Diplomarbeit zum Thema

Entwicklung einer Software zur optimierten Auslagerung in einem Medikamentenlager

zur Erlangung des akademischen Grades
Diplom-Informatiker (FH)

vorgelegt dem
Fachbereich Informatik und Medien der

René Peschmann
11. Mai 2009

Erstprüfer: Dipl.-Inform. Ingo Boersch
Zweitprüfer: Prof. Dr.-Ing. Michael Syrjakow

Inhaltsverzeichnis

1 Einführung

Optimierung in der Logistik ist für viele Firmen ein wichtiger Kostenfaktor. Ein großes und breites Angebot an Produkten bietet zwar einen höheren Marktanteil, bedeutet aber auch größere Lager. Dadurch werden die Wege zu den einzelnen Lagerpositionen länger. Das wiederum zieht nach sich, dass die Ein- und Auslagerung von Produkten länger dauert und letztendlich auch der Verschleiß an den Maschinen größer wird, die für diese Arbeit genutzt werden. Das Ziel ist also die Minimierung des Weges, der für die Lagerarbeiten benötigt wird.
Um kurze Wege zu schaffen kann man beim Einlagern die Produkte, die öfter zusammen ausgelagert werden, bereits nahe beieinander legen. Weiterhin kann man beim Einlagern auch die Position so wählen, dass häufig ausgelagerte Produkte bereits in der Nähe des Auslagerpunktes abgelegt werden.
Eine Option beim Auslagern ist es, den kürzesten Weg für die auszulagernden Produkte vorher zu berechnen. Die Suche nach dem kürzesten Weg wird Travelling-Salesman-Problem genannt. Es existieren dafür viele exakte und heuristische Lösungsansätze, je nachdem wie genau die Berechnung sein muss, wie zeitintensiv sie sein darf und welche Nebenbedingungen eingehalten werden müssen.

1.1 Aufgabenstellung

Das Ziel der Arbeit ist für ein Lager eine Auslagerungs-Strategie umzusetzen bzw. zu modifizieren, sodass die zu fahrende Gesamtroute für eine Auslagerung bestimmter Produkte so kurz wie möglich ist. Die einzige Nebenbedingung ist, dass der Lagerschlitten, also die Einheit, die für das Greifen der Produkte zuständig ist, für eine Tour nur eine bestimmte Anzahl an Produkten aufnehmen kann. Erst nachdem die aufgenommenen Güter dann an einem definierten Ort abgelegt wurden, kann eine weitere Tour gestartet werden. Abbruchbedingung für die Berechnung der Route kann eine bestimmte Zeit sein, welche vorher definiert wird. Zusätzlich ist es wichtig zu analysieren wie effektiv bekannte Verfahren sind und ob diese hinsichtlich des speziellen Problems noch optimiert werden können. Ziel der Diplomarbeit ist die Entwicklung und Umsetzung einer Auslagerungsstrategie für eine Lagerverwaltung. Hierbei sollen Aufträge zur Auslagerung von Gütern angenommen, eine kurze Route berechnet und ein reales Lagermodell angesteuert werden. Der theoretische Schwerpunkt liegt hierbei auf der Einordnung des Routingproblems und der Auswahl eines geeigneten Optimierungsverfahrens. Das Verfahren soll in eine funktionsfähige Applikation umgesetzt werden, welche die Planungskomponente als Modul enthält, sowie eine Visualisierung des Planungsprozesses und der erstellten Routen vornimmt. Die Funktionalität und Geschwindigkeit ist durch geeignete Experimente nachzuweisen.

1.2 Aufbau des Lagers

Ausgangspunkt der Arbeit ist ein Lager für Medikamente in Form eines Regals mit gleichgroßer Fachgröße. Jedes Fach kann eine Verpackung beinhalten. Der Prototyp des Lagers wird als Lehrmittel genutzt und soll als realistisches Modell für große Anlagen dienen. Das Modell beschränkt sich dabei auf 400 Fächern, die zweidimensional angeordnet sind. Realistisch sind Lagerautomaten mit 6000 und mehr Fächern, welche dann meistens in 4 dieser zweidimensionalen Ebenen unterteilt werden (Abbildung 1).

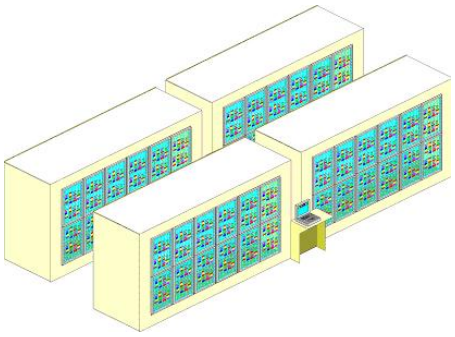


Abbildung 1: Schematische Ansicht von 4 Lagerautomaten

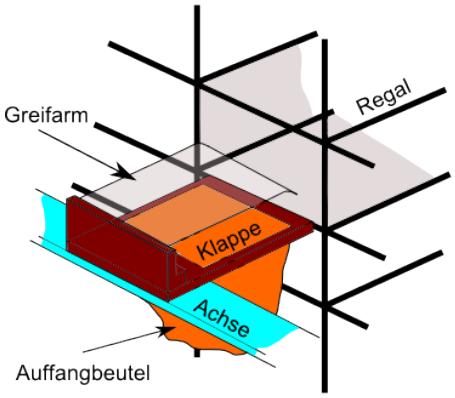


Abbildung 2: Ladekopf

Die Fachgrößen können je Lagerautomat auch unterschiedlich sein, sind aber üblicherweise innerhalb eines Automaten gleich groß. Der Lagerschlitten, welcher die Produkte an die einzelnen Fachpositionen bringen und holen kann, ist auf 2 Achsen montiert und wird mittels einer SPS gesteuert. Es können unterschiedliche Methoden für das Ein- und Auslagern der Verpackungen verwendet werden. Der erste, für Lehrzwecke optimierte Lagerautomat hat auf jeder Position eine eigene Umverpackung, die gegriffen und abgelegt werden kann. Ein weiterer Automat benutzt keine Umverpackung und kann die Verpackungen in die einzelnen Fächer schieben bzw. heraus ziehen. Beim Hineinschieben (vgl. schematische Abbildung **??**) in das Regal bleibt der Greifarm oben und die Packung wird durch einen pneumatischen Zylinder an die Lagerposition geschoben. Beim Herausnehmen fährt der Greifarm in das Fach und zieht mit einer Kante am Greifarm die Verpackung aus dem Fach auf eine Klappe, welche im Ladekopf integriert ist. Nachdem die Packung an dieser Stelle ist, öffnet sich eine Klappe, wodurch das Produkt in einen Behälter fällt.

Die Anbindung der SPS an den PC erfolgt mit der Schnittstelle prodave (DLL-Bibliothek). Für die Programmierung wird hauptsächlich Delphi7 und teilweise Turbo Delphi 2006 genutzt. Die Kommunikation findet durch einen Datenaustausch in einem Datenbaustein der SPS statt, welcher zyklisch gelesen wird und bei Bedarf beschrieben werden kann.

1.3 Pflichtenheft / Anforderungen an den Prototyp

Das Pflichtenheft ist für das komplette Medikamentenlager vorhanden. Da in der Diplomarbeit auf die Verwaltungs- und Optimierungssoftware eingegangen wird, wurden die hardwareseitige Kriterien weggelassen, welche den Verwaltungsvorgang in keiner Weise beeinflussen. Das Pflichtenheft ist weitestgehend dem Vorschlag von Helmut Balzert[Bal98] gegliedert.

1.3.1 Zielbestimmung

- Es muss eine übersichtliche Darstellung der Fächer und ihres Inhalts vorhanden sein.
- Es muss ermöglicht werden, die berechneten Wege für eine optimierte Auslagerung nur anzeigen zu lassen.
- Es muss eine Produktliste mit mindestens folgenden Eigenschaften editierbar sein: Artikelnummer, Produktname, Kommentar und Verbundprodukte.
- Der Auslagerungsvorgang sollte so weit wie möglich optimiert werden.
- Ein theoretischer Auslagerauftrag sollte durchgespielbar sein.

- Die einzelnen Parameter für Geschwindigkeit sollten eingestellt werden können.
- Eine manuelle Nachjustierung der Fachpositionen muss dem Endbenutzer ermöglicht werden. (Kann auch extern umgesetzt werden)
- Ein Einpflegung von Haltbarkeitsdaten ist nicht vorgesehen.
- Die Verwaltungssoftware ist kein Verkaufssystem und sollte daher kritische Bestandsmengen, die Mengenverwaltung- im Allgemeinen, Nachbestellung von Produkten ungeachtet lassen.

Für den Lehrautomaten gilt weiterhin:

- Die Kommunikation zwischen SPS und PC sollte visualisiert werden.
- Eine automatische Inventur sollte die Gültigkeit der Fachinhalte überprüfen können.
- Eine umfangreiche Statistik, welche die Belegung des Lagers anzeigt sollte realisiert werden.
- Neben dem Barcode-System wäre ein RFID-System für die Umverpackungsvariante wünschenswert.

1.3.2 Produkteinsatz

Der Prototyp ist als Lehrmittel vorgesehen und soll möglichst realitätsnah ein komplettes Medikamentenlager umfassen. Verschiedene Fachbereiche an Hochschulen sollen Aufgaben im Bereich der Logistik, Informatik, im Ingeneurwesen und in der Automatisierungstechnik anhand dieses reellen Beispiels lehren können.

1.3.3 Produktübersicht

Geliefert wird ein funktionstüchtiger Lagerautomat sowie eine vorinstallierte modifizierbare Lagerverwaltungssoftware mit Sourcecode und Compiler. Ausserdem bekommt der Kunde eine USV, Geräte für das Drucken und Lesen des Barcodes sowie das Lesen und Beschreiben von RFID-Chips.

1.3.4 Produktfunktionen

Die Lagerverwaltungssoftware ist ein Programm mit dem ein Lager komplett gesteuert werden kann. Dazu gehören nicht nur Ein- und Auslagervorgänge, sondern auch statistische Angaben und ein umfangreiches, aber stabiles System, welches typische Funktionen eines Lagers abdecken. Das Einlagern erfolgt über einen Barcode oder einem RFID-Systems, welche an einer Umverpackung den Inhalt kennzeichnet und so das Gut im Lager sicher einlagern kann. Beim Einlagern wird die Markierung (der Barcode oder der RFID-Chip) auf der Umverpackung ausgelesen. Somit kann in der Datenbank das Produkt eindeutig zugewiesen werden und an einem leeren günstigen Platz einlagern. Mit der Software ist es dann möglich über Netzwerk oder über eine Suchfunktion Produktgruppen oder einzelne Produkte auszulagern. Durch Überprüfung der Markierung kann immer überprüft werden, ob das Produkt was ausgelagert werden soll, wirklich noch vorhanden ist oder ob bspw. nach einem Stromausfall eine manuelle Entnahme stattfand.

1.3.5 Produktdaten

Die Daten sollten transparent gehalten werden, so dass beispielsweise auf die Datenbank auch mittels Drittprogrammen ein Zugriff erfolgen kann. Dadurch, dass der Quellcode offen liegt, kann so auch eine Modifizierung der Datenbank und in der Lagerverwaltungssoftware erfolgen.

1.3.6 Produktleistungen

Die Geschwindigkeit der Software sollte einen schnellen Zugriff auf die Datenbank ermöglichen und keine spürbaren Verzögerungen beim Einlagern oder Auslagern entstehen lassen. Hardwareseitig muss ein Kompromiss gefunden werden, welcher zwischen hoher Geschwindigkeit und den dadurch höher werden Materialverschleiss ein Optimum darstellt.

1.3.7 Qualitätsanforderungen

Der Computer sollte nicht die ganze Zeit voll ausgelastet sein, auch wenn ein schneller Datenaustausch mit der SPS eine ständige Überwachung des verwendeten Profibus erfordert.

1.3.8 Benutzungsoberfläche

Wichtige Funktionen sind mit einem integrierten Passwortprogramm geschützt. So können bspw. bestimmten Zonen im Lager nur von berechtigten Personen angefahren werden. Da die Verwaltungssoftware eine Komplettübersicht über das ganze Lager bieten soll und ebenfalls noch Platz für Suche und Statistik reserviert werden muss, ist die Umsetzung der Software über mehrere Monitore sinnvoll.

1.3.9 Nichtfunktionale Anforderungen

Die Sicherheitsanforderungen müssen primär hardwareseitig umgesetzt werden.

1.3.10 Technische Produktumgebung

Die Software sollte auf einem Windows 2000/XP/Vista - System laufen. Für die Verwaltungssoftware ist keine spezielle Hardware notwendig. Lediglich für die Anbindung an die SPS mittels einer Profibus-Kommunikationskarte muss ein PCI-Slot eingerechnet werden. Für den Zugriff und der Visualisierung über Netzwerk wird ein Ethernetanschluss benötigt. Die Statusmeldung der USV erfolgt über USB.

1.3.11 Spezielle Anforderungen an die Entwicklungsumgebung

Zur Programmierung soll eine Hochsprache benutzt werden. Ein gut kommentierter Quellcode in C++, DELPHI oder JAVA soll den Auszubildenden bzw. Studenten die Möglichkeit bieten, die Vorgehensweise detailiert nachzuvollziehen. Ebenfalls muss die Datenbankstruktur und die Kommunikation mittels SPS genau beschrieben sein.

1.3.12 Ergänzungen / Sonstiges

Backups von der Lagerverwaltungssoftware und der SPS-Software werden auf CD mitgeliefert.

2 Einordnung des Problems

Um einen brauchbaren Lösungsansatz umzusetzen muss das Problem genau kategorisiert werden. Es gibt eine Vielzahl von Problemformulierungen, die sich nicht unbedingt decken bzw. zu viel Spielraum der einzelnen Parameter zulassen.

2.1 Der Hamiltonkreis

Das Hamiltonkreisproblem ist ein Problem der Graphentheorie. Jeder Graph (G) enthält Knoten (V) und Kanten (E) mit $n = |V|$ Knoten und $m = |E|$ Kanten. Es gilt $G = (V, E)$. Ein Hamiltonkreis ist ein Pfad in dem Graphen G, der jeden Knoten V genau einmal enthält. Jeder vollständige Graph mit $n \geq 3$ ist hamiltonisch.

2.2 Das Travelling Salesman Problem (TSP)

Das Travelling Salesman Problem ist ein Optimierungsproblem der theoretischen Informatik. Die Aufgabe ist es, die Reihenfolge für den Besuch mehrerer Orte so zu wählen, dass der Gesamtweg möglichst kurz ist. Projeziert auf einen Graphen wird der kürzeste Hamiltonkreis gesucht.
Analog zu dem vorhandenen Lagerautomaten würde ein kurzer Weg gesucht werden, der ohne jegliche Nebenbedingung einmal die ausgewählten Fachpositionen anfährt.
Wegen der Analogie wird im Hauptteil der Arbeit kein Unterschied mehr zwischen Städte, Orte, Knoten und Fachpositionen gemacht. Wege, Pfade und Kanten sind somit ebenfalls gleichbedeutend.
Die optimale Route im Standardproblem (vgl. Kapitel 3.1) mit 48 Städten (att48) ist hier zu sehen:

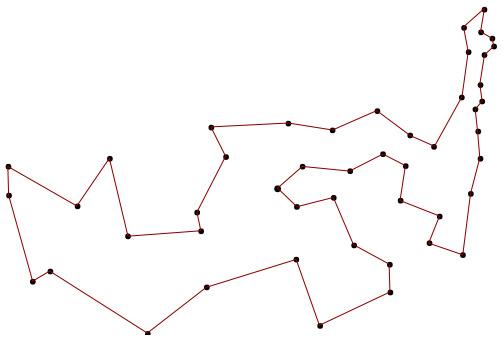


Abbildung 3: att48 - TSP optimal gelöst

Eine formale Beschreibung liefert eine gute Übersicht über den Grundgedanken des Problems. Minimiert werden soll die Zielfunktion f, welche die Länge der Tour repräsentiert. Dazu wird die zulässige Menge S der Touren auf die reellen Zahlen $\mathbb{R}$ abgebildet. [Pap82, S. 4]:

$$f : S \rightarrow \mathbb{R} \tag{2.1}$$

Gesucht wird die optimale Tour $s^* \in S$ von allen $s_k \in S$, für die gilt: $f(s^*) \leq f(s_k)$. Die Zielfunktion f benötigt hierbei alle Abstände zweier Städte i und j zueinander, welche in einer Distanzmatrix $[d_{ij}]$ eingetragen sind. Repräsentiert $\pi_k(i)$ dann die Nachfolgestadt in der Tour s_k, kann die zu minimierende Zielfunktion wie folgt dargestellt werden:

$$f(s_k) = \sum_{i=1}^{n} d_{i\pi_k(i)} \tag{2.2}$$

TSP gilt als NP-vollständig (vgl. [Lae08, S. 129], [DHF01, S. 174]), da wahrscheinlich kein effizienter Lösungsalgorithmus, also kein Algorithmus mit der Komplexität von $O(n^k)$ (polynomielle Komplexität) [Lan06, S. 145] existiert. Es müssen also gute Approximationen gefunden werden. Die Anzahl der verschiedener Routen $r = |S|$ bei einem TSP mit n Städten lässt sich berechnen mit:

$$r = \frac{(n-1)!}{2} \tag{2.3}$$

So existieren für ein 48-Städte-TSP $\approx 1,29 * 10^{59}$ mögliche Routen.
Es besteht die Möglichkeit, dass der Graph auf dem das TSP basiert nicht vollständig bzw. asymmetrisch ($d_{ij} \neq d_{ji}$) aufgebaut ist. Da aber alle Wege vorhanden sind und die Städte 2-dimensionale Koordinaten haben, ist der Graph und somit auch das TSP symmetrisch aufgebaut.

$$d_{ij} = d_{ji} \; \forall \, i, j \in \{1..n\} \tag{2.4}$$

2.3 Das Vehicle Routing Problem (VRP)

Das Vehicle Routing Problem ist ein TSP mit meist kapazitiven oder aber anderen Nebenbedingungen. Dieses Problem ist in der Praxis entstanden, dadurch das Logistikunternehmen aus einem oder mehreren Depots mit mehreren LKWs viele Kunden beliefern müssen, die Lastwagen aber nur eine begrenzte Kapazität haben. Andere Nebenbedingung sind beispielsweise das manche Lebensmittel gekühlt oder in einem bestimmten Zeitrahmen ausgeliefert werden müssen. Ziel ist es, die Kunden möglichst kostengünstig zu beliefern.
Es gibt sehr viele verschiedene bekannte Probleme, die dem VRP zuzuordnen sind. Einen kleiner Auszug (aus [Tot01]) über die wohl meist behandelten Probleme zeigt Tabelle 1.

	Name	Beschreibung
CVRP	Capacitated VRP	Alle Transporter haben eine gleiche eingeschränkte Ladekapazität.
DVRP	Dynamisches VRP	Es kann während der Berechnung weiterer Bedarf entstehen.
MDVRP	Multiple Depot VRP	Die Transporter können von mehreren verschiedenen Depots starten.
PVRP	Periodisches VRP	Der Bedarf der Kunden kann in zeitlichen Abständen nachwachsen.
SDVRP	Split Delivery VRP	Ein Kunde kann von verschiedenen Transportern beliefert werden.
SVRP	Stochastisches VRP	Auslieferungs- und Servicezeitkosten sind nicht eindeutig.
VRPB	VRP with Backhauls	Die Abgabemengen der Lieferanten werden berücksichtigt.
VRPPD	VRP with Pickup and Delivery	Zwischen den Auslieferungen können auch Waren aufgenommen werden.
VRPTW	VRP with Time Windows	VRP unter Berücksichtigung, dass Kunden nur in einem bestimmten Zeitfenster beliefert werden können.

Tabelle 1: Auszug aus bekannten VRP - Problemen

2.4 Das Capacitated Vehicle Routing Problem (CVRP)

Das kapazitierte Vehicle Routing Problem ist eine Variante des VRP bei der die Transporter alle die gleiche Kapazität haben. In der Literatur wird manchmal davon ausgegangen, dass auch der Platz, den ein Produkt benötigt bei allen Produkten gleich ist. Leider wird hier keine weitere Unterscheidung gemacht, welche das Problem spezifiziert. Die mathematische Formulierung ist wie beim TSP mit der zusätzlichen Nebenbedingung, dass die Kapazität Q eines einzelnen Transporters für seine Route R_i nicht überschritten werden darf: $Q : \sum_{i=1}^{m} k_i \leq Q$. Dabei sind k die Kosten einer Stadt, die beispielsweise durch die Größe des auszuliefernden Paketes entstehen.

2.5 Zusammenfassung

Im gegebenen Lagerautomaten existiert zwar im Gegensatz zur praxisrelevanten Problemstellung des CVRP nur eine Transportmöglichkeit, das Problem ist aber dennoch ähnlich. Die Hauptbedingung des CVRP, also die gleiche Lagerkapazität wird dadurch erfüllt, dass für die einzelnen Touren immer derselbe Transporter (der Ladekopf) existiert. Das die Touren nicht parallel durch mehrere Transporter sondern nur sequentiell abgearbeitet werden können, ist nicht ausschlaggebend und kann deshalb vernachlässigt werden.
Ein Sonderfall ist trotzdem gegeben, da das Lager gleiche Fachgrößen hat und somit die Güter immer eine gleichgroße Einheit an Platz im Transporter belegen. Es existieren also keine verschieden kostenverursachenden Städte, auf die manche Buchautoren eingehen. Auch wenn die allgemeine formale Nebenbedingung für diesen Spezialfall hinreichend ist, kann nun noch definiert werden, dass die Kosten für eine Stadt gleich sind ($k\ const.$) und nun die Kapazitätsrestriktion gekürzt werden kann auf $Q : m \leq Q$.

2.6 Analyse

Die Anzahl der möglichen Routen für dieses spezifische CVRP ist zwar berechenbar, vermutlicherweise gibt es aber keine zusammenfassende Formel. Die Gesamtroute besteht immer genau aus $\lceil n/c \rceil$ Teilrouten. Um die Anzahl zu berechnen kann zunächst eine Teilroute als eigenständiges TSP betrachtet werden, da bei einer Auswahl an h Städten ($h = |H|$ und $H \subset V$) lokal die optimale Tour gefunden werden muss. Die Anzahl der möglichen Routen r_h beträgt zwischen diesen h Städten:

$$r_h = h!/2 \tag{2.5}$$

Je nach Kapaziät werden h Städte für diese Teilberechnung herausgegriffen, sodass es für h Städte insgesamt

$$h_c = \binom{n}{h} \tag{2.6}$$

Möglichkeiten gibt. Die Formel für die Anzahl aller möglichen Routen der ersten h herausgegriffenen Städte lautet somit:

$$a_1 = h_c * r_h = \binom{n}{h} * h!/2 \tag{2.7}$$

Falls vorhanden, wird mit den verbleibenden $n - h$ Städten (a_2 bis a_x) genauso verfahren, nur mit der Berücksichtigung, dass nun die Kapazität der nächsten Teilroute(n) nicht ausgelastet sein muss. Für $V_z = 9$ Zielstädte und einer Kapazität $Q = 3$ kann somit die Gesamtroute berechnet werden mit:

$$\begin{aligned} r_{9,3} &= (\binom{9}{3} * (3!/2)) * (\binom{6}{3} * (3!/2)) * (\binom{3}{3} * (3!/2)) \\ &= \frac{9*8*7}{2} * \frac{6*5*4}{2} * \frac{3*2*1}{2} \end{aligned} \tag{2.8}$$

Falls die Anzahl an Zielstädten V_z nicht restlos durch die Kapazität Q geteilt werden kann, so können eventuell mehrere Kombinationen existieren, welche dennoch nur aus $\lceil n/c \rceil$ Teilrouten bestehen. So wären bei $V_z = 7$ mit $Q = 3$ die Kombinationen $\{3, 3, 1\}$ und $\{3, 2, 2\}$ existent.

Tritt so ein Fall ein, so müssen die verschiedenen Routenanzahlen der Kombinationen addiert werden. Für dieses Beispiel würde die Formel dann lauten:

$$\begin{aligned} r_{7,3} &= (\binom{7}{3} * (3!/2)) * \left((\binom{4}{3} * (3!/2)) + (\binom{4}{2} * (2!/2)) \right) \\ &= \frac{7*6*5}{2} * \left(\frac{4*3*2}{2} + \frac{4*3}{2} \right) \end{aligned} \tag{2.9}$$

Die Tabelle 2 gibt einige Parameterbeispiele für das spezielle CVRP sowie derer Gesamtroutenanzahl. Der Quellcode für die Berechnung dieser Daten ist im Anhang (Quellcode 11). Diese Analyse der Problemstruktur erfüllt den Zweck, das spezielle Problem besser zu verstehen und eventuell so verschiedene existene Herangehensweisen (vgl. Kapitel 3) besser in ihre Effektivität einzuordnen.

Zielstädte	Kapazität	Kombinationen	mögliche Routen
5	3	$\{3,2\}$	30
5	4	$\{4,1\}$, $\{3,2\}$	90
8	3	$\{3,3,2\}$	5040
8	4	$\{4,4\}$	10080
8	5	$\{5,3\}$, $\{4,4\}$	20160
12	4	$\{4,4,4\}$	$59'875'200$
12	5	$\{5,5,2\}$, $\{5,4,3\}$, $\{4,4,4\}$	$179'625'600$
12	6	$\{6,6\}$	$119'750'400$
12	7	$\{7,5\}$, $\{6,6\}$	$239'500'800$
17	4	$\{4,4,4,4,1\}$, $\{4,4,4,3,2\}$, $\{4,4,3,3,3\}$	$4.45 * 10^{13}$
48	14	15 verschiedene	$1.16 * 10^{61}$
48	15	34 verschiedene	$2.64 * 10^{61}$

Tabelle 2: Auszug aus dem speziellen CVRP und deren mögliche Routenanzahl

3 Bekannte Lösungsverfahren

3.1 Vergleichbarkeit / Standardprobleme

Das Travelling-Salesman-Problem als mathematisches Optimierungsproblem wurde bereits 1930 von Karl Menger [Men98, S. 12] in einem mathematischen Kolloquium formuliert:

> Wir bezeichnen als Botenproblem (weil diese Frage in der Praxis von jedem Postboten, übrigens auch von vielen Reisenden zu lösen ist) die Aufgabe, für endlich viele Punkte, deren paarweise Abstände bekannt sind, den kürzesten die Punkte verbindenden Weg zu finden.

Das damals noch Botenproblem genannte Thema erfreute sich schnell größer werdender Beliebtheit, sodass Christofides und Eilon 1969 konkrete Probleme mit einer bestimmten Anzahl von Städten und kapazitiven Bedingungen beschrieben. So wurden Muster geschaffen, welche für das CVRP genutzt und durch weglassen der Nebenbedingungen ebenfalls als VRP oder TSP angesehen werden konnten. Für sehr viele Spezialfälle wurden ebenfalls Standardprobleme geschaffen. Dadurch entstand die Möglichkeit, dass sich nun verschiedene Algorithmen an denselben Problemen messen konnten. Anzumerken ist allerdings, dass zwar die gefundenen Weglängen für diese Fälle vergleichbar sind, jedoch selten Aussagen über die benötigte Rechenzeit und Rechenkapazität getroffen werden.
Durch große Rechnernetzwerke konnten bis jetzt alle bedeutenden Problemstellungen perfekt gelöst werden. Mit verschiedenen Algorithmen wird nun allgemein probiert Zeit einzusparen oder nach Vorgabe einer Resourcenbeschränkung das globale Optimum um einen bestimmten Grad nicht mehr zu verfehlen. Mit steigender Rechenkapazität werden außerdem komplexere oder riesige Probleme gelöst, wie die Findung des optimalen Arbeitsweges auf einer Leiterplatte mit 33810 Bohrlöchern [Hus07, S. 113]. Eine Möglichkeit mit dieser Rechenkapazität eigene simple TSP-Probleme ohne Nebenbedingung zu lösen, findet sich im Internet unter den Namen Concorde TSP Solver[1].
Eine Problemformulierung ist auch für das Medikamentenlager zu gebrauchen, sodass eine Beispielaufgabe mit einer besten Lösung vorliegt. Die Anordnung der Städte ist bekannt als „att48“, wobei die Kosten je Stadt auf 1 angeglichen wurden. Der konkrete Name ist „att-n48-k4.vrp“. Dabei handelt es sich um 47 Städte, die mit Routen zu einer Kapazität von je 15 Einheiten optimal abgefahren werden sollen.
Falls das eine Beispiel nicht reichen sollte, so besteht die Möglichkeit weitere Probleme zu konstruieren, welche dann durch das Opensource-Paket SYMPHONY in vertretbare Zeit die optimale Lösung errechnet. In dieser Software wird das Branch-and-Cut-Verfahren (Kapitel 3.3.2) benutzt. Ein zufällig erstelltes 80-Städte-CVRP konnte innerhalb einer Minute auf dem aktuellen System gelöst werden. Ein 50-Städte-Problem mit einer Kapazität von $Q = 6$ hingegen benötigte knapp 6 Minuten.

3.2 Lösungsverfahren für das CVRP

Die Lösungsverfahren kann man in zwei Kategorien einteilen, welche miteinander kombiniert werden können. Exakte Lösungsverfahren finden nachweisbar eine Optimallösung. Heuristische Verfahren finden in kurzer Zeit gute Lösungen. Charakteristisch ist die recht lange Laufzeit für die exakten Verfahren und die schnelle Findung eines lokalen Optimums für die heuristischen Verfahren.

[1] http://neos.mcs.anl.gov/neos/

3.3 Exakte Verfahren

3.3.1 Vollständige Enumeration

Bei einer vollständigen Enumeration (auch brute-force genannt) werden alle Lösungen des Lösungsraums systematisch durchlaufen. In Bezug auf das CVRP würde jede mögliche Reihefolge der einzelnen Orte getestet werden, sodass das globale Optimum auf alle Fälle gefunden wird. Die Komplexität ist $O(n!)$ für ein n-Städte-CVRP. Der Nachteil bei dieser Methode ist, dass die Menge der Routen mit steigender Städteanzahl stark ansteigt (vgl. Tabelle 2) und somit die Rechenzeit ins Unermessliche treibt. Anzumerken ist jedoch die gute Parallelisierbarkeit dieser Methode, da das TSP in Entscheidungsbäume dargestellt werden kann [Ohr08, S. 57] und so bspw. jeder Zweig durch ein verfügbares System abgearbeitet werden kann.

3.3.2 Branch-and-Cut

Um Entscheidungsbäume nicht vollständig enumerieren zu müssen, wurden Verfahren entwickelt, die überprüfen ob einige Äste außer Betracht gelassen werden können. Diese Behandlungsmethode wird Branch-and-Bound genannt [Ohr08, S. 57 - 60] und kommt aus dem Bereich der ganzzahligen linearen Optimierung.
Bei diesem Verfahren werden zunächst eine untere und obere Schranke berechnet. Die untere Schranke gibt einen Zielfunktionswert an und wird relaxiert in dem Nebenbedingungen entfernt werden [Rie08, S. 53]. Diese relaxierte Variante des Problems kann mit Hilfe von Methoden der ganzzahlig linearen Optimierung gelöst werden. Da die Relaxation üblicherweise nicht ganzzahlig ist, kann verzweigt mit einer auf- und abgerundeten Variante weiter gerechnet werden [Ohr08, S. 58]. Eine obere Schranke wird durch ein heuristisches Verfahren gebildet (vgl. [Arn08, S. 50-51], [Rie08, S. 53-58]).
Nun werden für alle Unterknoten P_i, $i = 1, ..., k$ eines Knoten P die Relaxationen gelöst und die unteren Schranken (LB_i) und oberen Schranken (UB_i) berechnet. Die unterste Schranke wird LB, die oberste UB genannt. Im Bounding - Schritt werden nun die Äste des Baumes abgeschnitten für die gilt [Rie08, S. 54]:

- LB_i ist größer oder gleich UB
- Ist $LB_i < UB$ und LB_i ist eine zulässige Lösung für P, so ist UB nun LB_i
- Die Relaxation von P_i besitzt keine zulässige Lösung

Dieser Schritt wird solange wiederholt, bis ein globales Optimum gefunden wurde. Dieses Optimum ist gegeben, wenn die beiden globalen Schranken gleich groß sind $LB = UB$.
Branch-and-Cut stellt eine Erweiterung von Branch-and-Bound dar und wurde von Grötschel (1984) sowie Padberg und Rinaldi (1987) entwickelt [Rie08, S.59]. Die Idee ist sukzessive Nebenbedingungen in Form von Schnittebenen hinzuzufügen, durch die dann weitere Knoten ausser Betracht gelassen werden können. Diese Strategie wurde in der Dissertation [Bla99] auf das CVRP angewendet und ist selbst bei verschiedenen großen Problembeispielen sehr effektiv (vgl. Kapitel 7).

3.4 Heuristische Verfahren

Es gibt sehr viele heuristische Verfahren für das CVRP, die im Rahmen der Diplomarbeit nicht alle näher beschrieben werden können. Je nachdem, ob der heuristische Algorithmus eine Tour

konstruiert oder eine bestehende Tour zu verbessern versucht, werden sie als Konstruktions- oder Verbesserungsverfahren bezeichnet. Eine Kombination von mehreren Verfahren heißt Metaheuristik [Gei05, S. 51] und wird häufig verwendet, da durch ein gutes Konstruktionsverfahren bereits ein relativ guter Anfang vorliegt und somit Zeit eingespart werden kann.

3.4.1 Eröffnungs-Heuristiken

- **Nearest-Neighbor-Verfahren**

Bei der Nearest-Neighbor-Heuristik wird immer ein Weg zur nächstliegenden, noch nicht besuchten Stadt der Gesamtroute hinzugefügt. Die Ausgangssituation spielt dabei keine Rolle. Werden anfangs noch kurze und damit auch Erfolg versprechende Kanten generiert, so müssen je weniger unverbundene Städte verbleiben immer längere Wegstrecken in Kauf genommen werden. Ein großer Nachteil entsteht auch, weil nach einer ausgewählten Stadt der nächste kürzeste Weg gesucht werden muss und durch Vergleich aller übrig gebliebenen Wegstrecken die Komplexität des Algorithmus dadurch $O(n^2)$ ist. Die Länge der ermittelten Tour (Nb) zur optimalen Tour (Opt) ist nach [Ros77]:

$$\frac{Nb}{Opt} \leq \frac{1}{2}\lg(n) + \frac{1}{2} \tag{3.10}$$

- **Cheapest-Insertion-Verfahren**

Ein Einfügeverfahren fügt durch aufbrechen einer vorhandenen Wegstrecke nacheinander Städte in die bestehende Route ein. Das Cheapest-Insertion-Verfahren ist ein Einfügeverfahren, welches eine gewählte Stadt genau dort in die Route eingefügt, wo ein Ersetzen der Wegstrecke durch zwei Teilwegstrecken zu der neuen Stadt minimale zusätzliche Kosten benötigt. Die Länge ist nach [Ros77]:

$$\frac{Cheap}{Opt} \leq 2 \tag{3.11}$$

- **Cheapest-Farthest-Insertion-Verfahren**

Diese Methode erweitert das Cheapest-Insertion-Verfahren um eine Regel. Die am weitesten von den bereits erfassten Städten entfernte Stadt wird zum Einfügen in die Route verwendet. Laut [Wan06, S. 42] scheint dieser Algorithmus die beste Eröffnungsheuristik zu sein. Jedoch wurde diese Aussage für das TSP getroffen. Es bleibt zu überprüfen, ob die Qualität auch im speziellen CVRP erhalten bleibt.

- **Savings-Verfahren**

Das Savings-Verfahren geht zurück auf Clarke und Wright [Zim07, S. 291]. Zunächst führt zu jeder Stadt eine eigene Tour. Im Anschluss wird ein Savings-Wert für je zwei Touren berechnet, welcher die Kostenreduktion bei Zusammenlegung dieser Touren repräsentiert. In absteigender Reihenfolge dieser Savings-Werte werden nun die Touren zusammengefasst, die keine Nebenbedingung verletzen. Dieses Grundkonzept wurde von vielen Autoren modifiziert. So können unter anderem mehrere Touren bei der Zusammenlegung berücksichtigt werden [Ohr08, S. 65] und somit bessere Optima für eine Route gefunden werden.

- **Christofides Heuristik**

Die von Louis N. Christofides 1976 entwickelte Algorithmus kann für ein symmetrisches TSP eine Route errechnen, dessen Länge nachgewiesen das 1.5-fache des Optimums nicht überschreitet [Wan06, S. 43]. Dieser Algorithmus findet beim CVRP keine Anwendung, da unbekannt ist, wie Teilrouten aus diesem Algorithmus gewonnen werden können. Außerdem ist das Resultat zu berechnend, so dass für Algorithmen, welche Kombinationen von mehreren Ausgangsrouten voraussetzen, nicht genügend Vielfalt geboten wird (siehe Evolutionäre Algorithmen).

3.4.2 Verbesserungsverfahren

- **Kantentausch-Verfahren**

Beim Kantentausch-Verfahren wird eine bestimmte Anzahl an Kanten durch die gleiche Anzahl anderer Kanten ersetzt, falls dies eine Verkürzung der Tour zur Folge hat.

- **Das k-opt-Verfahren**

Die k-opt-Heuristik ist ein Kantentausch-Verfahren bei welchem jeweils k beliebige Kanten mit k anderen Kanten dann vertauscht werden, wenn sich die resultierende Route dadurch verbessert. Der Aufwand beträgt $O(n^k)$ [Ohr08, S. 68]. Die Anwendung von $k > 3$ scheint dabei aber nicht mehr sinnvoll zu sein [Wen95, S. 23]. Die häufigsten Umsetzungen sind also 2-opt und 3-opt, wobei diese auch hintereinander ausgeführt werden können, was schneller zu Ergebnissen führt [AIK08, S. 149]. Ein 2-opt bewirkt beim TSP einen Umtausch der Fahrtrichtung auf die Teilroute, welche zwischen den getauschten Kanten liegt. Dabei können Nebenbedingungen verletzt werden, welche allerdings beim normalen CVRP an dieser Stelle nicht vorhanden sind, da sich die Kapazität der Städte dieser Teilroute nicht ändert. Wird 2-opt über 2 verschiedene Teilrouten angewendet, so muss überprüft werden, ob die Kapazitätsrestriktion nicht verletzt wird. In Abbildung 4 würde die Gesamtroute bei einer Kapazität von $Q = 6$ valide sein, bei $Q = 5$ hingegen ungültig. Weiterhin ist in diesem Beispiel erkennbar, dass die Gesamtstrecke der Kanten, die ausgetauscht werden sollen, kürzer ist als die neu erstellten Kanten. Dadurch, dass die anderen Kanten gleichlang bleiben, kann schnell auf eine Verschlechterung geschlossen werden. Es reicht also ausschliesslich die Summe der Länge der zu verändernden Kanten zu vergleichen.

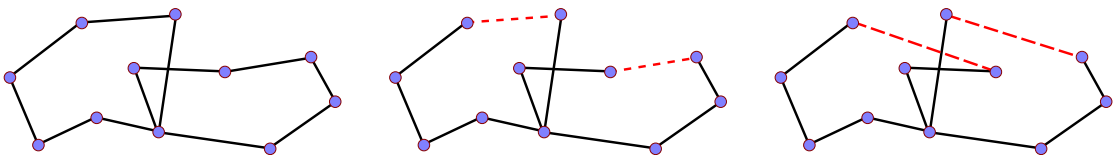


Abbildung 4: 2-opt über 2 Teilrouten

- **Evolutionäre Algorithmen**

Ein Evolutionärer Algorithmus ist ein Ansatz den natürlichen Lebenslauf einer Bevölkerung teilweise maschinell nachzuahmen. Schon viele Technologien wurden durch erfolgreiche Nachahmung der Natur verbessert, wie bspw. das Profil von Reifen mittels genetischer Algorithmen [Tec06]. Ein Ansatz ein evolutionäres Verfahren für das CVRP zu benutzen, wird durch Co-operated Simulated Annealing umgesetzt. Die Stärke dieses Algorithmus spiegelt sich in der Geschwindigkeit wieder und wird daher als Ausgangspunkt der Lösungsanpassung genommen.

3.4.3 Cluster first - Route second

Ein etwas andere Betrachtungsweise auf die Problemlösung wird durch eine Cluster-Route - Kombination dargestellt. Dabei werden bei der Methode „Cluster first - Route second“ vorerst Cluster mit den Städten gebildet, die zu einer Gruppe zusammengefasst werden können. Beim CVRP können diese Gruppen Städte sein, welche sich nah beieinander befinden oder in einem bestimmten Sektor sind. Im Anschluss werden dann Routen innerhalb der einzelnen Gruppen gebildet. Beim Sweep-Verfahren kann das gleiche Verfahren, welches für die Clusterfindung benutzt wird - der Winkel zum Depot - auch für die Routenkonstruktion benutzt werden.

- **Sweep-Verfahren**

Das Sweep-Verfahren wurde 1972 von Wren und Holliday veröffentlicht [Wre72] und ist der wohl bekannteste Vertreter der Methoden, die Cluster benutzen [Ohr08, S. 65].
Zunächst werden die Winkel des direkten Weges zu den einzelnen Städten berechnet und sortiert. Der Reihenfolge nach werden nun die Städte solange einer Tour zugeordnet bis eine Nebenbedingung verletzt wird. Dann wird eine neue Tour angelegt und die Liste weiter abgearbeitet.
Durch verschiedene Startwinkel können die Resultate unterschiedlich ausfallen. Als Beispiel wird hier das att48-Problem mit $Q = 15$ betrachtet, wobei der Algorithmus vom Depot aus einmal mit einem Winkel von 270° (Abbildung 5) und ein anderes Mal mit 0° (Abbildung 6) startet.

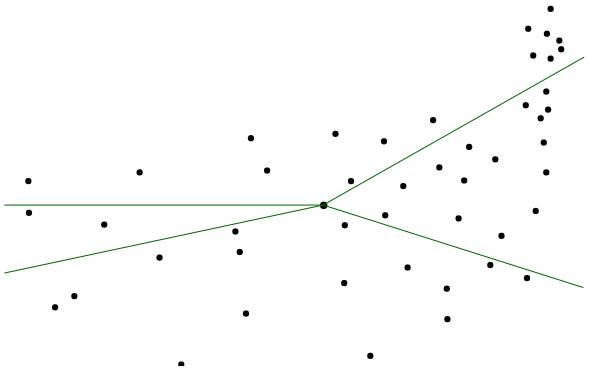


Abbildung 5: att48 - Einteilung 1

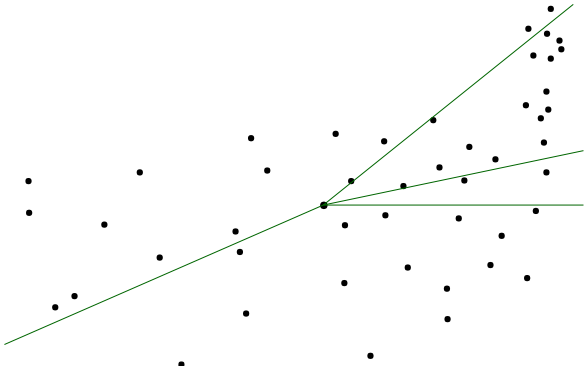


Abbildung 6: att48 - Einteilung 2

Wie in den beiden nachfolgenden Abbildungen können die entstandenen Sektoren als TSP-Problem aufgefasst werden und mittels einem nicht ganz so rechenaufwändigen Algorithmus gelöst werden. In diesem Beispiel könnte schon das Branch-and-Cut - Verfahren eingesetzt werden, was bei 15 Städten auf aktuellen Systemen innerhalb einer Sekunde die beste Lösung liefern kann. Mit steigender Kapazität kann zwar auch die Häufigkeit für diese Anwendung des Branch-and-Cut - Verfahrens sinken, aber die Rechenzeit nimmt dafür um ein vielfaches zu, wodurch überlegt werden müsste, ob diese Methode nicht auf das ganze Problem angewendet werden sollte. Die einfache und viel schnellere Methode wäre - wie auch im originalem Algorithmus vorgesehen - die Routen dann auch nach ihrer Sortierung bezüglich dem Winkel zum Depot abzufahren. Eine schnellere nicht zufällige Variante als die Pfadkonstruktion nach der Sortierung der Winkel gegenüber der Startposition ist auch in einschlägiger Literatur nicht bekannt. Die Pfadlänge in den beiden folgenden Abbildungen beträgt 52380 bzw. 52343 Längeneinheiten, was das bekannte Optimum (40002 LE) um 30,9% übertrifft.

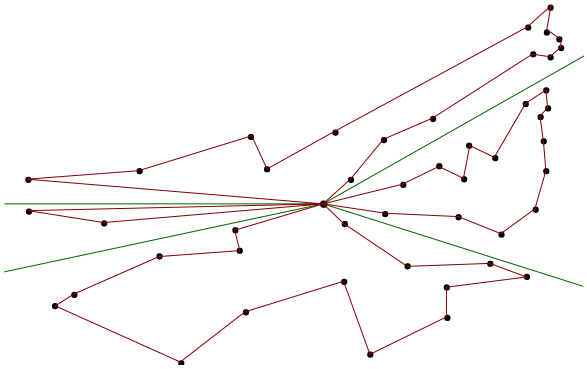


Abbildung 7: att48 Sweep-Lösung 1 (52380 LE)

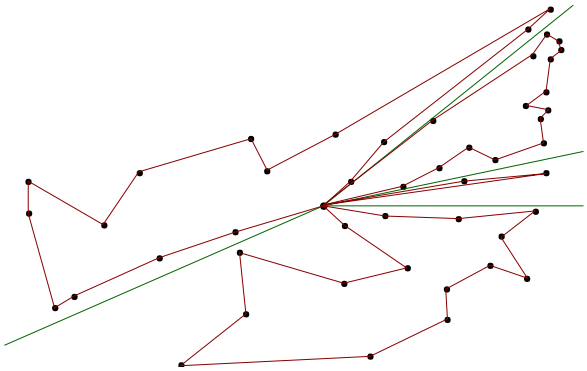


Abbildung 8: att48 Sweep-Lösung 2 (52343 LE)

3.5 Zusammenfassung

Da die zeitliche Vorgabe des Starts des Ladekopfes eingehalten werden muss, kommt nur ein heuristisches Verfahren in Frage. Alle bekannten exakten Methoden liefern in vorgegebener Zeit keine Ergebnisse, die mit heuristischen Verfahren mithalten können. Da laut Wendts eigenen Analysen [Wen95] sein Cooperative Simulated Annealing bei gleicher Rechenkapazität durchschnittlich die besten Lösungen liefert, fällt die Wahl auf diesen Algorithmus.
Anzumerken ist jedoch, dass ein exaktes Verfahren bei nur wenigen Städten durchaus Sinn machen kann. Gerade bei einem kleinen Medikamentenlager, bei welchem oftmals auch nur eine Hand voll Medikamente gleichzeitig angefordert werden, ist die Wahl von bspw. Branch-and-Cut der Heuristik vorzuziehen. Denn im schlimmsten Fall könnten bei 9 verschiedenen Medikamenten 453600 verschiede Routen existieren (mit $Q = 8$), welche auf aktuellen Systemen kaum merkbar auf ihre Güte durchprobiert werden könnten.

4 Aufbau des Grundalgorithmus

COSA beinhaltet zwei Methoden der Künstlichen Intelligenz. Der Grundgedanke ist Teile von genetischen Algorithmen mit dem heuristischen Optimierungsverfahren Simulated Annealing zu vermischen.

- Selektion
  Die Fitness gibt Auskunft über die Güte eines Individuums. Je nach Variante des Selektionsverfahrens werden mit dieser Information die Individuen ausgewählt, die an einem Reproduktionsprozess teilnehmen dürfen. Eine Begünstigung von den besten Individuen kann die Suche eines lokalen Optimums beschleunigen, bringt aber den Nachteil mit sich, dass Alternativen kaum berücksichtigt werden.

- Crossover
  Durch den Crossover - Operator werden Teile von verschieden ausgewählten Individuen ausgetauscht und an Nachfahren weiter vererbt. Dadurch können die erzeugten Nachfahren hauptsächlich eine Verbesserung im lokalen Suchraum erzielen. Ein Sprung zu einer entfernten Stelle im Suchraum ist aber auch möglich.

- Mutation
  Um einer Konvergenz der Individuen auf den lokalen Suchraum entgegen zu wirken, modifiziert der Mutations-Operator einzelne Teile des Individuums. Um die eventuell erzielten Fortschritte nicht zu vernichten ist es notwendig genau abzuschätzen zu welchem Anteil dieser Operator eingesetzt werden soll.

4.1 Der Genetische Algorithmus

Ein genetischer Algorithmus (GA) ist ein evolutionärer Algorithmus (EA), welcher mögliche Problemlösungen in Bitvektoren fester Länge kodiert und diese dann mit den für EA entworfenen Operatoren Selektion, Mutation und Rekombination (Crossover) behandelt [Hol75]. Ausgangspunkt sind zufällig erzeugte Individuen, welche die erste Generation bilden. Für jedes Individuum wird ein Fitnesswert berechnet, welcher die Qualität des Individuums widerspiegelt. Mittels geeigneten Selektionsverfahren werden Individuen aus der gesamten Generation ausgesucht, welche dann Nachfahren erzeugen sollen. Die Erschaffung dieser Nachkommen geschieht mittels Crossover und Mutation, welche jeweils zu vorgegebenen Wahrscheinlichkeiten eingesetzt werden. Dieser Prozess wird solange wiederholt, bis ein Abbruchkriterium erfüllt wurde.

Möchte man genetische Algorithmen für eine Routenplanung verwenden, so gilt es zuerst den Aufbau eines Individuums festzulegen. Hierbei muss beachtet werden, dass alle möglichen Lösungen repräsentiert werden können. Eine Einschränkung des Lösungsraumes ist ebenfalls möglich, findet bei den verschiedensten Problemlösungen aber selten Verwendung. Wahrscheinlich ist der Rechenaufwand den Lösungsraum für genetische Algorithmen vorher einzuschränken zu groß. Diese These wird auch dadurch unterstützt, dass genetische Algorithmen im Allgemeinen sehr schnell Positionen im schlechten Lösungsraum verlassen (vgl. [Jet07, S. 199]).
Die Städte durchzunummerieren ist wahrscheinlich die effektivste Variante. Man könnte auch die absolute Position einer Stadt nehmen, dann würden sich allerdings die genetischen Operatoren als schwierig gestalten. Die Rekombination aus 2 absoluten Positionen ist zu aufwendig. Für selbstorganisierende Karten bzw. neuronale Netze ist diese Kodierung der Städte eher eine mögliche Option.

Durch die Durchnummerierung wird eine Basis geschaffen, welche eine klare, einfach zu benutzende Struktur beinhaltet. Mit dieser Basis wird es dann möglich, schnell Knoten oder Kanten innerhalb eines Individuums auszutauschen, Teilrouten zu ändern oder herauszunehmen sowie nach einer kurzen Analyse das Individuum zu validieren. Für das Depot, also die Ausgangsstadt kann bspw. eine „0“ verwendet werden. Ein valides Beispiel für ein TSP mit 7 Städten wäre dann folgende Rundreise:

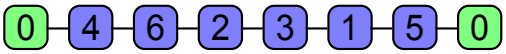


Abbildung 9: Ausgangssituation

Da jede anzufahrende Stadt genau einmal vorhanden ist, die Ausgangsstadt am Anfang und Ende der Route vorhanden ist und das Individuum aus acht Genen (zwei Depot- und sechs Zielstädte) besteht, ist die Route gültig.
Eine Stadt kann noch zusätzliche zu den Koordinaten eine Zeitrestriktion, einen kapazitiven Verbrauch oder weiterer Bedingungen unterliegen. Diese würden dann aber erst bei der Auswahl oder der Validierung berücksichtigt werden müssen.
Eine Pfadrepräsentation für ein VRP, was zwischendurch einmal das Depot anfahren muss, wäre:

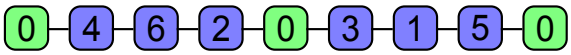
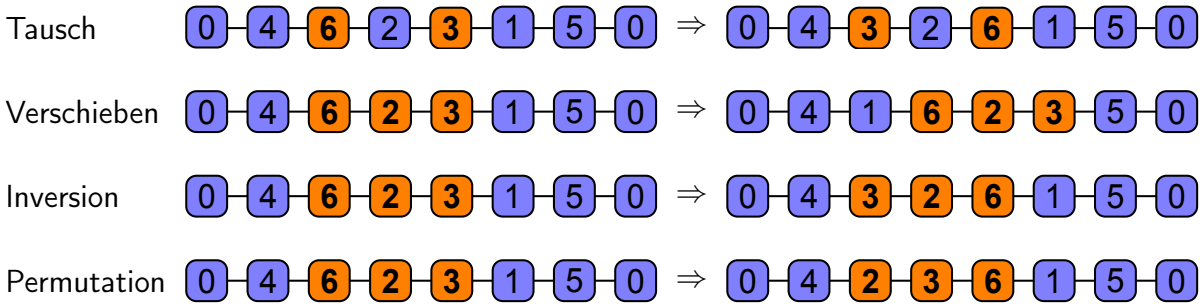


Abbildung 10: VRP mit zwei Routen

Durch diesen Aufbau müssen die genetischen Operatoren auf der Parameterebene angewendet werden. Standardmutationen wie das Kippen eines einzigen Bits fallen weg, da dadurch üblicherweise invalide Individuen erzeugt werden [BHS07, S. 81]. Die Gültigkeit der Individuen kann auf verschiedene Weise beibehalten werden. In der allgemeinen Literatur werden die Mutationsoperatoren Tausch, Verschieben, Inversion und Permutation benutzt.



4.2 Ein geeigneter Crossover - Operator

Der Crossover - Operator kombiniert vorhandene Lösungen, wodurch üblicherweise weitere Lösungen im lokalen Suchraum entstehen. Dazu werden grundsätzlich mindestens zwei Individuen (Elternteile) benötigt von denen im Allgemeinen jeweils ein Teil an ein neues Individuum weiter vererbt wird. Die Wahl eines oder mehrerer Verfahren kann somit über die Güte des Individuums in Bezug auf das globale Optimum entscheiden. Auch hier werden wie bei den Mutationsoperatoren die Funktionen auf der Parameterebene angewendet, da somit gegeben ist, dass jede Stadt mit Ausnahme des Depots nur einmal im jeweiligen Individuum vorkommt.

4.2.1 PMX Partially Matched Crossover

Ein 1988 von Goldberg und Lingle entwickeltes Verfahren heißt Partially Matched Crossover und ist hauptsächlich dafür bekannt, dass Positionen weitgehend erhalten bleiben. Hierzu wird aus den Elternteilen ein beliebiges Element zufälliger Länge vertauscht auf die neuen „Kind“ - Individuen an gleicher Stelle kopiert [BHS07, S. 80].
In einem Beispiel (Abbildung 11) werden ab der dritten Stadt die jeweiligen Teilrouten von vier Städten in die Kindindividuen übertragen (Abbildung 12).

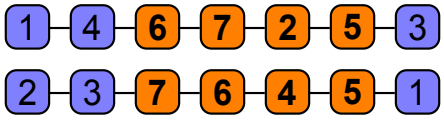


Abbildung 11: PMX: Beide Elternteile mit Markierung der zu tauschenden Strecke

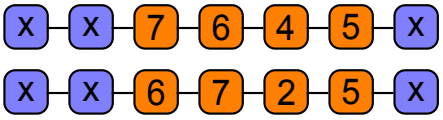


Abbildung 12: PMX: Beide Kinder mit vertauschten Teilstrecken der Eltern

Im Anschluss werden die Gene aus den Originalpositionen übernommen, welche keinen Konflikt innerhalb des Individuums entstehen lassen (Abbildung 13). Bei den Städten, die bereits vorhanden sind, werden der Reihe nach die Gene des jeweilig anderen Elternteils genommen (Abbildung 14).

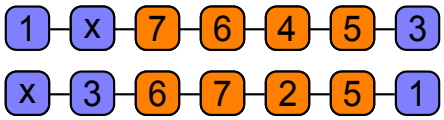


Abbildung 13: PMX: Übernahme der möglichen Originalgene der Eltern

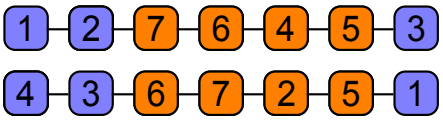


Abbildung 14: PMX: Beide Kinder nach der Rekombination

Wenn auch die Positionen weitestgehend erhalten bleiben, werden jedoch viele Kanten zerstört. So ist die Anwendung des Operators ohne Modifikation nicht sehr zielgerichtet, was auch durch empirische Tests bewiesen werden kann [Wen95, S. 97]. Hinzu kommt beim CVRP die Möglichkeit, dass ein Depot nicht unbedingt in beiden gewählten Segmenten existieren muss (Abbildung 15). So könnten Teilrouten ohne Stadt entstehen, welche dann üblicherweise keine valide Lösung mehr sind (Abbildung 16).

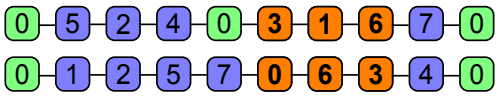


Abbildung 15: PMX-CVRP: Die Auswahl der Teilrouten bei den Eltern

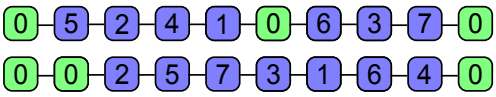


Abbildung 16: PMX-CVRP: Das zweite Individuum hat eine Teilroute ohne Zwischenstadt

4.2.2 OX Order Crossover

Um diesen Nachteil der vielen verlorenen Kanten entgegen zu wirken, führte Davis [Dav91] den Order-Crossover-Operator ein. Wie beim PMX wird hier zunächst ein beliebiges Segment (Abbildung 17) der Individuen auf die Nachkommen übertragen. Im Anschluss werden die restlichen Gene des jeweils anderen Individuums in bestehender Reihenfolge (Abbildung 18) an den Nachkommen vererbt (Abbildung 19).

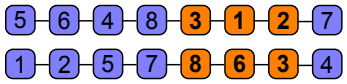


Abbildung 17: OX: Die Auswahl der Teilrouten bei den Eltern

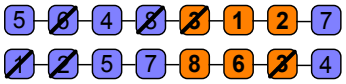


Abbildung 18: OX: Die Reihenfolge ohne die zu tauschenden Elemente

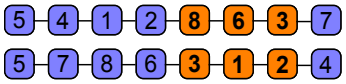


Abbildung 19: OX: Die beiden entstandenen Nachfahren

Wie in Abbildung 18 zu sehen ist, wird im oberen Individuum aus der Route die Stadt 6 entfernt, da diese schon Bestandteil des gewählten Teilsegments des anderen Individuums ist. Die Summe der beiden ursprünglichen Kanten $\{5, 6\}$ und $\{6, 4\}$ kann laut Dreiecksunglichung nicht länger sein als die resultierende Kante $\{5, 4\}$ [Wen95, S. 82]. Dadurch kann eine Verschlechterung nur dann bestehen, wenn die Kanten zu der neu eingefügten Teilroute (Kanten $\{2, 8\}$ und $\{3, 7\}$) addiert zu dem Weg der Teilroute selbst ($\{8, 6, 3\}$) größer ist als der eingesparte Weg. Hierbei ist zu beachten, dass die jeweils übernommene Teilroute durch die richtige Selektion mit der Zeit besser werden kann. Werden zum Anfang der Optimierung noch qualitativ schlechte Teilrouten entnommen, so steigt im Gesamtverlauf die Wahrscheinlichkeit, dass die entnommene Teilroute sogar die kürzeste Route zwischen den Städten repräsentiert.
Durch diesen Ansatz, die Kanten zu erhalten, konnte für das TSP die Effektivität gegenüber PMX und auch anderen Verfahren durchschnittlich um mindestens 11% überboten werden [Wen95, S. 83]. Wird die Strecke jedoch in mehrere Routen aufgeteilt wie es beim VRP sein kann, ist zu überprüfen ob eine Nebenbedingung verletzt wurde. Es gibt offensichtlich zwei Varianten. Behandelt man die Depots innerhalb der Gesamtroute wie jede andere Stadt, können wie beim PMX (ähnlich Abbildung 16) schnell kapazitive Probleme entstehen. Eine andere Variante wäre die Depots aus einem Individuum zu übernehmen, was Umwege über diese Depots mit sich bringt.

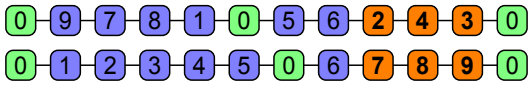


Abbildung 20: OX-CVRP: Ausgangssituation

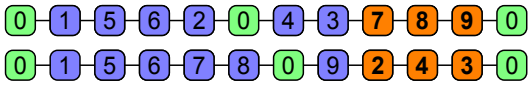


Abbildung 21: OX-CVRP: Resultat bei übernommenen Depotpositionen

Im oberen Nachkommen aus Abbildung 21 entstehen nicht nur die Kanten zu dem Segment $\{7, 8, 9\}$ sondern auch ein Umweg von den Städten 2 und 4 über das übernommene Depot. Dabei ist zu beachten, dass hier das ausgewählte Segment nur zwei Teilrouten beeinflusst und mit längeren Segmenten weiteren Teilrouten und somit auch mehr Depots hinzukommen können, über die ein Umweg entstehen könnte. Weiterhin muss beachtet werden, dass so bereits gewonnene Vorteile zur Auslastung einzelner Teilrouten schnell zerstört werden. Man zwingt förmlich der Teilroute eine neue Kapazität auf, welche bspw. bei unterschiedlich gewichteten Städten (im Sinne des kapazitiven Verbrauchs einer Stadt) genau das vernichtet, was ein Crossover-Operator erhalten soll: Die bis dahin gewonnene Qualität im Lösungsraum.
Da im vorliegenden Problem allerdings keine Stadt eine Nebenbedingung mit sich bringt, ist anzunehmen, dass der OX - Operator wohl einer der besseren Operatoren ist, solange die Depotpositionen aus einem Elternteil übernommen werden. Denkbar wäre auch das nachträgliche Einfügen der Depots in die Route unter der Prämisse, dass die eingefügten Depots dann zusammen den geringsten Umweg ergeben. Die Suche nach genau diesen Positionen stellt sich aber als eigenes Problem dar und gleicht dem Cheapest-Insert-Verfahren, wobei die Depots erschwerender Weise gleichzeitig eingefügt werden müssen um Kapazitätsrestriktionen valide zu halten.

4.2.3 CX Cycle Crossover

Durch Cycle-Crossover können ohne Kompromisse machen zu müssen die Positionen der Städte erhalten bleiben. Zuerst dient das erste Individuum als Gen-Spender und das Zweite liefert die Positionen an welche das Gen im Nachfolgeindividuum platziert werden soll. Es wird also eine beliebige Stadt aus dem ersten Individuum genommen und direkt auf das Kind übertragen (Abbildung 22). Die Stadt aus dem zweiten Elternteil, die nun nicht mehr an diese Position eingefügt werden kann, wird im ersten Elternteil gesucht. Von hier an wird solange der Vorgang des Städtesuchens und Einfügen wiederholt bis im zweiten Individuum die Startstadt gefunden wird. Die restlichen, fehlenden Gene des Nachfolgeindividuums werden direkt aus dem zweiten Elternteil kopiert.

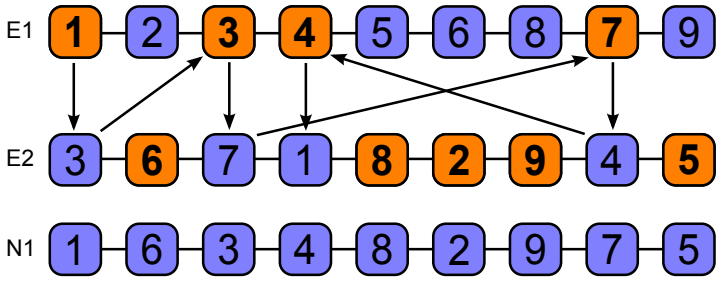


Abbildung 22: Cyclic Crossover

Das Resultat, dass nun ein Nachfolger Gen für Gen aus Vater oder Mutter besteht, kann schon für ein TSP kaum überzeugen, weil zu oft viele Kanten zerstört werden [Wen95, S. 83]. Erst recht für das CVRP verhindert die mögliche Abweichung der Depot-Positionen in beiden Elternteilen die Standardimplementierung ähnlich wie schon bei den anderen Verfahren.

4.2.4 ERX Edge Recombination Crossover

Da es offensichtlich effektiver ist gute Kanten zu erhalten als die Positionen im Genstrang beizubehalten, entwickelte Whitley [Whi89] das Edge Recombination Crossover. Durch dieses Verfahren, welches recht aufwendig ist, werden fast ausschließlich Kanten der Eltern vererbt. Aufwendig ist diese Methode, da zunächst eine Liste mit allen Städten und deren möglichen durch Kanten in den Eltern resultierenden Nachbarstädten erstellt werden muss. Im Anschluss kann eine Rundreise dadurch erstellt werden, dass vorrangig die Städte besucht werden, welche wenig unbesuchte Nachbarstädte aufweisen. Für das CVRP gibt es einige erfolgreiche Implementierungen dieses Operators wie auch in [AGP08, S. 386], wobei der Autor allerdings selbst einräumt, dass der Gesamtalgorithmus bei mehr als 50 Städten nur noch zu ca. 8% das bekannte globale Optimum erreicht.

4.2.5 Zusammenfassung

Schneiden die bekannten Crossover - Operatoren für die Lösung des TSPs noch einigermaßen gut ab, sind sie für das CVRP eher ungeeignet. Es ist zwar überwiegend möglich, diese Operatoren für das CVRP zu modifizieren, aber dennoch hindern die Depots im Chromosom eines Individuums eine gute Kantenerhaltung. Einzig und allein ERX hat laut [Cot07, S. 143] in einer bestimmten Modifikation ein großartiges Potential.

4.3 Vorraussetzung

4.3.1 Simulated Annealing

Ein Optimierungsverfahren für kombinatorische Problemstellungen ist Simulated Annealing, welches formell erstmals 1983 von Kirkpatrick beschrieben wurde [KGV83]. Als Vorbild dient der Abkühlungsprozess eines geschmolzenen Körpers zu einem Festkörper. Können sich die Moleküle in der Schmelze noch frei bewegen, nimmt mit sinkender Temperatur diese Bewegungsfreiheit ab [BHS07, S. 44]. Analog für das Optimierungsverfahren entspricht die Temperatur der Wahrscheinlichkeit, mit der sich ein Zwischenergebnis verschlechtern darf.
Es wird eine Anfangstemperatur $t > 0$, der Abkühlungsfaktor $k > 1$ und ein Abbruchwert $\varepsilon > 0$ festgelegt.

Solange $t > \varepsilon$:
    Wähle aus der Nachbarschaft von x einen Punkt y
    $\sigma = f(i_x) - f(i_y)$
    Ist $\sigma > 0$, so setze $x := y$
    Andernfalls: Wähle eine Zufallszahl $r \in (0, 1)$. Ist $r < e^{\sigma/t}$, so setze $x := y$
    $t := \frac{t}{k}$
Gib x als Lösung zurück.

Abbildung 23: Simmulated Annealing

4.3.2 Nebenbedingungen

Es gibt viele mögliche Nebenbedingungen für das allgemeine Travelling Salesman Problem. Beispielsweise können Zeitfenster vorgegeben sein, wann bestimmte Positionen angefahren werden dürfen (VRPWTW). Ein anderes Beispiel wäre die Benutzung mehrerer LKWs (MTSP). Bei der im Kapitel 3.1 angesprochenen Standard - Problemstellung beinhaltet jede Stadt genau eine Nebenbedingung, welche als „Kapazitätsverbrauch“ angesehen werden kann. Im CVRP51+1 stehen bspw. 160 Einheiten zur Verfügung. Die einzelnen Städte verbrauchen verschiedene Kapazitäten, so dass im günstigsten Fall eine Route 11 Städte umfassen kann, im schlechtesten Fall aber nur 8. Ein Spezialfall wäre, wenn alle Städte die gleiche Kapazität verbrauchen würde. Dieser Fall tritt ein, wenn das Gut, welches transportiert werden soll, immer die gleiche Größe hat oder die Transporteinheit wie bspw. ein LKW je Gut eine uniformierte Platzgröße vorsieht. Im Fall des Medikamentenlagers wird je Gut ein Platz reserviert, da sich die verschiedenen Medikamentenverpackungen in ihrer Größe üblicherweise ähneln. Die Nebenbedingung ist also das der Ladeschlitten nur eine vorher festgelegte Anzahl an Packungen aufnehmen kann, egal welches Gewicht oder welche Größe diese Packung hat. Eine baubare Variante wäre z.B. die Aufnahmefähigkeit von 6 Schachteln ($Q = 6$).

4.4 Cooperative Simulating Annealing

4.4.1 Motivation

Die Vorzüge von genetischen Algorithmen, schnell eine gute Lösung zu liefern sind eine optimale Voraussetzung für Probleme verschiedenen Größenordnung. Die Natur dabei nachzuahmen hat sich bei vielen Problemen als recht nützlich erwiesen (vgl. Kapitel 3.4.2). Da selbst 10

Jahre nach der Entwicklung von Cooperative Simulated Annealing (COSA) dieses Verfahren immernoch als effektive Lösung angesehen wird [Sto06, S. 89] und ausserdem eine Anpassung an das spezielle CVRP als annehmbare Lösung vorstellbar ist, wird COSA als Grundlage für die Implementierung genommen.

4.4.2 Grundidee

Beim genetischen Algorithmus besteht der Grundgedanke darin, dass zwei oder mehr Elternteile ein oder mehrere Kinder erzeugen. Dazu werden die in diesem Kapitel erklärten Operatoren benutzt. Es existiert das Problem, dass ein entstandenes Kind in der Gesamtbetrachtung relativ weit entfernt von beiden Eltern sein kann. Werden zufällig zwei Elternteil bestimmt, die beide an einer bestimmten Stelle schon eine recht effektive Teilroute beinhalten, muss üblicherweise ein Crossover - Operator schon zufällig genau die beiden Teilrouten für das Kind wählen, damit die Fitness von dem Kind auch die Eltern übertrifft. Gerade in einem fortgeschrittenen Stadium, in dem sich schon gute Teilrouten etabliert haben, schafft solch ein Operator meistens nur eine Verbesserung, wenn er in die Teilroute nicht eingreift [Wen95, S. 152]:

> „Sind sich Vater und Mutter hierbei sehr fremd, so besteht in der Regel kaum eine Chance, dass durch ein Gemich ihrer Chromosomen ein Punkt im Lösungsraum generiert wird, der von seinen Fitnesswert auch nur halbwegs mit seinen Eltern konkurieren kann.“

Dieses Problem scheint immer zu bestehen, wenn einer der gewöhnlichen Crossover - Operatoren des genetischen Algorithmus benutzt wird. Zwei Ansätze dieses Problem zu lösen wäre die Einführung eines anderen genetischen Operators oder in den grundlegenden Ansatz des genetischen Ablaufs einzugreifen.
Eine Idee, welche bei COSA umgesetzt wurde, ist eines der Elternteile ein wenig zum anderen heranzurücken. Dazu muss ein Elternteil als Informationsspender dienen (Abbildung 24). Nun wird aus dem Spender eine beliebige Stadt und eine Nachbarstadt gemerkt. Im nächsten Schritt wird überprüft, ob diese benachbarten Städte auch im anderen Elternteil benachbart sind. Befinden sich beide Städte noch nicht nebeneinander (Abbildung 25), wird eine Inversion genau so durchgeführt, dass die beiden Städte dann benachbart sind (Abbildung 26).

Abbildung 24: COSA (TSP) Informationsspender

Abbildung 25: COSA (TSP) zu modifizierendes Elternteil

Abbildung 26: COSA (TSP) Elternteil nach der Inversion

Falls sich die Städte schon vorher nebeneinander befanden, wird eine zufällige Transition ausgeführt. Diese zufällige Transition kann einer der genannten Operatoren für genetische Algorithmen sein, wie z.b. Tausch oder Permutation. Eine zufällige Inversion wäre ebenfalls denkbar. Dadurch kann eine Innovation in der Population gewahrt werden [BHS07, S. 79].

Der Gedanke der hinter diesem Algorithmus steht ist der, dass sich zwei Nachbarstädte eines Spenderindividuums besser etabliert haben als eine zufällig gewählte Stadt. Stabile Teilrouten, die bereits in einem Individuum existieren, bleiben so im Vergleich zu anderen Operatoren oftmals bestehen. Der Hauptgrund dafür ist, dass wie beim Order Crossover und auch beim k-opt-Verfahren (vgl. Kapitel 3.4.2) darauf gezielt wird, so viele Kanten wie möglich beizubehalten. Im Beispiel werden dem zu modifizierenden Elternteil zwei Kanten entnommen $(5, 2$

und $6,1$) und untereinander vertauscht wieder hinzugefügt ($5,1$ und $6,2$).

Um den Grad der Verbesserung des modifizierten Individuums zu berechnen, genügt es, wie beim k-opt-Verfahren die Längen an den eben genannten Kanten zusammen zu addieren und zu vergleichen. Ist die Summe der Kanten im neuen Individuum nun geringer, wurde eine bessere Position im lokalen Suchraum gefunden. Das Individuum kann übernommen werden. Wurde eine Verschlechterung der Fitness errechnet, so muss mit Hilfe von Simulated Annealing ermittelt werden, ob eine Übernahme der neuen Struktur für dieses Individuum zufällig angenommen werden kann.

Für das CVRP kann diese Idee ebenfalls verwendet werden. Jedoch mit dem Unterschied, dass hier vorher getestet werden muss, ob die Städte im zu modifizierenden Individuum in der gleichen Teilroute sind. Es muss so lange gesucht werden, bis diese Bedingung erfüllt ist. Bleibt keine Stadt mit dieser Eigenschaft übrig, so sollte eine zufällige Transition ausgeführt werden. Hier gibt es sehr ungünstige Fälle. Je mehr Teilrouten vorhanden sind, desto geringer ist auch die Wahrscheinlichkeit, dass zwei Städte in der gleichen Teilroute sind. Durch eine geringe Kapazität Q oder sehr vielen Knoten K in einem Individuum wird so die Suche nach Städten in gleichen Routen sehr rechenintensiv. Es bleibt zu untersuchen, ob es Techniken gibt, die routenübergreifend ein ähnliche kantenerhaltende Modifizierung zulassen.

4.4.3 Pseudo-Algorithmus

Der Entwickler Oliver Wendt hat einen Pseudoalgorithmus erstellt, der den Ablauf für eine Routenoptimierung beschreibt [Wen95, S. 156]:

procedure COSA
bestimme Populationsgröße $|POP|$
bestimme Anfangs-Population $POP_0 = (I_1 \in S, I_2 \in S, ..., I_{|POP|} \in S) \in S^{|POP|}$
bestimme Nachbarschaftsrelation $N \subseteq S \times S$
bestimme Transitionenzahl $K = n * |POP|$ mit $n \in \mathbb{N}$
bestimme Anfangstemperatur $T_1 \in \mathbb{R}_+$ und Abkühlungsfaktor $\alpha \in]0;1[\subset \mathbb{R}_+$
for k:=1 **to** $\frac{K}{|POP|}$ **do**
  for i:=1 **to** $|POP|$ **do**
  bestimme zufälligen Informationsspender $I_j \in POP_{k-1}$
    generiere $I_{new} := cotrans(I_i, I_j)$
    if $random(0,1) \leq \begin{cases} 1, & \text{wenn } f(I_{new}) < f(I_i) \\ e^{-\frac{f(I_{new})-f(I_i)}{T_k}}, & \text{wenn } f(I_{new}) \geq f(I_i) \end{cases}$
      then $I_i := I_{new}$
  next i
  $POP_k := (I_1, I_2, ..., I_{|POP|})$ $T_{k+1} := update(POP_{k-1}, POP_k, T_k, \alpha)$
next k
end.

Abbildung 27: COSA-Minimierungsalgorithmus

function update:$(POP_k \in S^{|POP|}, POP_{k-1} \in S^{|POP|}, T \in \mathbb{R}_+, \alpha \in]0;1[) \rightarrow T \in \mathbb{R}_+)$
$\Delta E := E(POP_k) - E(POP_{k-1}$
$T := \begin{cases} T, & \text{wenn } \Delta E < 0 \\ \alpha T, & \text{wenn } E \geq 0 \end{cases}$
return T.

Abbildung 28: Funktion 'update' zur adaptiven Absenkung der Temperatur

function cotrans:$(I_i \in S, I_j \in S) \rightarrow I_{new} \in S$
bestimme alle Nachbarn von I_i, die bzgl. N näher an I_j liegen als I_i:
CLOSER: $= \{s_k \in N(I_i) \| d(s_k, I_j) < d(I_i, I_j)\}$
if $\|$CLOSER$\| \geq 1$

then wähle zufälliges$I_{new} \in$ CLOSER
else wähle zufälliges$I_{new} \in N(I_i)$

return I_{new}.

Abbildung 29: Funktion 'cotrans' zur Generierung kooperatier Transitionsversuche

Diese Darstellung (Abbildung 27-29) des gewählten Algorithmus ist allgemein gültig und anwendbar auf das TSP sowie das CVRP. In den folgenden Unterpunkten, wird der Grundgedanke des beschriebenen Ablaufs erklärt.

4.4.4 Die Anfangspopulation

Zuerst wird wie bei evolutionären Algorithmen eine Anfangspopulation gewählt. Die Anzahl der Individuen kann entweder fest oder adaptiv an die Problemgröße angepasst werden. Durch Tests oder mittels Methoden der Künstlichen Intelligenz angepasst werden.
Diese Population muss ausschließlich valide Individuen enthalten, da nicht nachträglich ihre Gültigkeit überprüft wird. Hier greift der im Kapitel 4.1 angesprochene Vorteil, dass dadurch viel Berechnungszeit eingespart werden kann. Das ist unter anderem wichtig, weil erst mit Erhalten des Auslagerungsauftrags die anzufahrenden Positionen bekannt sind und die Zeit ab diesem Punkt gering gehalten werden muss. Eine Vorrausberechnung, was als nächstes ausgelagert werden könnte, ist nicht möglich und verbräuchte anhand der möglichen Kombinationen zu viele Ressourcen.
Zunächst wird vermutet, dass es ebenfalls möglich sein müsste, den Ladekopf erst einmal loszuschicken und während das erste Produkt auf den Schlitten geladen wird, die Berechnung der Anfahrtsreihenfolge stattfinden könnte. Es ist jedoch kein Verfahren bekannt nach dem ein optimiertes Anfangsprodukt ausgewählt werden könnte. Es ist offensichtlich, dass nicht immer das weitentfernteste Produkt genommen werden kann, da unterwegs bereits ein Produkt mitgenommen werden könnte, wodurch definitiv zu mindest in dieser lokalen Betrachtung Weg

eingespart wird. Würde die Wahl der ersten Anfahrtsstelle auf die Nächstgelegene fallen, kann bewiesen werden, dass so die beste Route ausgeschlossen sein könnte:
Der Beweis, dass bei der Routenoptimierung die Wahl der dichteste Stadt als Anfangsstadt die beste Route ausschliessen kann, ist erbracht, wenn mindestens eine Route gefunden wurde, bei der dieser Fall eintritt.

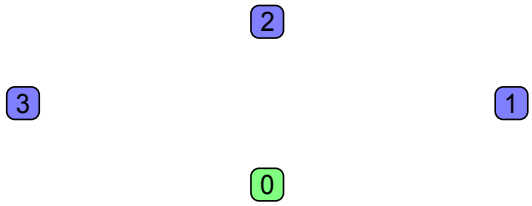


Abbildung 30: Ausgangssituation des Minimalbeispiels

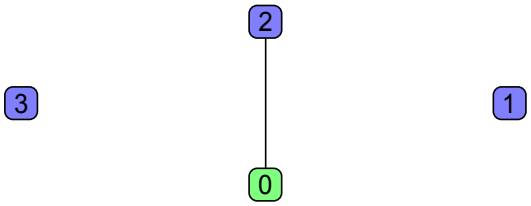


Abbildung 31: erste Strecke zur dichtgelegensten Stadt

Ein Minimalbeispiel ist die die Suche nach dem kürzesten Weg mit der Ausgangssituation in Abbildung 30. Die zum Depot 0 nächstgelegene Stadt 2 wird angefahren. Während dieser Weg gefahren wird (Abbildung 31), würde ein Algorithmus die restliche Route (Abbildung 32) errechnen. Die entstandene Route kann keine optimale Lösung sein, da die Strecken $\overline{02}$ und $\overline{31}$ zusammen länger sind als $\overline{12}$ und $\overline{30}$. (Satz des Pythagoras)

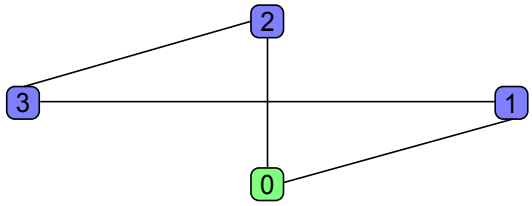


Abbildung 32: Beste Route nach Schnellstart

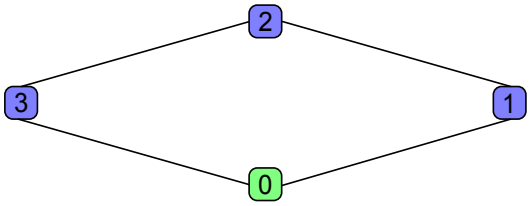


Abbildung 33: Der eigentlich kürzeste Weg

Folglich kann also die Variante die naheliegenste Stadt als Start für die Route zu wählen eine optimale Route verhindern. Eine weitere Erkenntnis kann aus diesem Beispiel gewonnen werden: Eine Überkreuzung der eigenen Route ist ungünstig, so lange es nicht die Problemstellung vorgibt und keine weiteren Nebenbedingungen existieren. Eine Problemstellung, die eine Überkreuzung mit sich bringen könnte, wäre zum Beispiel beim Periodisches VRP (PVRP) gegeben, bei dem der Bedarf der Kunden nachwächst und eine optimalere Route ständig nachberechnet werden sollte. Aber auch durch Nebenbedingungen können schnell Überkreuzungen auftreten, wie es beim CVRP der Fall ist. Hier ist es meist sogar von Vorteil, wenn verschiedene Subrouten sich überschneiden. Auch hier ist ein Minimalbeispiel möglich:
Bei einer Ausgangssituation von 4 Städten und einem Depot sowie einer Kapazität $Q = 2$, existieren genau 3 Varianten. Kann nachgewiesen werden, dass eine Variante mit Überkreuzung eines Pfades besser ist als alle Varianten ohne Überkreuzung, so besteht die Möglichkeit auch in komplexeren Strukturen, dass eine Überschneidung vorteilhaft ist.

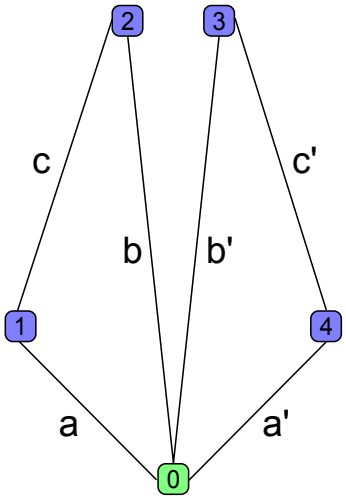


Abbildung 34: Variante 1

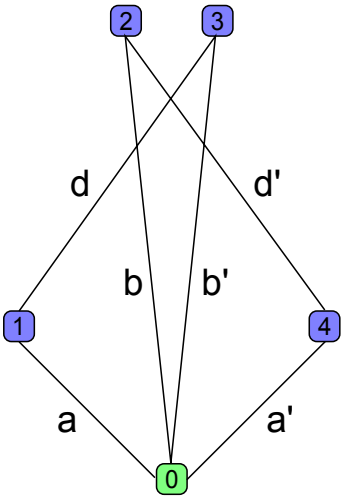


Abbildung 35: Variante 2

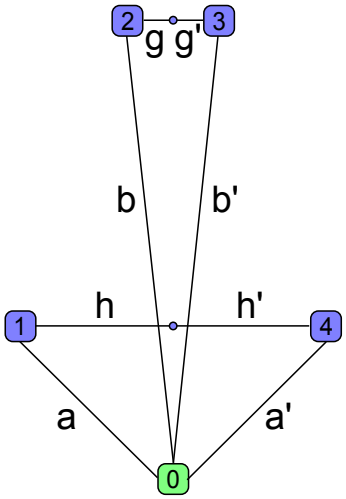


Abbildung 36: Variante 3

Der mathematische Aufwand kann um einiges verringert werden, wenn vorher überlegt wird, welche Varianten für den Vergleich wichtig sind und welche Strecken ausser Acht gelassen werden können. Es reicht aus, wenn eine der beiden Varianten mit Überschneidung (also Variante 2 oder 3) eine kürzere Gesamtroute hat als die Variante ohne Überschneidung. Weiterhin müssen nicht alle Strecken errechnet werden. Es ist offensichtlich, dass das Beispiel symetrisch aufgebaut ist und somit die Strecken a', b', c', d', g' und h' die Gesamtroute nur verdoppeln. Ferner sind in allen Varianten die Strecken a und b vorhanden. Auch diese würden nur einen mathematischen Offset bilden. Um also den gewünschten Nachweis zu Erbringen muss nur die Strecke c mit $g + h$ aus Variante 1 und 3 analysiert werden. Um den mathematischen Beweis gänzlich weglassen zu können, muss nur darauf verwiesen werden, dass c völlig unabhängig von den Strecken g und h ist. Die Strecke c kann also um ein vielfaches länger sein als $g + h$. Wodurch der Beweis erbracht wäre, dass eine Überschneidung bei Problemen mit Nebenbedingungen wie beim CVRP durchaus eine bessere Lösung bieten kann als Überschneidungsfreie.

4.4.5 Die Nachbarschaftsrelation

Eine Nachbarschaftsrelation stellen alle von einem Individuum erreichbaren Individuen dar. Diese können durch Anwendung von genetischen Operatoren erreicht werden. Da die Anzahl an Nachbarn mit der Anzahl der Population und der Problemgröße exponentiell ansteigt, ist es günstiger, einen zufälligen Nachbarn erst dann zu bestimmen, wenn er gebraucht wird. Bei exakten Verfahren, wie dem Branch-and-Cut, ist eine Nachbarschaftsrelation vorher aufzustellen sehr wichtig, da verschiedene Methoden ähnliche Knoten vergleichen müssen. Dabei reicht es sich die Nachbarschaftsstruktur zu überlegt. Eine Liste mit den möglichen Nachbarn muss nicht aufgestellt werden.

4.4.6 Abbruchbedingung

Mit der Transitionenzahl legt Wendt die Abbruchbedingung seinen Algorithmusses fest. Um einen Vergleich zu erhalten, wie schnell der Algorithmus zum Optimum konvergiert, ist diese Festlegung hinreichend. Jedoch muss für den Lagerautomaten eine Zeit als Abbruchbedingung genommen werden, da nicht sichergestellt werden kann, dass eine bestimmte Anzahl an Optimierungsversuchen in einer bestimmten Zeit getestet werden. Eine weitere Abbruchbedingung wäre durch die Überprüfung des Optimums gegeben. Falls dieses Optimum sich nach einer

festgelegten Spanne nicht mehr ändert, kann davon ausgegangen werden, dass eine weitere Verbesserung unwahrscheinlich ist. Der Nachteil bei dieser Methode ist, dass Rechenleistung eingesetzt wird, von der garantiert ist, dass sie keine Verbesserung liefern kann sondern nur für eine Validitätsprüfung genutzt wird. Eine Kombination von mehreren Abbruchbedingungen wäre für ein Medikamentenlager denkbar. Verlängern sollte die Abbruchzeit sich aber nur bei einer Inventur oder einem größeren Auftrag, bei welchem so viel Weg zustande käme, dass sich eine längere Rechenzeit lohnt.

4.4.7 Abkühlungsprozess

Der Abkühlungsprozess umfasst die Anfangstemperatur und die Geschwindigkeit, mit der die Temperatur abnimmt. Die Anfangstemperatur muss abhängig sein von dem Abstand zwischen den Städten. Eine hohe Temperatur lässt eine große Verschlechterung der Qualität der einzelnen Individuen zu, sodass sichergestellt ist, dass diese auch einen lokalen Suchraum verlassen können. Wird die Anfangstemperatur zu niedrig gewählt, ist eine Verschlechterung zwar möglich aber zu unwahrscheinlich, wodurch eine geringe Flexibilität vorprogrammiert ist. Das primäre Ziel von Simulated Annealing besteht darin, die Aufenthaltswahrscheinlichkeit eines Individuums im kompletten Suchraum so weit wie möglich von den restlichen Individuen zu setzen [Wen95, S. 126]. Gerade bei Individuen, die alle durch den gleichen Algorithmus (Cheapest - Insert) generiert wurden, kann so ein breiteres Spektrum an Lösungsmöglichkeiten ausgeschöpft werden. Der Abkühlungsfaktor muss so gewählt werden, dass die Temperatur auf einem Minimum ist, sobald die Abbruchbedingung erreicht wird. Der typische Faktor ist in verschiedener Literatur meist zwischen 0.9 und 0.999 angegeben. Für Traveling-Salesman-Probleme hat sich ein adaptiver Abkühlungsplan als nützlich erwiesen. So hat jedes Individuum einen eigenen Abkühlungsprozess. Eine Abkühlung findet hier nur statt, wenn auch eine Transition stattfand.

5 Konzeption und Implementierung von COSA

5.1 Objektorientierter Aufbau

Die Darstellung der Individuen, der Routen und des Problems ist im Rechner unterschiedlich möglich. Eine komplizierte Struktur im Quellcode kann zwar anfangs schwierig und aufwendig sein, ist aber bei Änderungen und Erweiterungen oft hilfreich. Eine Objektorientierte Umsetzung dient zum besseren Überblick bei komplexen Programmierungen. Die Pfadrepräsentation wie schon im Kapitel 4.1 beschrieben, wird hier noch weiter in ihre Teilrouten zerlegt. Außerdem hat jede Teilroute in der Datenstruktur das Depot am Anfang und Ende gespeichert. Da nur mit Referenzen und Zeiger die Optimierung vollzogen wird, benötigt der Algorithmus selbst kaum zusätzlich Speicher. Die Routen sind also ausschließlich Zeigerlisten, wobei ein Individuum eine Zeigerliste auf die einzelnen Teilrouten hat und die einzelnen Teilrouten wiederum auf die Städte und ihre Eigenschaften zeigen. Gesteuert wird alles von einem Manager, welcher ebenfalls eine Zeigerliste mit den Individuen beinhaltet. Eine grafische Darstellung des generellen Aufbaus liegt in der UML-Ansicht vor (Anhang: Abbildung 63).

Folgend werden die wichtigsten Funktionen erklärt, die ausschlaggebend für eine Optimierung sind. Die Basis bei den genetischen Algorithmen bietet eine Population mit einer Menge an Individuen. Gemeinsam je nach Problem ist nur gegeben, dass diese Individuen anhand ihrer Fitness selektiert werden. Das mindeste, was also ein Individuum enthalten sollte ist ein Fitnesswert. Alle Objekte, welche die Problemlösung des TSPs behandeln, haben für eine bessere Unterscheidung das Präfix „Taic“.

```
1 type TaicIndividual = class(TObject)
2   protected
3     rFitness: real;
4   public
5     function getFitness: real;
6 end;
```

Quellcode 1: Individuum - Basiseigenschaften

Ein Individuum besteht aus einem Genstrang, welcher entweder hintereinander betrachtet oder in einzelnen Teilrouten aufgeteilt werden kann. Beim CVRP kann ein Individuum aus mehreren Teilrouten bestehen. Aus diesem Grund wird eine Struktur bevorzugt, welche die einzelnen Teilrouten separiert. So wird ein normalerweise hintereinander aufgebauter Genstrang (Abbildung 37) als parallel existierende Routen dargestellt (Abbildung 38). Der Nachteil, der dadurch entsteht, das Depot am Anfang und Ende jeder Teilroute zu speichern, wird dadurch wieder wet gemacht, dass nun schneller und gezielt auf einzelne Teilrouten zugegriffen werden kann. Desweiteren ändert sich innerhalb eines Problems nicht die Anzahl der Teilrouten, so dass die Depots unberührt bleiben. Durch die Verwendung von Zeigern wird der Speicheraufwand gering gehalten. Ein Vergleich von Städten kann somit über den Zeiger direkt geschehen und ersparrt eine inhaltliche Auswertung wie bspw. die Position der Stadt.

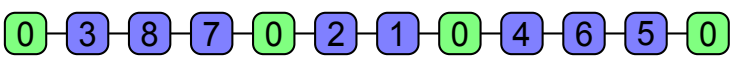


Abbildung 37: Darstellung eines Individuums als ein Genstrang

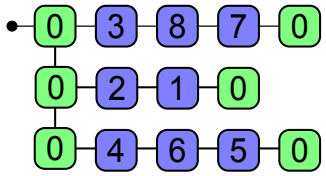


Abbildung 38: Darstellung eines Individuums mit parallelen Gensträngen

In der Liste *multiChromosom* werden diese einzelnen Teilrouten gehalten.

```
1 type TaicMcIndividual = class(TaicIndividual)
2   protected
3     multiChromosom: TList;
4   public
5     constructor Create;
6     destructor Destroy; override;
7 end;
```

Quellcode 2: Individuum - Basiseigenschaften

Eine weitere Separierung für verschiedene Individuen findet nicht statt. Es könnten zwar Gemeinsamkeiten beim CVRP und dem speziellen CVRP vorliegen, aber eine Separierung im Sinne einer weiteren Vererbungsstufe, welche dann diese Gemeinsamkeiten zusammenfasst, ist unnötig. Für ein CVRP mit gelockerten Nebenbedingungen werden Optimierungsmöglichkeiten vermutet, welche auf das normale CVRP nicht zutreffen. Um dieses spezielle CVRP vom normalen CVRP mit der Nebenbedingung, dass verschiedene Städte auch unterschiedlichen Kapazitätsverbrauch haben können, zu unterscheiden, wurde CCVRP als Name verwendet. Das vorangestellte „C“ spiegelt dabei den konstanten Kostenverbrauch ($k = const.$) wider.
Die wichtigen Basiseigenschaften werden in Quellcode 3 aufgelistet:

Wegen der adaptiven Abkühlung erhält jedes Individuum eine eigene Variable für die Temperatur, da nach [Wen95, S. 150] dadurch die Lösungsqualität ein wenig gesteigert werden kann.
Da durch den Vorteil von genetischen Algorithmen mehrere Threads (vgl. Kapitel 5.2) an einem Problem arbeiten können, muss ein Manager wissen können, ob ein Thread bereits an einem Individuum arbeitet. Um diese Angaben zu verwalten, wurde die Variable *locked* hinzugefügt. Die Abbildung des Genstranges auf mehrere Routen erfordert eine Funktion, die ermittelt, in welcher Route und an welcher Position eine bestimmte Stadt vorhanden ist *procedure getRouteAndPosition*.
Weiterhin wird für COSA ein Elternteil als Informationsspender benötigt. Die Nachbarstädte dieses Spenders werden durch *makePropsal* ermittelt. Eine Optimierung wird durch *tryToOptimizeWith* durchgeführt, falls die Fitness besser ist oder im Rahmen des durch Simulated Annealing ermittelten Wertes. Eine Initialisierung des Individuums findet durch *prepare* und *insertCheapest* statt, wodurch die Eröffnungsheuristik Cheapest-Insertion (vgl. Kapitel 3.4.1 und 5.3.1) angewendet wird.

```
type
  TaicIndividualCCVRP = class(TaicMcIndividual)
    protected
      coolingRate: real;
      temperature: real;
      ...
      oneRoute: PTaicCcvrpRoute;
      secRoute: PTaicCcvrpRoute;
      capacity: integer; // how much cities per route
      depot: TPointIdXY; // CCVRP has one single depot
    public
      locked: integer; // 0 = unlocked
      procedure calcFitness; // calc the fitness of the individual
      constructor Create(cap: integer); overload;
      destructor Destroy; override;
      procedure getRouteAndPosition(var cX: PTPointIdXY; var
          pRoute: PTaicCcvrpRoute; var posInRoute: Integer);
      procedure prepare(var depotCity: TPointIdXY; const cities:
          Integer);
      procedure insertCheapest(var newPoint: TPointIdXY);
      ...
      procedure makePropsal(var cM, cL, cR: PTPointIdXY);
      procedure tryToOptimizeWith(var cM, cL, cR: PTPointIdXY);
  end;
```

Quellcode 3: Individuum - Basiseigenschaften

Die ursprüngliche Klasse *TaicMcIndividual* enthält eine Liste, welche wiederum die einzelnen Teilrouten beinhaltet. Diese Teilroutenliste wurde mit den Klassen TaicTspRoute bzw. TaicCcvrpRoute definiert. Die Klasse *TaicCcvrpRoute* kann hauptsächlich den Abstand zwischen 2 Städten in ihrer Route ermitteln, den besten Platz für das Einfügen einer Stadt zurückgeben und bei Bedarf auch einfügen.

```
type
  TaicTspRoute = class(TObject)
    public
      cities: TList;
      ...
  end;

type
  TaicCcvrpRoute = class(TaicTspRoute)
    protected
      function distance: real;
      ...
    public
      procedure addOneCity(var onePoint: TPointIdXY);
      function getBestInsertFitness(var onePoint: TPointIdXY):
          TBestInsertPos;
      procedure insertCity(var onePoint: TPointIdXY; const
          position: integer);
      ...
  end;
```

Quellcode 4: Individuum - Teilroute

5.2 Manager

Mit dieser Grundstruktur können nun Individuen erzeugt werden und es kann auf einzelne Elemente zugegriffen werden. Eine Managerklasse soll dafür sorgen, dass gezielt Individuen mittels genetischen Operatoren manipuliert werden, sodass insgesamt eine Verbesserung der Fitness die Folge ist. Dabei müssen im Manager Abbruchbedingungen und der allgemeine Vorgang

festgelegt werden, welches Individuum welchen Spender bekommt und wie die Rechenleistung optimal ausgenutzt wird. Die Managerklasse kann ebenfalls Probleme und Lösungen aus einer Datei laden, wodurch vordefinierte Probleme vergleichsweise getestet werden können und eine Güte gegenüber exakten Lösungen und anderen Verfahren bestimmt werden kann.

Eine Koordinatenliste *coords* enthält dabei alle für die Optimierung verwendeten Städte. An die Individuen werden die Speicherzellen übergeben, welche dann durch Zeiger ausgelesen werden können. In einer weiteren Liste *individuals* sind alle Individuen eingetragen. Um die aktuellen Prozessoren auch auszureizen, sollte sich kein normaler itterativer Algorithmus dem Problem widmen. Durch die Aufteilung der Problemlösung auf mehrere Threads kann gewährleistet werden, dass mehrere Prozessorkerne für die Lösungssuche genutzt werden. Die Funktion *checkFreeForProposal* gibt an, ob ein Thread ein Individuum für einen Vorschlag benutzen darf. Ist das Individuum nicht als Informationsspender geeignet, wird ein anderes Individuum gewählt. Dadurch das mehrere Threads auf die selben Speicherbereiche zugreifen könnten, ist so eine Abfrage unumgänglich. Es ist ebenfalls möglich durch eine kritische Sektion den Rechner zu zwingen, dass ein einzelner Prozess Daten von dem Individuum holt. Während eines solchen Vorgangs kann aber kein anderer Thread arbeiten. Falls gerade ein anderer Thread damit beschäftigt ist, bspw. einen Fitnesswert zu berechnen, so wird dieser auch unterbrochen. Die Wahl fiel somit auf ein eigenes Sperrsystem, welches durch die dafür eingefügte Variable keinen Thread zu keinem Zeitpunkt blockiert. Eine Analyse des Vorgangs wird durch ein eigenes Fenster dargestellt (siehe Kapitel 6). Ermittelt werden kann so, wann wie lange und in welchem Maß eine Optimierung stattfand.

```
1  type
2    TaicCcvrpManager = class(TObject)
3      protected
4        coords: TList;
5        individuals: TList;
6        ...
7        criticalSection: TCriticalSection;
8        numberOfProcessors: integer;
9        threads: TList;
10     public
11       function checkFreeForProposal(picked: integer): boolean;
12       procedure createIndividuals(const count, cap: Integer; var
             progressBar: TProgressBar);
13       procedure createProblemFromFile(const filename: String);
14       procedure createProblemRandom(const quantity: Integer);
15       function getIndividualCount: integer;
16       function getIndexOfBestIndividual: integer;
17       procedure optimize(maxTransitions: Int64; maxTime: Integer;
             startTemp: real; const coolingRate: Real);
18       procedure showAnalyseWindow;
19       ...
20   end;
```

Quellcode 5: Managerklasse

Die für den Manager entwickelte Threadklasse wird bei Bedarf durch ein Event gestartet und enthält den für die Optimierung aufzurufende Prozedur *drift*, welche ein Individuum nach Möglichkeit wie in Kapitel 4.4 beschrieben ein Stück näher an ein anderes Individuum rückt.

```
1  type
2    TaicCcvrpThread = class(TThread)
3      protected
4        mngCcvrp: TaicCcvrpManager;
5        startEvent: TEvent;
6        comeToAnEnd: boolean;
7        individualProposal,
8        individualToModify: PTaicIndividualCCvrp;
9        ...
10       procedure drift(var cM, cL, cR: PTPointIdXY; var
             individualToModify: PTaicIndividualCCvrp);
11   end;
```

Quellcode 6: Threadklasse des Managers

Ein Programmablaufplan (Abbildung 39) vermittelt das Konzept und prinzipiellen Ablauf der Transition eines Individuums mittels eines Spenders. Dieses Flussdiagramm ist komplett in den Threads umgesetzt. Nur der Test, ob eine Abbruchbedingung erfüllt ist, könnte teilweise vom Manager selbst übernommen werden.

Der Eignungsabfragen, ob ein Individuum als Spender brauchbar ist bzw. modifiziert werden darf ist nur durch die Möglichkeit des parallelen Rechnens notwendig. Auf einem Rechner, der nur einen Prozessor hat wird auch nur ein Thread gestartet. Ein Weglassen dieser Sperren würde dann den Prozess leicht beschleunigen. Da aber aktuelle Prozessoren meistens mehrere Kerne haben, wird auf eine extra Funktion ohne Sperre verzichtet.

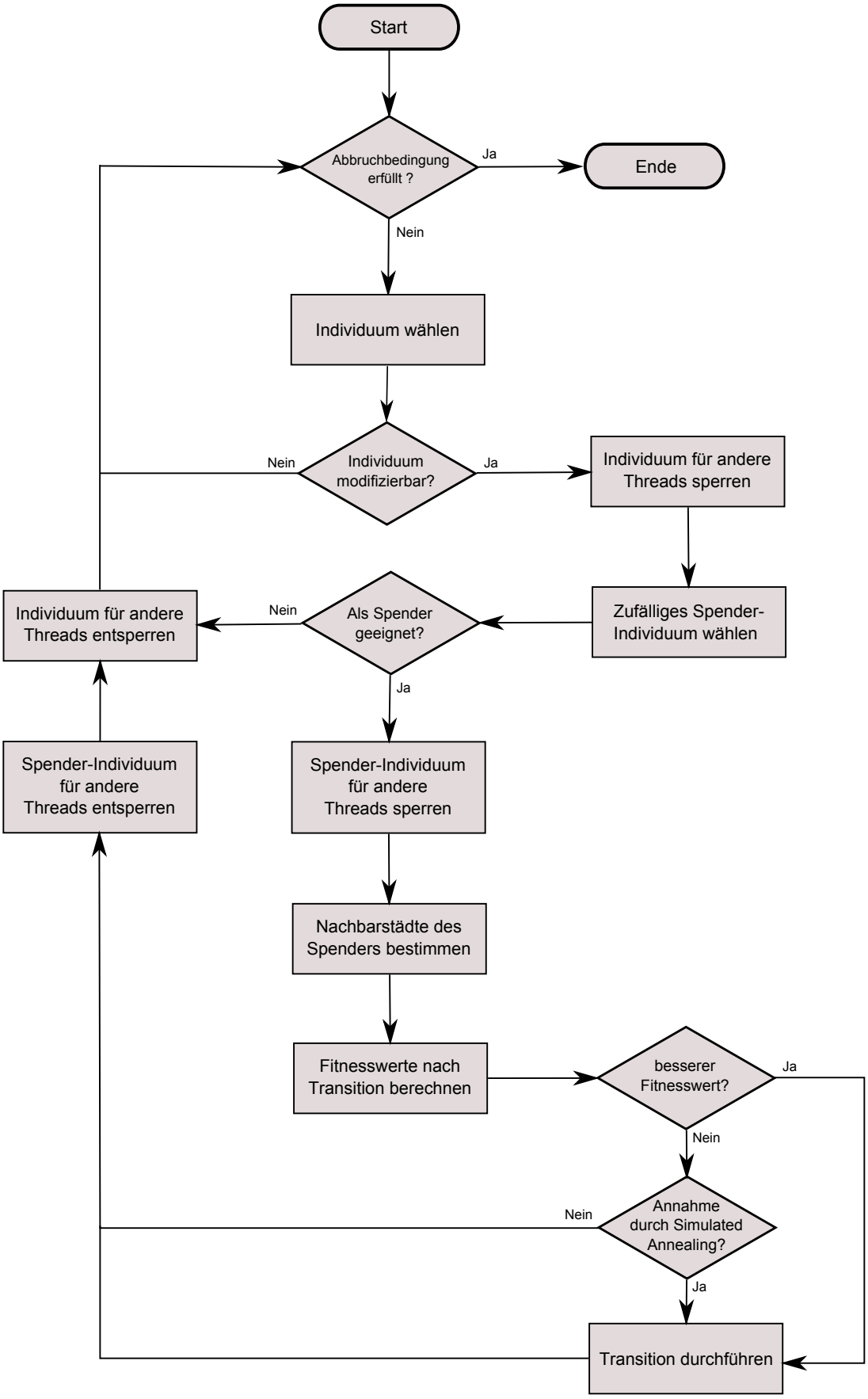


Abbildung 39: Programmablaufplan für die threadsichere Optimierung

5.3 Der Optimierungsablauf

5.3.1 Cheapest-Insertion

Die Eröffnungsheuristik wird mit der Erstellung eines Individuums gestartet. Zunächst wird in einer Schleife ermittelt, welche Teilroute und an welcher Position dieser Teilroute die Einfügemöglichkeit die geringsten Umwegskosten hinzufügt. Im Anschluss wird diese Stadt dann an dieser Stelle eingefügt. Würde nur ein Genstrang existieren, so müssten nicht zusätzlich noch die Teilroute ermittelt werden. Allerdings fällt dieser Umstand nicht weiter ins Gewicht, da die Anzahl der möglichen Einfügepositionen gleich ist.

```
1  procedure TaicIndividualCCVRP.insertCheapest(var newPoint:
       TPointIdXY);
2  var
3    i: integer;
4    bestRoute: integer;
5    bestLen: real;
6    bestPosInRoute: integer;
7    bestResultForThisRoute: TBestInsertPos;
8  begin
9    bestRoute:=-1;
10   bestLen:=MaxInt;
11   bestPosInRoute:=-1;
12
13   for i := 0 to multiChromosom.Count - 1 do
14   begin
15     // foreach subroute
16     oneRoute:=multiChromosom[i];
17
18     // check capacity problems
19     if oneRoute^.cities.Count < capacity + 2 then
20     begin
21
22       // get the best best len and the best insert point
23       bestResultForThisRoute:=oneRoute^.getBestInsertFitness(
             newPoint);
24
25       // is the best fitness of the best insert point better than
             the old one?
26       if bestResultForThisRoute.len < bestLen then
27       begin
28         // keep the best len in mind
29         bestLen:=bestResultForThisRoute.len;
30         // notice the number of the subroute
31         bestRoute:=i;
32
33         // notice the position of the city in this subroute
34         bestPosInRoute:=bestResultForThisRoute.position;
35       end;
36
37     end;
38   end;
39   // select the subroute that has the best insert possibility
40   oneRoute:=multiChromosom[bestRoute];
41   // and insert it
42   oneRoute^.insertCity(newPoint, bestPosInRoute + 1);
43 end;
```

Quellcode 7: Cheapest Insertion

5.3.2 Ursprüngliche Umsetzung und Erweiterung

In der ursprünglichen Umsetzung von [Wen95, S. 177] für das CVRP um ein Individuum an ein anderes heranzurücken heisst es:

> Eine beliebige Tour t_i wird aus dem Tourenplan I_i herausgegriffen. Zu einer beliebigen Stadt s aus t_i wird sodann ein neuer Nachbar gesucht. Hierzu wird zunächst diejenige Tour t_j ermittelt, der die Stadt s in der Informationsspender-Konfiguration I_j zugeordnet ist. Der in der Pfadrepräsentation von t_j rechts von s stehende Kunde sei als u bezeichnet. Gehört u nun auch in Konfiguration I_j zur gleichen Tour (nämlich t_i) wie s so wird eine Inversion der Tour t_j genau so durchgeführt, dass s und u anschliessend benachbart sind. Ist u dagegen in I_i einer anderen Tour zugeordnet als s, können beide nicht zusammen kommen.

Falls beide Städte nicht zusammenkommen können wird empfohlen einen zufälligen validen Nachbarn für die Inversion zu bestimmen. Unerwähnt bleibt jedoch wie genau die Inversion durchgeführt werden soll. Es ist wichtig, dass s in t_j an u heranrückt, da u nur so auch ein Depot sein darf. Eine Inversion, in der u gerückt wird, kann eine invalide Teilroute entstehen lassen. Im folgenden Beispiel fällt die Wahl auf $s = 3$ bei der Teilroute des Spenders, wodurch $u = 0$ ist (Abbildung 40). Würde u jetzt an s heranrücken, entsteht eine invalide Teilroute (Abbildung 42). Hingegen steht einer Inversion mit s an u nichts im Wege (Abbildung 43).

Abbildung 40: Inversion Spenderindividuum

Abbildung 41: Inversion Zielindividuum

Abbildung 42: Inversion Ungültige Inversion

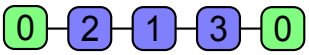


Abbildung 43: Inversion Gültige Inversion

Ein entscheidener Vorteil des Algorithmus ist der, dass keine Kapazitätsprobleme auftauchen können. Es werden keine Städte der Teilroute hinzugefügt und auch keine Städte entnommen. Da aber Städte im speziellen CVRP keine unterschiedlichen Kosten haben, können routenübergreifende Operatoren entwickelt werden.

Der erste Gedanke ist der, aus dem Spender mehr Informationen zu holen. Wird sich bspw. der linke und rechte Nachbar einer Stadt s gemerkt, ist die Wahrscheinlichkeit diese Nachbarstadt auch in der Teilroute t_j zu finden, etwas höher. Falls beide Nachbarstädte in der Teilroute existieren, kann ein Fitnessvergleich oder der Zufall über die Inversion entscheiden. Routenübergreifende Operatoren wären wie folgt möglich:

Ohne die Kapazität in einer Teilroute zu beeinflussen können die Routen getauscht werden, wenn die Anzahl der Städte in den rechten oder linken Armen zweier Teilrouten gleich lang sind (Abbildungen 45, 46).

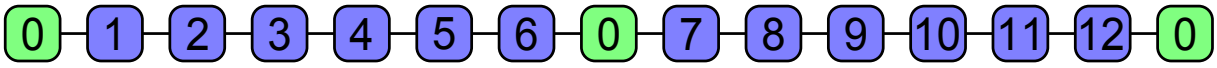


Abbildung 44: Routenübergreifende Operatoren: Ausgangssituation

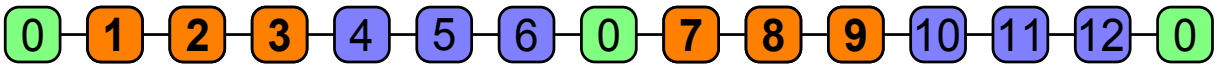


Abbildung 45: Routenübergreifende Operatoren: gleichlange rechte Arme

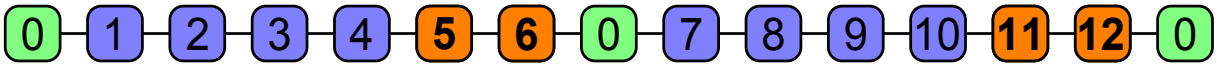


Abbildung 46: Routenübergreifende Operatoren: gleichlange linke Arme

Auch gespiegelt betrachtet gleichlange Arme können vertauscht werden. So wäre eine innere Inversion (Abbildung 47) und eine äußere Inversion (Abbildung 48) implementierbar.

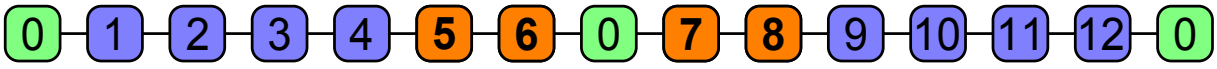


Abbildung 47: Routenübergreifende Operatoren: gleichlange mittlere Arme

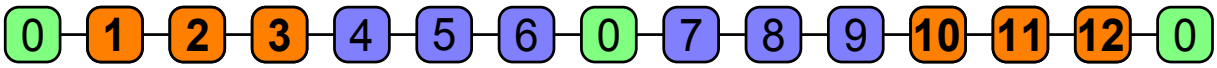


Abbildung 48: Routenübergreifende Operatoren: gleichlange äußere Arme

Der Vorteil bei allen diesen Inversion ist der, dass auch nur zwei Kanten geändert werden wie beim ursprünglichen Ansatz. So ist beispielsweise in Abbildung 48 nach einer Inversion die Kante $\overline{3\ 4}$ und die Kante $\overline{9\ 10}$ getauscht worden zu $\overline{10\ 4}$ und $\overline{9\ 3}$, wie es in Abbildung 49 zu sehen ist.

Abbildung 49: Routenübergreifende Operatoren: äußere Inversion nach dem Tausch

Diese Verfahren sind möglich, da durch eine Symmetrie die Routenlänge nicht beeinträchtigt wird und die Städtekosten identisch sind. Die Wahrscheinlichkeit, nun also einen gültigen Nachbarn zu finden, ist durch einen linken und rechten Nachbarn des Spenders sowie den vier weiteren Möglichkeiten eine Inversion zu vollziehen, ein wenig gestiegen. Je nach Länge der Gesamtroute und der Kapazität der Transporteinheit kann dieser Vorteil unterschiedlich gut sein. Bei kleineren Kapazitäten wie dem Ladeschlitten des Medikamentenlagers mit $Q = 6$ kann jedoch so, falls die Stadt s nicht in der gleichen Route wie u_1 (rechter Nachbar) oder u_2 (linker Nachbar) ist, bei mindestens zwei Städten pro Teilroute der Operator verwendet werden.

Es bleibt zu überprüfen, ob dadurch wirklich eine schnellere Optimierung stattfindet und ob die Vielfaltigkeit, welche durch zufällige Transitionen gegeben ist, nicht zu stark unterbunden wird. Ein genetischer Algorithmus hat die negative Eigenschaft, durch zu viel „perfektes“ Material, sich frühzeitig auf ein lokales Optimum einzupendeln (vgl. [BHS07, S. 81]).

5.3.3 Verbesserungsalgorithmus

Die Optimierungsfunktion vergleicht zuerst ob beide Städte des Spender-Individuums in der gleichen Teilroute ist und ob diese schon nebeneinander liegen. Wenn das der Fall ist, kann abgebrochen werden und eine zufällige Transition vollzogen werden. Im weiteren Fall könnten die Städte zwar in der gleichen Teilroute sein, aber dennoch nicht nebeneinander liegen. Der Beispielcode gibt an, wie eine Situation behandelt wird, in welche der linke Nachbar des Spenders auch im Ziel-Individuum links von der ursprünglich gewählten Stadt liegt. Mit *getDistance* wird die Kantenlänge zwischen zwei Städten ermittelt. Dabei ist in *oldFitness* die aktuelle Fitness gespeichert und in *newFitness* die Fitness, welche durch einen Kantentausch entstehen würde. Ist die Verbesserung kleiner oder gar besser als die Temperatur kann eine Inversion durchgeführt werden. Die richtige Anwendung von Simulated Annealing, die Transition nur zu einem definierten Prozentsatz durchzuführen hat sich als folge der Experimente zu mindest für das spezielle CVRP als nicht wesentlich herausgestellt. Eine Anpassung von *temperature* wird adaptiv durchgenommen, falls auch eine Transition durchgeführt wurde.

```
1  // check if cL is already neighbour
2  if ((cMpRoute = cLpRoute) and (abs(cMposInRoute-cLposInRoute)=1))
       then
3  begin
4    exit;
5  end;
6
7  if (cLpRoute = cMpRoute) then
8  begin
9
10   if cLposInRoute < cMposInRoute then
11   begin
12     oldFitness:=getDistance(cL, cLR) + getDistance(cM, cMR);
13     newFitness:=getDistance(cL, cM) + getDistance(cLR, cMR);
14     daFitness[LRL]:=deltaFitness(newFitness, oldFitness);
15
16     if daFitness[LRL] < temperature then
17     begin
18       iU := cLposInRoute + 1;
19       iD := cMposInRoute;
20       repeat
21         cMpRoute.cities.Exchange(iU, iD);
22         iU:=iU+1;
23         iD:=iD-1;
24       until (iU >= iD);
25       temperature := temperature * coolingRate;
26     end;
27
28   end;
29 end;
```

Quellcode 8: Optimierungsalgorithmus - Standard

Ein Beispiel für eine routenübergreifende Transition ist in Quellcode 10 gegeben. Hier wird noch zusätzlich zu den in Kapitel 5.3.2 überlegten Verfahren überprüft, ob ein ungleich langer Austausch von „Armen“ möglich ist. Für das ursprüngliche Beispiel in Abbildung (46) würden hier nun unterschiedlich lange Arme möglich sein, wenn dadurch die Kapazität in keine der modifizierenden Teilrouten verletzt wird.

```
1     // check capacity problems
2     if ((cLposInRouteD+cMposInRouteD <= capacity) and
3         ((routeCapM+routeCapX)-(cLposInRouteD+cMposInRouteD) <=
              capacity)) then
4     begin
5       // to figure result if we swaping the both left arms (XLL)
            arms
6       oldFitness:=getDistance(cL, cLL) + getDistance(cM, cMR);
7       newFitness:=getDistance(cL, cM) + getDistance(cLL, cMR);
8       daFitness[LLL]:=deltaFitness(newFitness, oldFitness);
9     ...
10    end;
11    if (daFitness[LLL] < temperature) then
12      begin
13        for iU := 1 to cMposInRoute do
14        begin
15          oC:=cMpRoute^.cities[1];
16          cMpRoute.cities.Insert(1, oC);
17          cMpRoute.cities.Delete(1);
18                                              ...
19        end;
20        for iU := 1 to cLposInRoute - 1 do
21        begin
22          oC:=cLpRoute^.cities[1];
23          cLpRoute.cities.Insert(1, oC);
24          cLpRoute.cities.Delete(1);
25        end;
26    end;
```

Quellcode 9: Optimierungsalgorithmus - Routenübergreifend

Für die Distanz zweier Städte untereinander wird der euklidische Abstand genommen. Eine weitere Möglichkeit für ein Lager wäre die Verwendung des Manhattan-Abstands, welcher den absoluten Wert der X-Koordinaten und den absoluten Wert der Y-Koordinaten beider Städte zusammenaddiert. Die Manhatten-Metrik zu benutzen hat sogar den Vorteil, dass hier direkt der Verschleiß an den Achsen berücksichtigt wird, da je eine Achse für den Fahrtweg auf einer Koordinate zuständig ist. Da ein Ladeschlitten aber nicht zwingend auf zwei Achsen positioniert werden muss, ist der euklidische Abstand hinreichend.

```
1   // euclidean distance
2   function getDistanceE(var pFrom, pTo: PTPointIdXY): real;
3   begin
4     result:=Sqrt(Sqr(pFrom.X - pTo.X) + Sqr(pFrom.Y - pTo.Y));
5   end;
6
7   // taxicab geometry (manhattan)
8   function getDistanceM(var pFrom, pTo: PTPointIdXY): real;
9   begin
10    result:=abs(pFrom.X - pTo.X) + abs(pFrom.Y - pTo.Y);
11  end;
```

Quellcode 10: Optimierungsalgorithmus - Routenübergreifend

6 Benutzeroberfläche

Die anzufahrenden Fachpositionen des Auslagerungsauftrages können schematisch dargestellt werden. Da die Städte nur eine X- und Y-Position haben, werden sie auf eine zweidimensionale Fläche als Punkte angezeigt. Das Depot (1) ist ein wenig hervorgehoben, um besser sehen zu können, von wo und wohin die Medikamente gefahren werden müssen.

Auf der rechten Seite befindet sich eine Box „Probleme“, in der verschiedene Probleme geladen und auch zufällig generiert werden können. Optimallösungen, weche bspw. mittels dem Branch-and-Cut - Verfahren ermittelt wurden, können so für einen Vergleich herangezogen werden. Durch die Box „Lösungsbasis“ kann die Anfangspopulation mit der festgelegten Größe generiert werden. Die Parameter für das Simulated Annealing werden in der Box „Abkühlung“ zusammengefasst. Die Anfangstemperatur und der Abkühlungsfaktor können hier auch automatisch an das Problem angepasst werden. Die Temperatur errechnet sich dabei aus der aktuellen Temperatur multipliziert mit diesem Faktor. Es ist nicht der eigentliche Abkühlungsfaktor, der vermuten lässt, dass ein steigender Wert eine schnellere Abkühlung zur Folge hat. Als Abbruchkriterium sind mehrere Einstellungen möglich. Als Defaultwert wurde die Zeit genommen, da diese ausschlaggebend für ein echtes Medikamentenlager ist, in welchem der Ladeschlitten innerhalb einer kurzen Zeit losfahren sollte. Das Textfeld oben rechts gibt Informationen über Vorgänge an, wie bspw. das Laden einer Datei. Im Container „Optimierung“ befindet sich der Button „Optimieren“ mit welchem der Verbesserungsprozess gestartet werden kann. Für eine Optimierung bei der günstige Parameter automatisch gewählt werden sollen, kann der Button „Automatische Einstellungen“ verwendet werden. Im unteren Bereich befindet sich eine Box, welche die Struktur der einzelnen Individuen anzeigt.

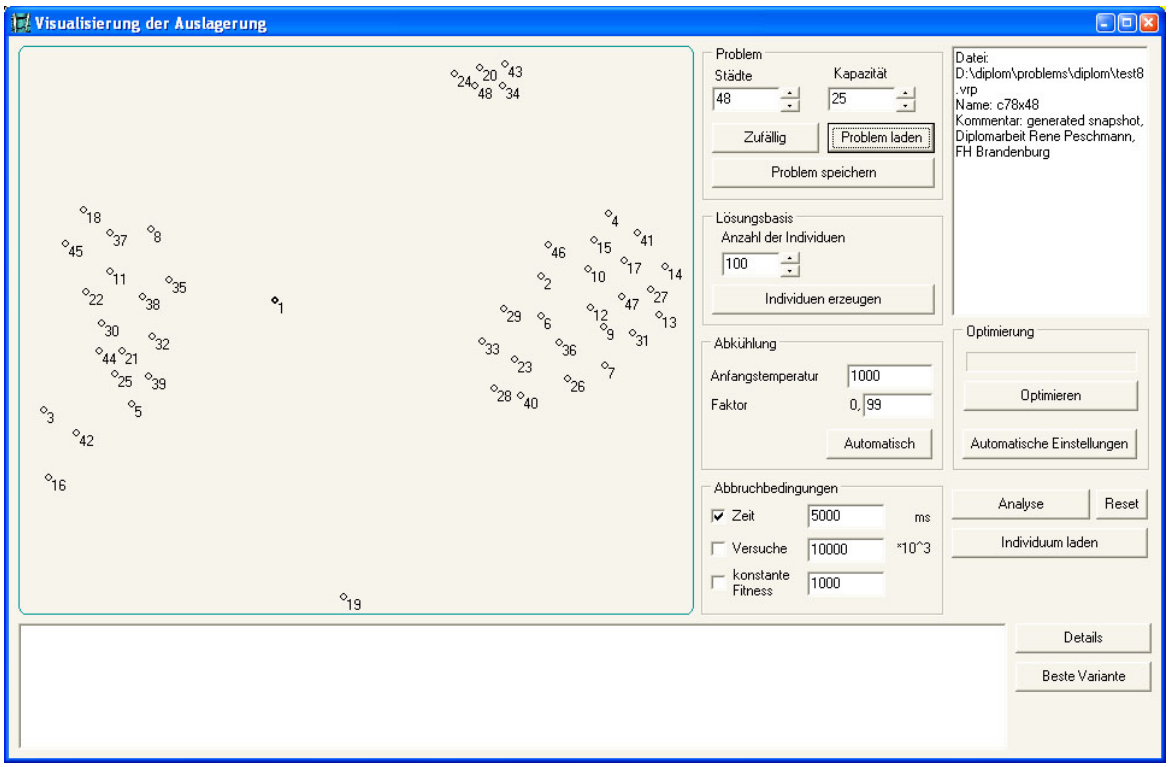


Abbildung 50: Oberfläche des Programms zum Lösen des speziellen CVRPs

Ein Problem kann nachträglich verändert werden. Dazu muss eine Stadt mit der linken Maustaste ausgewählt werden. Im Anschluss kann mit gedrückter linker Maustaste eine Art Vektor auf

die Fläche gezeichnet werden. Die Stadt bewegt sicht dann zu den neuen Koordinaten. Neue bzw. modifizierte Problemstellungen können mit „Problem speichern“ als vrp-Datei gespeichert werden. Die Dateistruktur ist durch TSPLib gegeben, welche um 1991 vom Institut für Mathematik in Heidelberg durch Gerhard Reinelt veröffentlicht wurde. Eine typische Datei, welche kompatibel ist mit dem implementierten Lösungsverfahren befindet sich im Anhang (Quellcode 12). Durch die einfache, selbst erklärende Datenstrtuktur können auch Probleme mit exakten Werten generiert werden.

Sobald ein Problem vorliegt, können mit der eingestellten Anzahl Individuen erzeugt werden. Durch das teilweise vom Zufall abhängige Cheapest-Insertion entsteht die erste valide Population. Das Route des besten Individuums wird dargestellt.

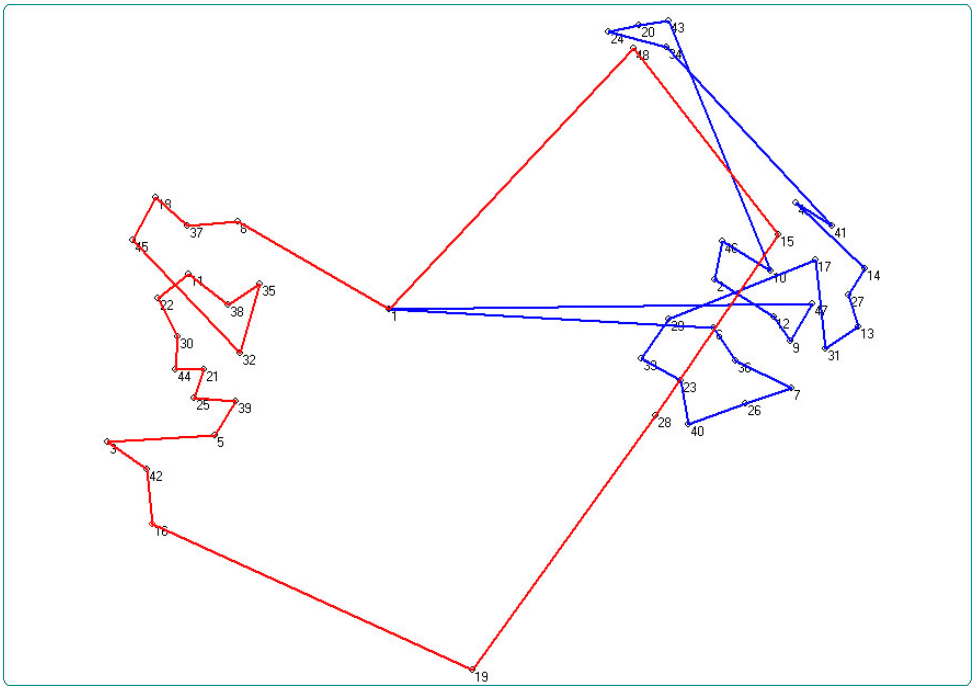


Abbildung 51: Bestes Individuum der ersten Generation

Die Population wird in einer Liste angezeigt. Das beste Individuum ist markiert. In der Liste ist für jedes Individuum zu sehen, welche Nummer es hat, welche Teilrouten es beinhaltet und welche Gesamtlänge dessen Route hat (Abbildung 52). Mittels „Details“ kann das Individuum auch vollständig als Text und mit den im Fachpositions-Fenster dargestellten Farben angezeigt werden (Abbildung 53).

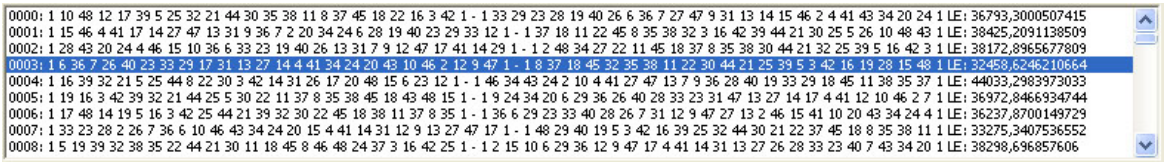


Abbildung 52: Liste aller Individuen mit Länge der Gesamtroute

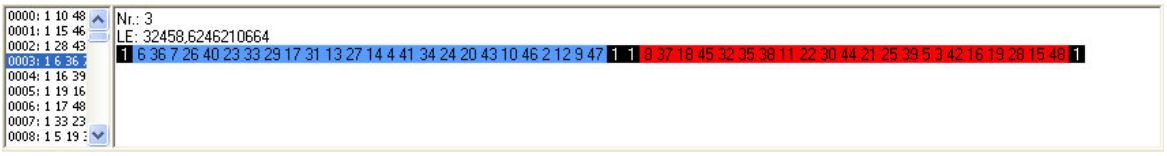


Abbildung 53: Liste aller Individuen mit Länge der Gesamtroute

Durch „Optimieren“ wird nun der Verbesserungsalgorithmus gestartet. Nachdem die Abbruchbedingung greift, wird nun wieder das beste Individuum angezeigt. Zusätzlich öffnet sich ein Analyse-Fenster, welches auch über „Analyse“ geöffnet werden kann. Die 3 farbigen Kurven (Abbildung 54) stehen für verschiedene Werte. Die blaue Kurve zeigt die Fitness des bestbekannten Individuums. Je weiter diese Kurve unten verläuft, desto mehr steigt die Fitness bzw. desto kürzer ist die Gesamtstrecke der stärksten Elements. Diese Darstellung darf nicht verwundern, denn durch Zufall kann ein relativ gutes Individuum gleich am Anfang gefunden und gemerkt werden. Im weiteren nahen Verlauf ist die Temperatur aber noch so hoch, dass fast jegliche Verschlechterung zugelassen wird und ein besseres Individuum vorläufig nicht gefunden werden könnte. Die rote Kurve zeigt den durchschnittlichen Fitnesswert der letzten Individuen an. Auch diese Kurve kann durch Simulated Annealing vorerst eine Verschlechterung anzeigen, zeigt aber üblicherweise eine sich langsam verbessernde Population an. Die gelbe Kurve zeigt die durchschnittliche Temperatur der adaptiven Abkühlung für Simulated Annealing an. Eine schwache Krümmung spiegelt einen hohen Faktor beim Abkühlungsprozess wieder. Ist der Faktor für einen Abkühlungsdurchlauf verhältnismäßig klein gewählt, kühlt die Temperatur sehr stark ab und diese Kurve läuft schneller gegen Null.
Das Analysefenster gibt ausserdem als Text an, welches die am Ende beste und schlechteste Fitness war sowie welche Temperaturen durch den Abkühlungsprozess erreicht wurden.

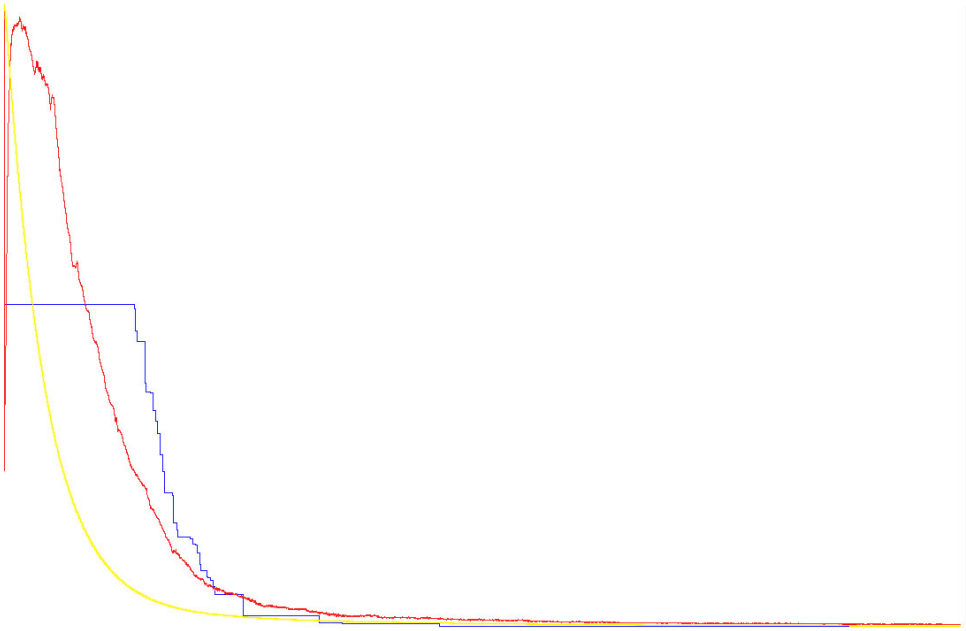


Abbildung 54: Kurvendarstellung des Optimierungsverlaufs

7 Experimente

7.1 Probleminstanzen

Da nur wenige wie in Kapitel 3.1 beschriebene Standardprobleme für das spezielle CVRP existieren, mussten weitere Problematiken erzeugt werden. Durch das SYMPHONY-System (Version 5.1.10), welches von Ted Ralphs seit 2000 immer weiter entwickelt wurde (siehe [GRW05, S. 61]) konnten einige weitere Problemklassen erstellt werden. Durch Branch and Cut, welches sich dieses System zu Nutze macht, wurde in folgenden Zeiten die optimale Lösung gefunden. In folgender Tabelle befinden sich die jeweiligen Instanzen mit den dazugehörigen Berechnungszeiten in Sekunden. Anzumerken ist dabei, dass nur ein Prozessor für die Berechnung benutzt wurde. Unklar bleibt, ob eine Verteilung der Rechenkapazität auf mehrere Prozessoren möglich wäre.

Städte	Kapazität	Berechnungszeit (Sek.)
20	5	1
30	6	2
48	15	13
48	25	8
50	6	351
50	9	45
50	10	165
50	11	32

Tabelle 3: Berechnungszeiten für exaktes Lösen mittels Branch-and-Cut

Da hier die Berechnungszeit bei wenig Städten kaum bemerkbar ist, besteht wenig Hoffnung, dass ein evolutionärer Algorithmus mithalten kann. Auffällig ist jedoch die unterschiedliche Berechnungszeit bei gleichbleibender Städteanzahl. Gerade bei der Ladekapazität des Medikamentenlagers $Q = 6$ schnellt die Berechnungszeit in die Höhe. Vermutlich kann beruhigt bis zu 20 Städten das exakte Verfahren verwendet werden. Bei größeren Problemen bleiben die Tests nicht aus.

7.2 Kleine Probleme

Hinsichtlich der Tabelle kann bei 20 Städten und einer kaum spürbaren Berechnungszeit von einem kleineren Problem die Rede sein. Nach Erstellen der ersten Generation ist das beste Individuum bereits 12% am globalen Optimum dran (Abbildung 55). Nach weniger als einer Sekunde kann allerdings schon keine bessere Lösung mehr gefunden werden als in Abbildung 56 erkennbar. Diese Lösung ist mit 23088LE nur um $1,36\%$ vom Optimum (Abbildung 57, 22779LE) entfernt. Auch in weiteren Tests mit dieser Problemklasse konnten kaum andere Werte ermittelt werden.

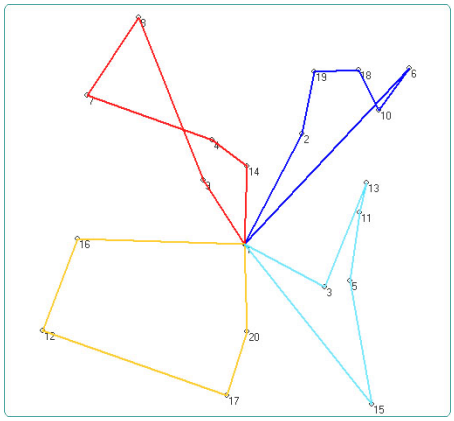


Abbildung 55: CCVRP20, Q=5, Bestes Individuum nach Cheapest-Insertion

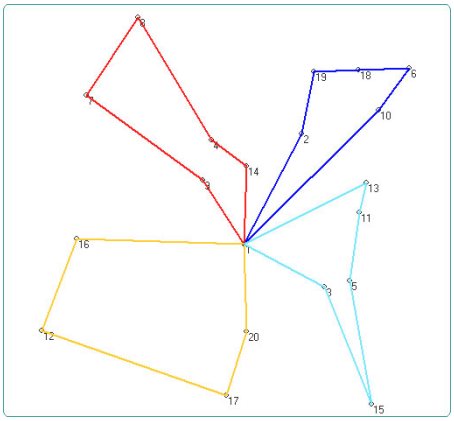


Abbildung 56: CCVRP20, Q=5, Bestes ermitteltes Individuum nach 0.5 Sekunden

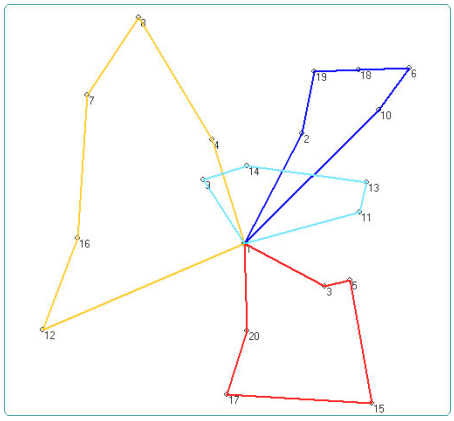


Abbildung 57: CCVRP20, Q=5, Beste Lösung mit 22779LE

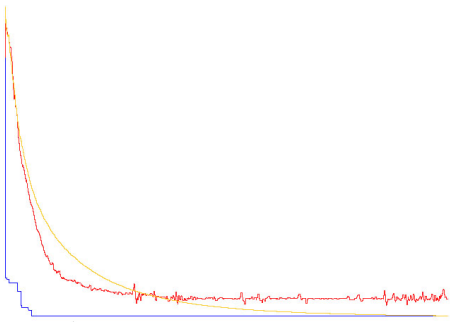


Abbildung 58: CCVRP20, Q=5, Diagramm

7.3 Mittlere Probleme

Mittlere Probleme wie das schon bekannte att48 mit $Q = 15$ liegen nach 5 Sekunden Berechnungszeit ca. 10% über dem Optimum. Auch eine Änderung der Parameter für das Simulierte Abkühlen konnte keine Verbesserung erwirken.

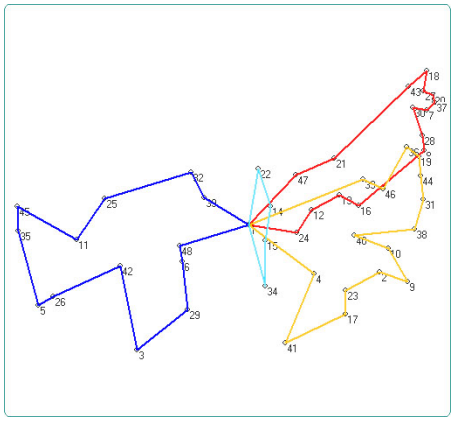


Abbildung 59: ATT48, Q=15, Beste ermittelte Lösung mit 42781LE

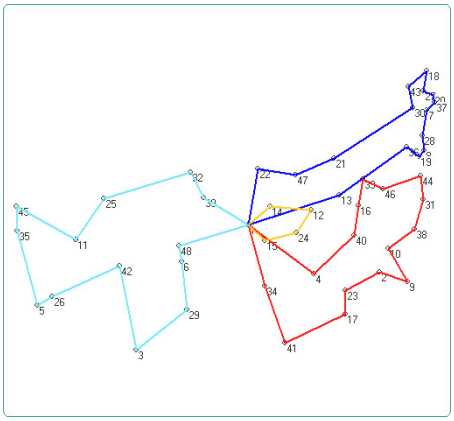


Abbildung 60: ATT48, Q=15, Globales Optimum mit 40002LE

Das erstellte CCVRP50 kann mit 18% über dem globalen Optimum nur in Sachen Zeit relativ gut plaziert. Bei einem Weg, der um so viel größer ist als die beste bekannte Lösung, muss schon abgewogen werden, ob eventuell ein kürzerer Weg die fast sechs Minuten Rechenzeit armortiesiert.

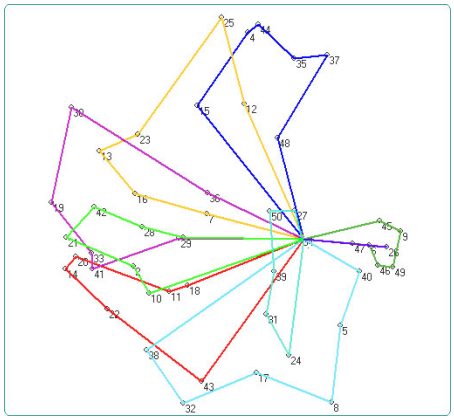


Abbildung 61: CCVRP50, Q=6, Beste ermittelte Lösung mit 44687LE

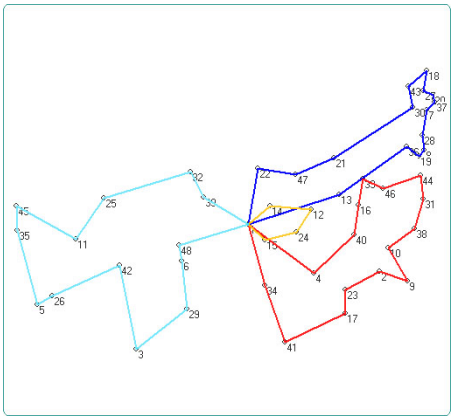


Abbildung 62: CCVRP50, Q=6, Globales Optimum mit 37622LE

Andere Problemklassen in dieser Größe verhalten sich mit COSA ähnlich. Um eine wirklich gute Lösung zu bekommen, muss eine längere Zeit und eine geringere Abkühlung genommen werden. Auf dem aktuellen System ist nach 2 Minuten bei dem gleichen Problem auch eine Lösung mit 40444LE $(7,5\%)$ möglich. Die Parameter hierfür waren für die Abkühlung eine Starttemperatur von 10000 und ein Faktor von 0.999. Nach 2 Minuten konnten ungefähr 200'000 Generationen erschaffen werden.

7.4 Weitere Probleminstanzen

Bei größeren Problemen liegt die Stärke des Algorithmus in der schnellen Konvergenz zu einem brauchbaren Optimum. Die Erwartung im schlechtesten Fall maximal 10% der geringsten bekannten Weglänge nicht zu überschreiten, kann leider nur mit dem erwünschten Zeitdruck von fünf Sekunden bei kleinen Problemen erfüllt werden. Um gute Ergebnisse auch noch bei sehr großen Mengen an Städten zu bekommen, muss sehr viel mehr Rechenzeit eingeplant werden. Auch das globale Minimum dann noch zu erreichen ist nur in seltenen Fällen gelungen.

Zusammenfassend bleibt zu sagen, dass COSA für das CVRP ein Algorithmus ist, welcher durch die Geschwindigkeit zu mindest in handelsüblichen Mengen für ein Medikamentenlager eine brauchbare und gute Lösung darstellt.

8 Aussicht

Der Auslagerungsvorgang, wie in der Diplomarbeit beschrieben, ist nur ein kleiner, wenn auch wichtiger Prozess. Es gibt viele Möglichkeiten auf einen der Vorgänge des Einlagerns, der Routenberechnung für das Auslagern, die Zugriffszeit des Greifarms usw. einzuwirken. In den weiteren Abschnitten werden zeitsparende Mittel aufgelistet, welche bereits in anderen Lagerverwaltungsprogramme erfolgreich eingesetzt werden.

8.1 Suche des richtigen Lagerplatzes

Der Lagerplatz selbst kann an vielen Bedingungen geheftet sein. Medikamente brauchen eine bestimmte Umgebungstemperatur und müssen somit an warmen, kühlen oder gefrorenen Lagerplätzen gebracht werden. Sommerartikel werden wahrscheinlich im Winter nicht sehr häufig benötigt. Manche Medikamente sind apothekenpflichtig und dürfen bei einem Ausfall einer Lagerautomatik nicht den Zugriff hindern. Hier wäre es möglich die wichtigen Produkte so zu verteilen, dass ein Teilausfall die Anlage nicht stört. Aber abgesehen von den Bedingungen, welche eventuell an ein Medikament geknüft sind, kann beim Einlagern auch schon ein optimaler Platz gesucht werden. Werden zwei Produkt sehr oft zusammen ausgelagert, lohnt es sich zwei dicht aneinander gelegene Lagerplätze zu benutzen. Arzneimittel, die häufig gebraucht werden können in Nähe der Auslagerposition gelagert werden, damit der Weg und somit die Zeit zum Auslagern so gering wie möglich ist.

8.2 Das Benutzen einer Warteschlange

Durch eine Warteschlange kann der Benutzer mehrere Artikel mit einmal einlagern und dann einer anderen Arbeit nachgehen. Sind die Güter nicht zerbrechlich, kann auch eine Schütte verwendet werden. Der Nachteil von Lagern mit verschiedenen Gütern ist, dass das System automatisch erkennen muss, um welches Gut es sich handelt. Möglichkeiten der Erkennung sind bspw. Erfassen mit der Videokamera, das Auslesen des Barcodes sowie das Auslesen der mittels Funk übertragenen Daten wie es bspw. bei RFID der Fall ist. Die Kosten für die Erfassung mit der Videokamera sind sehr hoch und lohnen nur, wenn viele solcher Erfassungssysteme verkauft werden. Auch das Lesen des Barcodes gestaltet sich schwierig, denn dieser kann sich auf allen Seiten der Verpackung befinden (wie bspw. bei Medikamentenverpackungen). Mittels des RFID-Systems können mehrere Attribute gespeichert und verarbeitet werden. Jedes Gut muss einen RFID-Chip besitzen und das Lese- und Schreibgerät ist auch relativ teuer.

8.3 Priorisierung

Generell ist das Einlagern nicht zu priorisieren. Kommt ein Auslagerungsauftrag, wird auch zuerst der Greifarm bzw. Ladeschlitten für das Auslagern benutzt, denn der Kunde, welcher das Produkt bekommen soll muss zeitnah bedient werden. Wenn das Gut bereits in der Warteschlange zum Einlagern liegt, ist ein direkter Zugriff nur schwer realisierbar. Da aber die meisten Produkte mehrfach im Lager vorhanden sind, gibt es nur selten den Fall, dass ein Produkt während des Einlagerungsprozesses gebraucht wird.

8.4 Optimierung durch Umlagern

In Ruhezeiten, wenn das Lager nicht gebraucht wird wie bspw. in der Nacht können in manchen Lagern die Güter auch umsortiert werden. Eine Umsortierung ist sinnvoll wenn:

1. eine Ware gerade sehr gefragt ist. Wie zum Beispiel Husten-Bonbons während einer Grippewelle.

2. ein Produkt welches aus mehreren Gütern besteht modifiziert wurde. Zum Beispiel hat ein Tisch jetzt andere Tischbeine.

3. das Verfallsdatum einer Ware abgelaufen ist. Wobei dann das Produkt meistens gleich ausgelagert wird.

Falls nur sehr wenige Umlagerplätze existieren, sollte ein Umlagern lieber wegfallen, da durch weniger freie Plätze auch mehr Weg verbraucht wird. Dazu wird kurz auf die Komplexität des Umlagerns eingegangen. In citeTUL95 wird auf die Umschlaghäufigkeit eingegangen. Im allerbesten Fall gäbe es nichts umzulagern. Dabei wäre egal wie voll das Lager ist. Im Fall, dass genau ein Gut am falschen Platz liegt, muss auch nur eine Umlagerung gemacht werden. Es kann davon ausgegangen werden, dass der Zielplatz leer ist, da sonst ja mehr als ein Gut an der falschen Position wäre. Im schlimmsten Fall wäre jedes Gut an einem falschen Platz. Hier wiederum muss man unterscheiden: Im besten Fall sind nur 2 Güter im Lager, die beide auf einer falschen Position sind. Im schlimmsten Fall ist das Lager ausgeschöpft. (Ein Umlagerplatz muss natürlich frei sein.) Die Anzahl der Wege für die Umlagerung bestimmen die Komplexität.

8.5 Hardwarenahe Optimierung

Zu einer qualitativ guten Umsetzung gehört unter anderem auch die komplette Benutzung der möglichen Rechenleistung, gerade weil die Zeit die wichtigste Rolle spielt. Der offensichtlichste Vorteil geht von dem verwendeten genetischen Algorithmus aus, welcher durch seine Parallelisierbarkeit verteiltes Rechnen ermöglicht. Eine zurzeit zahlreich von der Industrie umgesetzte Möglichkeit die Rechenpower zu erhöhen ist die Konstruktion von Multicore-CPUs, also CPUs mit mehreren Kernen. Um diese CPUs vollkommen mit den genetischen Algorithmen auszulasten müssen zunächst die Individuen auf verschiedene Prozesse (Threads) aufgeteilt werden. Im Normalfall muss diese Aufteilung bei jeder neuen Generation gemacht werden. Dadurch, dass die Individuen bei COSA aber nur modifiziert werden und keine Nachkommen ausbilden, fällt diese neue Aufteilung weg und spart somit Rechenkapazität ein. Eine weitere Variante die CPU-Auslastung noch zu verbessern ist die Benutzung erweiterter Befehlssätze wie SSE. Dazu ist es jedoch notwendig eine bestimmte Struktur in den Individuen zu schaffen. Würden die Fitnesswerte in einem extra dafür angelegten Speicherbereich behandelt werden, wäre es möglich mehrere aufeinander folgende Werte gleichzeitig zu setzen.

A Anhang

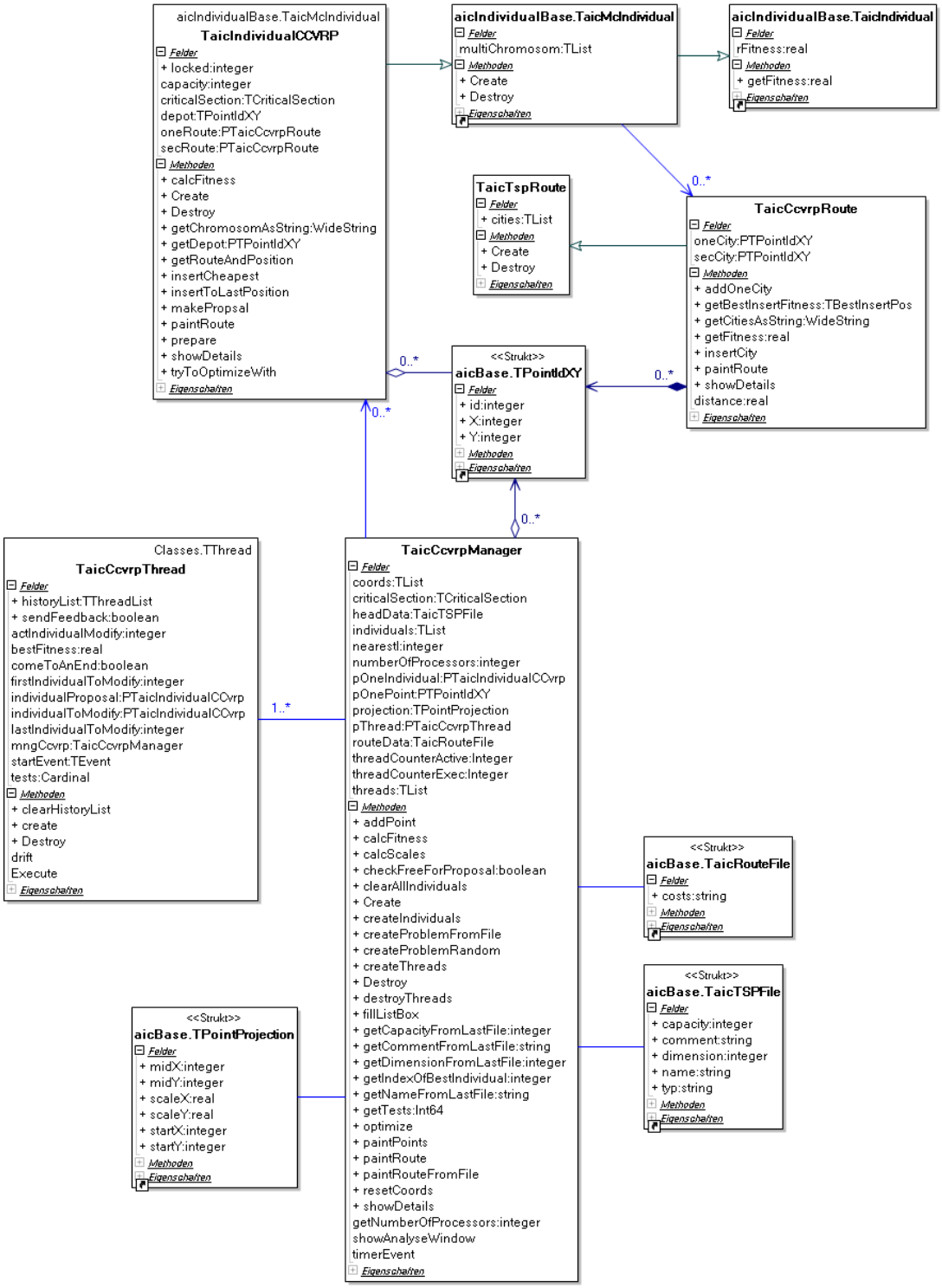


Abbildung 63: UML-Ansicht des Grundaufbaus (Erstellt mit Turbo Delphi 2006)

```
var
  comb: integer;
  routes: extended;
  gc, gn: integer;

function tsproutes(tl: string): extended;
var
  i, ti, tb, tu: integer;
  tr, tn: Extended;
  sl: TStringList;
begin
  sl:=Split(tl,' '); (* String an den Leerzeichen teilen *)
  tr:=1;
  tu:=gn;
  for i := 0 to sl.Count - 1 do
  begin
    tb:=StrToInt(sl[i]);
    tn:=1;
    for ti := tu downto tu-tb+1 do
      tn:=tn*ti;
    tu:=tu-tb;
    if tn>1 then
      tr:=tr*(tn / 2);
  end;
  Result:=tr;
end;

procedure combinations(n, c, l: Integer; s: string);
var
  tr: extended;
begin
  if l=0 then
  begin
    println(s);
    tr:=tsproutes(s);
    println('Kombination #'+IntToStr(comb+1)+': '+FloatToStr(tr));
    routes:=routes+tr; (* Routenanzahl der Kombination addieren *)
    inc(comb); (* Anzahl der Routenkombinationen erhöhen *)
    exit;
  end
  else
  begin
    repeat
      combinations(n-c, Min(c, (n-c)), l-1, s + ' '+IntToStr(c));
      c:=c-1;
    until (l*c<n) or (c=0);
  end;
end;

(* calcRoutes wird vom Hauptprogramm aufgerufen *)
(* n = Anzahl der Zielstädte, c = Kapazität *)
procedure calcRoutes(n, c: integer);
var
  l: integer;
begin
  l:=Ceil(n / c);
  comb:=0;
  routes:=0;
  gn:=n;
  gc:=c;
  combinations(n, c, l, '');
end;
```

Quellcode 11: Anzahl der Routen im speziellen CVRP

```
NAME : att48
COMMENT : (Rinaldi,Yarrow/Araque, No of trucks: 4, Optimal value: 40002)
TYPE : CVRP
DIMENSION : 48
EDGE_WEIGHT_TYPE : EUC_2D
CAPACITY : 15
NODE_COORD_SECTION
1 6823 4674
2 7692 2247
3 9135 6748
4 7721 3451
5 8304 8580
6 7501 5899
7 4687 1373
8 5429 1408
9 7877 1716
10 7260 2083
11 7096 7869
12 6539 3513
13 6272 2992
14 6471 4275
15 7110 4369
16 6462 2634
17 8476 2874
18 3961 1370
19 5555 1519
20 4422 1249
21 5584 3081
22 5776 4498
23 8035 2880
24 6963 3782
25 6336 7348
26 8139 8306
27 4326 1426
28 5164 1440
29 8389 5804
30 4639 1629
31 6344 1436
32 5840 5736
33 5972 2555
34 7947 4373
35 6929 8958
36 5366 1733
37 4550 1219
38 6901 1589
39 6316 5497
40 7010 2710
41 9005 3996
42 7576 7065
43 4246 1701
44 5906 1472
45 6469 8971
46 6152 2174
47 5887 3796
48 7203 5958
DEMAND_SECTION
1 0
2 1
3 1
...
47 1
48 1
DEPOT_SECTION
 1
 -1
EOF
```

Quellcode 12: Datei att-n48-k4.vrp (leicht gekürzt) - 48 Städte CVRP

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Literaturverzeichnis

[AGP08] A. Abraham, C. Grosan, W. Pedrycz. *Engineering Evolutionary Intelligent Systems*. Springer, 2008. ISBN 3540753958.

[Arn08] D. Arnold. *Handbuch Logistik*. Springer, 2008. ISBN 3540729283.

[AIK08] D. Arnold, H. Isermann, A. Kuhn, K. Furmansm, H. Tempelmeier. *Handbuch Logistik*. Springer, Berlin, 2008. ISBN 3540729283.

[Bal98] H. Balzert. *Lehrbuch der Software-Technik*. Elsevier-Verlag, 1989.

[Bla99] U. Blasum. *Anwendung des Branch & Cut Verfahrens auf das kapazitierte Vehicle-Routing Problem*. Doktorarbeit, Universität zu Köln, 1999.

[BHS07] I. Boersch, J. Heinsohn, R. Socher. *Wissensverarbeitung*. Spektrum Akademischer Verlag, 2007. ISBN 9783827418449.

[Cot07] C. Cotta, J. Van Hemert. *Evolutionary computation in combinatorial optimization*. Springer, 2007. ISBN 3540716149.

[DHF01] K. Danzer, H. Hobert, C. Fischbacher, K.-U. Jagemann. *Chemometrik*. Springer Verlag, 2001. ISBN 3540412913.

[Dav91] L. Davis. *Handbook of Genetic Algorithms*. New York, Van Nostrand Reinhold, 1991. Zuerst 1989 verfasst.

[Gei05] M. J. Geiger. *Multikriterielle Ablaufplanung*. Gabler, 2005. ISBN 383500171X.

[GRW05] B. L. Golden, S. Raghavan, E. A. Wasil. *The Next Wave in Computing, Optimization, and Decision Technologies*. Springer, 2005. ISBN 0387235280.

[Hol75] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975. ISBN 0472084607.

[Tec06] G. Honsel. Wettkampf der Profile. *Technology Review*, 11/2006.

[Hus07] S. Hussmann, B. Lutz-Westphal. *Kombinatorische Optimierung erleben in Studium und Unterricht*. Vieweg+Teubner, 2007. ISBN 3528032162.

[Jet07] S. Jetzke. *Grundlagen der modernen Logistik*. Hanser Verlag, 2007. ISBN 3446404597.

[KGV83] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi. Optimization by Simulated Annealing. *Science Vol. 220, Nr. 4598*, 05/1983.

[Lan06] H.-W. Lang. *Algorithmen: In Java*. Oldenbourg Wissenschaftsverlag, 2006. ISBN 348657938X.

[Lae08] U. Lämmel, J. Cleve. *Künstliche Intelligenz*. Hanser Verlag, 2008. ISBN 3446413987.

[Men98] K. Menger, E. Dierker, K. Sigmund, J. W. Dawson. *Ergebnisse eines mathematischen Kolloquiums*. Springer, 1998. ISBN 3211831045.

[Ohr08] C. Ohrt. *Tourenplanung im Straßengüterverkehr*. Dissertation Universität Hamburg, 2008. ISBN 9783834909558.

[Pap82] C. H. Papadimitriou. *Combinatotial Optimization*. Dover Publ., 1998. ISBN 0486402584. Sec. edition.

[Rie08] J. Rieck. *Tourenplanung mittelständischer Speditionsunternehmen*. Gabler Edition Wissenschaft, 2008. ISBN 9783834913982.

[Ros77] D. J. Rosenkrantz, R. E. Stearns, P. M. Lewis II. *An Analysis of Several Heuristics for the Traveling Salesman Problem*. SIAM J. Comput. 6, 1977.

[Sto06] T. Stockheim. *Supply Network Optimization*. Books on Demand, 2006. ISBN 3833461578.

[Tot01] P. Toth, D. Vigo. *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics, 2001. ISBN 0898714982.

[Wan06] R. Wanka. *Approximationsalgorithmen*. Vieweg+Teubner, 2006. ISBN 9783519004448.

[Wen95] Oliver Wendt. *Tourenplanung durch Einsatz naturanaloger Verfahren*. Deutscher Universitäts Verlag; Wiesbaden : Gabler, 1995. ISBN 3824461811.

[Whi89] D. Whitley, T. Starkweather, D. Fuquay. *Scheduling Problems and Traveling Salesman: The Genetic Edge Recombination Operator*. New York, Van Nostrand Reinhold, 1989.

[Wre72] A. Wren, A. Holliday. *Computer scheduling of vehicles from one or more depots to a number of delivery points*. Operation Research Quarterly, 1972.

[Zim07] H. J. Zimmermann. *Operations Research*. Vieweg+Teubner, 2007. ISBN 383480455X.

Hiermit versichere ich, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

René Peschmann
Brandenburg, 11. Mai 2009