die Lösung entsteht letztlich durch die endgültige Zuordnung der Knoten auf die Neuronen auf den Ringen. Im folgenden wird näher auf die selbstorganisierende Karte von El Ghaziri [1993] eingegangen.

5.2 Das hierarchisch verformbare Netz

Das hierarchisch verformbare Netz von El Ghaziri [1993] entwickelt einen Wettbewerb auf zwei verschiedenen Ebenen: ein deterministischer Wettbewerb der Neuronen innerhalb jedes Ringes und einen stochastischen der Ringe untereinander. Die Zahl der Ringe ist festgelegt, jedoch wird dieses Modell mit wachsender Anzahl von Ringen angewandt, bis eine zulässige Lösung gefunden wurde.

- Schritt 1, *Initialisierung*: setze die Knoten in zufälliger Reihenfolge. Erzeuge einen Ring per Tour. Initialisiere die Parameter h, η und G.

- Schritt 2, *Wahl des Vektors:* wähle den Koordinatenvektor I des nächsten Knotens v und setze ihn als aktuellen Eingabevektor (wenn alle Knoten behandelt wurden, starte mit Knoten 1).

- Schritt 3, *Wettbewerb innerhalb jedes Ringes*: Stelle die gewinnende Einheit $u_{j^\star k}$ innerhalb jedes Ringes r_k fest. Der Gewichtsvektor dieses Neurons ist dem aktuellen Koordinatenvektor I am nächsten.

- Schritt 4, *Wettbewerb der Ringe untereinander*: Jeder Ring r_k erhält folgende Gewinnchanchen:

$$p(r_k) := \frac{f(r_k, h)}{\sum\limits_{l=1}^{m} f(r_l, h)},$$

$$f(r_k, h) := \frac{e^{-d(T_{j^\star k}, I)/h}}{1 + e^{-\Delta Q/h}},$$

wobei h ein Parameter des Algorithmus, m die Anzahl der Ringe, $d(T_{j^\star k}, I)$ der euklidische Abstand zwischen dem Koordinatenvektor I und dem Gewichtsvektor $T_{j^\star k}$ des gewinnenden Neurons $u_{j^\star k}$ auf Ring r_k und ΔQ die Differenz zwischen der Kapazität des Fahrzeuges und der gesamten Nachfrage auf Ring r_k (inklusive Knoten v) ist.

- Schritt 5, *Auswahl des Ringes*: wähle den Ring $r_{k^\star}$ gemäß einer Auswahlwahrscheinlichkeit für jeden Ring r_k proportional zu $p(r_k)$. Knoten v wird dem Neuron $u_{j^\star k^\star}$ zugeordnet.

- Schritt 6, *Gewichtsanpassung*: bewege den Gewichtsvektor $T_{jk^\star}$ jeder Einheit j auf dem Gewinnerring $r_{k^\star}$ dem Koordinatenvektor I von Knoten v wie folgt entgegen:

$$T_{jk^\star} := T_{jk^\star} + \eta f(j, j^\star, k^\star)(I - T_{jk^\star}),$$

$$f(j, j^\star, k^\star) := \frac{1}{2}\sqrt{2} e^{-dL(j,j^\star,k^\star)^2/G^2},$$

wobei $dL(j, j^\star, k^\star)$ die seitliche Distanz zwischen Neuron j und Gewinnerneuron $j^\star$ auf Ring $r_{k^\star}$ ist.

- Schritt 7, *Parameteranpassung*: Erhöhe die Werte der Parameter h, η und G leicht.

Wiederhole die Schritte 2 bis 7 bis sich jeweils ein Gewichtsvektor nahe genug an jedem Knoten befindet.

Parameter h wird in Schritt 4 dazu benutzt, die Wahrscheinlichkeitverteilung der Ringe zu modifizieren. Wenn h groß ist, ist die Auswahlwahrscheinlichkeit jedes Ringes etwa gleich. Wenn h klein ist, wächst sie sehr rasch, da das Gewinnerneuron auf einem Ring näher zum aktuellen Knoten rückt. Weiterhin geht die Auswahlwahrscheinlichkeit eines Ringes, der die Kapazitätsbedingung verletzt, rasch gegen 0. Aus diesem Grund wird der Wert von h langsam verkleinert, um die Konvergenz gegen eine gute zulässige Lösung zu begünstigen. Es kann durchaus sein, daß die Zuordnung eines Knotens zu einem Neuron in Schritt 5 sich von Durchlauf zu Durchlauf verändert, weil der Gewichtsvektor jedes Neurons gegen ein bestimmtes Gebiet in der euklidischen Ebene konvergiert. An diesem Punkt verändern sich die Gewichte nicht sehr, und die Zuordnung der Knoten zu den Neuronen stabilisiert sich. Die Gewichtsupdates aus Schritt 6 erklären dieses Verhalten: alle Neuronen aus dem Gewinnerring bewegen sich in Richtung des aktuellen Knotens, jedoch ist die Geschwindigkeit für das Gewinnerneuron und seine unmittelbaren Nachbarn weitaus größer als für die übrigen Neuronen des Ringes. Ferner ist die Geschwindigkeit der Bewegung klein für kleine Parameter η und G. Deswegen wird der Wert dieser Parameter langsam verkleinert, um Konvergenz gegen einen stabilen Zustand zu erreichen. El Ghaziri hat dieses Modell auf die Probleme 1,3,4,5,11 und 12 von Christofides et al. [1979] angewendet, indem er reale Distanzen verwendet. Die Ergebnisse sind gut, jedoch können mit anderen Approximationsalgorithmen bessere gefunden werden. Rechenzeiten werden keine genannt.

		Hierarchical deformable net	Best known solution value
Problem	n	f^{*a}	f^{*a}
1	50	545.1	524.61[b,c,d]
3	100	911.2	826.14[b,c]
4	150	1 133.2	1 028.42[b]
5	199	1 421.5	1 298.79[b]
11	120	1 133.4	1 042.11[b,c]
12	100	826.6	819.56[b,c]

[a] Best distance.
[b] Taillard [1993a].
[c] Gendreau, Hertz & Laporte [1994].
[d] Osman [1993].

Tab5: Ergebnisvergleich für das hierarchisch verformbare Netz

6 Zusammenfassung

Die mit lokalen Suchstrategien erzielten Ergebnisse zeigen eindeutig das Potential solcher Verfahren zur Lösung komplexer und schwieriger kombinatorischer Optimierungsprobleme. Jedoch sind die angesprochenen Methoden nicht gleich erfolgreich, und es ist wichtig, einen großen Anteil problemspezifisches Wissen zu investieren. Dies kommt bei den Simulated Annealing Strategien deutlich zur Geltung, indem Algorithmen mit demselben Ansatz sehr unterschiedliche Ergebnisse erzielen. Die späteren Simulated Annealing und Tabu Search Implementationen sind zwar weitaus schwieriger zu realisieren, jedoch liefern sie exzellente Ergebnisse. Der oft gewaltige Unterschied zwischen den Rechenzeiten ist haupsächlich der Definition der Nachbarschaftsrelation und dem Spielraum der Suchstrategie zuzuschreiben. Hier ist jedenfalls eine ausreichende Suchdiversifikation notwendig, um einen Großteil des Suchraumes zu erkunden.
Vergleicht man die verschiedenen Ansätze, so fällt auf, daß neuronale Netze zur Lösung des VRP nicht sehr geeignet sind. Dies ist nicht verwunderlich, zumal diese Methode ursprünglich auch für einen anderen Zweck entwickelt wurde.
Die besten Simulated Annealing Strategien sind zwar sehr erfolgreich, jedoch nicht sonderlich robust.
Tabu Search liefert in guten Implementationen exzellente Ergebnisse. Um genetische Algorithmen erfolgreich auf VRPs anwenden zu können, ist ein großer Betrag an Findigkeit nötig. Folglich gibt es hier wenige Implementationen und wenig Erfahrung, um deren Potential wirk-

lich auszuschöpfen. Die Ergebnisse für VRPTW sind jedoch vielversprechend, was vermutlich daran liegt, daß GA mit einer Population von Lösungen arbeiten, was in einer weiter gefächerten Erforschung des Suchraumes resultiert, als es andere Methoden tun.
In Zukunft soll es mehr Versuche geben, GA und Tabu Search auf andere Varianten des VRP anzuwenden. Tabu Search wurde beispielsweise auf das $m - TSP$ mit *minmax* Zielfunktion angewandt, ein Problem mit Anwendungen in der Produktionsplanung [Franca et al., 1995]. Andere Anwendungen dieser Technik auf das VRP sind am Laufen, und die vorläufigen Ergebnisse sind sehr gut. Von methodologischer Seite aus sind einige Forschungsprojekte zu erwarten, die darauf abzielen, die besten Merkmale verschiedener Verfahren in hybriden Algorithmen miteinander zu kombinieren.