Lösung des Travelling Salesman Problem mit Zeitfenstern mittels Heuristiken im Rahmen einer dynamischen Nachlieferung

Bachelorthesis

Bergische Universität Wuppertal
Fachbereich C - Mathematik und Naturwissenschaften

vorgelegt von

Teresa Schnepper

Prüfer: Prof. Dr. Ernst-Peter Beisel, Angewandte Mathematik
Arbeitsgruppe für Optimierung und Approximation

Wipperfürth, 18. April 2011

Inhaltsverzeichnis

Abkürungs- und Variablenverzeichnis

$[a_i, b_i]$	Zeitfenster des Knoten i
A	Menge der Kanten/Bögen
a_i	Zeitfensteranfang von Knoten i
b_i	Zeitfensterende von Knoten i
C	Distanzmatrix
C_1	Zielfunktionswert/Tourkosten
$C_1(v_j, v)$	Kosten für das Einfügen des Knotens v hinter v_j in die Tour
$C_2(v)$	Gesamtkosten einer veränderten Tour
c_{ij}	Kosten für das Zurücklegen der Strecke von i nach j
c_i^-	Strafkosten für zu frühes Ankommen am Knoten i
c_i^+	Strafkosten für zu spätes Ankommen am Knoten i
DK	Dummykunden
G	Digraph
$\gamma(v_i)$	Nachfolgerknoten von v_i
h	Anzahl Knoten, nach denen der Cut durchgeführt wird
IC	Interchange - Verfahren
j^*	Optimale Position für das Einfügen eines Knotens
k	Länge der Tabuliste
λ	Gewichtung der Entfernung des Knotens vom Depot
M	Konstante
m	Anzahl der Kunden
$max\_iteration$	Anzahl Iterationen von Tabu Search
max_same	Anzahl Iterationen ohne Kostenverbesserung
$Move$	Nachbarschaftsoperation
μ_{ij}	Die der Kante (i, j) zugeordnete Entfernung/Kosten
N	Menge der Kunden
$\bar{N}$	Menge der Kunden, die noch nicht zur Tour gehören
$\mathcal{N}(x)$	Nachbarschaft von x
$Nutzen(v)$	Nutzen für das Einfügen von Knoten v an seiner optimalen Position
o	Startdepot
OR	Or-Opt - Verfahren
P	Zulässigkeitsbereich der Lösungen des TSPTW
$\mathcal{P}(P)$	Potenzmenge von P
p	Anzahl der Cluster
r	Länge der Knotenkette im Or-Opt-Verfahren
$\rho(v_i)$	Vorgängerknoten von v_i
S	Zulässiges Segment einer Tour bzw. vollständige Tour

S_0	aktuell ausgeführte Tour
S_a	Eine Tour aus dem Tourenplanpool, $a = 1, ..., s$
SIT	Shift Inside Tour - Verfahren
s	Anzahl der Touren im Tourenplanpool
s_i	Servicezeit am Knoten i
T_i	Ankunftszeit am Knoten i
TK_i	Teilkosten bis Knoten i
$Tlist$	Tabuliste
TS	Transferred Sequence - Verfahren
TSP	Travelling Salesman Problem
$TSPTW$	Travelling Salesman Problem with Timewindows
t	Taktzeit
t_{ij}	Fahrzeit zwischen den Knoten i und j
u_i	Verspätung im Knoten i
V	Menge der Kunden einschließlich des Startdepots
v_1	Erster Knoten der Tour nach dem Depot
v_{q-1}	Letzter Knoten der Tour vor dem Depot
w_i	Wartezeit am Knoten i
x_{ij}	Entscheidungsvariable
ξ	Zufallszahl
y_i	Hilfsvariable; Summe der μ_{ij}
z	Zeitpunkt des Auftretens einer Reklamation

1. Einleitung

Jedes Unternehmen steht heute durch die freie Marktwirtschaft und die zunehmende Globalisierung immer mehr unter dem Druck, sich den häufig widersprüchlichen Erwartungen und Zielsetzungen ihrer Umwelt anpassen zu müssen. Eines der daraus resultierenden Hauptziele ist die Umsatzsteigerung bei gleichzeitiger Kostenminimierung ohne dabei einen Qualitätsverlust zu erleiden oder die Zufriedenheit der Kunden aufs Spiel zu setzen.
Aus diesen Gründen gewinnt die Optimierung von Betriebsabläufen an Bedeutung und findet in nahezu jeder Branche ihre Anwendung.
Vielfach wird hierfür auf die Tourenplanung zurückgegriffen, die einen großen Teil solcher Optimierungsprobleme zu lösen vermag.

1.1. Ziel der Arbeit

Diese Bachelorarbeit ist Teil eines Gesamtprojektes, das sich mit einem Praxisbeispiel der Tourenplanung im Rahmen einer dynamischen Nachlieferung beschäftigt.
Die Aufgabe ist, für ein Logistikunternehmen, welches Zeitungen für ein Verlagshaus ausliefert, ein Konzept zur Bewältigung von Kundenreklamationen zu entwickeln. Dabei sollen Kunden, die ihre Zeitung fälschlicherweise nicht erhalten haben, spätestens eine halbe Stunde nach Eingang ihrer Beschwerde mit Ersatz versorgt werden, um so ihre Zufriedenheit mit dem Unternehmen zu erhalten.
Das Auftreten der Reklamationen an einem Tag ist zufallsabhängig, allerdings können anhand von historischen Daten durch Zeitreihenanalyse mögliche Orte und Zeitpunkte für das Auftreten einer Beschwerde ermittelt werden. Mit den so gewonnenen Informationen wird das gesamte Auslieferungsgebiet nach bestimmten Kriterien in einzelne kleinere Gebiete, die sogenannten Cluster, aufgeteilt. Für jedes Cluster ist ein Auslieferer zuständig, der seine Tour an einem zu bestimmenden optimalen Standpunkt, dem Zentraldeopt, beginnt. Innerhalb jedes Gebietes muss dann eine Route für den Fahrer gefunden werden, so dass der Weg kurz und somit die anfallenden Kosten für das Unternehmen möglichst gering gehalten werden aber trotzdem die Zeitvorgabe zur Bearbeitung der Reklamationen nicht überschritten wird.
Diese Arbeit beschäftigt sich mit dem letztgenannten Punkt des Gesamtprojekts, mit der Findung der optimalen Tour in jedem Cluster. Diese Problemstellung ist eine der bekanntesten der Optimierung und trägt den Namen *Travelling Salesman Problem with Timewindows*.

Es ergeben sich folgende Teilaufgaben im Rahmen dieser Arbeit:

- **Erstellung eines mathematischen Modells:**
  Der oben beschriebenen Problematik entsprechend soll ein mathematisches Modell formuliert werden, das alle notwendigen Bedingungen umfasst, so dass es als Basis der zu lösenden Nachlieferungssteuerung eingesetzt werden kann. Dabei wird ein Modell aus der allgemeinen Theorie der Tourenplanung auf den Spezialfall angepasst.

- **Entwicklung eines Verfahrens zur Lösung des Problems:**
  Aus den vielfältigen Lösungsverfahren für das Travelling Salesman Problem sollen geeignete Varianten ausgewählt, genau beschrieben und mathematisch als Algorithmus formuliert werden. Dabei soll eine übergeordnete Metaheuristik eingesetzt werden, die die einzelnen Lösungverfahren steuert.

- **Programmierung und Parameteranalyse:**
  Der entwickelte Lösungsansatz soll mit der Programmiersprache VBA implementiert und getestet werden. Dabei ist eine Analyse der in den Algorithmen auftretenden Parametern durchzuführen und eine Empfehlung für die geeignetste Wahl dieser Größen auszusprechen.

1.2. Aufbau der Arbeit

Zu Anfang erfolgt eine Einführung in die Tourenplanung, um das notwendige Basiswissen zu schaffen. Darauf aufbauend werden Problematik und Zielsetzung des allgemeinen Travelling Salesman Problem näher erläutert und kurz einige bekannte Varianten des allgemeinen Falls erwähnt. Auf das Travelling Salesman Problem mit Zeitfenstern wird daraufhin näher eingegangen. Es erfolgt die Modellierung eines mathematischen Modells und dessen genaue Interpretation sowie die Beschreibung der Dynamik in diesem Projekt und die damit einhergehende Verkomplizierung des Problems. Im nächsten Kapitel folgt die genaue Darstellung des Steuerungskonzeptes der Nachlieferung und dessen Komponenten.
Der eigentliche Optimierungsprozess bildet den Hauptteil der Arbeit. Hier wird ein Lösungsverfahren für das Travelling Salesman Problem erarbeitet. Dies besteht aus mehreren Komponenten, den Eröffnungs- und Verbesserungsverfahren, deren Ideen beschrieben und anschließend mathematisch als Algorithmen formuliert werden. Zur sinnvollen Steuerung der verschiedenen Verfahren wird die Tabu-Search Metaheuristik eingesetzt. Sie wird zunächst allgemein und dann auf das vorliegende Problem bezogen vorgestellt.
Im letzten Teil wird das Verfahren implementiert und mit verschiedenen Werten der vorhandenen Parametern getestet. Am Schluss erfolgt eine Empfehlung für die Parameterwahl.
Im Anhang befinden sich Tabellen mit den Ergebnissen der Parameteranalyse sowie der Programmcode zu den in VBA implementierten Eröffnungs- und Verbesserungsverfahren.

2. Einführung in die Tourenplanung

Bei der Tourenplanung geht es darum, durch konkrete Zuordnung jedes Kundens zu einer Tour, einen möglichst kostengünstigen Ablaufplan, den sogenannten *Tourenplan*, zu erstellen mit dem die Abholung oder Auslieferung bestimmter Güter geregelt werden kann. Dabei ist eine Tour gekennzeichnet durch eine genaue Reihenfolge, in der der Fahrer seine Kunden bedient.
Je nachdem, ob jede Kante (Straße) mindestens einmal besucht werden soll oder jeder Knoten (Kunde), unterscheidet man die *kanten-* von den *knotenorientierten* Planungsproblemen. Gibt es sehr viele nah beieinander liegende Knoten, so macht es Sinn diese als Kante zusammenzufassen. Ein Beispiel dafür ist die Müllentsorgung in einzelnen Straßen. Liegen dagegen eher große Entfernungen zwischen den Kunden, so werden diese als einzelne anzufahrende Punkte des Planes betrachtet. Diese Arbeit beschäftigt sich mit einem solchen knotenorientierten Problem.
Die Lösung eines Tourenplanungsproblems hat meist zwei Aspekte: Die *Clusterung* gibt an, welche Aufträge zu einer Tour zusammengefasst werden. Das *Routing* definiert, in welcher Reihenfolge diese Kunden innerhalb einer Tour bedient werden. Je nachdem welcher Aspekt zuerst ausgeführt wird, unterscheidet man zwischen 'Route-first-Cluster-second' und 'Cluster-first-Route-second'-Verfahren. Im ersten Fall wird zunächst eine Gesamttour erstellt, die alle Knoten umfasst und anschließend wird diese in einzelne kürzere Touren zerlegt (s. [9]). In diesem Gesamtprojekt wird das zweite Verfahren verwendet, da zunächst die Knoten zu sinnvollen Clustern zusammengefasst werden und dann innerhalb der Cluster die optimale Reihenfolge der Kunden gesucht wird.
Tourenplanungsprobleme gehören zu den $\mathcal{NP}$-schweren Problemen, das heißt, dass es keinen Algorithmus gibt, der diese Aufgabe in polynomialer Laufzeit löst. Oft ist schon das Auffinden einer Lösung, die überhaupt zulässig ist, nicht trivial.

Das wohl bekannteste Tourenplanungsproblem ist das Travelling-Salesman-Problem.

2.1. Das Travelling-Salesman-Problem

Das Travelling-Salesman-Problem (TSP) ist eines der meist untersuchten kombinatorischen Optimierungsprobleme. Trotz der einfachen Formulierbarkeit und Verständlichkeit zeichnet sich das Problem vor allem durch seine sehr schwere Lösbarkeit aus.
Anwendungsgebiete des TSP sind unter anderem die Bestimmung von optimalen Bedienreihenfolgen auf Maschinen, Optimierungen in der Verkehrsplanung und Logistik, auch beim Entwurf integrierter Schaltungen und Mikrochips ist dieser Aspekt von großer Bedeutung.

Das Grundproblem des Travelling Salesman besteht darin, dass ein Handlungsreisender n vorgegebene Städte innerhalb einer Rundreise (*Tour*) besuchen muss, wobei der dabei zurückgelegte Weg so kurz wie möglich sein soll.
Gesucht ist also eine *entferunungs- oder kostenminimale Reihenfolge* der Städte innerhalb der Tour, so dass der Reisende an einem Ursprungsort startet, alle Städte genau einmal besucht und am Schluss wieder zu dem Startunkt zurückkehrt. Jede Tour wird also am sogenannten *Depot* begonnen und auch beendet.
Die Entfernungen zwischen allen Paaren von Städten sind hierzu (z.B. in Form einer Tabelle) gegeben. Neben den tatsächlichen geometrischen Abständen kann es sich dabei auch um Reisezeiten oder um andere Kosten (z.B. für Treibstoff) handeln.

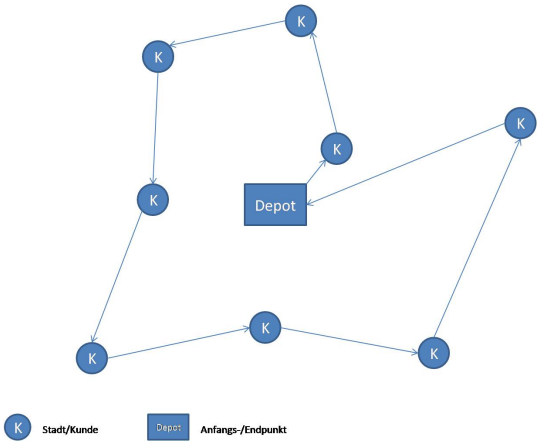


Abbildung 1: Beispiel einer Tour

Zur mathematischen Formulierung des symmetrischen[1] TSP betrachtet man den ungerichteten, bewerteten *Graphen* $G = (V, A, c)$, wobei $A = \{\{i, j\} : i, j \in V, j \neq i\}$ die Menge der ungerichteten *Kanten* zwischen Stadt i und Stadt j in G darstellt. Die Menge der Städte, allgemein auch *Knoten* genannt, werden durch die Menge $V = \{1, ..., m\}$ repräsentiert.
Das TSP ist dabei eindeutig durch die Anzahl m der Knoten bestimmt. Dazu wird eine $m \times m$-Matrix C, die *Distanzmatrix*, in folgender Weise definiert:
$C = (c_{ij}), \quad i, j \in V.$
C stellt also die Entfernungen bzw. die Kosten für das Zurücklegen des Weges zwischen je zwei Knoten der Menge V dar, wobei die Diagonaleinträge auf 0 gesetzt werden. Außerdem gilt bei dem symmetrischen TSP stets $c_{ij} = c_{ji}$.

Aus dem klassischen Travelling-Salesman-Problem hat sich eine nahezu unerschöpfliche Menge beliebig kombinierberarer Varianten mit Erweiterungen um verschiedene Restriktionen entwickelt. Die wesentlichsten Typen von Restriktionen sind Präzedenzen, auch Vorrangigkeitsbeziehungen genannt, eingeschränkte Kapazitäten oder Zeitfenster. Auf letztere soll nun näher eingegangen werden.

2.2. Das Travelling-Salesman-Problem mit Zeitfenstern

Das Travelling-Salesman-Problem with Timewindows (TSPTW) ist eine Erweiterung des klassischen TSP um zusätzliche zeitliche Restriktionen. Bei dieser Variante wird jedem zu besuchenden Kunden ein *Zeitfenester* zugewiesen, innerhalb dessen er von dem Handlungsreisenden bedient werden muss.
Diese Variante ist in der Praxis sehr häufig anzutreffen, gerade dann wenn der Auslieferer Zugang zu Lagerhallen oder eine Empfangsbestätigung für die gelieferte Ware benötigt. Dafür hat er sich zum Beispiel an die Öffnungszeiten der zu beliefernden Geschäfte zu halten.

Genau wie beim klassischen TSP geht man hier davon aus, dass der Handlungsreisende sich zu Beginn an einem ausgezeichneten Ort, dem *Startdepot* o befindet und auch zu diesem zurückkehren soll, wobei er dazwischen sämtliche Kunden genau einmal bedient. Hier sind die Kosten (bzw. Distanzen) wieder in Form der Distanzmatrix C gegeben. Hinzu kommen die Zeitfenster, wobei man zwei Arten von Fenstern unterscheidet: *Harte* Zeitfenster erlauben keine Überschreitung des vorgegebenen Zeitraums, führen also bei Nichteinhaltung zu Nachfrageverlust. Das heißt, bei zu spätem Ankommen kann der Kunde nicht mehr bedient werden. *Weiche* Zeitfenster dagegen er-

[1] Beim asymmetrischen TSP können Hin- und Rückweg zwischen zwei Knoten unterschiedlich lang sein, weshalb die Distanzmatrix nicht symmetrisch ist und deshalb $c_{ij} \neq c_{ji}$ ist.

lauben in Maßen eine Verletzung der gegebenen Zeit nach oben oder unten, allerdings wird dies mit Strafkosten geahndet. Durch übermäßige Erhöhung der Strafkostensätze werden ursprünglich weiche Zeitfenster hart.
Der Unterschied zum TSP wird dadurch deutlich, dass nicht jede beliebige Permutation der Städte eine zulässige Lösung des Problems darstellt, da durch die Zeitfenster eine gewisse Ordnung auf den Knoten impliziert wird. Durch diesen Aspekt ist das TSPTW deutlich schwerer zu lösen als das klassische TSP, schon das Auffinden einer überhaupt zulässigen Lösung ist ungleich komplizierter.

Das **Beispiel** ([5]) verdeutlicht, dass nicht jede beliebige Reihenfolge der Städte eine Lösung darstellen kann:
Gegeben ist die Tabelle mit dem Startdepot o sowie 5 Kunden mit den dazugehörigen Zeitfenstern.

Kunde i	o	1	2	3	4	5
früheste Zeit a_i	1	8	11	21	21	30
späteste Zeit b_i	50	22	16	28	40	56

An der Abbildung erkennt man, dass man gezwungen ist, den Kunden 3 vor den Kunden 4, 5 und 6 zu besuchen um die gegebenen Zeitfenster einhalten zu können.

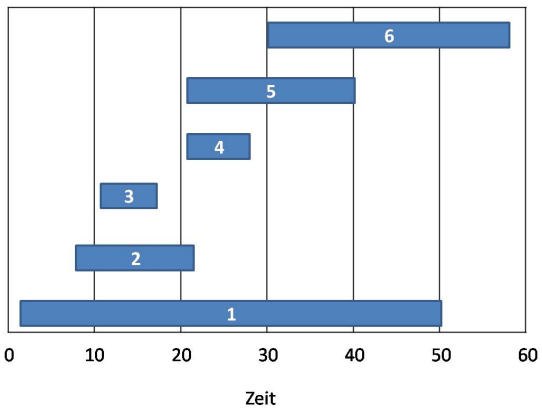


Abbildung 2: Zeitfenster

2.2.1. Mathematisches Modell TSPTW

Zur Formulierung des Modells wird zunächst vorausgesetzt, dass jeder Kunde i vom Startdepot aus erreichbar ist und genauso, dass es einen Weg von jedem Knoten zurück zum Startdepot gibt. Ist dies nicht der Fall, so existiert kein *Hamiltonkreis*[2] im Graphen und es kann keine Lösung für das Problem geben.

Dem Modell wird der gerichtete Digraph $G = (V, A, c)$ zugrunde gelegt. Eine Menge $N = \{1, ..., m\}$ von Kunden soll von einem Handlungsreisenden bedient werden, der seine Tour im Startdepot o beginnt und beendet. Daraus ergibt sich die Menge der Standorte zu $V := N \cup \{o\}$.
Zu jedem dieser Standorte $i \in N$ existiert ein Zeitfenster $[a_i, b_i]$, in dem der Kunde bedient werden muss. In dieser Arbeit soll ein Modell mit weichen Zeitfenstern betrachtet werden, also ist eine Verletzung des Zeitraumes zwar erlaubt, allerdings fallen dadurch zusätzliche Kosten an. Kommt der Handlungsreisende zu früh an einem Standort i an, so muss er dort bis zum Zeitpunkt a_i warten.

[2]Ein Hamiltonkreis ist ein Kreis, der alle Knoten des Graphen genau einmal enthält.

Diese *Wartezeit* wird mit w_i bezeichnet, und verursacht pro Zeiteinheit die Kosten c_i^-. Wird die obere Grenze um u_i überschritten, so kostet dies c_i^+ pro Zeiteinheit.
Die Fahrzeit zwischen zwei Standorten $i, j \in V$ beträgt t_{ij}, fällt eine zusätzliche Servicezeit (z.B. für das Entladen der Ware) bei dem Kunden i an, so ist sie durch s_i gegeben.
Die Menge A des Digraphen G beinhaltet alle zeitlich zulässigen Bögen (i, j), das heißt

$$A = \{(i, j) \in V \times V : a_i + t_{ij} \leq b_j, i, j \neq o\}.$$

Im Gegensatz zum TSP ist die Richtung der Pfeile hier also zu beachten, da es aufgrund der Zeitfenster ja gerade darauf ankommt, welcher Knoten zuerst besucht werden muss.
Zusätzlich werden folgende Variablen und Größen benötigt:
$x_{ij} \in \{0, 1\}$ ist eine Entscheidungsvariable, die besagt, ob der Reisende die Strecke (i, j) fährt oder nicht, $y_i \geq 0$ wird als Hilfsvariable eingesetzt und die Konstante M wird genügend groß gewählt.

Unter Einbezug obiger Größen folgt dann das **Modell TSPTW:**

$$\text{minimiere} \quad \sum_{(i,j)\in A} c_{ij}x_{ij} + \sum_{i\in N} c_i^- w_i + \sum_{i\in N} c_i^+ u_i$$

unter den Nebenbedingungen:

$$\begin{aligned}
&(1) && \sum_{j\in V} x_{ij} = 1 && \forall i \in N \\
&(2) && \sum_{j\in V} x_{oj} = 1 && \\
&(3) && \sum_{i\in V} x_{io} = 1 && \\
&(4) && \sum_{i\in V} x_{ih} - \sum_{j\in V} x_{hj} = 0 && \forall h \in N \\
&(5) && y_i + \mu_{ij} - M(1 - x_{ij}) \leq y_j && \forall i, j \in N \\
&(6) && \mu_{oj} x_{oj} \leq y_j && \forall j \in N \\
&(7) && a_i - w_i \leq y_i \leq b_i + u_i && \forall i \in N \\
&(8) && x_{ij} \in \{0, 1\} && \forall i, j \in N \\
&(9) && y_j \geq 0 && \forall j \in N
\end{aligned}$$

Das Modell ist folgendermaßen zu interpretieren:

Die Zielfunktion besteht aus drei Teilen. Pro durchlaufener Kante von i nach j fallen die Kosten c an. Werden diese mit x_{ij} multipliziert, so erhält man die variablen Fahrkosten, die natürlich nur anfallen, wenn der Reisende den Weg von i nach j durchfährt (in disem Fall ist x_{ij} gerade gleich 1).
Die zweite Summe bestraft das zu frühe Ankommen bei einem Knoten. Je länger die Zeit w_i ist, die der Fahrer warten muss, desto mehr Zeit geht verloren. Die Strafkosten wachsen also mit steigender Wartezeit an und erhöhen die Gesamtkosten. Gleiches gilt für den dritten Summanden. Hier wird die Überschreitungszeit u_i des Zeitfensterendes betrachtet. Je öfter die Zeitfenster also verletzt werden, desto stärker steigt der Zielfunktionswert an. Insgesamt wird durch die Minimierung ein kostenoptimaler Tourenplan gesucht.

Die erste Restriktion stellt sicher, dass der Fahrer von jedem Kunden über genau einen Pfeil wegfährt. Zusammen mit der Nebenbedingung (4), die veranlasst, dass jeder Knoten genau so oft angefahren wie verlassen wird, ist also festgelegt, dass alle $i \in N$ *genau einmal* besucht werden.
Die Bedingungen (2) und (3) binden auch das Startdepot ein, indem sichergestellt wird, dass es über genau einen Bogen verlassen und über einen anderen am Ende der Tour wieder angefahren wird.
Restriktion (5) soll sogenannte *Subzyklen* verhindern. Diese entstehen, wenn eine Tour existiert, die nicht alle Knoten umfasst und vor allem das Startdepot nicht enthält. Die y_i kann man als die bis einschließlich Knoten i aufsummierten μ_{ij} verstehen, die wiederum die den Pfeilen (i, j) zugeordneten Entfernungen c_{ij}, die Fahrzeiten t_{ij} oder auch die Bedarfe am Endknoten darstellen können [1]. Hier ist eine Interpretation der y_i als der jeweilige Beginn der Servicezeit beim Knoten i sinnvoll. x_{ij} ist gleich 1, falls der Bogen (i,j) verwendet wird. In diesem Fall fällt der letzte Teil der linken Seite weg und die Ungleichung ist genau dann erfüllt, wenn die Tour im Depot beginnt und auch endet [9]. Ist dies nicht der Fall, gilt also $x_{ij} = 0$, so ist die Restriktion ebenfalls erfüllt (siehe Beweis von Satz 1). Die Ungleichung (6) ist nötig, um auch das Startdepot einzuschließen.
Interpretiert man y_i als den jeweiligen Beginn des Services beim Kunden i, so legt Nebenbedingung (7) fest, dass dieser Beginn in dem um Warte- und Verspätungszeiten geöffneten Fenster $[a_i - w_i, b_i + u_i]$ liegen muss.

Dass dieses Modell das TSPTW richtig beschreibt, zeigt der folgende Satz [1].

Satz 1: Vorgegeben sei das Modell TSPTW. Es sei M größer gleich der Summe aller $\mu_{ij} \geq 0$, $i, j \in N$. Dann gilt:

1. Sei $(\bar{x}, \bar{y})$ eine zulässige Lösung des TSPTW. Dann ist $\bar{x}$ ein geeigneter Tourenplan.
2. Sei $\bar{x}$ ein geeigneter Tourenplan. Dann gibt es einen Vektor $\bar{y} \geq 0$ so, dass $(\bar{x}, \bar{y})$ eine zulässige Lösung vom TSPTW ist.

Beweis
Ad 1: Es sei $(\bar{x}, \bar{y})$ zulässig zum TSPTW und T der Träger von $\bar{x}_{ij}$.
Aufgrund der ersten 4 Restriktionen ist klar, dass in jeden Knoten je genau ein Pfeil aus T hinein und herausgeht, allerdings kann T aus einer disjunkten Vereinigung geschlossener Untertouren, die nicht alle Knoten umfassen, bestehen.
Sei nun T^1 die Untertour von T, die durch das Depot o geht. Dann gilt entweder $T = T^1$, d.h. T^1 umfasst alle Knoten oder es gibt eine zweite Untertour T^2 außerhalb von T. Gäbe es eine zweite Untertour mit der Spur $(i_1, ..., i_s)$, so müsste diese einen Zyklus bilden. Aus Restriktion (5) würde dann folgen:

$$\bar{y}_{i_1} - \bar{y}_{i_2} + \mu_{i_1 i_2} \leq M(1 - \bar{x}_{i_1 i_2}) = 0$$
$$\bar{y}_{i_2} - \bar{y}_{i_3} + \mu_{i_2 i_3} \leq M(1 - \bar{x}_{i_2 i_3}) = 0$$
$$...$$
$$\bar{y}_{i_s} - \bar{y}_{i_1} + \mu_{i_s i_1} \leq M(1 - \bar{x}_{i_s i_1}) = 0$$

Addiert man diese Ungleichungen, so ergibt sich

$$0 < \mu_{i_1 i_2} + \mu_{i_1 i_2} + ... + \mu_{i_s i_1} \leq 0 \qquad \rightarrow \text{Widerspruch!}$$

Das heißt, es kann keine zweite Untertour geben und somit beschreibt T eine Tour, die alle Knoten umfasst!
Aus Bedingung (1) folgt dann die Behauptung.

Ad 2: Sei nun $(\bar{x}, \bar{y})$ eine geeignete Tour. Für beliebiges $i \in N$ definiert man ein y_i wie folgt:

$$y_i = \sum_{j=0}^{j=i-1} \mu_{i\gamma(j)}$$

wobei $\gamma(j)$ den eindeutigen Nachfolger des Knotens j bezeichnet.
Zu zeigen bleibt, dass $(\bar{x}, \bar{y})$ sämtliche Restriktionen erfüllt.
(1) bis (4), (7) sind offensichtlich erfüllt.

Fall 1: $x_{ij} = 1$, die Knoten i und j folgen in der Tour unmittelbar aufeinander.
Dann gilt

$$\underbrace{y_i + \mu_{ij}}_{=y_j} - \underbrace{M(1 - x_{ij})}_{=0} = y_j \qquad \Rightarrow \text{Bedingung (5)}$$

$$\mu_{oj} x_{oj} = y_j \cdot 1 = y_j \qquad \Rightarrow \text{Bedingung (6)}$$

Fall 2: $x_{ij} = 0$, die Knoten i und j folgen nicht unmittelbar nacheinander.
Bedinung (6) ist dann automatisch erfüllt.
Liegt nun i vor j, so gilt per Definition $y_i \leq y_j$, also gilt wie in Fall 1

$$y_i + \mu_{ij} - M(1 - x_{ij}) = y_i + \mu_{ij} - M < y_j \qquad \Rightarrow \text{Bedingung (5)}$$

Liegt aber i hinter j, so gilt $y_i - y_j = \alpha$ mit $\alpha > 0$. Also gilt wegen der Wahl von M

$$y_i - y_j + \mu_{ij} - M(1 - x_{ij}) < \alpha + \mu_{ij} - M \leq 0 \qquad \Rightarrow \text{Bedingung (5)}$$

q.e.d.

2.3. Das dynamische TSPTW

Neben der Unterscheidung der verschiedenen Varianten des klassischen TSP nach Restriktionstypen gibt es noch eine andere wichtige Klassifizierung von Tourenplanungsproblemen, also speziell des TSPTW. Je nachdem, welche Informationen wann bekannt sind unterscheidet Larsen [11]

- *Statische Probleme*
  Hierbei wird angenommen, dass alle für die Planung benötigten Informationen dem Planer vor Beginn des Planungsprozesses bekannt sind und dass sich diese relevanten Informationen auch nicht mehr ändern, nachdem der Tourenplan erstellt wurde.

- *Dynamische Probleme*
  Nicht alle für den Planer benötigten Informationen sind zu Beginn des Planungsprozesses bekannt. Außerdem können sich diese Informationen, nachdem der Tourenplan erstellt wurde, wieder ändern.

Der wesentliche Unterschied zeigt sich also darin, ob der Tourenplan während seiner Ausführung geändert werden muss oder nicht. Beim statischen TSPTW reicht es aus, am Anfang - aufbauend auf den bekannten Informationen - einmal einen optimalen Plan zu generieren. Dieser bleibt nach seiner Erstellung dauerhaft gültig, da sich die grundlegende Situation nicht verändert. Dies hat den Vorteil, dass zur Berechnung des Tourenplans auch sehr komplexe Verfahren mit hohem Aufwand genutzt werden können, da man nicht auf zeitliche Änderungen achten muss. Häufig sind solche statischen Probleme aber nicht für die Praxis geeignet, da sie wenig flexibel sind. Auch in dem hier zu bearbeitenden Problem der zufällig auftretenden Reklamationen ist eine statische Planung nicht möglich.

Bei einem dynamischen Planungsproblem verändert sich die gegebene Situation laufend, was ein ständiges Neuplanen und Anpassen der vorhanden Tour bedeutet. Nach dem Erstellen eines anfänglichen Plans muss ein Steuerungsprozess einsetzen, der dafür sorgt, dass die neuen Informationen zeitnah und korrekt in einen neuen Tourenplan eingefügt werden können. Auf diese Weise hat man die Möglichkeit, jederzeit flexibel auf die sich verändernde Problemstellung zu reagieren. Allerdings können hier aufgrund des Zeitdrucks keine allzu aufwendigen Lösungsverfahren herangezogen werden, weshalb man eventuell mit weniger guten Lösungen arbeiten muss.

3. Grundstruktur der Nachlieferungssteuerung

Die Bearbeitung der Reklamationen soll jeweils in einem Zeitfenster von 30 Minuten geschehen, das zur Aufrechterhaltung der Kundenzufriedenheit so gut wie möglich eingehalten werden sollte. In dem Modell wird von weichen Zeitfenstern ausgegangen, die Grenzen dürfen also generell kostenpflichtig überschritten werden. Für dieses Projekt ist es nun nicht sinnvoll, auch zu frühes Ankommen bei einem Kunden zu bestrafen, da die Kunden nicht unzufrieden sein werden, wenn sie ihre Reklamation früher als gedacht erhalten. Deshalb werden für diesen Fall keine Konventionalkosten angesetzt. Die Wartezeit, die die Tourdauer unnötig erhöht, wird in den Verfahren berücksichtig um sie möglichst gering zu halten. Eine Verspätung darf sich der Fahrer allerdings nicht erlauben, um den Kunden nicht zu verärgern. Deshalb werden hier sehr hohe Strafkosten angesetzt um die weiche Zeitobergrenze in eine harte umzuwandeln. Zu Anfang werden die Kosten auf 100€ pro Minute gesetzt, durch die Parameteranalyse anhand des VBA-Programmes wird dieser Betrag getestet und eventuell angepasst.
Außerdem werden hier keine Servicezeiten angesetzt, da die Zeitungen lediglich im Briefkasten deponiert werden müssen, was keine größere Zeitspanne in Anspruch nimmt.

Im Vorfeld dieser Arbeit werden in einem anderen Teil des Gesamtprojekts mit Hilfe von Erfahrungwerten, die in der Vergangenheit gemacht wurden, die *Dummykunden* DK bestimmt. Dafür wird das gesamte Auslieferungsgebiet in gleichgroße Sektoren aufgeteilt. Dummykunden sind Kunden, die noch nicht reklamiert haben, aber für die aufgrund der bisherigen Erfahrungen eine hohe Wahrscheinlichkeit besteht, dass diese am Ort ij und Zeitpunkt z in einem der Sektoren auftreten [9]. Sie werden benötigt, um überhaupt einen Plan erstellen zu können, da ein großteil der Informationen aufgrund der Dynamik nicht bekannt sind. Nachdem dann im nächsten Teil des Projektes die Cluster gebildet wurden, in denen jeweils ein Lieferant die Zeitungen austeilt, reduziert sich das Problem der optimalen Routenfindung innerhalb der Cluster auf ein Travelling Salesman Problem mit Zeitfenstern.
Bei der Bearbeitung des Problems sind jeweils *Entscheidungen in Echtzeit* zu treffen, um den Austeilungsprozess des Lieferanten nicht zu unterbrechen. Während der Plan noch ausgeführt wird, wird er gleichzeitig an die neuen Daten angepasst und optimiert, um dann sofort ausgeführt werden zu können. Dies wird folgendermaßen erreicht:
Begonnen wird der Arbeitstag mit der ersten Erstellung eines Tourenplans für jedes Cluster. Zu diesem Zeitpunkt sind sowohl die ermittelten Dummykunden, also auch eventuell bereits die ersten realen Rekalmationen bekannt. All diese Informationen fließen mit in die Findung der optimalen Tour ein. Sobald diese feststeht, erhält der Lieferant die Information, zu welchem Kunden er fahren soll; hat er Diesen bedient erhält er die nächste Anweisung. Da es sich bei dem vorliegenden Problem um ein dynamisches TSPTW handelt, ist die Optmierung nicht mit der einmaligen Erstellung eines Planes beendet. Ständig kommen neue Reklamationen hinzu, die dann mit in die Tour einzuarbeiten sind.

3.1. Die Taktzeit

Um diesen Anforderungen gerecht zu werden, muss permanent optimiert und gleichzeitig der aktuelle Plan ausgeführt werden. Dafür wird die sogenannte *Taktzeit* t angesetzt. Die Taktzeit ist die Zeitspanne, die das Optimierungsmodul Zeit hat, die neuen optimalen Tourenpläne zu erstellen und in der gleichzeitig Informationen über neu auftretende Beschwerden angesammelt werden. Während dieser Optimierungszeit führt der Lieferanten in ihrem jeweiligen Cluster weiterhin den aktuellen Plan aus. Nach den t Minuten wird dann der neue Plan an den Fahrer weitergeleitet, damit er weiß, welche Änderungen vorgenommen wurden und welche Kunden er deshalb als nächste besuchen muss. Die Wahl einer geeigneten Taktzeit ist dabei von großer Bedeutung. Reklamierende Kunden werden nicht sofort bei Anruf in den Plan mit eingebunden, sondern können erst nach Ablauf der Taktzeit in den nächsten Optimierungsprozess mit einbezogen werden. Um nicht zu viele Minuten vom Zeitfenster der Kunden verstreichen zu lassen, sollte die Taktzeit nicht zu groß gewählt werden. Andererseits bedeutet eine sehr kurze Taktzeit, dass nur wenig Zeit zur Verfügung steht, einen optimalen Tourenplan zu finden. Je kürzer die Zeit angesetzt wird, desto schlechter fallen die Lösungen gegebenenfalls aus. Zunächst wird die Taktlänge auf 3 Minuten gesetzt, allerdings ist dies die Zeit, in der die Pläne für alle Cluster erstellt werden. Diese werden nacheinandern abgehandelt, so dass für jedes Cluster eine Taktzeit von $\frac{3}{p}$ Minuten bleibt, wenn p die Anzahl der Cluster angibt. Das heißt, dass das Programm in jedem Cluster $\frac{3}{p}$ Minuten lang optimiert, dann einen neuen Tourenplan ausgibt, Informationen über neue Reklamationen einliest und sofort mit der Erstellung des nächsten Planes beginnt. In wieweit dies ein sinnvoller Wert für t ist, wird später anhand des VBA-Programmes analysiert.
Bei jedem erneuten Durchlaufen des Optimierungsprozesses wird von einem *Warmstart* ausgegangen. Das Programm startet also mit dem aktuell ausgeführten Tourenplan als Ausgangslösung und verändert diesen den neuen Informationen entsprechend, es muss nicht jedes mal eine komplett neue Tour erstellt werden.

3.2. Der Tourenplanpool

Um eine gewisse Auswahl an Tourenplänen zu erhalten, wird mit einem *Tourenplanpool* gearbeitet. Dies ist ein Speicher, in dem pro Durchlauf die s besten Pläne gespeichert werden, die das Lösungsverfahren generiert.
Dafür wird im ersten Durchlauf des Insertion-Verfahrens[3] die beste Lösung bezüglich des Zielfunktionswertes s-mal abgespeichert. Auf alle diese Lösungen wird Tabu-Search angewandt; da die Verbesserungsverfahren zufällig ausgewählt werden, entstehen während dem Verlauf s unterschiedliche Lösungen, die parallel betrachtet und optimiert werden. Am Ende jeder Taktzeit wird der Plan mit den aktuell niedrigsten Kosten ausgeführt.
Alle im Tourenplanpool gespeicherten Touren müssen nach jeder Taktzeit an die aktuelle Situation der neu hinzugekommenen Reklamationen angepasst und aktualisiert werden, damit diese immer auf dem neuesten Stand sind. Es müssen bereits abgearbeitete Kunden gelöscht werden, alle noch nicht bedienten Kunden verbleiben in der Tour und neue Reklamationen werden mit Hilfe des Insertion-Verfahrens in die Tour eingefügt.
Es sein kann, dass nach einer Neuoptimierung nicht der Plan aus dem Tourenplanpool gewählt wird, der auf dem im vorherigen Takt ausgeführten Plan beruht, sondern ein auf einer anderen Tour basierender Plan, falls dieser einen besseren Zielfunktionswert aufweist, als die Übrigen. Vor der Optimierung muss daher berechnet werden, bis zu welchem Kunden der Fahrer mit dem aktuellen Plan in der Taktzeit t kommt, damit dieser dann als Startknoten für alle Touren des Pools eingesetzt und so die Tour nach der Aktualisierung reibungslos fortgesetzt werden kann. Diese Berechnung ist aufgrund der gegebenen Abstände zwischen den Kunden sowie den Geschwindigkeiten

[3] siehe Kapitel 4.2.3.1

der Fahrzeuge leicht durchzuführen. Sollte der Fahrer sich nach Ablauf der Taktzeit nicht genau bei einem Kunden, sondern auf dem Weg zum nächsten Kunden hin befinden, so wird dieser nächste Kunde als Startpunkt eingefügt.
Würde man auf diese Weise alle Lösungen auf jeder Stufe betrachten und auf diese alle möglichen Änderungen anwenden, so würde das Vorgehen sicher zu dem globalen Optimum führen. Da dies aber natürlich aufgrund der Vielzahl der möglichen Lösungen sehr aufwendig wäre, wird hier nur ein Teil der Lösungen gespeichert was aber gegenüber nur einer betrachteten Lösung, schon eine deutlich höhere Chance bietet tatsächlich das globale Optimum zu finden.

3.3. Behandlung der Dummykunden

Die Dummykunden spielen eine wichtige Rolle im Bezug auf die Dynamik des Nachlierferungsproblems. Sie sollen mit in den Optimierungsprozess eingebunden werden, um auf das Auftreten einer Reklamation in der näheren Umgebung vorbereitet zu sein. Auf diese Weise kann der Fahrer sich vorsorglich in die angegebene Richtung bewegen und so dort auftretende Kunden schnell und effizient in die Tour einbeziehen.
Zu Beginn der Taktzeit, vor der Erstellung des nächsten Tourenplans, wird überprüft, wo tatsächlich Reklamationen aufgetreten sind. Dabei wird für jede reale Beschwerde ein Dummy aus demselben Sektor gelöscht; die Vorhersage ist dann eingetroffen und der reale Kunde wird statt des Dummys bedient. Dummykunden, für die (noch) keine realen Reklamationen aufgetreten sind, verbleiben in der Tour und werden in die Optimierung mit einbezogen. Da die Dummykunden also oftmals keine realen Kunden sind, muss definiert werden, was geschieht, wenn ein solcher laut dem aktuellen Tourenplan als nächster bedient werden soll.
Es können zwei Möglichkeiten auftreten. Entweder, der Dummykunde ist zufällig auch ein realer Kunde, dann wird er wie ein solcher angefahren und bedient. Sollte der Dummykunde dagegen kein realer Kunde sein und auch kein realer Kunde in dem gleichen Sektor auftreten, so hat sich das Forecasting nicht erfüllt. Muss laut Tourenplan nun als nächstes ein Dummy angefahren werden, so wird dieser gelöscht und stattdessen der nächste tatsächliche Kunde, den der Plan vorsieht, dem Fahrer übermittelt. Dabei muss berücksichtigt werden, ob durch die Zeitersparnis, die das Überspringen des Dummies bewirkt, ein Kunde mehr als geplant in der Taktzeit bedient werden kann. Dies wird folgendermaßen umgesetzt: Nach der Überprüfung, welche Knoten des besten Plans innerhalb der nächsten Taktzeit bedient werden können, werden aus dieser Menge sämtliche Dummykunden gelöscht. Dadurch können nun andere Knoten in diese Zeitspanne nachrücken und abgearbeitet werden. Auch aus dieser Menge werden wieder die Dummies entfernt; dies wird fortgesetzt, bis die Knoten der Taktzeit nur noch aus realen Kunden bestehen.

4. Der Optimierungsprozess

In diesem Kapitel wird das gesamte Optimierungsmodul genau betrachtet.
Die Aufgabe des Optimierungsprozesses ist es, aufgrund der aktuellen Situation von Dummy- und realen Kunden den optimalen Tourenplan für den nächsten Takt zu generieren.
Zur Lösung des TSPTW gibt es wesentlich weniger Lösungsverfahren als für das allgemeinere TSP, da die Zeitfenster eine erhebliche Erschwernis für die Tourensuche darstellen. Die Ansätze der existierenden Verfahren bauen auf den entsprechenden TSP-Ansätzen auf und erweitern diese. Deshalb gibt es für das TSPTW sowohl exakte Algorithmen als auch Heuristiken.

Die *exakten Lösungsverfahren* finden das optimale Ergebnis einer Problemstellung oder stellen fest, dass es kein solches gibt. Sie basieren meist auf der vollständigen Enumeration aller möglichen Lösungen. Allerdings steigt die Anzahl dieser Lösungen mit der Anzahl der Kunden sowie der Zeitfensterlänge immer weiter an, was klare Grenzen für die Anwendbarkeit dieser Verfahren in der Praxis setzt. Aktuell exakt lösbare Probleme umfassen nur wenige 100 Kunden [2]. Aufgrund dessen wird hier auf eine heuristische Lösung zurückgegriffen.

4.1. Heuristik und Metaheuristik

Die *heuristischen Verfahren* haben nicht den Anspruch eine exakte Lösung zu finden, sondern suchen nach einer annähernd optimalen Tour. Deshalb wird dieser Verfahrenstyp oft auch als *Approximationsverfahren* bezeichnet. Eine Definition könnte folgendermaßen aussehen:

Definition 1: Heuristik
Heuristiken sind Algorithmen, die ein gegebenes Optimierungsproblem mit einem akzeptablen Aufwand möglichst gut zu lösen versuchen [4].

Eine „gute Lösung“ bedeutet, dass sie eine möglichst geringe Abweichung vom tatsächlichen Optimum aufweisen sollte. Eine solche Ausgestaltung reicht für das TSPTW völlig aus, da es aufgrund der Dynamik wichtiger ist, schnell eine gute Lösung zu generieren als lange nach einer exakten zu suchen.
Im Gegensatz zu exakten Verfahren, die dann terminieren, wenn eine optimale Lösung gefunden wurde, brauchen Heuristiken spezielle *Stopp-Regeln*. Diese hängen ganz von der Ausgestaltung der verwendeten Heuristik ab.
Desweiteren unterscheidet man zwischen Eröffnungs- und Verbesserungsverfahren. Die *Eröffnungsverfahren* suchen nach einer (guten) zulässigen Lösung und terminieren automatisch, sobald eine solche gefunden wurde. *Verbesserungsverfahren* setzen danach an und versuchen zum Beispiel innerhalb einer vorgegebenen Anzahl an Iterationen eine zulässige Lösung weiter zu verbessern.
Die Gefahr bei Approximationsverfahren besteht darin, sich in einem lokalen Optimum festzufahren. Deshalb wurden in den sechziger Jahren die sogenannten *Metaheuristiken* eingeführt.

Definition 2: Metaheuristik
Eine Metaheuristik ist ein übergeordneter Algorithmus, der die Lösungssuche eines oder mehrerer abhängiger Algorithmen steuert. Sie beruht auf einer Sammlung von Strategien, die unabhängig vom zugrundeliegenden Problem und den abhängig gesteuerten Algorithmen sind [4]

Eine Metaheuristik ist also ein Algorithmus, der die Lösungssuche einer oder mehrerer Heuristiken steuert. Im Gegensatz zu problemspezifischen Heuristiken, die nur auf ein bestimmtes Optimierungsproblem angewendet werden können, definieren Metaheuristiken eine abstrakte Folge von Schritten, die theoretisch auf beliebige Problemstellungen angewandt werden können.

Die meisten Metaheuristiken lassen es zu, dass sich die Lösungen während der Suche im Bezug auf den Zielfunktionswert verschlechtern. Auf diese Weise ist die Möglichkeit gegeben lokale Optima wieder zu verlassen und dadurch eine wesentlich näher am globalen Optimum gelegene Lösung zu finden. Ob das globale Optimum erreicht wurde, kann allerdings zu keinem Zeitpunkt festgestellt werden.
Bei der Anwendung eines solchen Verfahrens beobachtet man anfangs meist eine starke Verbesserung der Ausgangslösung, bis das erste lokale Optimum aufgefunden wurde. Ändert sich die zu optimierende Eigenschaft der betrachteten Lösungen über eine gewisse Anzahl von Iterationen nicht, so setzt eine Verschlechterungsphase ein, um in einen anderen Bereich des Lösungsraumes vorrücken zu können.
Mittlerweile gibt es eine große Auswahl an Metaheuristiken. Zu den Bekanntesten zählen die generischen Algorithmen sowie Simulated Annealing und Tabu-Search. *Generische Verfahren* generieren Lösungsvorschläge, die dann verändert und miteinander kombiniert und einer Auslese unterzogen werden, so dass diese Lösungsvorschläge den gestellten Anforderungen immer besser entsprechen. Beim *Simulated Annealing* wird die Verschlechterung einer neuen Lösung nur mit einer bestimmten Wahrscheinlichkeit akzeptiert.
In diesem Optimierungsmodul wird das Verfahren des *Tabu-Search* verwendet und im nächsten Abschnitt näher erläutert.

4.2. Tabu-Search Metaheuristik

Der Algorithmus Tabu-Search zur Steuerung der Suche beim Verlassen von lokalen Optima wurde 1987 von Fred Glover veröffentlicht und seither ständig weiterentwickelt, so dass heute eine Vielzahl ganz verschiedener Varianten dieser Metaheuristik existieren.
Die Grundidee dieses Verfahrens ist es, durch die Führung einer *Tabuliste* die Bildung von Zyklen in der Suche zu vermeiden, indem sie sich bereits angelaufene Lösungen merkt und diese für eine gewisse Zeit verbietet.
Es wird also verhindert, dass der Algorithmus, nachdem er sich einige Schritte durch Verschlechterung vom lokalen Optimum entfernt hat, durch Ausführung der inversen Schritte, die aus lokaler Sicht wieder eine Verbesserung versprechen, in das gleiche lokale Optimum zurückbewegt. Hierbei wird nicht nur die akutelle Lösung betrachtet, sondern auch der bisherige Verlauf der Suche.
Die Tabuliste wird auch als *attributiver Speicher* bezeichnet, da hierin nur bestimmte Eigenschaften (Attribute) der vorangegangenen Lösungen oder des Übergangs von einer Lösung zur Nachbarlösung gespeichert werden.
Einen solchen Übergang erhält man durch eine sogenannte *Nachbarschaftsoperation*. Dies ist eine Operation, die aus einer Initiallösung neue Lösungen, die Nachbarschaft $\mathcal{N}(x)$, generiert. So ein Übergang kann zum Beispiel ein Or-Opt-Schritt sein, wie er in Kapitel 5.2.2 dargestellt wird.
Ein häufig verwendetes Attribut ist der Zielfunktionswert. Es sollen also keine kompletten Lösungen abgespeichert werden, sondern nur festgelegte Teile davon, wodurch man den benötigten Speicherplatz relativ klein halten kann.
Glover selbst beschrieb sein Verfahren folgendermaßen:

> *„The allover approach is to avoid entrainment in cycles by forbidding or penalizing moves which take the solution, in the next iteration, to points in the solution space previously visited[3].“*

Gelangt der Algorithmus an den Punkt, in dem eine neue Nachbarschaft $\mathcal{N}(x)$ erstellt wurde, wird die bezüglich des Zielfunktionswertes beste Lösung gewählt. Vor dem Wechsel muss allerdings in der Tabuliste nachgesehen werden, ob die Lösung Eigenschaften aufweist, die darauf hindeuten, dass sie schon einmal besucht wurde. Ist das der Fall, so ist dieser Übergang *tabu*.

4.2.1. Das klassische Tabu-Search

Die einfachste Form der Tabu-Search Metaheuristik arbeitet mit einem Kurzzeitgedächtnis, also einer Tabuliste in der die Eigenschaften der letzten k Lösungen gespeichert sind. Diese Liste hat dann die Form einer Warteschlange der Länge k, die nach dem FIFO-Prinzip gehandhabt wird. Die Eigenschaften werden in der Reihenfolge ihres Auftretens in der Liste abgelegt. Ist diese voll, so wird die sich bereits am längsten im Speicher befindende Lösung geschlöscht. Das Verfahren „vergisst“ diese dann, sie wird im weitern Fortschritt des Verfahrens nicht berücksichtig.
Über die Länge der Tabuliste kann man den Verlauf von Tabu-Search maßgeblich beeinflussen. Wird k zu groß gewählt, so wird die Anzahl der Nachbarlösungen sehr stark eingeschränkt, was in eine Sackgasse führen kann. Die Größe des Speichers entscheidet auch über die Länge der Zyklen, die verhindert werden. Bei kleinem k muss also bedacht werden, dass nur Zyklen der Länge k verhindert werden können. Eine angemessene Länge für die Tabuliste soll später anhand des VBA-Programmes experimentell ermittelt werden.
Es kann im Laufe des Verfahrens vorkommen, dass aufgrund der Länge der Tabuliste und der gewählten Attribute keine Nachbarlösung existiert, die nicht tabu gesetzt wurde. Damit der Algorithmus an dieser Stelle nicht abbricht, gibt es das sogenannte *Aspirationskriterium*. Dabei handelt es sich um eine Bedingung, die den Tabu-Status bestimmter Lösungen aufhebt. Das Kriterium greift nicht nur in Ermangelung neuer Lösungen, sondern kann auch bei anderen, vorher definierten Situationen in Kraft treten, zum Beispiel falls der Zielfunktsonwert einer Tabu-Lösung unerwartet besser ist als der der bisher besten Lösung.
Daraus folgt:

Algorithmus 1: Klassisches Tabu-Search

1. **(Initialisierung)** Bestimme eine zulässige Initiallösung x des Problems. Setze den Iterationszähler $t := 0$. Setzte $x^* = x$.
2. **(Schleife)** Wiederhole, bis ein Stopp-Kriterium erfüllt ist:
    3. **(Nachbarschaft)** Setze $t = t + 1$ und bilde die Nachbarschaft $\mathcal{N}(x^*, t)$.
    4. **(Bestensuche)** Durchsuche die gesamte Nachbarschaft und wähle das bezüglich einer bestimmten Eigenschaft beste $x^{'} \in \mathcal{N}(x^*, t)$, das nicht tabu ist oder das Aspirationskriterium erfüllt.
    5. **(Iteration)** Setze $x^* = x^{'}$ und $t = t + 1$.
    6. **(Aktualisierung)** Akutalisiere die Tabuliste und das Aspirationskriterium.
    Gehe zu 2.

Diese Form des Tabu-Search ist noch sehr allgemein gehalten und somit auf ganz verschiedene Optimierungsprobleme anwendbar.
Nun soll der allgemeine Ansatz noch etwas erweitert und anschließend auf den im Projekt vorliegenden Fall präzisiert werden.

4.2.2. Generisches Tabu-Search

Die klassische Variante des Tabu-Search wird häufig um zwei Komponenten erweitert, die eine noch bessere Steuerung der Suche nach dem globalen Optimum zulassen: Intensivierung und Diversifikation.
Unter *Intensivierung* versteht man die besonders aufwendige und genaue Durchsuchung eines vielversprechenden Bereichs des Lösungsraumes. Dies kann geschehen, indem zum Beispiel in einem

Langzeitgedächtnis abgespeichert wird, ob gute Lösungen häufig ein bestimmtes Attribut aufweisen, das dann für den weiteren Verlauf des Verfahrens fixiert wird.
Den gegenteiligen Effekt erhält man durch *Diversifikation*. Sie dient dazu, den Lösungsraum weiträumig abzusuchen, um in andere, bislang nicht untersuchte Bereiche zu gelangen. Auch hierbei kann mit einem Langzeitgedächtnis gearbeitet werden; häufig auftretende Eigenschaften werden bewusst abgeändert um sich von der aktuellen Lösung zu entfernen.
Mit diesen beiden Steuerungsmöglichkeiten wird also das Verbleiben in einem lokalen Optimum verhindert. Nach anfänglicher Intensivierung der Initiallösung wird, wenn nach einer gewissen Anzahl von Iterationen keine Verbesserung der Lösung mehr stattfindet, in den Diversifikationsmodus gewechselt. Ist dann in einem anderen Bereich des Lösungsraumes eine gute Lösung gefunden worden, wird erneut zur Intensivierung gewechselt.
Dieses Vorgehen bei der Suche nach dem globalen Optimum wird in Abbildung 3 veranschaulicht.
Die Verwendung eines Langzeitgedächtnisses ist ein deterministischer Ansatz zur Steuerung von Intensivierung und Diversifikation, allerdings hat es sich in der Praxis als nützlich erwiesen, diese Koordination stochastisch zu gestalten. Eine solche Umsetzung soll hier gewählt werden, indem im Punkt 3 des Algorithmus die Nachbarschaft $\mathcal{N}(x)$ durch einen zufällig ausgewählten Operator erstellt wird.

Definition 3: Nachbarschaft [4]
Es sei P der Zulässigkeitsbereich der Lösungen des TSPTW. Eine *Nachbarschaft* $\mathcal{N}(x)$ einer Lösung $x \in P$ ist durch die Abbildung $\mathcal{N} : P \to \mathcal{P}(P), \quad x \mapsto \mathcal{N}(x) \subset P$ gegeben.
Die Nachbarschaft einer Lösung x besteht also aus allen zulässigen Lösungen, die in der „Nähe“ von x liegen.

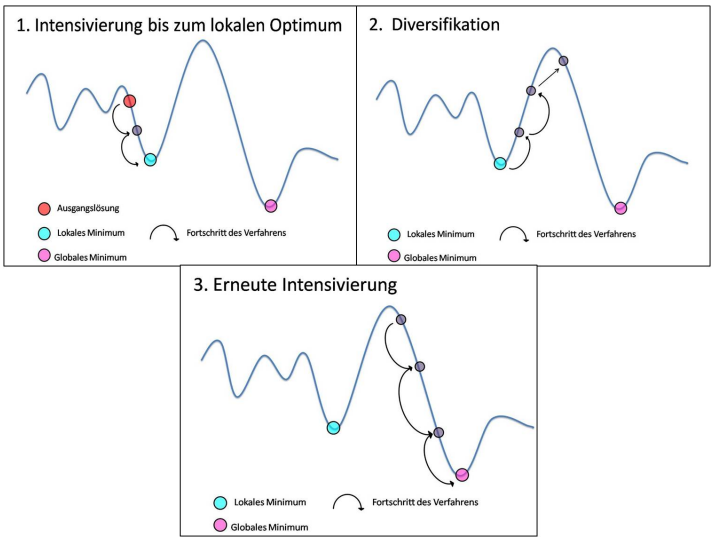


Abbildung 3: Möglicher Verlauf Tabu-Search am Bsp. einer Minimierungsaufgabe

Lösungen „in der Nähe“ von x werden durch die sogenannte Nachbarschaftsoperationen, auch *Move* genannt, erstellt. Wie genau diese aussieht hängt von dem verwendeten Verfahren ab. Eine mögliche Operation beim TSPTW wäre das Austauschen von zwei beliebigen Kanten der Tour, wobei sich die Nachbarschaft als Vereinigung aller so entstanden zulässigen Touren ergibt.

In dieser Arbeit wird eine Menge von vier Nachbarschaftsoperatoren betrachtet. Dabei handelt es sich um die Verbesserungsverfahren[4] Shift Inside Tour (SIT), Interchange (IC), Or-Opt (OR) und Transferred Sequence (TS). Aus dieser Menge soll nun ein Operator, aufgrund einer gewissen

[4]siehe Kapitel 5.2.4

Auswahlwahrscheinlichkeit, zufällig ausgewählt werden um die Nachbarschaft von x zu entwickeln. Die Operatorentypen wurden gerade so gewählt, dass sie durch ihre verschiedenen Vorgehensweisen unterschiedlich große Veränderungen an der Ausgangslösung x verursachen.
Verfahren, die nur minimale Veränderungen vornehmen, sorgen dafür, dass die direkte Umgebung der Lösung x erkundet wird. Sie weisen also einen intensivierenden Charakter auf. Diversifizierend dagegen wirken Operatoren, die die Ausgangslösung stark abändern.
Welches der genutzten Verbesserungsverfahren wie wirkt, zeigt die folgende Darstellung.

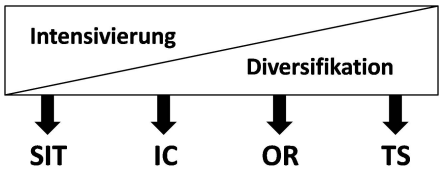


Abbildung 4: Intensivierungs-/Diverisfikationscharakter der Operatoren

Anhand der Auswahlwahrscheinlichkeiten kann, trotz der eigentlich zufälligen Auswahl aus der Operatorenmenge, eine Bevorzugung von intensivierenden bzw. diversifizierenden Verfahren vorgenommen werden. Wäre also in der nächsten Iteration eine Intensivierung vorteilhaft, so werden die Auswahlwahrscheinlichkeiten entsprechend gesetzt, das heißt hohe Wahrscheinlichkeiten für Shift Inside Tour und Interchange, niedrige dagegen für OR-Opt und Transferred Sequence; für den Diversifitkations-Status gilt das Gegenteil. Im Verlauf des Tabu-Search werden dann Zufallszahlen $\xi \in [0, 1]$ generiert, die festlegen, welches Verbesserungsverfahren angewandt wird.
Die Wahrscheinlichkeiten werden im Rahmen dieses Projektes gesetzt, wie in Abbildung 5 dargestellt. Diese Aufteilung berücksichtigt den entsprechenden intensivierenden beziehungsweise diversifizierenden Charakter der Verbesserungsverfahren und es wird sichergestellt, dass die Verfahren zielgerichtet angewendet werden können.

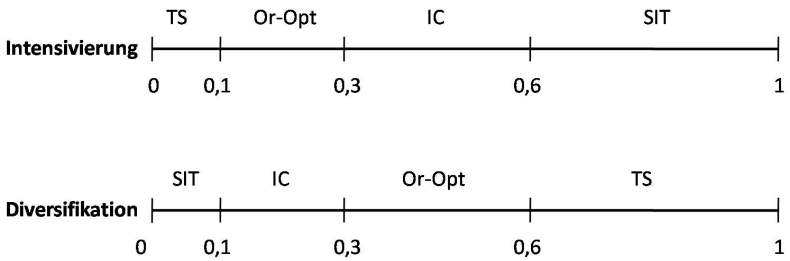


Abbildung 5: Mögliche Auswahlwahrscheinlichkeiten

Wird die Idee der Intensivierung und Diversifikation mit in den klassischen Tabu-Search Algorithmus eingebunden, so erhält man folgendes, speziell auf die Problemstellung des TSPTW zugeschnittenes Verfahren. Dabei wird der Zielfunktionswert C_1 als zu speicherndes Attribut der Lösung gewählt, terminieren soll der Algorithmus nach einer festgelegten Anzahl $max\_iteration$ an Durchgängen.

Algorithmus 2: Generisches Tabu-Search [9]

1. **(Initialisierung)** Setze die Zähler $j, t = 0$, initialisiere die Tabuliste *Tlist*.
2. **(Ausgangslösung)** Bestimme eine Initiallösung x mittels des Insertion-Verfahrens. Setze den Auswahlstatus auf Intensivierung.
3. **(Schleife)** Solange Iterationszähler $t < max\_iteration$:
    4. **(Auswahl)** Wähle aus der Menge {SIT, IC, OR-Opt, Name} entsprechend des Auswahlstatus einen Operator zufällig aus.
    5. **(Nachbarschaft)** Generiere mit diesem Operator die Nachbarschaft $\mathcal{N}(x)$.
    6. **(Evaluation)** Untersuche jeden Nachbarn $x_n \in \mathcal{N}(x)$. Speichere diese Werte.
    7. **(Bestensuche)** Wähle die beste Lösung x_{sel} , sofern sie nicht tabu ist.
    8. **(Zähler)** Falls $C_1(x_{sel}) \geq C_1(x)$, setze $j = j + 1$
        9. **(Diversifikation?)** Falls $j > max\_same$, setze Auswahlstatus auf Diversifikation.
    10. **(Iteration)** Setze $x := x_{sel}$ und aktualisiere *Tlist*.
    11. **(Aktualisierung)** Falls $C_1(x) < C_1(x_{best})$, setze $x_{best} := x$.
    Setze Auswahlstatus auf Intensivierung. Gehe zu 3.

Begonnen wird stets im Intensivierungs-Status, um zunächst die nähere Umgebung der Initiallösung zu durchsuchen. Falls nach max_same Iterationen (Schritt 9.) keine Verbesserung des Zielfunktionswertes mehr stattgefunden hat, so werden die Auswahlwahrscheinlichkeiten auf Diversifikation gesetzt. Nun wird der Tourenplan stark verändert, um in anderen Regionen des Lösungsraumes nach dem globalen Optimum zu suchen. Wird auf diese Weise ein besserer Plan als der bisher Beste x_{best} gefunden, konzentriert man sich erneut auf die genaue Untersuchung der direkten Umgebung (Schritt 11.).

4.2.3. Konstruktion der Initiallösung

Die Tabu-Search Metaheuristik benötigt in Punkt 2. eine Initiallösung, aus der die erste Nachbarschaft entwickelt werden kann.
Dafür verwendet man die sogenannten *Eröffnungsverfahren*, die nach einer möglichst guten, zulässigen, meist aber noch nicht optimalen Lösung des Problems suchen. Es existieren viele verschiedene Eröffnungsverfahren für das TSPTW, die Bekanntesten werden kurz vorgestellt. Es sei $S = (v_p, ..., v_q)$ ein fortlaufend numeriertes, zeitlich zulässiges Segment einer Tour oder eine vollständige Tour.

- **Nearest-Neighbour-Verfahren**
  Dies ist das intuitivste Verfahren. Hierbei ist zunächst nur das Startdepot o in der Tour erhalten. Diese wird dann skuzessive um den am nächsten liegenden Kunden erweitert, indem er ans Ende der Tour angehängt wird.
  Der große Nachteil dieses Verfahrens ist, dass man zum Ende häufig Kanten mit sehr hohen Kosten einfügen muss, wodurch die gefundene Lösung meist weiter weg vom tatsächlichen Optimum liegt als bei anderen Verfahren.

- **Savings-Verfahren**
  Idee dieses Verfahrens ist es, anfangs mit Pendeltouren zu jedem Knoten, d.h. vom Startdepot zum Kunden und wieder zum Depot zurück, zu beginnen. Danach werden dann je zwei Pendeltouren (o, v_i, o, v_j, o) durch eine Tour (o, v_i, v_j, o) ersetzt. Diese Zusammenlegungen werden solange wiederholt, bis alle Knoten in einer Tour enthalten sind. Dabei entsteht durch jede Zusammenlegung eine Ersparnis $s_{ij} = c_{io} + c_{oj} - c_{ij}$. Realisiert wird die Verbindung, die die größte Ersparnis bringt.
  Die Anwendung dieses Verfahrens ist im Rahmen des Projektes allerdings nicht sinnvoll, da bei jeder erneuten Optimierung ein Warmstart erfolgen soll, man also vom letzten optimalen Plan aus startet. Das erneute Anfangen bei Pendeltouren wäre unnötiger Aufwand.

- **Insertion-Verfahren**
  Eine Subtour, bestehend aus dem Startdepot, wird iterativ durch das Einfügen neuer Knoten an der jeweils richtigen Position ergänzt. Dies wird so lange wiederholt, bis alle Knoten in der Tour enthalten sind. Der Vorteil hierbei ist, dass nicht unbedingt mit einer Subtour aus einem Knoten begonnen werden muss. Es ist durchaus möglich, eine bereits bestehende Route zu betrachten und diese zu verändern. Dieser Ansatz ist also gut für den vorgesehenen Warmstart geeignet und soll deshalb verwendet werden.

4.2.3.1. Insertion-Verfahren

Das Insertion-Verfahren für die Tourenplanung mit Zeitfenstern wurde erstmals 1987 von Solomon vorgeschlagen [2].
Die Grundidee ist es, mit einer bestehenden Tour zu starten, der immer weiter Knoten hinzugefügt werden, bis alle Kunden in die Tour eingebunden sind. Dafür betrachtet man jeden Knoten $v \in N$, der noch nicht zur aktuellen Tour gehört und sucht für diesen durch Ausprobieren jeder möglichen Einfügestelle die optimale Position des Knotens in der Tour.
Im allerersten Duchlauf des Verfahrens, wenn noch kein Tourenplan aus der vorherigen Taktzeit existiert, beginnt das Verfahren mit einer Subtour vom Startdepot o über den Kunden, dessen Reklamation als erste an diesem Tag eingeht, zurück zum Depot o. Die übrigen Realen und Dummykunden werden nun sukzessive zu dieser Subtour hinzugefügt.
Bei den folgenden Neuoptimierungen wird in jedem Takt ein Warmstart verwendet. Das heißt, dass der Tourenplan, der im letzten Takt erstellt wurde und nun aktuell ausgeführt wird, als Startsegment für das erneute Durchführen des Optimierungsprozesses und somit des Insertion-Verfahrens verwendet wird. Es werden also in den aktuellen Plan der Startknoten und die neu hinzukommenden Reklamationen eingebaut, ohne dass wieder bei einer Subtour aus zwei Knoten gestartet werden muss.
Nun stellt sich die Frage, wie genau die Knoten in die bestehende Tour eingebunden werden sollen. Dabei wird zum einen nach einem günstigen *Einfügekriterium* (Welcher Knoten wird als nächster eingefügt?) und zum anderen nach der optimalen *Einfügeposition* (Wo wird dieser Knoten eingefügt?) gesucht.
Antworten auf diese Fragen gibt die sogenannte I_1-*Heuristik*. Diese geht folgendermaßen vor:
Zuerst wird für jeden noch nicht hinzugefügten Knoten die optimale Einfügeposition im bereits bestehenden Segment S bestimmt, indem die Einfügekosten berechnet werden. Anschließend wird anhand dieser Information entschieden, welcher Knoten an seiner optimalen Position eingefügt wird.

Es sei $\bar{N} := N \setminus S$ die Menge aller Knoten, die noch nicht zur Tour gehören. Betrachtet wird das Einfügen eines Knotens $v \in \bar{N}$ zwischen zwei Knoten v_j und v_{j+1} in ein bereits bestehendes Segment S.
Im i-ten Iterationsschritt liegt das Segment $(o, v_1, ..., v_i, o)$ mit $i+2$ Knoten vor, da das Startdepot o als Anfang und Ende der Tour immer zum Segment dazugehört.

Die I_1-Heuristik basiert auf der Idee, nicht nur die Kosten, sondern auch die durch das Einfügen des Knotens zusätzlich benötigte Zeit als Einfügekriterium einzubeziehen. Betrachtet man eine Problemstellung mit weichen Zeitfenstern, so wird die Verletzung dieser über die Strafkosten in die Kosten mit einbezogen.
Die *Einfügekosten* $C_1(v_j, v)$ für den Knoten v werden nun folgendermaßen berechnet:
Zunächst werden iterativ die Teilkosten der Tour bis einschließlich des eingefügten Knotens berechnet. Diese ergeben sich aus den Teilkosten TK_i bis zum Knoten v_i und den Kosten für die Verbindung von v_i und v. Außerdem muss eine mögliche Verspätung im Knoten v berücksichtigt werden, da bei einer Verletzung der Zeitfenster pro Minute die Strafkosten c_v^+ anfallen. So wird die zusätzliche Zeit über die Strafkosten in eine Geldgröße umgewandelt und bei der Entscheidung berücksichtigt:

$$Kosten = TK_i + c_{i,v} + u_v \cdot c_v^+$$

Ein zu frühes Ankommen beim Kunden hat keine Strafkosten zur Folge, allerdings muss eine eventuelle Wartezeit $w_i = max(0, a_i - (T_{i-1} + t_{i-1,i}))$ natürlich als zusätzlich benötigte Zeit mit eingerechnet werden. $T_{i-1} \in [a_i, b_i]$ ist dabei die Ankunftszeit am Knoten v_{i-1}.
Daraufhin werden die Kosten der Resttour neu berechnet. Sollte durch die Erweiterung des Segments um v das Zeitfenster von einem der auf v folgenden Knoten verletzt werden, müssen die zusätzlichen Strafkosten aufaddiert werden.
Die gesamten Einfügekosten $C_1(v_j, v)$ ergeben sich dann als Summe der Teikosten bis einschließlich v und den Kosten für die Resttour.
Auf Basis dieser Einfügekosten wird die *optimale Position* $j^*(v)$ von v in der Route bestimmt:

$$j^*(v) = \arg\min_j C_1(v_j, v) \qquad \text{und} \qquad C_1(v) = \min_{j=0,\ldots,i-1} C_1(v_j, v)$$

Die optimale Einfügeposition zeichnet sich also durch die niedrigsten Einfügekosten aus.
Wenn für alle $v \in \bar{N}$ die beste Position bestimmt wurde, dann erfolgt die Auswahl des Knotens, der tatsächlich als nächster an seiner jeweiligen optimalen Stelle in das Segment eingebracht wird. Dazu vergleicht man die Einfügekosten $C_1(v_j, v)$ mit den Kosten einer Pendeltour (o, v, o) vom Startdepot zum Kunden und wieder zurück. Es wird nicht nur die günstigste Lösung im Sinne von $C_1(v_j, v)$ einbezogen, sondern eine, die auch im weiteren Verlauf des Verfahrens sinnvoll ist. Deshalb wird derjenige Knoten gewählt, der möglichst weit vom Depot entfernt liegt und niedrige Kosten $C_1(v_j, v)$ hat. Durch diese Wahl kann man spätere Probleme im Hinblick auf die Verletzung der Zeitfenster reduzieren, da die Einbeziehung der Depotdifferenz tendenziell weit entfernte und somit schwerer zu integrierende Kunden bevorzugt. Gewählt wird also der Knoten v^*, der folgenden Ausdruck maximiert:

$$Nutzen(v) = \lambda(c_{o,v} + c_{v,o}) - C_1(v), \qquad \text{d.h.} \qquad v^* = \arg\max Nutzen(v)$$

Der Parameter λ regelt, wie stark Kunden bevorzugt werden, die weit weg vom Depotknoten liegen. Günstige Werte für λ werden anhand des Programmes experimentell ermittelt. Im folgenden sei $\rho(v_j) = v_{j-1}$ der Vorgängerknoten von von v_j.
Zusammenfassend folgt damit der Algorithmus:

Algorithmus 3: I_1-Heuristik

1. **(Initialisierung)**
   Falls es noch keinen Tourenplan gibt

   wähle einen Startknoten $v_i \in V \setminus \{o\}$ mit $a_i = min\{a_j\}\ \forall j \in \{1, ..., m\}$, $S := (o, v_i, o)$ und $\bar{N} := N \setminus \{v_i\}$;

   Sonst beginne mit dem aktuellen Tourenplan als Startsegment.

2. **(Schleife)** Wiederhole solange $\bar{N} \neq \emptyset$:

   3. **(Schleife über alle noch nicht eingefügten Knoten)** Wiederhole $\forall\ v \in \bar{N}$:

      4. **(Schleife über alle Einfügepositionen)** Wiederhole $\forall$ Positionen j

         (Teilkosten bis einschließlich Knoten v bestimmen):

         - setze $u_o = 0$, $w_o = 0$, $TK_o = 0$, $Kosten = 0$
         - Für alle Knoten $i = (v_1, ..., v_j)$:
           Wartezeit in i: $w_i = max(0, a_i - (T_{\rho(i)} + t_{\rho(i),i}))$
           Ankunftszeit in i: $T_i = T_{\rho(i)} + t_{\rho(i),i} + w_i$
           Verspätung in i: $u_i = max(0, T_i - a_i - (b_i - a_i))$
           Teilkosten bis i: $TK_i = TK_{\rho(i)} + c_{\rho(i),i} + u_i \cdot c_i^+$
         - Setze Kosten $= TK_{\rho(v)} + c_{\rho(v),v} + u_v \cdot c_v^+$
           Wartezeit in v: $w_v = max(0, a_v - (T_{\rho(v)} + t_{\rho(v),v}))$
           Ankunftszeit in v: $T_v = T_{\rho(v)} + t_{\rho(v),v} + w_v$

         (Kosten der Resttour aufaddieren:)

         - Für alle Knoten $l = (v_{j+1}, ..., v_{q-1})$ aus der Resttour:
           Wartezeit in l: $w_l = max(0, a_l - (T_{\rho(l)} + t_{\rho(l),l})$
           Ankunftszeit in l: $T_l = T_{\rho(l)} + t_{\rho(l),l} + w_l$
           Verspätung in l: $u_l = max(0, T_l - a_l - (b_l - a_l))$
           Teilkosten bis l: $Kosten = Kosten + c_{\rho(l),l} + u_l \cdot c_l^+$
         - Setze Gesamtkosten $C_1(v_j, v) = Kosten + t_{v_{q-1},o}$

      5. Bestimme:
         Einfügeposition $j^* = \arg\min_j C_1(v_j, v)$
         Einfügekosten $C_1(v) = \min_{j=0,...,i-1} C_1(v_j, v)$
         Nutzen $Nutzen(v) = \lambda(c_{o,v} + c_{v,o}) - C_1(v)$
         Speichere diese.

      Bestimme:
      optimalen Knoten $v^* = \arg\max Nutzen(v)$

   6. **(Einfügen)** Setze $\bar{N} := \bar{N} \setminus v^*$, füge v^* an der optimalen Position j^* in S ein.

Da der Knoten v zuerst an den Anfang der Tour gesetzt und dann sukzessive immer eine Position weiter nach hinten verschoben wird, verändern sich die Kosten vom Anfang der Tour bis zum Vorgänger v_i in den folgenden Iterationen nicht mehr. Wird also diese zweiteilige Berechnung der Tourkosten verwendet, können die anfänglichen Teilkosten TK_i gespeichert werden und müssen so in den nächsten Schritten nicht erneut berechnet werden, was das Verfahren beschleunigt.

4.2.4. Konstruktion der Nachbarschaft

Im 5. Schritt des Tabu-Search Algorithmus wird die Nachbarschaft $\mathcal{N}(x)$ mit Hilfe eines zufällig ausgewählten Operators erstellt. Dafür steht eine Menge von vier Nachbarschaftsoperatoren zur Verfügung:

- *Shift Inside Tour* (SIT): Entfernen eines Knotens aus der Tour und erneutes Einfügen an einer anderen Stelle
- *Interchange* (IC): Tausch zweier Knoten
- *Or-Opt*: Versetzen einer Knotenkette der Länge drei
- *Transferred Sequence*: Übernehmen der Knotenreihenfolge von einer anderen Tour des Tourenplanpools

Es handelt sich hierbei um *Verbesserungsverfahren*, die im Folgenden genauer vorgestellt werden.

4.2.4.1. Shift Inside Tour

Dieses Verbesserungsverfahren läuft ähnlich ab wie das Insertion-Verfahren.
Die Grundidee ist, einen Knoten aus der Tour auszuwählen, der dann entfernt wird. Anschließend wird dieser der Reihe nach an sämtlichen denkbaren Zwischenpositionen der Tour wieder eingesetzt. Dieses Vorgehen wird für alle Knoten der Tour wiederholt. Es ist die Operation zu wählen, die den Zielfunktionswert am stärksten herabsetzt.

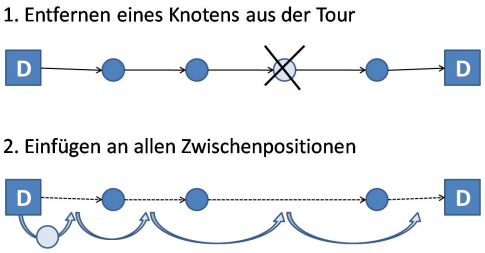


Abbildung 6: SIT-Operator

Auf diese Weise werden durch die Auswahl eines Knotens gleich mehrere Nachbarn erzeugt und analysiert. Umfasst die Tour n Kunden (ohne das Depot), so werden $n+1$ Nachbarlösungen generiert, wobei eine dem alten Tourenplan entspricht [9]. Diese kann allerdings nicht realisiert werden, da sie im vorherigen Schritt in die Tabuliste aufgenommen wurde und deshalb nun verboten ist.
Nur falls in der gesamten Nachbarschaft keine Verbesserung existiert, wird die Operation gewählt, die die kleinste Verschlechterung verursacht.

Algorithmus 4: Shift Inside Tour

1. **(Initialisierung)** Betrachte den aktuellen Tourenplan mit den Kosten C_1.
2. **(Schleife)** Für alle Knoten $v_{j_0} = (v_1, ..., v_{q-1})$ aus dem Tourenplan:
    3. **(Entfernen)** Lösche den Knoten v_{j_0} aus der Tour.
    4. **(Schleife über alle Einfügepositionen)** Wiederhole $\forall$ Positionen j:
    Setze v_{j_0} an jeder Position der Tour ein und berechne die Kosten

    (Teilkosten bis einschließlich Knoten v_{j_0} bestimmen):

    - setze $u_o = 0$, $w_o = 0$, $TK_o = 0$, $Kosten = 0$
    - Für alle Knoten $i = (v_1, ..., v_{j_0-1}, v_{j_0+1}, ..., v_j)$:
    Wartezeit in i: $w_i = max(0, a_i - (T_{\rho(i)} + t_{\rho(i),i}))$
    Ankunftszeit in i: $T_i = T_{\rho(i)} + t_{\rho(i),i} + w_i$
    Verspätung in i: $u_i = max(0, T_i - a_i - (b_i - a_i))$
    Teilkosten bis i: $TK_i = TK_{\rho(i)} + c_{\rho(i),i} + u_i \cdot c_i^+$
    - Setze Kosten $= TK_{\rho(v_{j_0})} + c_{\rho(v_{j_0}),v_{j_0}} + u_{v_{j_0}} \cdot c_{v_{j_0}}^+$
    Wartezeit in v_{j_0}: $w_{v_{j_0}} = max(0, a_{v_{j_0}} - (T_{\rho(v_{j_0})} + t_{\rho(v_{j_0}),v_{j_0}}))$
    Ankunftszeit in v_{j_0}: $T_{v_{j_0}} = T_{\rho(v_{j_0})} + t_{\rho(v_{j_0}),v_{j_0}} + w_{v_{j_0}}$

    (Kosten der Resttour aufaddieren:)

    - Für alle Knoten $l = (v_{j+1}, ..., v_{j_0-1}, v_{j_0+1}, ..., v_{q-1})$ aus der Resttour:
    Wartezeit in l: $w_l = max(0, a_l - (T_{\rho(l)} + t_{\rho(l),l}))$
    Ankunftszeit in l: $T_l = T_{\rho(l)} + t_{\rho(l),l} + w_l$
    Verspätung in l: $u_l = max(0, T_l - a_l - (b_l - a_l))$
    Teilkosten bis l: $Kosten = Kosten + c_{\rho(l),l} + u_l \cdot c_l^+$
    - Setze Gesamtkosten $C_2(v_{j_0}, j) = Kosten + t_{v_{q-1},o}$
    - Setze $Shiftkosten(v_{j_0}, j) = C_2(v_{j_0}, j) - C_1$

    5. **(Rückgängig)** Setze v_{j_0} wieder an seiner Ausgangsposition in die Tour ein.
6. **(Einfügen)** Setze den Knoten $v_{j_0}^*$ an seiner optimalen Position $j^* = \arg\min_j Shiftkosten(v_{j_0}^*, j)$ ein. Setze Tourkosten $C_1 = Shiftkosten(v_{j_0}^*, j^*)$.

Eine Verbesserung der Lösung ist genau dann eingetreten, wenn in Schritt 4 die Variable $Shiftkosten$ negativ ist, da dann der alte Zielfunktionswert b größer ist als der der neuen Tour.
Bei der Berechnung der Kosten wird der entfernte Knoten sowohl bei den ersten Teilkosten, als auch bei der Berechnung der Resttourkosten berücksichtigt. Dies geschieht nur der Übersichtlichkeit halber; natürlich kann sich die ursprüngliche Position des Knotens nur entweder vor oder nach der gerade betrachteten Einfügeposition befinden.

4.2.4.2. Interchange

Das λ-Interchange-Verfahren wurde 1989 zum ersten Mal von Osman und Christofides publiziert. Der Operator erzeugt, ausgehend von einer zulässigen Tour, neue Lösungen, indem er Abschnitte mit λ aufeinanderfolgenden Knoten vertauscht.
Das hier verwendete IC-Verfahren ist ein Spezalfall des λ-Interchange mit $\lambda = 1$, das heißt zwei Knoten der Tour werden getauscht.

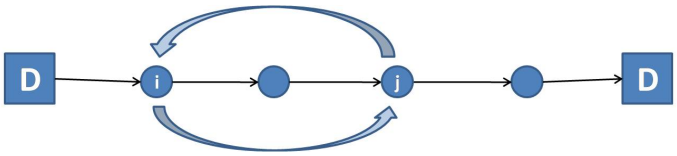


Abbildung 7: IC-Operator

Diese Wahl für λ wird getroffen, da der Operator eine intensivierende Wirkung haben soll. Wird λ auf einen größeren Wert gesetzt, so bewirkt dies eher eine Diversifikation.
Das Verfahren arbeitet in zwei Schritten: Die beiden zu vertauschenden Knoten werden zunächst aus der Tour gelöscht, woraufhin sie an der jeweiligen Position des anderen entfernten Knoten eingefügt werden. Ein wichtiger Unterschied zum SIT ist also, dass die neuen Einfügepositionen der beiden entfernten Knoten eindeutig festgelegt sind, es werden nicht alle möglichen Stellen ausprobiert. Eine zweiteilige Kostenberechnung der neu hergestellten Tour wie bei SIT kann deshalb durch die Berechnung der kompletten Tourkosten ersetzt werden, da die Zwischenspeicherung der Teilkosten nicht weiter verwendet werden kann.
Der Algorithmus verwendet den „Best-Accept“ - Ansatz, das heißt, es wird die gesamte Nachbarschaft durch den Tausch aller möglichen Knotenpaare erstellt und analysiert. Der Tausch, der die größte Herabsetzung des Zielfunktionswertes erreicht, wird durchgeführt. Im folgenden sei $\gamma(v_j) = v_{j+1}$ der Nachfolgerknoten von v_j.

Algorithmus 5: Interchange

1. **(Initialisierung)** Betrachte den aktuellen Tourenplan mit Kosten C_1.
2. **(Schleife 1. Knoten)** $\forall\ i = (v_1, ..., v_{q-1})$:
    3. **(Schleife 2. Knoten)** $\forall\ j = (i, ..., v_{q-1})$:
        4. **(Ursprungsstelle)** Speichere Ausgangspositionen $(\rho(i), i)$ und $(\rho(j), j)$.
        5. **(Vertauschen)** Tausche die Position der beiden Knoten in der Tour:
           $help := \rho(i), \quad \rho(i) := \rho(j), \quad \rho(j) := help$
           $help := \gamma(i), \quad \gamma(i) := \gamma(j), \quad \gamma(j) := help$
        6. **(Tourkosten)** Berechne Kosten des neuen Tourenplans:
            - setze $u_o = 0$, $w_o = 0$, $TK_o = 0$, $Kosten = 0$
            - Für alle Knoten $l = (v_1, ..., v_{q-1})$:
              Wartezeit in l: $w_l = max(0, a_l - (T_{\rho(l)} + t_{\rho(l),l}))$
              Ankunftszeit in l: $T_l = T_{\rho(l)} + t_{\rho(l),l} + w_l$
              Verspätung in l: $u_l = max(0, T_l - a_l - (b_l - a_l))$
              Setze $Kosten = Kosten + c_{\rho(l),l} + u_l \cdot c_l^+$
            - Setze Gesamtkosten $C_2(v_i, v_j) = Kosten + t_{v_{q-1},o}$
        7. **(Veränderung)** Speichere Änderung des Zielfunktionswertes:
           $Changekosten_{i,j} = C_2(i, j) - C_1$
        8. **(Zurücktauschen)** Mache den Tausch rückgängig:
           $help := \rho(i), \quad \rho(i) := \rho(j), \quad \rho(j) := help$
           $help := \gamma(i), \quad \gamma(i) := \gamma(j), \quad \gamma(j) := help$
9. **(Best-Accept)** Führe den Tausch durch, der die Kosten am stärksten verringert:
   $swap(i, j) = \arg\min_{i,j} Changekosten_{i,j}, \qquad C_1 = C_1 + \min_{i,j} Changekosten_{i,j}$

Wie bei SIT findet eine Verbesserung des Zielfunktionswertes dann statt, wenn die Variable $Changekosten_{i,j} < 0$ ist.

4.2.4.3. Or-Opt

Der Or-Opt-Operator wurde 1976 von Or als Kompromiss zwischen der schnelleren Laufzeit des 2-opt-Verfahrens und der besseren Lösungsgüte des 3-opt-Verfahrens entwickelt.
2-Opt und 3-Opt funktionieren analog; im ersten Fall werden genau zwei Kanten des aktuellen Tourenplans gegen zwei andere Kanten ausgetauscht, im zweiten Fall werden drei Kanten ersetzt. Der 3-Opt-Algorithmus weist bessere Ergebnisse auf als der 2-Opt-Algorithmus, allerdings hat er mit $O(n^3)$ einen wesentlich höheren Aufwand im Vergleich zum 2-Opt mit $O(n^2)$[8].
Or-Opt ist eine Modifikation von 3-Opt, bei der die Anzahl der Kantenvertauschungen gegenüber 3-Opt wesentlich reduziert wird, die Güte der erhaltenen Näherungslösung aber kaum schlechter ist.
Die Idee hinter dem Verfahren ist, eine komplette Kette von r aufeinanderfolgenden Kunden aus der Tour zu entfernen und an einer anderen Stelle zwischen zwei Konten in der gleichen Reihenfolge wieder einzusetzen. Da immer komplette Ketten verschoben werden, bleibt, anders als zum Beispiel beim 2-Opt-Verfahren, die Orientierung aller Tourenabschnitte erhalten. Aus diesem Grund ist dieses Verfahren gut für Probleme mit Zeitfenstern geeignet.
Beim klassischen Or-Opt wird nur $r = 1, 2, 3$ betrachtet. Zuerst werden alle Tauschmöglichkeiten der Dreierketten überprüft, als nächstes jene mit Zweierketten und zum Schluss die Tauschmöglichkeiten, die mit nur einem Knoten entstehen.

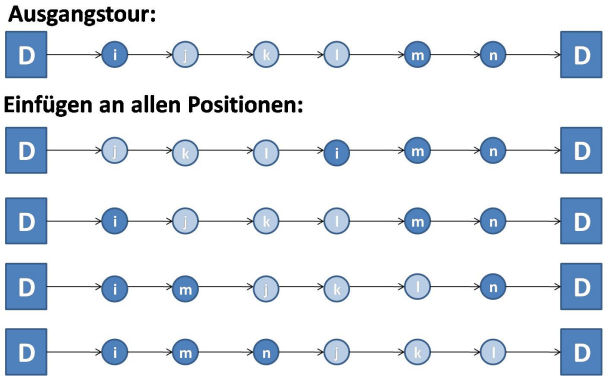


Abbildung 8: Or-Opt Operator

Zur Umsetzung wird jeder Knoten v_{j_0} der Tour einmal als Voränger der zu entfernenden Kette ausgewählt. Allerdings darf keiner der nachfolgenden Knoten v_{i+r} $\forall r \leq 3$ der Depotknoten sein. Die Knotenkette von v_{i+1} bis v_{i+r} wird dann aus der Tour entfernt und sukzessive an allen möglichen Stellen der Tour wieder eingefügt. Der so entstehende neue Tourenplan wird ausgewertet; am Ende wird die Operation mit der größten Kostenersparnis durchgeführt.
Auf diese Weise werden mit jedem durchlaufenen Knoten v_{j_0} insgesamt $3N - 3$ Nachbarn erzeugt und ausgewertet, wobei eine Lösung dem ursprünglichen Tourenplan entspricht. Diese Lösung wurde in der vorherigen Tabu-Search Iteration auf tabu gesetzt. Die Prozedur ähnelt dem SIT-Verfahren, sie unterscheiden sich nur in der Menge der einzufügenden Kunden. Der Algorithmus ist ähnlich aufgebaut, da sich auch hier die anfänglichen Teilkosten bis zur Einfügestelle nicht mehr verändern und so zwischengespeichert werden können.

Algorithmus 6: Or-Opt

1. **(Initialisierung)** Betrachte den aktuellen Tourenplan mit den Kosten C_1.
2. **(Schleife)** Für alle Knoten $v_{j_0} = (v_1, ..., v_{q-1})$ aus dem Tourenplan:
    3. **(Anzahl der Knoten)** Für $r = 3, 2, 1$:
        4. **(Entfernen)** Lösche die Kette $(v_{j_0+1}, ..., v_{j_0+r})$ aus der Tour.
        5. **(Schleife über alle Einfügepositionen)** Wiederhole $\forall$ Positionen j: Setze die Kette an jeder Position der Tour ein und berechne die Kosten

            (Teilkosten bis einschließlich Knoten v_{j_0+r} bestimmen):

            - setze $u_o = 0$, $w_o = 0$, $TK_o = 0$, $Kosten = 0$
            - $\forall$ Knoten $i = (v_1, ..., v_{j_0}, v_{j_0+r+1}, ..., v_j)$:
              Wartezeit in i: $w_i = max(0, a_i - (T_{\rho(i)} + t_{\rho(i),i}))$
              Ankunftszeit in i: $T_i = T_{\rho(i)} + t_{\rho(i),i} + w_i$
              Verspätung in i: $u_i = max(0, T_i - a_i - (b_i - a_i))$
              Teilkosten bis i: $TK_i = TK_{\rho(i)} + c_{\rho(i),i} + u_i \cdot c_i^+$
            - $\forall$ Knoten $i = (v_{j_0+1}, ..., v_{j_0+r})$:
              Setze Kosten $= TK_{\rho(v_{j_0})} + c_{\rho(v_{j_0}),v_{j_0}} + u_{v_{j_0}} \cdot c_{v_{j_0}}^+$
              Wartezeit in v_i: $w_{v_i} = max(0, a_{v_i} - (T_{\rho(v_i)} + t_{\rho(v_i),v_i}))$
              Ankunftszeit in v_i: $T_{v_i} = T_{\rho(v_i)} + t_{\rho(v_i),v_i} + w_{v_i}$

            (Kosten der Resttour aufaddieren:)

            - $\forall$ Knoten $l = (v_{j+1}, ..., v_{j_0-1}, v_{j_0+r+1}, ..., v_{q-1})$ aus der Resttour:
              Wartezeit in l: $w_l = max(0, a_l - (T_{\rho(l)} + t_{\rho(l),l}))$
              Ankunftszeit in l: $T_l = T_{\rho(l)} + t_{\rho(l),l} + w_l$
              Verspätung in l: $u_l = max(0, T_l - a_l - (b_l - a_l))$
              Teilkosten bis l: $Kosten = Kosten + c_{\rho(l),l} + u_l \cdot c_l^+$
            - Setze Gesamtkosten $C_2(v_j, j, r) = Kosten + t_{v_{q-1},o}$
            - Setze $Ersparnis(v_j, j, r) = C_2(v_j, j, r) - C_1$

        6. **(Rückgängig)** Setze die Kette $(v_{j_0+1}, ..., v_{j_0+r})$ wieder an ihrer Ausgangsposition in die Tour ein.
7. **(Einfügeposition)** Setze $j^* = \arg\min_j(\min_s Ersparnis(v_j, j, r))$ und $r^* = \arg\min_s Ersparnis(v_{j^*}, j^*, r)$
8. **(Einfügen)** Setze die Kette $(v_{j_0+1}, ..., v_{j_0+r^*})$ an der optimalen Stelle j^* in die Tour ein. Setze Tourkosten $C_1 = Ersparnis(v_j^*, r^*)$

4.2.4.4. Transferred Sequence

Das Transferred Sequence-Verfahren beruht auf einer Idee von Prof. Dr. Beisel und wird nun in dieser Arbeit ausgebaut und konkretisiert.
Im Gegensatz zu den vorherigen Verfahren handelt es sich bei diesem um einen Vertreter der *genetischen Algorithmen*. Hierbei wird, angelehnt an die biologische Evolution, eine Menge von Lösungskandidaten erzeugt und diejenigen ausgewählt, die einem bestimmten Gütekriterium am besten entsprechen. Deren Eigenschaften werden dann leicht verändert und zufällig miteinander kombiniert um so eine neue Generation von Lösungskandidaten zu erzeugen. Durch Auslese, Mutation und Rekombination erhält man so neue Lösungen, die den Anforderungen des Problems immer besser entsprechen.

Da es sich bei den Lösungen des TSPTW um Reihenfolgen handelt, können die ersten Knoten bis zu einer zufällig gewählten Position h einer Tour nicht mit den letzten $(n-h)$ Knoten einer anderen Tour kombiniert werden, denn dies hätte zur Folge, dass einige Kunden doppelt oder auch gar nicht mehr in der neuen Tour enthalten wären. Deshalb wird der Rekombinationsoperator modifiziert: Beim Transferred Sequence-Verfahren wird also mit der aktuellen Tour S_0 und zusätzlich mit einer weiteren Tour S_1 aus dem Tourenplanpool gearbeitet. Die Grundidee ist es nun, in der aktuellen Tour nach den ersten h Knoten einen *Cut* durchzuführen, das heißt alle Kunden nach dem Knoten h aus der Tour zu entfernen und diese anschließend in der Reihenfolge, in der sie in dem Plan S_1 angeordnet sind, wieder hinter h in die Tour einzufügen. Übernommen wird also die Reihenfolge der aus S_0 entfernten Knoten, in der der Handlungsreisende diese in S_1 besucht. Das Vorgehen wird in Abbildung 7 verdeutlicht.

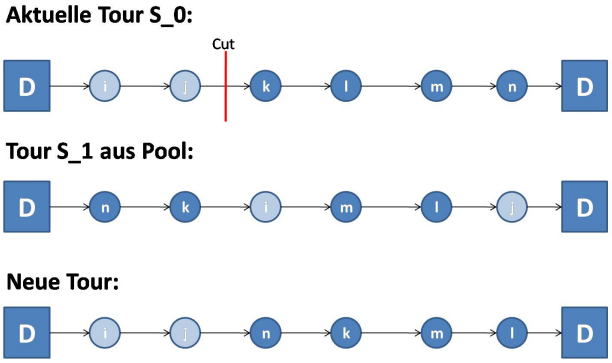


Abbildung 9: TS-Operator

Die Knotenanzahl h, nach der der Cut vorgenommen wird, sollte nicht zu groß gewählt werden. Da die ersten h Kunden in ihrer Reihenfolge nicht verändert werden, würde dies bei einem hohen h-Wert keine große Veränderung der Ausgangslösung bewirken. Der TS-Operator soll aber gerade eine starke diversifizierende Funkton haben, was eine deutliche Veränderung des aktuellen Tourenplans impliziert. Wird der Schnitt also weit am Anfang der Tour gesetzt, so bleibt genug Spielraum, um das Verfahren in einen anderen Teil des Lösungsraumes zu bewegen.
Dieses Vorgehen wird mit allen Touren aus dem Pool durchgeführt und der beste sich daraus ergebende Plan wird umgesetzt.

Algorithmus 7: Transferred Sequence

1. **(Start)** Starte mit der aktuellen Tour S_0 mit Kosten C_1.
2. **(Teilkosten bis h)** Berechne Teilkosten der ersten h Knoten in S_0:
   - Setze $u_o = 0$, $w_o = 0$, $TK_o = 0$
   - $\forall\ l = (v_{o,1}, ..., v_{0,h})$:
     Wartezeit in l: $w_l = max(0, a_l - (T_{\rho(l)} + t_{\rho(l),l}))$
     Ankunftszeit in l: $T_l = T_{\rho(l)} + t_{\rho(l),l} + w_l$
     Verspätung in l: $u_l = max(0, T_l - a_l - (b_l - a_l))$
     Setze $TK_l = TK_{\rho(l)} + c_{\rho(l),l} + u_l \cdot c_l^+$
3. **(Cut)** Lösche die Knoten $(v_{0,h+1}, ..., v_{o,q-1})$ aus S_0.
4. **(Nachbarschaft)** $\forall$ Pläne S_a, $a \in \{1, ..., k\}$ aus dem Tourenplanpool:

5. **(Knoten entfernen)** Entferne das Depot aus der Tour.

   $\forall$ Knoten $i = (v_{a,1}, ..., v_{a,q-1})$:

   $\forall\ j = 1, ..., h$:

   Falls $\quad i = v_{0,j} \quad$ dann lösche i aus S_a

   Nenne die so entstehende Tour S_a^{neu}.

6. **(Reihenfolge übernehmen)** Erstelle Tour S_0^{neu} durch Aneinanderhängen der beiden Touren $(o, v_{0,1}, ..., v_{0,h})$ und S_a^{neu}:
   Setze $\rho(v_{a,1}^{neu}) = v_{0,h}$

7. **(Kosten)** Berechne die Kosten von S_0^{neu}:
   - Setzen Kosten = 0
   - $\forall$ Knoten $l = (v_{a,1}^{neu}, ..., v_{a,q-1}^{neu})$:
     Wartezeit in l: $w_l = max(0, a_l - (T_{\rho(l)} + t_{\rho(l),l}))$
     Ankunftszeit in l: $T_l = T_{\rho(l)} + t_{\rho(l),l} + w_l$
     Verspätung in l: $u_l = max(0, T_l - a_l - (b_l - a_l))$
     Setze $Kosten = Kosten + c_{\rho(l),l} + u_l \cdot c_l^+$
   - Setze Gesamtkosten $C_2(a) = TK_l + Kosten + t_{v_{a,q-1}^{neu},o}$

8. **(Veränderung)** Berechne $Transferkosten_a = C_2(a) - C_1$

9. **(Auswahl)** Wähle die Reihenfolge der Tour S_a mit $a^* = \arg\min_a Transferkosten_a$ und $C_1 = C_1 + Transferkosten_{a^*}$.

Zunächst werden die Knoten, die in der aktuellen Tour an den ersten h Plätzen stehen, aus der Tour S_a gelöscht, damit die verbleibenden Knoten in der richtigen Reihenfolge an den Knoten v_h der Tour S_0 angehängt werden können. Die Kosten der ersten h Knoten müssen natürlich nur einmal berechnet werden, da sich diese nicht ändern.

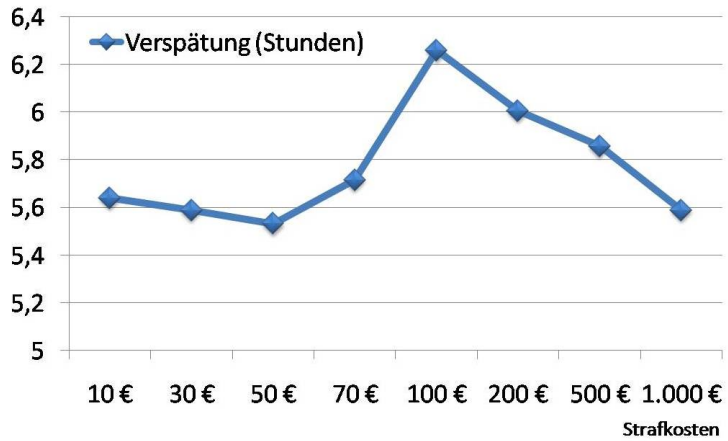
5. Parameteranalyse

In diesem Kapitel werden die Ergebnisse untersucht, die das VBA-Programm „Dynamische Tourenplanung“ für die vorliegende Problemstellung mit Hilfe der erarbeiteten Verfahren liefert. Implementiert wurden das Insertions-Verfahren sowie sämtliche Verbesserungsverfahren, die durch die Metaheuristik Tabu-Search gesteuert werden. Diese Ansätze hängen von einer Reihe von Parametern ab, für die nun ein möglichst optimaler Wert bestimmt werden soll, so dass das Gesamtverfahren sinnvoll auf die gegebene Datenmenge und die Problemstellung angepasst werden kann.
Es werden sämtliche Parameter und ihr Einfluss auf den errechneten Zielfunktionswert untersucht um so eine Empfehlung für eine sinnvolle Parameterwahl aussprechen zu können. Abschließend werden die einzelnen Verbesserungsverfahren getrennt voneinander getestet um deren individuelle Leistungsfähigkeit zu untersuchen.

Dem Programm wird als Datenmenge eine Dokumentation der Reklamationen in einem bestimmten Auslieferungsgebiet im Zeitraum von Januar 2006 bis Mai 2007 zugrunde gelegt. Die Tabellen geben pro Tag durchschnittlich 150 aufgetretene Kunden an, es handelt sich also um eine relativ kleine Datenmenge. Zusätzlich wurden die Ergebnisse der vorangegangen Arbeiten des Gesamtprojekts verarbeitet: Die Sektoreinteilungen und die Dummymenge aus der Zeitreihenanalyse und zwei verschiedene Clusterungen, die mit unterschiedlichen Ansätzen gebildet wurden. Diese Angaben wurden für insgesamt sechs Testtage erstellt. Die folgende Parameteranalyse wird anhand der Daten des 18.07.2006 und unter Verwendung des Clusterungsverfahrens k-Means durchgeführt, da dieser Tag mit 179 aufgetretenen Kunden eine überdurchschnittlich große Menge an Daten liefert und so mehr Möglichkeiten lässt, die implementierten Verfahren zu testen und geeignete Werte zu bestimmen. Im Anhang befindet sich zu jedem Parameter eine Tabelle mit den exakten Zielfunktions- und Verspätungswerten der getesteten Größen.

5.1. Abhängigkeit von den Strafkosten

Die Strafkosten c_i^+ spielen eine wichtige Rolle im Hinblick auf die Einhaltung der halbstündigen Zeitfenster. Pro angefangene Minute Überschreitung des Zeitfensters nach hinten fallen diese Konventionalkosten an. Über ihre Höhe kann reguliert werden, wie stark die Überschreitung ausfällt, da durch hohe Kosten die weichen Zeitfenster in harte umgewandelt werden können.
Es wurde zunächst ein Wert von 100 Euro angesetzt, allerdings stellte sich dieser Wert als zu hoch heraus, wie die folgende Abbildung zeigt. Auf der x-Achse sind die Strafkosten aufgetragen, auf der y-Achse die Gesamtverspätung aller Fahrer in Stunden.

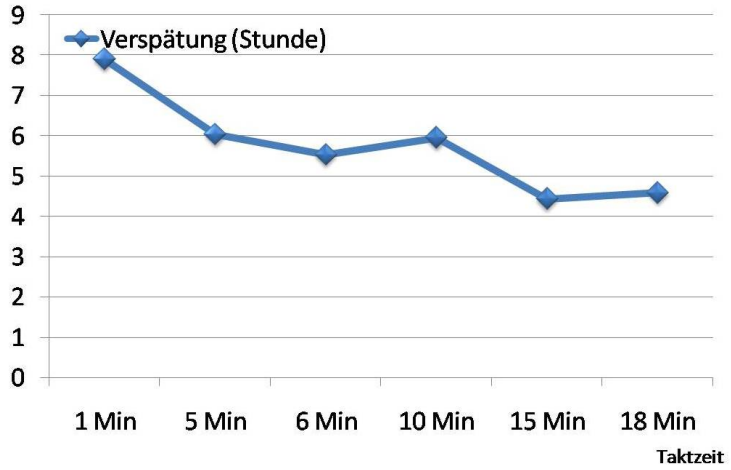


Man stellt fest, dass die Überschreitung der Zeitfenster nicht zwangsläufig mit steigenden Konven-

tionalkosten zurückgeht. Nur bis zu einem Wert von $c_i^+ = 50$ Euro zeigt sich dieser Effekt, zwischen 50 und 100 Euro steigt die Verspätung wieder stark an. Dies lässt sich über die relativ kleine Ausgangsdatenmenge erklären, durch die sich in den einzelnen Takten nur sehr wenige Kunden in einer Tour befinden. Durch hohe Strafkosten wird die gesamte Optimierung sehr undynamisch, da die Zeitfenster viel stärker fokussiert werden als die Entfernungen und so kann schlecht auf Änderungen reagiert werden. Bei geringeren Kosten dagegen wird auch die geometrische Entfernung angemessen berücksichtigt und ergibt so insgesamt ein besseres Ergebnis. Bei Werten über 100 Euro verringert sich die Gesamtverspätung wieder, allerdings sinkt sie nicht unter die bereits bei 50 Euro erreichte Zeit, sondern erreicht nur einen überproportionalen Anstieg des Zielfunktionswertes. Deshalb wird hier für die Höhe der Strafkosten ein Wert von $c_i^+ = 50$ Euro empfohlen.

5.2. Abhängigkeit von der Taktzeit

Während der Taktzeit t werden die Pläne aus dem Tourenplanpool optimiert und gleichzeitig neu auftretende Reklamationen gesammelt, die in der nächsten Taktzeit mit in die Erstellung des optimalen Tourenplans eingebunden werden. Hier wurde bei einem Ausgangswert von 3 Minuten begonnen, der sich allerdings als nicht optimal herausgestellt hat.

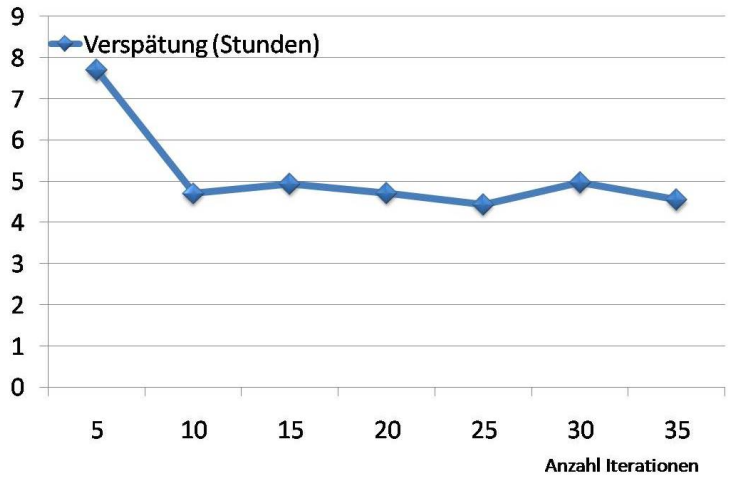


Es zeigt sich deutlich, dass die gesamte Verspätung mit steigender Taktzeit bis zu einem Wert von 15 Minuten abnimmt. Wird die Taktlänge weiter erhöht, so verschlechtert sich das Ergebnis, was drauf zurückzuführen ist, dass schon ein zu großer Anteil der Zeitfenster der neu auftretenden Kunden verstrichen ist, eher sie überhaupt in die Optimierung miteinbezogen werden konnten.
Die doch überraschend hohe Taktzeit lässt sich ebenfalls durch die geringe Menge an Kundendaten erklären. Je kürzer die Taktzeit ist, desto weniger Kunden werden in die Optimierung mit einbezogen und sie werden fast vollständig in der Reihenfolge ihres Auftretens abgearbeitet, da die Verbesserungsverfahren aus Mangel an einer ausreichenden Knotenanzahl nicht verwendet werden können. Bei einer größeren Zeitspanne dagegen sind genug Knoten vorhanden, um eine geeignete Reihenfolge zu finden.
Die Taktzeit wird also mit $t = 15$ Minuten angesetzt.

5.3. Abhängigkeit von der Iterationsanzahl

Die maximale Iterationsanzahl $max\_iteration$ gibt an, wieviele Durchläufe des Tabu-Search pro Taktzeit ausgeführt werden.
Zunächst ist deutlich zu erkennen, dass die Lösungsgüte mit steigender Anzahl an Iterationen zunimmt. Dies ist nicht verwunderlich, da das Verfahren von einer Ausgangslösung startend zunächst

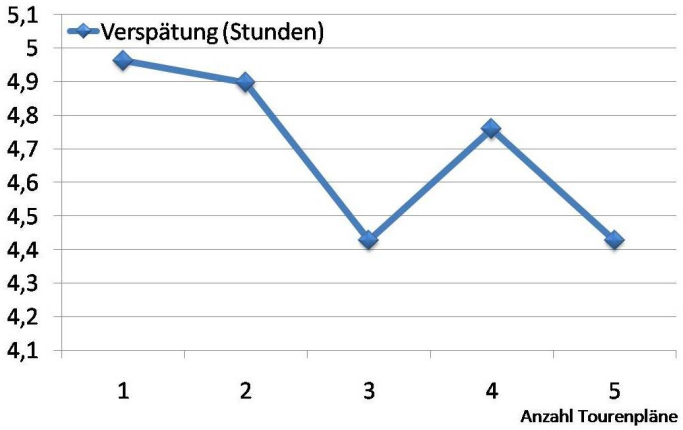
zu einem lokalen Optimum finden muss, wofür meistens einige Durchläufe der Verbesserungsverfahren notwendig sind.



Bis zu einem Wert von 10 Durchgängen ist eine starke Verbesserung des Zielfunktionswertes sichtbar. Weitere Erhöhungen der Durchlaufanzahl führen noch zu etwas besseren Ergebnissen, ab einem Wert von 25 Iterationen ist diese Verkleinerung der Zeitfensterverletzung allerdings nur noch minimal. Dieses Verhalten hängt stark mit der Länge der Tabuliste zusammen, weil sich die Optimierung ab einer gewissen Anzahl an Durchläufen in einem Zyklus verliert. Eine sehr hohe Iterationsanzahl liefert also nicht zwangsläufig bessere Ergebnisse; dies trifft nur auf einzelne der Testdurchläufe zu. Da sich die Laufzeit des Programms mit weiter steigender Iterationstiefe stark verlangsamt, wird ein Wert von 25 Tabu-Search Durchgängen pro Taktzeit für sinnvoll gehalten.

5.4. Abhängigkeit von der Poolgröße

Die Größe des Tourenplanpools gibt an, wieviele Tourenpläne pro Cluster parallel geführt und optimiert werden. Dieser Parameter hat keinen allzu großen Einfluss auf die Ergebnisse der Optimierung, wie die Tabelle zeigt. Die Änderung der Anzahl der Pläne wirkt sich bei der Verspätung nur in den Nachkommastellen aus. Lediglich der Zielfunktionswert zeigt einen leicht fallenden Trend bei zunehmender Poolgröße, wobei auch dieser ab einem Wert von 3 nur noch minimal ist.

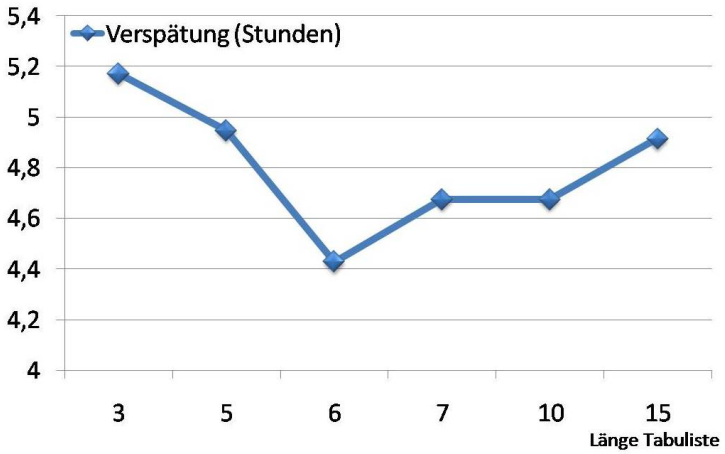


Die Größe des Pools auf nur einen Plan zu beschränken wäre wenig sinnvoll, weil dann das Transferred Sequence Verfahren nie angewendet werden könnte, da es auf mindestens 2 Pläne

angewiesen ist. Der Parameter soll hier nun auf einen Wert von 3 Plänen pro Cluster gesetzt werden, damit das TS Verfahren wenigstens zwei Variationsmöglichkeiten hat. Ein noch größerer Tourenplanpool bringt keine Verbesserung mit sich, allerdings steigt die Laufzeit des Programmes extrem mit der Anzahl der Pläne, was also nur zu einer Verlangsamung der Optimierung führt.
Bei einer größeren Datenmenge wäre der Tourenplanpool sicherlich weit erfolgreicher, da sich die Touren dann stärker voneinander unterscheiden würden und der Wechsel am Ende der Taktzeit von einem Plan zu einem besseren deutlich höhere Erfolge mit sich bringen würde.

5.5. Abhängigkeit von der Tabulistenlänge

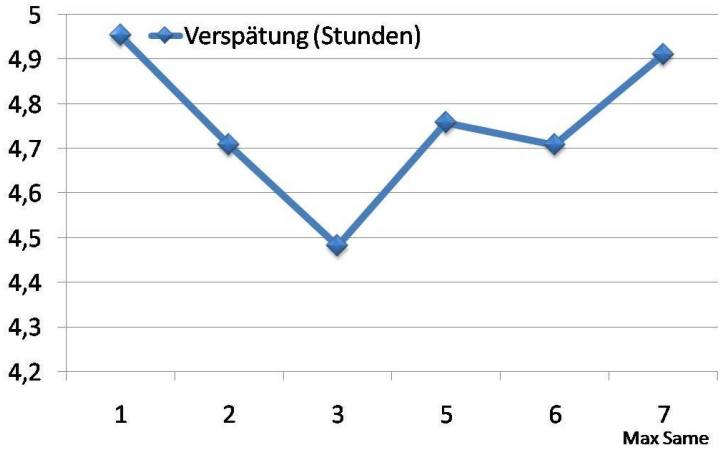
Die Hauptaufgabe der Tabuliste ist es, die Bildung von Zyklen in der Suche nach dem globalen Optimum zu verhindern, indem sie sich die Zielfunktionswerte der letzten Iterationen merkt und so die zugehörigen Lösungen für eine gewisse Zeit verbietet.
Die Länge der Tabuliste ist ein wichtiger Parameter. Die Abbildung zeigt deutlich, dass eine zu kurze Tabuliste nicht ihren Zweck erfüllt, da nur Zyklen der Länge der Tabuliste verhindert werden können, denn die Tabuliste wird in diesem Fall als FiFo-Liste geführt. Das heißt, dass neu hinzukommende Werte am Ende der Liste angehängt werden, gelöscht wird dagegen am Anfang, also der Wert, der bereits am längsten den Tabustatus trägt. Ist die Tabuliste dagegen zu lang, so werden die Verbesserungsverfahren stark eingeschränkt, so dass unter Umständen keine sinnvollen Nachbarschaftsoperationen mehr stattfinden können. Auf der x-Achse ist die Anzahl der Zielfunktionswerte aufgezeigt, die in der Tabuliste gespeichert werden.



Eine sinnvolle Länge der Tabuliste hängt sowohl von der Anzahl der Ausgangsdaten, als auch von der Anzahl der Gesamtiterationen ab.
Da die Datenmenge in diesem Fall nicht sehr groß ist, hat sich gezeigt, dass ein guter Wert für diesen Parameter eine Listenlänge von 6 ist. Bei der Parameteranalyse ist aufgefallen, dass sich gute Lösungen oft dadurch auszeichnen, dass die Tabulistenlänge ca. $\frac{1}{4}$ der Gesamtitertionsanzahl beträgt.

5.6. Abhängigkeit von der Anzahl der Verschlechterungen

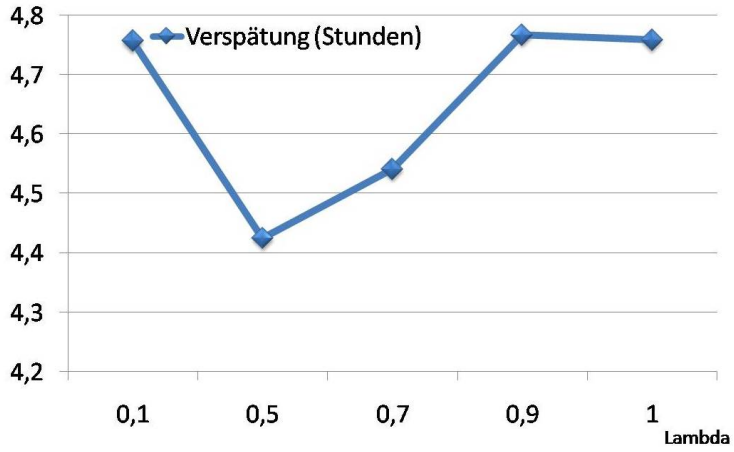
Die Anzahl *max_same* gibt an, wie oft hintereinander im Tabu-Search ein schlechterer oder höchstens gleich guter Zielfunktionswert als der aktuell beste akzeptiert wird, bis vom Intensivierungsmodus zur Diverisfikation übergegangen wird. Dieser Parameter ist nicht von zentraler Bedeutung, da er keinen allzu großen Einfluss auf das Ergebnis hat, wie folgende Abbildung zeigt.



Dabei ist zu beachten, dass sich der Unterschied zwischen dem schlechtesten Ergebnis für eine Anzahl von einem Durchlauf und dem besten Ergebnis bei drei Iterationen nur in den Nachkommastellen der Verspätung zeigt.
Während der Erstellung des Programms ist aufgefallen, dass sich der Zielfunktionswert oft ungefähr alle drei Iterationen verbessert, also nachdem zwei bis drei Verschlechterungen stattgefunden haben. Um diesem Muster zu folgen, wird max_same auf 3 gesetzt, was sich auch in der Analyse als bester Wert herausgestellt hat.

5.7. Abhängigkeit von λ

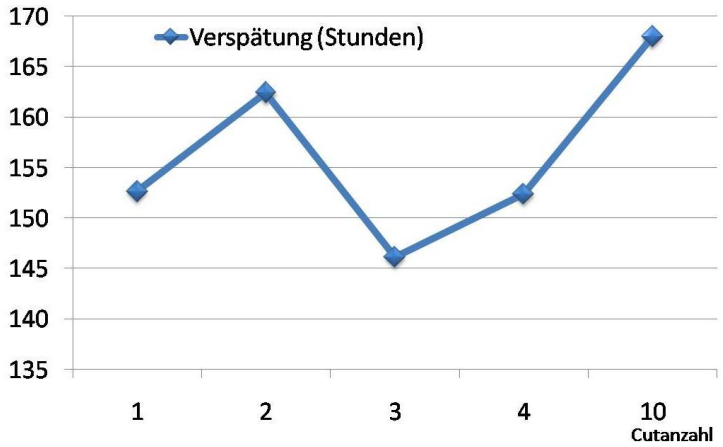
Der Parameter λ wird beim Insertions-Verfahren verwendet, um bei der Nutzenbestimmung die Bedeutung der Kosten der Pendeltour vom Depot zum einzufügenden Knoten und zurück zu gewichten.



Auch bei diesem Wert wirkt sich eine Veränderung nicht ausschlaggebend auf das Ergebnis der Optimierung aus, da die Verbesserungsverfahren schon so mächtig sind, dass die Güte der Anfangslösung nur von untergeordneter Wichtigkeit ist. Bei der gegebenen kleinen Datenmenge könnten die neu auftretenden Kunden ohne Effizienzverlust auch einfach an die bestehende Tour angehängt werden, ohne mit größerem Aufwand durch das Insertions-Verfahren eingefügt zu werden, weil die Reihenfolge der Knoten durch die Verbesserungsverfahren sowieso sehr stark verändert wird.
Hier wird nun $\lambda = 0,5$ gesetzt, da dies in den Tests die kleinste Verspätung einbrachte, wenn auch ohne erheblichen Unterschied zu anderen Belegungen für diesen Wert.

5.8. Abhängigkeit vom Cut

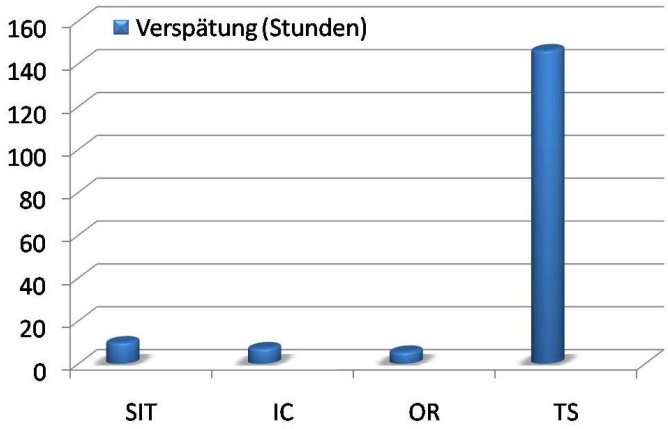
Beim Transferred Sequence Verfahren wird nach einer Anzahl h von Knoten ein Cut durchgeführt, also alle Knoten nach der h-ten Position werden aus der Tour entfernt. Zur Analyse dieses Parameters wurde die zufällige Auswahl der Verbessungsverfahren ausgeschaltet und lediglich mit dem TS-Verfahren gearbeitet, um einen besseren Eindruck vom Einfluss dieser Größe zu erhalten. Deshalb fallen die Verspätungsstunden auf der y-Achse in der Abbildung im Vergleich zu den vorhergehenden Diagrammen sehr hoch aus.



Es fällt auf, dass die Verspätungzeit in Abhängigkeit von h sehr stark schwankt. Dies wird auch wieder auf die geringe Datenmenge zurückzuführen sein. Da sich in jeder Taktzeit nur wenige Knoten in einer Tour befinden, ist ein hoher Wert für h nicht sinvoll, da man so oft nur die letzten zwei bis drei Knoten einer Tour entfernen würde, wodurch keine große Veränderung an der Reihenfolge der Kunden entstehen kann. Dies widerspricht dem Grundgedanken der Diversifikation, die das Verfahren herbeiführen soll. Aus dem gleichen Grund ist eine zu kleine Knotenanzahl vor dem Cut nicht erstrebenswert. Die Analyse zeigt, dass die Verspätung mit steigender Anzahl von h immer mehr zunimmt. Deshalb wird hier der ermittelte Optimalwert von $h = 3$ gesetzt.

5.9. Vergleich der Verbesserungsverfahren

In dem Programm „Dynamische Tourenplanung“ wurden alle vier vorgestellten Verbesserungsverfahren implementiert und ergeben zusammen mit der Metaheuristik Tabu-Search ein komplettes Verfahren, in dem die Nachbarschaftsoperatoren zufällig ausgewählt werden. Bei einem kompletten Durchlauf des Programms kann also die Leistungsfähigkeit der einzelnen Algorithmen nicht beurteilt werden.
Um festzustellen, wie sinnvoll die Verbindung der einzelnen Nachbarschaftsoperatoren zu einem Gesamtverfahren ist, wird das Programm leicht modifiziert, indem die Zufallsgenerierung wegfällt und jeweils alle bis auf ein Verbesserungsverfahren auskommentiert werden. Die gesamte Optimierung basiert dann nur noch auf dem Insertions-Verfahren und dem ausgewählten Algorithmus. Auf diese Weise konnte man klare Unterschiede in Leistung und Laufzeit der einzelnen Verfahren feststellen, wie man auch der folgenden Abbildung entnehmen kann.
Zunächst fällt deutlich auf, dass das Transferred Sequence Verfahren mit großem Abstand die schlechteste Lösung liefert. Mit einer Gesamtverspätung von 146,12 Stunden fällt das Ergebnis gegenüber der Verspätungshöhe von ungefähr 200 Stunden direkt nach dem Insertions-Verfahren nicht wesentlich besser aus.



TS ist für dieses Projekt also eine eher ungeeignete Lösungsvariante, allerdings heißt das nicht, dass dieses Verfahren generell unbrauchbar ist. Das schlechte Ergebnis im Rahmen dieser Arbeit ist auf die sehr kleine Datenmenge zurückzuführen, wodurch Transferred Sequence aufgrund der geringen Knotenanzahl einer Tour nicht in voller Weise anwendbar ist und gerade in den selten frequentierten Nachmittagsstunden oft sogar gar nicht angewendet werden kann und so keine Verbesserung liefert. TS wäre eine sinnvolle Variante bei Problemen mit sehr großem Datenumfang und auch einem großen Tourenplanpool. So könnte durch die vielfältigen Mutations- und Rekombinationsmöglichkeiten eine sehr gut diversifizierte Lösung generiert werden.
Das leistungsfähigste Verbesserungsverfahren dagegen ist Or-Opt. Werden alle Nachbarschaften mit Or-Opt Schritten generiert, so kommt man mit einer Verspätung von 5,44 Stunden fast an den Optimalwert von ca. 4 Stunden des Gesamtverfahrens heran. Dies ist wohl auf die Flexibilität des Verfahrens zurückzuführen, da es zumindest bei größeren Touren stärkere Veränderungen herbeiführen kann, indem eine Knotenkette der Länge drei neu platziert wird. Aber auch bei kürzeren Touren können noch Schritte mit zwei oder nur einem Knoten durchgeführt werden. So erhält man fast in jeder Iteration einen echt besseren Zielfunktionswert. Für diese Leistung ist die Laufzeit der Optimierung mit ca. 4 Minuten wesentlich höher als bei den anderen genannten Verfahren.

Verfahren	Gesamtkosten	Verspätung (Stunden)
Shilft Inside Tour	29.683,69 €	9,79072704
Interchange	21.447,65 €	7,04797542
Or-Opt	16.608,95 €	5,43975238
Transferred Sequence	438.780,74 €	146,1207

Shift Inside Tour und Interchange liefern ebenfalls sehr gute Ergebnisse, mit Gesamtverspätungen von 9,79 beziehungsweise 7,05 Stunden sind sie nur wenig schlechter als Or-Opt und deutlich besser als Transferred Sequence. Auffällig ist, dass sie trotz der ähnlich guten Lösung wie der des Or-Opt Verfahrens wesentlich schneller zu einem Ergebnis kommen.
Insgesamt ist die Nutzung des Gesamtverfahrens, bestehend aus allen vier Verbesserungsvarianten, empfehlenswert, da so ein Zielfunktionswert erreicht werden kann, der die Leistung der einzeln eingesetzten Verfahren übersteigt. Außerem ist die Laufzeit des Programms in dieser Komplettversion deutlich schneller als die des besten Einzelverfahrens. Die Vorteile der Verwendung liegen in der großen Anpassungsfähigkeit an den aktuellen Stand der Optimierung, die durch die zwar zufällige aber dennnoch durch entsprechende Wahrscheinlichkeiten gesteuerte Auswahl des jeweiligen Verbesserungsverfahrens beruht.

6. Fazit

Im Rahmen dieser Arbeit wurde ein realitätsnahes Konzept zur dynamischen Nachlieferungssteuerung entwickelt und umgesetzt, das zusammen mit den Ergebnissen anderer Arbeiten ein Gesamtprojekt darstellt.
Zunächst wurde ein mathematisches Modell erstellt, dass die gegebenen Restriktionen möglichst gut abbildet und so als Grundlage für die Entwicklung eines komplexen Verfahrens dienen konnte. Dieses Verfahren besteht aus mehreren Einzelteilen, die alle ausführlich vorgestellt und jeweils als Algorithmus formuliert wurden. Einen Schwerpunkt bildete die Tabu-Search Metaheuristik zur Koordination der verschiedenen Komponenten, deren Idee erst allgemein erläutert und anschließend auf den vorliegenden Praxisfall angepasst wurde. Auch die genaue Steuerung der Nachlieferungsbewältigung wurde dargestellt, so dass auf dieser Basis ein VBA-Programm erstellt werden konnte, das die optimale Reihenfolge für die Bearbeitung der Nachlieferung bestimmt.

Insgesamt kann man feststellen, dass das entwickelte Optimierungsverfahren zu mächtig ist für die vorliegende, relativ kleine Menge an Daten, ist. In einigen Iterationen können die Verbesserungsverfahren aufgrund zu weniger Knoten in einer Tour nicht richtig angewendet werden, vor allem in den späteren Nachmittagsstunden des Auslieferungstages ist dies der Fall. Auch die in einem anderen Teil des Gesamtprojekts, dem Forecasting, berechneten Dummykunden waren entsprechend der kleinen Menge an Ausgangsdaten nicht sehr zahlreich, so dass sie ihren Zweck, die Tour für potentielle Kunden offen zu halten, nicht wirklich erfüllen konnten. Die kritische Zeit war auch hier der Nachmittag, da ab 14 Uhr keine Dummies mehr vorhergesagt waren und auch nur noch vereinzelt reale Kunden auftraten, so dass der Fahrer oft aufgrund eines fehlenden Ziels an einem Punkt stehen bleiben musste, anstatt sich in eine sinnvolle Richtung bewegen zu können. Daher wäre es interessant, das Programm mit einer größeren Datenmenge zu testen, da dies von der Leistungsfähigkeit der Verfahren sicherlich zu guten Ergebnissen führen würde. Allerdings müsste dann das Programm in einer anderen Programmiersprache implementiert werden, da sich VBA als sehr ungeeignet für ein solch komplexes Programm herausgestellt hat und vor allem durch die Dynamik an seine Leistungsgrenzen stieß.
Die Ergebnisse der Optimierung hingen zudem stark von der Verwendung der Clusterdaten ab. Es lagen zwei verschiedene Clusterungen vor, die im Vorfeld einmal mit dem K-Means-Verfahren und einmal mit dem P-Median-Verfahren erstellt wurden. Die erste Variante lieferte deutlich bessere Ergebnisse was den Zielfunktionswert und die Gesamtverspätung aller Fahrer angeht, auch im Bezug auf die Laufzeit des Programms erwies sich diese Methode als vorteilhafter. Dies könnte darauf zurückzuführen sein, dass die Clusterung nach dem K-Means-Verfahren regelmäßiger ist, während das P-Median-Problem meist zwei große und mehrere kleinere Cluster lieferte.

Es sind sicherlich noch einige weitere Modifikationen an dem Optimierungsmodul denkbar. Eine Möglichkeit wäre es, die Länge der Tabuliste des Tabu-Search nicht konstant auf einen Wert zu setzen, sondern diese Größe variabel zu halten und anhand bestimmter Charakteristika der jeweiligen Lösung zu bestimmen. Auf diese Weise wäre ein noch gezielteres Eingehen auf den Verlauf der Optimierung möglich.
Da aufgrund der unterschiedlichen Ergebnisse der Clusterung der Nutzen dieser Vorarbeit nicht gut einzuschätzen ist, könnte man auch vom Travelling Salesman Problem in den einzelnen Clustern zu einem allgemeineren Fall, dem Vehicle Routing Problem übergehen. Dabei geht es um das kostenminimale Zuordnen der Kunden zu bestimmten Touren, sowie um das kostenminimale Aufreihen der Kunden innerhalb der Touren, welches dann dem TSP entspricht.
Abschließend ist zu sagen, dass die theoretische Ausarbeitung und die anschließende Programmierung gezeigt haben, dass das vorgestellte Optimierungsverfahren durchaus gute Lösungen liefert und unter zusätzlicher Beachtung weiterer Restriktionen auch in der Praxis Anwendung finden kann.

Anhang

A. Tabellen zur Parameteranalyse

Abhängigkeit von den Strafkosten:

Strafkosten	Gesamtkosten	Verspätung (Stunden)
10€	3.674,87 €	5,63904656
30€	10.347,75 €	5,5879014
50€	16.881,42 €	5,53157129
70€	24.297,07 €	5,71553883
100€	37.842,79 €	6,25794553
200€	72.342,87 €	6,0044432
500€	175.976,37 €	5,8562413
1000€	335.563,61 €	5,5879014

Abhängigkeit von der Taktzeit:

Taktzeit	Gesamtkosten	Verspätung (Stunden)
1 Min	24.005,80 €	7,8973403
5 Min	18.417,31 €	6,03764748
6 Min	20.200,37 €	5,53157129
10 Min	18.134,37 €	5,94460372
15 Min	13.572,54 €	4,42777614
18 Min	14.030,43 €	4,58309114

Abhängigkeit von der Iterationsanzahl:

Anzahl Iterationen	Gesamtkosten	Verspätung (Stunden)
5	23.357,08 €	7,68583156
10	14.392,85 €	4,699816
15	15.092,54 €	4,93375052
20	14.418,25 €	4,70825359
25	13.572,54 €	4,42777614
30	15.172,92 €	4,96181543
35	13.916,66 €	4,54253668

Abhängigkeit von der Poolgröße:

Größe Tourenplanpool	Gesamtkosten	Verspätung (Stunden)
1	15.174,35 €	4,96181543
2	14.983,17 €	4,89671974
3	13.572,54 €	4,42777614
4	14.565,13 €	4,75844741
5	13.572,54 €	4,42777614

Abhängigkeit von der Tabulistenlänge:

Länge Tabuliste	Gesamtkosten	Verspätung (Stunden)
3	15.801,16 €	5,17013453
5	15.129,73 €	4,94638587
6	13.572,54 €	4,42777614
7	14.311,15 €	4,67356903
10	14.311,15 €	4,67356903
15	15.031,71 €	4,91381408

Abhängigkeit von der Anzahl der Verschlechterungen:

Max Same	Gesamtkosten	Verspätung (Stunden)
1	15.152,04 €	4,95414615
2	14.418,25 €	4,70825359
3	13.735,76 €	4,48220173
5	14.565,13 €	4,75844741
6	14.418,25 €	4,70825359
7	15.019,16 €	4,91011965

Abhängigkeit von λ:

Lambda	Gesamtkosten	Verspätung (Stunden)
0,1	14.560,07 €	4,75672221
0,5	13.562,32 €	4,4244011
0,7	13.911,61 €	4,54081148
0,9	14.589,35 €	4,7670417
1,0	14.565,13 €	4,75844741

Abhängigkeit vom Cut:

Cutanzahl	Gesamtkosten	Verspätung (Stunden)
1	428.307,44 €	152,629097
2	487.722,35 €	162,430473
3	438.780,74 €	146,1207
4	457.540,38 €	152,372134
10	504.380,17 €	167,978036

B. Code zum VBA-Programm

Die gesamte Nachlieferungssteuerung wurde im Rahmen dieses Projektes in der zu den Microsoft-Office Programmen gehörenden Scriptsprache Visual Basic for Applications (VBA) implementiert. Im letzten Teil des Anhangs vefindet sich die Hauptteile der Programmcodes zu den in dieser Bachelorarbeit vorgestellten Eröffnungs- und Verbesserungsverfahren sowie der Metaheuristik Tabu-Search. Generelle Hinweise zur Implementierung, die Verbindung der Einzelverfahren zu einem laufenden Programm und kleinere Unterfunktionen können dem Benutzerhandbuch zum Programm „Dynamische Tourenplanung“ bzw. dem Programm selbst entnommen werden.

Tabu-Search:

```
Public Sub runTick(tickTimeStart As Double, tickTimeEnd As Double)

Dim plan As Tourplan
For Each plan In pool
  plan.runInsertion cloneCollection(tickNodes), tickTimeStart
Next plan
Set tickNodes = New collection
Dim minimalCosts As Double
Dim minimalTour As collection
minimalCosts = -1
For Each plan In pool
  Dim i As Integer
  i = 0
  Do Until i = instance.parameters.iterationCount
  i = i + 1
    plan.runTabuStep tickTimeStart
  If (minimalCosts = -1) Or (plan.currentCosts < minimalCosts) Then
    minimalCosts = plan.currentCosts
    Set minimalTour = cloneCollection(plan.tour)
  End If
Loop
  Next plan
  Dim removedNodes As collection
  Set removedNodes = vehicle.runTick(minimalTour, tickTimeStart, tickTimeEnd)
  For Each plan In pool
  plan.deleteNodes removedNodes
  Next plan

End Sub
```

```
Public Sub runTabuStep(ByVal time As Double)

Dim count As Integer
count = tour.count
Dim xi As Double
xi = rnd()
If xi >= tabuParameter.icMin And xi < tabuParameter.icMax Then
  runIC time
End If
If xi >= tabuParameter.orMin And xi < tabuParameter.orMax Then
  runOR time
End If
If xi >= tabuParameter.tsMin And xi < tabuParameter.tsMax Then
  runTS time
End If
If xi >= tabuParameter.sitMin And xi < tabuParameter.sitMax Then
  runSIT time
End If

End Sub



Public Sub processMinimalCosts(minimalCosts As Double)

tabuList.Add minimalCosts
tabuList.Remove 1
If tabuParameter Is instance.parameters.intensTabuParameter Then
  If minimalCosts >= currentCosts Then
    divCount = divCount + 1
  Else
    divCount = 0
  End If
  If (divCount = instance.parameters.maxSame) Then
    Set tabuParameter = instance.parameters.diversTabuParameter
  End If
Else
  If minimalCosts < currentCosts Then
    Set tabuParameter = instance.parameters.intensTabuParameter
  End If
End If
currentCosts = minimalCosts

End Sub
```

Shift Inside Tour:

```
Public Sub runSIT(time As Double)
If tour.count <= 2 Then
   Exit Sub
End If
Dim i, j As Integer
Dim minimalCosts As Double
Dim minimalTour As collection
i = 1
Do Until i = (tour.count + 1)
   Dim tempTour As collection
   Set tempTour = cloneCollection(tour)
   Dim node As node
   Set node = tour.Item(i)
   tempTour.Remove i
   j = 1
   Do Until j = (tempTour.count + 1)
      If (j + 1) <> i Then
         Dim tempTour2 As collection
         Dim costs As Double
         Set tempTour2 = cloneCollection(tempTour)
         tempTour2.Add node, after:=j
         costs = calculateCosts(vehicle, vehicle.nextNode, tempTour2, time)
         If ((minimalTour Is Nothing) Or (costs <= minimalCosts)) And Not isTabuMember(costs) Then
            minimalCosts = costs
            Set minimalTour = tempTour2
         End If
      End If
      j = j + 1
   Loop
   i = i + 1
Loop
If Not minimalTour Is Nothing Then
   Set tour = minimalTour
   processMinimalCosts (minimalCosts)
End If

End Sub
```

Interchange:

```
Public Sub runIC(time As Double)

If tour.count <= 2 Then
   Exit Sub
End If
Dim i As Integer
Dim j As Integer
Dim minimalCosts As Double
Dim minimalTour As collection
i = 1
Do Until i = (tour.count + 1)
   Dim node As node
   Set node = tour.Item(i)
   j = i + 1
   Do Until j = (tour.count + 1)
      Dim partner As node
      Set partner = tour.Item(j)
      Dim tempTour As collection
      Set tempTour = cloneCollection(tour)
      tempTour.Add node, after:=j
      tempTour.Remove j
      tempTour.Add partner, after:=i
      tempTour.Remove i
      Dim costs As Double
      costs = calculateCosts(vehicle, vehicle.nextNode, tempTour, time)
      If ((minimalTour Is Nothing) Or (costs <= minimalCosts)) And Not isTabuMember(costs) Then
         minimalCosts = costs
         Set minimalTour = tempTour
      End If
      j = j + 1
   Loop
   i = i + 1
Loop
If Not minimalTour Is Nothing Then
   Set tour = minimalTour
   processMinimalCosts (minimalCosts)
End If

End Sub
```

Or-Opt:

```
Public Sub runOR(time As Double)

Dim i, j As Integer
Dim s As Integer
Dim minimalCosts As Double
Dim minimalTour As collection
For s = 3 To 1 Step -1
  i = 1
  Do While i <= (tour.count - s + 1)
    Dim tempTour As collection
    Dim nodes As collection
    Set tempTour = cloneCollection(tour)
    Set nodes = extractSubCollection(tempTour, i, i + s - 1)
    removeNodes tempTour, nodes
    j = 1
    Do Until j = (tempTour.count + 1)
      If (j + 1) <> i Then
        Dim tempTour2 As collection
        Dim costs As Double
        Set tempTour2 = cloneCollection(tempTour)
        addSubCollection tempTour2, nodes, j
        costs = calculateCosts(vehicle, vehicle.nextNode, tempTour2, time)
        If ((minimalTour Is Nothing) Or (costs <= minimalCosts)) And Not isTabuMember(costs) Then
          minimalCosts = costs
          Set minimalTour = tempTour2
        End If
      End If
      j = j + 1
    Loop
    i = i + 1
  Loop
Next s
If Not minimalTour Is Nothing Then
  Set tour = minimalTour
  processMinimalCosts (minimalCosts)
End If

End Sub
```

Transferred Sequence:

```
Public Sub runTS(time As Double)

Dim removedNodes As collection
Dim newTour As collection
Dim minimalTour As collection
Dim minimalCosts As Double
Dim plan As Tourplan
Dim node As node
Dim costs As Double
If (tour.count <= instance.parameters.tsCut) Or (pool.count = 1) Then
  Exit Sub
End If
Set removedNodes = extractSubCollection(tour, instance.parameters.tsCut + 1, tour.count)
For Each plan In pool
  If id <> plan.id Then
    Set newTour = extractSubCollection(tour, 1, instance.parameters.tsCut)
    For Each node In plan.tour
      If nodeExistsInCollection(removedNodes, node) Then
        newTour.Add node
      End If
    Next node
    costs = calculateCosts(vehicle, vehicle.nextNode, newTour, time)
    If ((minimalTour Is Nothing) Or (costs <= minimalCosts)) And Not isTabuMember(costs) Then
      minimalCosts = costs
      Set minimalTour = newTour
    End If
  End If
Next plan
If Not minimalTour Is Nothing Then
  Set tour = minimalTour
  processMinimalCosts (minimalCosts)
End If

End Sub
```

Literatur

[1] Beisel, Ernst-Peter: *VRP-Modelle*, unveröffentlichtes Paper

[2] Engeler, Katja: *Mehrdepot-Tourenplanung mit Zeitfenstern*, Josef Eul Verlag, Lohmar, 2002

[3] Glover, Fred und Laguna, M.: *Tabu Search*, Springer Verlag, 1998

[4] Grünert, Tore und Irnich, Stefan: *Optimierung im Transport*, Band 1: *Grundlagen*, Shaker Verlag, Aachen, 2005

[5] Grünert, Tore und Irnich, Stefan: *Optimierung im Transport*, Band 2: *Wege und Touren*, Shaker Verlag, Aachen, 2005

[6] Heppner, Clemens: *Tabu-Search*, Universität Hamburg, Fachbereich Informatik

[7] Hizgilov, Tirza: *Anwendung eines Standardsoftwarepaketes für die Lösung von Tourenplanungsproblemen*, Universität Wien, 2009,

[8] Kämpf, Michael: *Probleme der Tourenbildung*, Technische Universität Chemnitz, 2006

[9] Kortmann, Britta: *Entwicklung und Analyse von Algorithmen zum dynamischen Vehicle Routing Problem für eine Nachliefersteuerung von Tageszeitungen unter Berücksichtigung von Forecastingmethoden*, Universität Wuppertal, 2008

[10] Lackner, Andreas: *Dynamische Tourenplanung mit ausgewählten Metaheuristiken: Eine Untersuchung am Beispiel des kapazitätsrestriktiven dynamischen Tourenplanungsproblems mit Zeitfenstern*, Cuvillier Verlag, 2004

[11] Larsen, A.: T*he dynamic Vehicle Routing Problem, Informatics and Mathematical Modelling*, Technische Universität Dänemark, 2000

[12] Lohwasser, Richard und Müller, Jürgen: *Dynamische Tourenplanung mit Zeitfensterrestriktionen*, Georg-August-Universität Göttingen, 2005

[13] Meissner, Jörg-Detlef: *Heuristiken beim Rundreiseproblem*, Technische Universität Berlin, 1976

[14] Müller, Kai: *Analyse und Verbesserung von iterierter lokaler Optimierung für das Kapazitive Vehicle-Routing-Problem mit Zeitfenstern*, Fachhochschule Konstanz, 2003