# Vehicle Routing mit Subtouren und Zeitfenstern

Thorben Tröbst

Geboren am 5. April 1996 in Bonn

7. August 2017

Bachelorarbeit Mathematik

Betreuer: Prof. Dr. Jens Vygen

Zweitgutachter: Prof. Dr. Stephan Held

Forschungsinstitut für Diskrete Mathematik

Mathematisch-Naturwissenschaftliche Fakultät der

Rheinischen Friedrich-Wilhelms-Universität Bonn

# Inhaltsverzeichnis

# Kapitel 1

# Einleitung

Das klassische "Vehicle Routing Problem" ist eines der zentralsten Probleme in der kombinatorischen Optimierung. Es handelt sich dabei um eine Verallgemeinerung des Traveling Salesman Problems, bei dem eine Menge von Zielpunkten optimal auf *mehrere* Fahrzeuge mit beschränkten Kapazitäten aufgeteilt werden soll. Erstmals wurde dieses Problem von Georg Dantzig und John Ramser in den Fünfzigern studiert und schließlich in der einflussreichen Arbeit "The Truck Dispatching Problem" im Jahr 1959 veröffentlicht (siehe [DR59]). Damals bemerkten die Autoren:

> No pratical applications of the method have been made as yet. A number of trial problems have been calculated, however.

In den darauf folgenden Jahrzehnten hat sich dies allerdings stark geändert: heutzutage verwenden Logistik-Unternehmen weltweit Planungs-Software, um täglich große Instanzen des Vehicle Routing Problems zu lösen. Da die Transport und Logistik Industrie zu einer der größten Industrien überhaupt gehört, ist das VRP nicht nur als schweres theoretisches Problem interessant, sondern hat weitgreifende Anwendungen. Im Rest dieses Abschnitts der Arbeit werden zunächst die grundlegenden Definitionen sowie die Problemstellung des Vehicle Routing Problems eingeführt. Schließlich werden ein paar einfache Variationen des Problems vorgestellt und es wird ein Überblick über die gängigen Lösungsverfahren gegeben.

In den letzten Jahren hat sich der Planungs- und Lieferhorizont in der Verbraucher-Logistik stark reduziert. Internet-Händler wie Amazon garantieren nun Lieferzeiten, die zum Teil noch am selben Tag liegen. Es werden daher zur Zeit verschiedene Verfahren getestet, die mit den immer knapper werdenen Zeitplänen umgehen können. Ein solches Verfahren ist das Umladen in sogenannte Subtouren, welches wir in Kapitel 2 behandeln werden.

Eine für die Praxis besonders relevante Klasse von Zusatzbedingungen für Vehicle Routing Probleme, die diese allerdings aus theoretischer Sicht deutlich verändert, entsteht durch das Hinzufügen von Zeitfenstern. Unter dieser zusätzlichen Einschränkung dürfen die Fahrzeuge nicht mehr zu beliebigen Zeiten an den Lieferpunkten ankommen. Stattdessen gibt es vorgeschriebene Lieferzeiten und Fahrzeuge müssen eventuell warten, um diese

einhalten zu können. In Kapitel 3 dieser Arbeit soll das Vehicle Routing Problem mit Subtouren aus Kapitel 2 zu dem Vehicle Routing Problem mit Subtouren und Zeitfenstern (VRPSTW) erweitert werden.

Für beide Probleme werden effiziente Heuristiken präsentiert, die praktische Probleme in geringer Laufzeit lösen können. Das Verhalten dieser Heuristiken, sowie den Einfluss den Zeitfenster und Subtouren auf Vehicle Routing Probleme haben, wird in Kapitel 4 behandelt werden. Dabei werden wir zufällig generierte Instanzen betrachten, die auf einem Modell von Berlin basieren.

## 1.1 Grundlegene Definitionen und Notation

Die grundlegenden Definitionen sowie die Notation dieser Arbeit folgen den Konventionen von [KV08]. Eventuelle Besonderheiten sollen in diesem Abschnitt geklärt werden.

Graphen sind bei uns — wenn es nicht wie in Kapitel 3 anders erwähnt wird — stets ungerichtet, endlich und einfach. Falls wir eine Funktion $f : V(G) \to \mathbb{R}$ gegeben haben, so setzen wir $f$ auf natürliche Weise auf die Potenzmenge von $V(G)$ fort:

$$f(M) \coloneqq \sum_{v \in M} f(v).$$

Analog gehen wir bei Funktionen $g : E(G) \to \mathbb{R}$ vor, wo wir zusätzlich oft $g(x, y)$ anstatt $g(\{x, y\})$ oder $g((x, y))$ schreiben werden. Zudem schreiben wir auch statt $f(V(H))$ einfach $f(H)$ (bzw. $g(H)$ anstatt $g(E(H))$) für Teilgraphen $H \leq G$.

Sei $P$ ein Weg und $v, w \in V(P)$, dann bezeichnen wir mit $P_{v,w}$ den eindeutigen Teilweg, welcher $v$ und $w$ als Endpunkte hat. Ist $C$ ein gerichteter Kreis mit $v, w \in V(P)$ so verwenden wir die Notation $C_{v,w}$ für den eindeutigen gerichteten $v$-$w$-Weg in $C$. Schließlich seien $P$ und $Q$ zwei disjunkte Wege in einem gerichteten Graphen $G$, sodass der Endpunkt von $P$ über eine Kante $e \in E(G)$ mit dem Anfangspunkt von $Q$ verbunden ist. Dann bezeichnen wir mit $P +\!\!+ Q$ die Konkatenation der beiden Wege entlang der Kante $e$.

## 1.2 Das klassische Vehicle Routing Problem

Das Szenario des klassischen Vehicle Routing Problems sieht wie folgt aus: Gegeben ist ein Depot, von dem aus mehrere Fahrzeuge losfahren können und zu dem sie wieder zurückkehren müssen. Weiterhin ist eine Menge von Zustellpunkten vorgegeben, die vom Depot aus beliefert werden müssen. Gesucht ist eine Aufteilung dieser Punkte auf Routen für jedes Fahrzeug, sodass kein Fahrzeug seine maximale Kapazität überschreitet, und sodass die Gesamtlänge aller Routen minimiert wird.

Eine optimal gelöste Instanz für das VRP mit 15 Zustellpunkten ist in Abbildung 1.1 zu sehen. Gelöst wurde diese Instanz mit dem Open-Source VRP Solver aus der `SYMPHONY` Suite. Im Folgenden soll ein kurzer Überblick über die Theorie und Praxis dieses klassischen Problems gegeben werden.

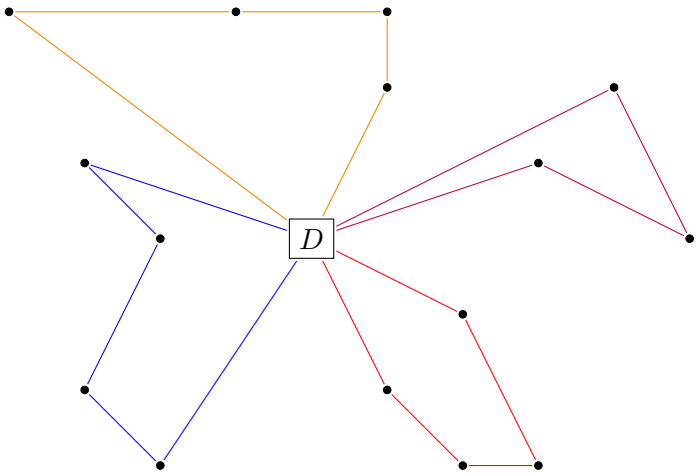



Abbildung 1.1: Gelöste Instanz des VRP mit Fahrzeug-Kapazität 4.

### 1.2.1 Vehicle Routing auf Graphen

Das klassische Vehicle Routing Problem kann als ein kombinatorisches Optimierungsproblem auf Graphen mit gewissen Zusatzdaten definiert werden. Der Graph ist dabei offenbar eine Abstraktion des Straßennetzes, welches in der Praxis extrem komplex ist. Insbesondere sind Wegzeiten in der Realität asymmetrisch und zeitabhängig, was wir in dieser Arbeit ignorieren werden.

**Definition 1.** Eine *Instanz des VRP* ist ein Tupel $(G, b, d, K)$ mit

- $G$ einem endlichen Graphen mit einem ausgezeichneten Knoten $D \in V(G)$,
- $b : V(G) \to \mathbb{R}_{\geq 0}$ mit $b(D) = 0$,
- $d : E(G) \to \mathbb{R}_{\geq 0}$ und
- $K$ einer positiven reellen Zahl.

Wir nennen $d(e)$ die *Länge* einer Kante $e \in E(G)$ und $b(v)$ die *Nachfrage* am Knoten $v \in V(G)$. Den Knoten $D$ nennen wir das *Depot* und die Zahl $K$ heißt die *Fahrzeug Kapazität*.

**Definition 2.** Sei $(G, b, d, K)$ eine VRP Instanz, dann heißt ein Kreis $C$ in $G$ eine *Route*, falls $D \in V(C)$ und $b(C) \leq K$. Die *Länge* von $C$ ist gegeben durch $d(C)$.

**Definition 3.** Für eine Instanz $(G, b, d, K)$ des VRP und ein $k \in \mathbb{N}_+$ sei eine *k-Partition* gegeben durch Knotenmengen $V_1, \ldots, V_k \subseteq V(G)$, sodass gilt:

- $D$ ist in jedem $V_i$ enthalten
- $V_i \cap V_j = \{D\}$ für alle $i \neq j$,

- $b(V_i) \leq K$ für $i = 1, \ldots, k$,
- jeder Teilgraph $G[V_i]$ enthält eine Hamilton-Tour und
- $V_1 \cup \cdots \cup V_k = V(G)$.

Weiterhin sei $k_{min}$ das minimale $k$, sodass eine $k$-Partition von $(G, b, d, K)$ existiert.

Offenbar ist $k_{min}$ gerade die minimale Anzahl an Fahrzeugen, welche wir für eine gegebene Instanz benötigen. Falls der zugrundeliegende Graph vollständig ist (was wir in der Regel vorraussetzen), so ist $k_{min}$ die minimale Größe einer Bin-Packing Lösung mit den Gewichten $b(v)$ für $v \in V(G)$. Dieses Problem ist zwar NP-schwer, lässt sich aber sehr einfach approximieren und ist in der Praxis selbst für große $n$ noch exakt lösbar (siehe etwa [CJCG+13]). Daher gehen viele Algorithmen in der Literatur davon aus, dass man eine vorgegebene Anzahl von Fahrzeugen hat.

**Definition 4.** Sei $(G, b, d, K)$ eine Vehicle Routing Instanz und $k \geq k_{min}$ dann nennen wir $R = (G_1, \ldots, G_k)$ eine *$k$-Route* falls

- jedes $G_i$ ein Route ist und
- $V(G_1), \ldots, V(G_k)$ eine $k$-Partition ist.

Die *Länge* von $R$ sei $d(G_1) + \cdots + d(G_k)$.

| Vehicle Routing Problem (VRP) | |
|---|---|
| **Instanz:** | Eine Instanz $(G, b, d, C)$ des Vehicle Routing Problems. |
| **Aufgabe:** | Finde eine $k_{min}$-Route von minimaler Länge. |

In dieser Arbeit werden wir stehts davon ausgehen, dass $G$ ein vollständiger Graph ist und dass $d$ die Dreiecks-Ungleichung erfüllt. Wir betrachten nun noch kurz zwei alternative Formulierungen des Vehicle Routing Problems als ganzzahliges lineares Programm, da diese bei den üblichen Lösungsverfahren eine Rolle spielen.

### 1.2.2 Vehicle-Flow Formulierung

Sei $(G, b, d, C)$ eine Instanz des Vehicle Routing Problems und $k \geq k_{min}$. Die sogennante *Vehicle-Flow* Formulierung des VRP als ganzzahliges lineares Programm erhalten wir,

indem wir jeder Kante $e \in E(G)$ eine ganzzahlige Variable $x_e$ zuweisen:

$$\begin{aligned}
\text{(VRP1)} \quad \min_{x_e} \quad & \sum_{e \in E(G)} d(e) x_e && && \text{(1.1a)}\\
\text{s.t.} \quad & \sum_{e \in \delta(v)} x_e = 2 && \forall v \in V(G) \setminus \{D\}, && \text{(1.1b)}\\
& \sum_{e \in \delta(D)} x_e = 2k, && && \text{(1.1c)}\\
& \sum_{e \in \delta(S)} x_e \geq 2\nu(S) && \forall S \subseteq V(G) \setminus \{D\}, && \text{(1.1d)}\\
& x_e \in \{0,1\} && \forall e \notin \delta(D), && \text{(1.1e)}\\
& x_e \in \{0,1,2\} && \forall e \in \delta(D) && \text{(1.1f)}
\end{aligned}$$

Dabei sei $\nu(S) \in \mathbb{R}_+$ eine untere Schranke für die minimale Anzahl an Routen, die benötigt wird, um jeden Knoten in $S$ zu besuchen.

Eine Kante $e$ kann in der Regel nur einmal in einer Lösung vorkommen, außer wenn die Lösung eine Route enthält, die nur aus $e$ besteht. Dies wird durch die Bedingungen 1.1e und 1.1f bewerkstelligt. Die Nebenbedingungen 1.1b und 1.1c drücken aus, dass wir nach Kreisen suchen, von denen $k$ das Depot enthalten. Dazu kommen die exponentiell vielen Nebenbedingungen 1.1d, welche in der Literatur als *Capacity Cut Constraints (CCC)* bezeichnet werden. Sie garantieren sowohl den Zusammenhang der Lösung als auch das Einhalten der Kapazitätsschranke $K$. Details sind in [TV01] zu finden.

Die Qualität der LP-Relaxierung dieser Formulierung hängt stark von der Wahl von $\nu$ ab. Gängige Wahlen in der Literatur sind

$$\begin{aligned}
&\nu(S) \coloneqq 1, && \text{(1.2)}\\
&\nu(S) \coloneqq b(S)/K, && \text{(1.3)}\\
&\nu(S) \coloneqq \lceil b(S)/K \rceil \text{ oder} && \text{(1.4)}\\
&\nu(S) \coloneqq \min\{l \in \mathbb{N}_+ \mid \exists S_1 \cup \cdots \cup S_l = S \text{ mit } b(S_i) \leq K\}. && \text{(1.5)}
\end{aligned}$$

Die Definitionen 1.2 und 1.3 sind zwar relativ schwach, lassen sich aber in polynomieller Zeit separieren (siehe [ABB+98]). Für 1.4 und 1.5 ist das Separations-Problem dagegen deutlich schwieriger.

### 1.2.3 Set-Covering Formulierung

Eine weitere übliche Formulierung des Vehicle Routing Problems ist durch ein gewisses Set-Covering Problem gegeben. Sei $(G, b, d, K)$ eine Instanz und $k \geq k_{min}$, dann sei $\mathcal{R}$ die Menge aller Routen. Für jede Route $R \in \mathcal{R}$ führen wir eine Binärvariable $x_R$ ein. Die *Set-Covering* Formulierung des VRP ist dann durch das folgende ganzzahlige Lineare

Programm gegeben:

$$\text{(VRP2)} \quad \min_{x_R} \quad \sum_{R \in \mathcal{R}} d(R) x_R \tag{1.6a}$$

$$\text{s.t.} \quad \sum_{\substack{R \in \mathcal{R} \\ v \in V(R)}} x_R \geq 1 \qquad \forall v \in V(G) \setminus \{D\}, \tag{1.6b}$$

$$\sum_{R \in \mathcal{R}} x_R = k, \tag{1.6c}$$

$$x_R \in \{0, 1\} \quad \forall R \in \mathcal{R} \tag{1.6d}$$

Durch die Nebenbedingungen 1.6c und 1.6d wird ausgedrückt, dass eine Lösung aus $k$ Routen besteht. Die Überdeckungsbedingung 1.6b erzwingt zudem, dass jeder vom Depot verschiedene Knoten in mindestens einer Route enthalten ist. Da wir davon ausgehen, dass $G$ vollständig und $d$ eine Metrik ist, wird eine Optimallösung nie den gleichen Knoten in mehreren Routen enthalten.

Im Gegensatz zur Vehicle-Flow Formulierung aus dem vorherigen Abschnitt hat die Set-Covering Formulierung exponentiell viele Binärvariablen. Dafür hat ihre LP-Relaxierung in der Praxis eine deutlich geringere Ganzzahligkeitslücke.

## 1.3 Varianten des VRP

Das klassische Vehicle Routing Problem ist stark durch seine praktische Anwendung inspiriert. Allerdings sind realistische Vehicle Routing Modelle — etwa der innerstädtischen Paketlieferung — viel komplexer und haben oft dutzende bis hunderte zusätzliche Parameter und Nebenbedingungen. Sie bieten daher ständig Inspiriation für neue Varianten des mathematischen Optimierungsproblems. In den vergangenen fünfzig Jahren sind auf diese Weise unzählige Vehicle Routing Probleme formuliert worden, die in der Regel alle ihre eigenen Abkürzungen haben (VRP, CVRP, VRPTW, SPDP etc.). Ein paar für uns besonders relevante Varianten seien hier kurz umrissen:

- *Asymmetrie*: Der Graph $G$ kann durch einen Digraphen ersetzt werden. Dann wird in der Regel gefordert, dass $d$ eine Quasimetrik bzw. eine asymmetrische Metrik ist. Aus theoretischer Sicht ist dies zwar eine grundlegende Generalisierung des Problems (bereits das asymmetrische TSP ist deutlich schwieriger als das gewöhnliche TSP), allerdings können die meisten praktischen Algorithmen ohne Probleme mit Asymmetrie umgehen. Dies wird auch der Fall für die hier dargestellten Verfahren sein.

- *Längen- oder Zeitbegrenzungen*: Einige Varianten des VRP fügen zusätzliche Ressourcen ein — etwa die Gesamtlänge oder Dauer einer Tour —, welche bestimmten Schranken unterliegen. Falls sich der Rsesourcenverbrauch einer Tour aus dem Verbrauch von einzelnen Segmenten ergibt, lassen sich die meisten heuristischen und metaheuristischen Algorithmen (siehe nächster Abschnitt) auf solche Varianten des VRP erweitern.

- *Zeitfenster*: Bei dem VRPTW (*Vehicle Routing Problem with Time Windows*) bekommt jedes Paket in zugeschriebenes Zeitfenster $[l_p, u_p] \subseteq \mathbb{R}$. Eine Tour ist nur dann zulässig, wenn sie jedes ihrer Pakete innerhalb dieses Zeitfensters besucht, wobei eventuell Wartezeiten anfallen können. Das VRPTW ist deutlich schwieriger als das gewöhnliche Vehicle Routing Problem. Bereits das Problem zu entscheiden, ob es für eine vorgegebene Paketmenge eine zulässige Tour gibt, ist NP-schwer, da dieses das gewöhnliche TSP enthält (siehe [Sav85]).

- *Mehrere Depots*: Anstatt dass jedes Paket bei einem zentralen Depot startet, gibt es mehrere Depots und die Touren müssen zunächst an verschiedenen Depots vorbeifahren, bevor sie die eingesammelten Pakete schließlich ausliefern können. Sind zusätzlich gemischte Touren erlaubt, die gewisse Pakete einsammeln nachdem andere bereits ausgeliefert wurden, so wird vom VRPSPD (*Vehicle Routing Problem with Simultaneous Pickup and Delivery*) gesprochen. Beide diese Varianten sind ebenfalls erheblich schwieriger als das VRP und erfordern andere Lösungsansätze.

## 1.4 Lösungsverfahren für das VRP

Das klassische Vehicle Routing Problem und seine zahlreichen Varianten haben wegen ihrer großen theoretischen und praktischen Relevanz eine Vielzahl von Lösungsverfahren inspiriert. Diese Lösungsverfahren lassen sich grob in drei Klassen unterteilen: exakte Verfahren, Heuristiken und Metaheuristiken. Wir wollen uns nun einen Überblick über diese Verfahren verschaffen.

### 1.4.1 Exakte Verfahren

Exakte Lösungsverfahren zeichnen sich dadurch aus, dass sie in endlicher Zeit eine optimale Lösung des VRP berechnen können. Da das VRP NP-schwer ist, kann offenbar keines dieser Verfahren eine polynomielle Laufzeit vorweisen. Trotzdem sind sie deutlich effizienter als eine reine Enumeration des Lösungsraums und selbst Instanzen mit 200 Paketen können in der Praxis exakt gelöst werden.

Typischerweise basieren diese Verfahren auf einer der beiden MILP Formulierungen 1.1a oder 1.6a. Dabei werden klassische Verfahren wie *branch and bound* oder *branch and cut* verwendet. Bei MILP 1.6a muss zusätzlich Spaltengenerierung zum Einsatz kommen, da dieses exponentiell viele Variablen hat. Das entsprechende Subproblem ist ein gewöhnliches Traveling Salesman Problem, welches in der Praxis sehr gut gelöst werden kann. Schließlich gibt es auch Lösungsansätze, die auf dynamischer Programmierung beruhen.

In dieser Arbeit werden wir uns nicht weiter mit exakten Verfahren beschäftigen, da diese bei größeren Instanzen keine sinnvollen Ergebnisse liefern können. Dies gilt besonders bei Varianten des VRP mit vielen zusätzlichen Variablen und Nebenbedingungen, wie das Vehicle Routing Problem mit Subtouren und Zeitfenstern aus Kapitel 3. Außerdem wollen wir garantieren können, dass wir stets in relativ kurzer Zeit zumindest eine zulässige

Lösung finden. Exakte Methodne sind auf großen Instanzen allerdings oftmals nicht dazu in der Lage, innerhalb von Minuten oder Stunden eine solche zulässige Lösung zu finden.

### 1.4.2 Heuristische Verfahren

Die Klasse an Lösungsverfahren, mit der wir uns in dieser Arbeit hauptsächlich beschäftigen wollen, sind die klassischen Heuristiken. Darunter versteht man Lösungsansätze, die in (garantiert) kurzer Laufzeit eine zulässige Lösung garantieren. Über die Qualität dieser Lösung lässt sich zwar a priori nicht viel sagen, praktische Experimente zeigen jedoch, dass sich auf diese Weise konsistent Lösungen generieren lassen, die bis auf 5–20% an den Wert einer Optimallösung herankommen. Außerdem sind sie ein kritischer Bestandteil von vielen Metaheuristiken (siehe nächster Abschnitt), die noch kleinere Optimalitätslücken aufweisen können.

Unter den heuristischen Verfahren gibt es wiederum drei Arten:

- *Savings* Heuristiken starten mit einer trivialen zulässigen Lösung — etwa einer Tour für jeden Lieferpunkt — und kombinieren dann Touren, sodass die dadurch entstehenden Ersparnisse optimiert werden. Dieses Verfahren basiert auf der Arbeit [CW64] von Clarke und Wright.

- *Sequentielle Konstruktionsheuristiken* behalten stets eine *aktive* Tour und eine Menge von bereits abgearbeiteten Knoten. In jeder Iteration wird ein noch nicht abgearbeiteter Knoten gewählt und in die aktive Tour eingefügt. Falls dies nicht möglich ist, so wird eine neue aktive Tour aufgemacht und die alte Tour wird nicht mehr betrachtet. Der Ansatz geht auf die Arbeit [JM76] von Jameson und Mole zurück.

- *Parallele Konstruktionsheuristiken* arbeiten auf mehreren Touren gleichzeitig. Hierbei wird in jedem Schritt sowohl ein noch nicht abgearbeiteter Lieferpunkt als auch eine Tour gewählt, in die jener schließlich eingefügt wird. In der Regel beginnen diese Verfahren bereits mit einer (ebenfalls heuristisch gewählten) Menge an Touren. Eine besonders effektive Heuristik dieser Art wurde von Christofides et. al in [CMT13] angegeben.

Es hat sich gezeigt, dass parallele Konstruktionsheuristiken in der Praxis die besten Ergebnisse erzielen. Dies lässt sich unter anderem dadurch begründen, dass die anderen Verfahren dazu neigen, „schwierige“ Pakete möglichst lange aufzuschieben. Oftmals entstehen dadurch einige wenige sehr schlechte Touren.

### 1.4.3 Metaheuristiken

Neben den exakten und den heuristischen Lösungsverfahren haben sich in den letzten Jahren einige sogenannte Metaheuristiken etabliert. Eine Metaheuristik besteht dabei stets aus einem Verfahren, welches aus alten Lösungen neue Lösungen generiert, sowie einer Heuristik die entscheidet, welche Lösungen wir im Laufe des Algorithmus weiterverwenden wollen.

- *Evolutionäre Algorithmen* behalten stets eine Menge von Lösungen, die als *Population* bezeichnet wird. Eine Iteration eines solchen Algorithmus besteht darin, dass die Lösungen aus der aktuellen Population miteinander kombiniert und mutiert werden. Die so gefundenen Lösungen werden dann lokal nachoptimiert und schließlich wird nach einem heuristischen Verfahren entschieden, welche Lösungen aus der aktuellen Population entfernt werden, damit die Anzahl der Lösungen beschränkt bleibt.

- *Tabu-Suche* ist ein Verfahren, welches auf dem klassischen *Hill Climbing* Ansatz beruht: eine ursprüngliche Lösung wird generiert und mittels lokalen Austauschschritten greedy verbessert. In jeder Iteration werden dabei besondere Eigenschaften der letzten Lösung auf eine *Tabu-Liste* gesetzt, sodass wir vorerst nicht zu dieser Lösung zurückkehren können. Dies erlaubt es dem Verfahren aus lokalen Minima zu entkommen.

- Unter *Large Neighborhood Search* versteht man eine Reihe von Verfahren, die ebenfalls auf lokaler Suche basieren, dessen lokaler Suchraum jedoch exponentielle Größe hat. Dabei werden mit Hilfe eines Sampling-Verfahrens Nachbarn der aktuellen Lösung generiert. Ob ein solcher Nachbar die aktuelle Lösung ersetzt wird in der Regel durch simulierte Abkühlung entschieden: zunächst werden alle Lösungen akzeptiert und im Verlauf des Algorithmus sinkt die Wahrscheinlichkeit, dass wir den Lösungswert verschlechternde Lösungen akzeptieren. Bei *adaptive* LNS (siehe [PR07]) werden zudem mehrere Nachbarschaften betrachtet von denen in jeder Iteration eine (zufällig) gewählt wird abhängig davon, welche Nachbarschaft in den vergangenen Iterationen die besten Ergebnisse erzielt hat.

Zudem gibt es noch zahlreiche weitere Metaheuristiken wie etwa *Particle Swarm*, *Ant Colony* oder *Path Relinking* Heuristiken. Es hat sich gezeigt, dass Metaheuristiken in der Praxis sehr gute Ergebnisse erzielen können. Allerdings ist mit diesen Verfahren ein großer Tuning-Aufwand verbunden, da sie in der Regel höchst sensitiv bezüglich ihrer vielen Parameter sind. Algorithmen wie Adaptive Large Neighborhood Search können hier Abhilfe verschaffen, bieten jedoch keine vollständige Lösung für dieses Problem.

Viele Metaheuristiken müssen zudem unzulässige Lösungen während ihres Verlaufs erlauben, besonders für das Vehicle Routing Problem mit Zeitfenstern. Es gibt dann keine Garantie, dass das Verfahren jemals wieder zu einer zulässigen Lösung zurückkehrt, oder dass sich die unzulässigen Lösungen legalisieren lassen. In diesem Fall muss eine Fallback-Lösung verwendet werden und der gesamte Rechenaufwand geht verloren. Daher wollen wir zunächst eine möglichst gute Konstruktionsheuristik entwickeln, bevor eine metaheuristische Nachoptimierung in Frage kommen kann.

# Kapitel 2

# Subtouren

Traditionell funktioniert die Paketlieferung bereits in mehreren Phasen. Zwischen der Abholung des Pakets beim Versender (bzw. dessen Abgabe) und der Lieferung zum Endkunden wird ein Paket möglicherweise in mehreren Sortierzentren zwischengelagert und umgeladen. Ein solches Verfahren ermöglicht Konsolidierungseffekte auf mehreren Stufen, welche die Kosten der Lieferung im Vergleich zu einer Direktlieferung von Kunde zu Kunde deutlich senken. Insbesondere auf nationaler oder gar internationaler Ebene sind diese mehrstufigen Systeme daher sehr effizient.

Allerdings stoßen Systeme, die auf dem Zwischenlagern von Paketen beruhen, bei besonders zeitkritischen Lieferungen offenbar auf Schwierigkeiten. Besonders im innerstädtischen Raum werden zudem Eilbestellungen immer häufiger. Weiterhin kann dort die notwendige Infrastruktur für ein effizientes Verteilsystem sehr teuer sein. Daher haben sich in den letzten Jahren — besonders im nordamerikanischen Raum — eine Vielzahl von Startup-Unternehmen etabliert, die Direktlieferungen in Großstädten anbieten.

Eine Möglichkeit, um die Konsolidierungseffekte von Umverteilungen zumindest teilweise auf innerstädtische, zeitkritische Lieferungen zu erweitern, besteht darin, dass Pakete direkt zwischen verschiedenen Fahrzeugen ausgetauscht werden. So kann etwa eine Firma, die am Rand der Stadt einen Online-Shop betreibt, vormittags eine Reihe von LKWs vorfahren lassen. Diese befördern die Pakete dann in verschiedene Stadtteile und treffen sich dort mit kleineren Lieferfahrzeugen, in welche jeweils eine Ladung von Paketen umgeladen wird, bevor die eigentliche Lieferung zu den Endkunden beginnt.

Im Folgenden wollen wir das theoretische Problem — das Vehicle Routing Problem mit Subtouren — beschreiben und studieren, welches sich aus dem oben beschriebenen zwei-stufigen Liefersystem ergibt. Unser Ziel wird es sein, eine effiziente parallele Konstruktionsheuristik für das Problem zu entwickeln. Außerdem werden wir ein paar für die Praxis relevante Varianten betrachten.

## 2.1 Das Vehicle Routing Problem mit Subtouren

Analog zu dem gewöhnlichen Vehicle Routing Problem aus Kapitel 1 wollen wir jetzt das Vehicle Routing Problem mit Subtouren auf Graphen definieren. Dabei kommen nun

zwei verschiedene Arten von Fahrzeugen vor — Haupttour und Subtour Fahrzeuge — für die wir unterschiedliche Kosten und Kapazitäten betrachten wollen.

**Definition 5.** Eine *Instanz des VRPS* ist ein Tupel $(G, b, d, K, c)$, sodass gilt:

- $G$ ist ein endlicher und — so nehmen wir hier an — vollständiger Graph mit einem ausgezeichneten Knoten $D$,
- $b : V(G) \to \mathbb{R}_{\geq 0}$ mit $b(D) = 0$,
- $d = (d_m, d_s)$ mit $d_m : E(G) \to \mathbb{R}_{\geq 0}$ und $d_s : E(G - D) \to \mathbb{R}_{\geq 0}$ Metriken,
- $K = (K_m, K_s)$ mit $K_m, K_s \in \mathbb{R}_{\geq 0}$ und
- $c = (c_{mv}, c_{sv})$ mit $c_{mv}, c_{sv} \in \mathbb{R}_{\geq 0}$.

Wie bereits beim VRP heißt $b(v)$ die *Nachfrage* am Knoten $v$ und $D$ das *Depot*. Für jede Kante $e$ unterscheiden wir nun zusätzlich die *Subtour Kosten* $d_s(e)$ und die *Haupttour Kosten* $d_m(e)$. Die Komponenten $c_{mv}$ und $c_{sv}$ geben die Haupt- bzw. Subtour *Fixkosten* an. Schließlich nennen wir $K_m$ die *Haupttour Kapazität* und $K_s$ die *Subtour Kapazität*.

Im Folgenden fixieren wir eine konkrete Instanz $(G, b, d, K, c)$ des Vehicle Routing Problems mit Subtouren.

**Definition 6.** Wir nennen einen Weg $P$ in $G - D$ eine *Subtour*, falls der Gesamtbedarf entlang $P$ die Subtour Kapazität nicht überschreitet, also $b(P) \leq K_s$. Für disjunkte Subtouren $S_1, \ldots, S_k$ nennen wir einen Kreis $C$ in $G$ eine *Haupttour*, wenn gilt:

- $D$ liegt auf $C$,
- $b(S_1) + \cdots + b(S_k) \leq K_m$,
- $C$ enthält genau einen der beiden Endknoten von jedem $S_i$ und
- sonst enthält $C$ keine weiteren Knoten.

**Definition 7.** Für ein $k \in \mathbb{N}_+$ besteht eine *$k$-Route* aus $k$ Haupttouren $H_1, \ldots, H_k$ jeweils mit assoziierten Subtouren $S_{i,1}, \ldots, S_{i,l_i}$, sodass gilt:

- $V(H_i) \cap V(H_j) = \{D\}$ für $i \neq j$,
- $V(S_{i,j}) \cap V(S_{i',j'}) = \emptyset$ für $(i, j) \neq (i', j')$ und
- $\bigcup_{i=1}^{k} \bigcup_{j=0}^{l_i} S_{i,j} = V(G) \setminus \{D\}$.

Die *Kosten* einer solchen $k$-Route seien:

$$\sum_{i=1}^{k} \left( c_{mv} + d_m(H_i) + \sum_{j=1}^{l_i} (c_{sv} + d_s(S_{i,j})) \right).$$

Ein Beispiel für eine 2-Route ist Abbildung 2.1 zu sehen.

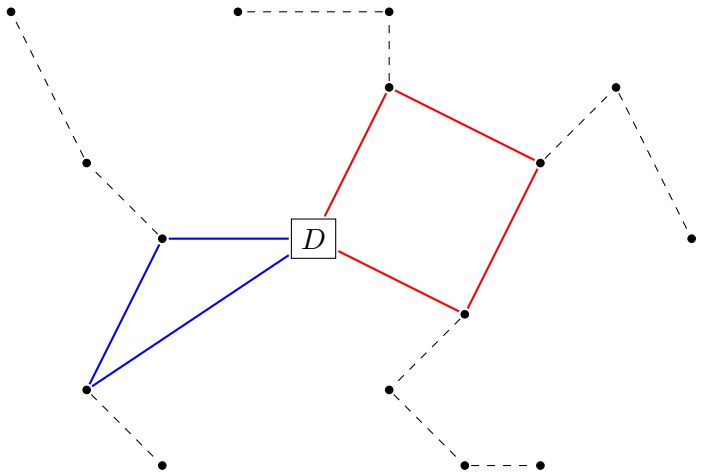



Abbildung 2.1: 2-Route auf 15 Punkten mit 5 Subtouren.

Mit Hilfe dieser Definitionen können wir nun eine formale Defintion des VRPS geben:

VEHICLE ROUTING PROBLEM MIT SUBTOUREN (VRPS)

**Instanz:** Gegeben sei eine Instanz $(G, b, d, K, c)$ des Vehicle Routing Problems mit Subtouren.

**Aufgabe:** Finde ein eine $k$-Route von minimalen Kosten für ein $k \in \mathbb{N}_+$.

### 2.1.1 Varianten des Problems

Wie beim klassischen Vehicle Routing Problem gibt es eine Vielzahl von möglichen Verallgemeinerungen, die wir am VRPS durchführen können. In dem nächsten Kapitel werden wir eine besonders wichtige Variante — das VRPS mit Zeitfenstern — betrachten. Außerdem sind folgende Verallgemeinerungen für die Praxis interessant:

- *Depots für Subtouren:* Subtouren starten und enden nicht an beliebigen Lieferpunkten sondern an einem von mehreren Subtour Depots. Dies macht das Problem zunächst nur unwesentlich schwieriger. Fügt man allerdings die zusätzliche Bedingung hinzu, dass an einem Subtour Depot jeweils nur beschränkt viele Fahrzeuge vorrätig sind — so ist es in der Praxis offenbar der Fall —, dann wird das Problem dadurch deutlich komplexer.

- *Übergabepunkte:* Übergaben von Haupttouren zu Subtouren finden nicht notwendigerweise an dem Startknoten der Subtour statt, sondern an beliebigen Übergabepunkten. Wir werden sehen, wie wir unseren Algorithmus auf diese Variante erweitern können. In der Praxis kann man dieses Problem allerdings auch einfach durch eine Nachoptimierung der Haupttouren lösen.

- *Deadlines:* Jede Aktion (fahren, einladen, entladen etc.) erhält gewisse Zeitkosten und muss vor einer globalen Deadline geschehen. Im Gegensatz zu der Zeitfenster Variante des VRPS lässt sich diese zusätzliche Einschränkung ebenfalls in unserem Algorithmus berücksichtigen.

- *Gemischte Lieferung:* Im Gegensatz zum VRPS dürfen hier nicht nur Subtour Fahrzeuge tatsächlich Pakete ausliefern, sondern dies kann auch von Haupttour Fahrzeugen ausgeführt werden. Dabei fallen eventuell unterschiedliche Kosten und Zeiten (bei Deadlines oder Zeitfenstern) an.

### 2.1.2 Verwandte Probleme aus der Literatur

Das hier studierte Vehicle Routing Problem mit Subtouren basiert auf dem ähnlichen Problem in der Arbeit [HKV17] von Held, Könemann und Vygen. Die Autoren geben darin einen $O(1+\epsilon, 1+1/\epsilon)$ Bikriterium-Algorithmus bezüglich des spätesten Lieferzeitpunkts und der Gesamtkosten an. Ein fundamentaler Unterschied besteht allerdings darin, dass wir nur eine Ebene von Subtouren erlauben. Außerdem betrachten wir verschiedene Fahrzeuge für Haupttouren und Subtouren.

In der Vehicle Routing Literatur gibt es zudem bereits einige studierte Probleme, bei denen mehrere unterschiedliche Fahrzeuge betrachtet werden, welche untereinander Pakete austauschen. Das klassische Beispiel ist das 2E-VRP (Two-Echelon Vehicle Routing Problem). Hierbei werden die Pakete zunächst von einem Depot zu mehreren Umladestationen (*satellites*) geliefert. An diesen Umladestationen werden dann wieder andere Fahrzeuge beladen, die schließlich die Lieferungen zu den Endkunden bringen. Ein Überblick über verschiedene Varianten dieses Problems ist in [CGS15] zu finden.

Mehrstufige Liefermodelle sind in der Logistik-Industrie de facto standard, sodass das 2E-VRP und seine Varianten in der Praxis sehr relevant sind. Allerdings benötigen diese Modelle einen vergleichsweise langen Planungshorizont. Ein Paket, welches vormittags erst eingesammelt wurde, kann kaum noch in einem Verteilzentrum umgeladen werden, wenn es noch am Nachmittag des selben Tages ausgeliefert werden soll. Insbesondere wird beim 2E-VRP implizit angenommen, dass die Fahrten der zweiten Phase erst stattfinden, nachdem alle Fahrten der ersten Phase abgeschlossen wurden. Zudem erfordert eine praktische Implementierung eines solchen Modells für innerstädtische Paketlieferung erhebliche Infrastrukturinvestitionen.

Eine weitere Variante von mehrstufigen Vehicle Routing Problemen ist durch das Routing von sogenannten *Carrier Vehicle Systems* gegeben. Bei einem Carrier Vehicle System oder CVS handelt es sich um ein Fahrzeug, welches ein anderes Fahrzeug befördern kann. Beispielsweise werden in [GNC11] Schiffe betrachtet, die einen Helikopter tragen. Besondere Relevanz haben solche CVS Routing Probleme in der Paketlieferung durch autonome Lieferfahrzeuge — inbesondere Drohnen — erhalten (siehe etwa [DHMM17] oder [WPG17]).

Eigenheiten des CVS Routing Problems sind dabei, dass die Subfahrzeuge stets wieder zu ihren Hauptfahrzeugen zurückkehren müssen und dass sie in der Regel nur sehr wenige Pakete — typischerweise sogar nur ein einziges — tragen können. Die gesuchten

Subtouren haben also eine deutlich andere Form als in dem hier studierten VRPS. Außerdem kann stets nur eine beschränkte Anzahl an Subtouren gleichzeitig aktiv sein, die von der Anzahl der getragenen Fahrzeuge — typischerfalls ebenfalls nur ein einziges — abhängt. Weiterhin müssen Drohnen-basierte Ansätze mit den Ladezeiten und niedrigen Kapazitätsbeschränkungen der Subfahrzeuge umgehen.

Schließlich wollen wir noch auf die Arbeit [KZL17] von Kafle, Zou und Lin eingehen. Darin wird ein dem VRPS sehr ähnliches Problem aufgestellt, bei dem es ebenfalls um innerstädtische Paketlieferung mit Übergaben geht. Die Subtouren werden dabei von Fußgängern oder Fahrradfahrern durchgeführt, die mittels eines Crowdsourcing-Modells entlohnt werden. Sie entstehen durch eine Art Auktion, bei denen die Interessenten jeweils endlich viele Gebote auf Paketmengen abgeben.

Zur Lösung dieses Problems verwenden die Autoren eine Tabu-Suche. Dabei wird abwechselnd ein kombinatorisches Auktionsproblem und ein Vehicle Routing Problem gelöst bzw. verbessert. Dieser Ansatz basiert offenbar darauf, dass die Anzahl der Subtour Gebote nicht zu groß ist. Daher schlagen die Autoren vor, dass jeder potentielle Subliferant nur eine beschränkte Anzahl an Geboten abgeben darf, da es unter Umständen für die Lieferanten optimal ist, exponentiell viele Gebote abzugeben.

An dieser Stelle findet sich dann auch der größte Unterschied zu dem in dieser Arbeit betrachteten Problem: die möglichen Subtouren sind bereits Teil der Eingabe und in ihrer Zahl beschränkt. Dadurch ergeben sich — besonders unter Hinzunahme von Zeitfenstern (siehe Kapitel 3) — nur geringe Vorteile gegenüber der klassischen Paketlieferung. Wir wollen stattdessen alle Subtouren zulassen, auch wenn es derer exponentiell viele gibt.

### 2.1.3 Approximierbarkeit des VRPS

Das Vehicle Routing Problem mit Subtouren verbindet ähnlich wie das klassische Vehicle Routing Problem NP-schwere Aspekte von Routing und Bin-Packing Problemen. Daher ist es nicht verwunderlich, dass das VRPS selbst in eingeschränkten Fällen nicht beliebig approximierbar ist.

**Satz 8.** *Das VRPS ist APX-schwer, selbst wenn keine Fixkosten vorkommen und die Fahrzeugkapazitäten unbeschränkt sind.*

*Beweis.* Wir reduzieren das metrische Weg-TSP mit einem festgehaltenen Endpunkt auf das VRPS. Dieses ist nach Appendix A von [HKV17] APX-schwer.

Sei $\alpha > 0$, $G$ ein vollständiger Graph mit Kantengewichten $c : E(G) \to \mathbb{R}_{\geq 0}$. Weiterhin sei $v_1 \in V(G)$ der festgehaltene Startpunkt und $M \in \mathbb{R}_{\geq 0}$ eine obere Schranke für einen kürzesten Hamilton-Pfad in $G$ beginnend mit $v_1$. Dann konstruieren wir eine Instanz des VRPS wie folgt:

- $G'$ ist ein vollständiger Graph auf $\{D\} \dot\cup V(G)$ mit dem ausgezeichneten Knoten $D$.
- $b(v) \coloneqq 1$ für alle $v \in V(G)$ und $b(D) \coloneqq 0$.
- $c_{mv} \coloneqq 0$ und $c_{sv} \coloneqq 0$.

- $d_m(\{D, v_1\}) \coloneqq 0$ und $d_m(e) \coloneqq (1 + 2\alpha)M$ für $e \neq \{D, v_1\}$ sowie $d_s \coloneqq c$.
- $K_m \coloneqq K_s \coloneqq |V(G)|$.

Angenommen wir können eine Lösung $L$ für die Instanz $(G', b, d, K, c)$ finden, welche bis auf den Faktor $1 + \alpha$ an das Optimum heran kommt. Diese Lösung kann nur die Kante $\{D, v_1\}$ für ihre Haupttouren verwenden, da die Kosten der Lösung sonst bereits $(1+2\alpha)M$ sind. Es gibt allerdings immer eine Lösung mit Kosten höchstens $M$ bestehend aus einer einzigen Haupttour $Dv_1D$ und einem kürzesten Hamilton-Pfad in $G$ beginnend mit $v_1$ als Subtour. Da verschiedene Haupttouren disjunkt sind, kann $L$ demnach ebenfalls nur diese Haupttour enthalten und daher auch nur eine einzige Subtour $S$ beginnend bei $v_1$. Die Kosten der Lösung $L$ ergeben sich genau aus den Wegkosten von $S$, sodass $S$ eine $1 + \alpha$ Approximation für die Instanz $(G, v_1, c)$ des metrischen Weg-TSP mit einem festgehaltenen Endpunkt liefert. □

## 2.2 Subtour Partitionierung

Unser Ziel ist es weiterhin, eine effiziente Heuristik zu entwickeln, welche auf den klassischen Konstruktionsheuristiken für das VRP beruht und in der Praxis gute Resultate erzielt. Das Kernstück dieser Heuristik wird ein schnelles Verfahren sein, welches aus einer Tour (im VRP-Sinne) eine Haupttour mit mehreren Subtouren erzeugt.

### 2.2.1 Partitionierung von Touren

Eine klassische Heuristik für das VRP besteht darin, zunächst eine große Tour durch alle Lieferpunkte zu bilden und diese dann in mehrere Touren zu unterteilen. Das Bestimmen einer solchen Tour lässt sich effizient mit bestehenden Heuristiken für das Traveling Salesman Problem — etwa die Lin-Kernighan Heuristik — lösen. Zudem kann das Bestimmen einer optimalen Partition mit Hilfe von dynamischer Programmierung ebenfalls effizient gelöst werden. Wir wollen ein ähnliches Verfahren anwenden, allerdings um Touren in Haupttouren mit Subtouren zu partitionieren.

**Definition 9.** Einen Kreis $C$ in $G$ nennen wir eine *Pre-Haupttour*, falls $C$ eine Tour in der VRP Instanz $(G, b, d_m, K_m)$ ist.

**Definition 10.** Sei $H$ eine Pre-Haupttour und $H'$ eine Haupttour mit Subtouren $S_1, \ldots, S_k$. Dann heißt $H'$ eine *Subtour Partition* von $H$, falls

- jedes $S_i$ ein Weg in $H$ ist und
- $H'$ seine Knoten in der selben Reihenfolge abläuft, wie diese in $H$ vorkommen.

Schießlich definieren wir das Problem der Subtour Partitionierung:

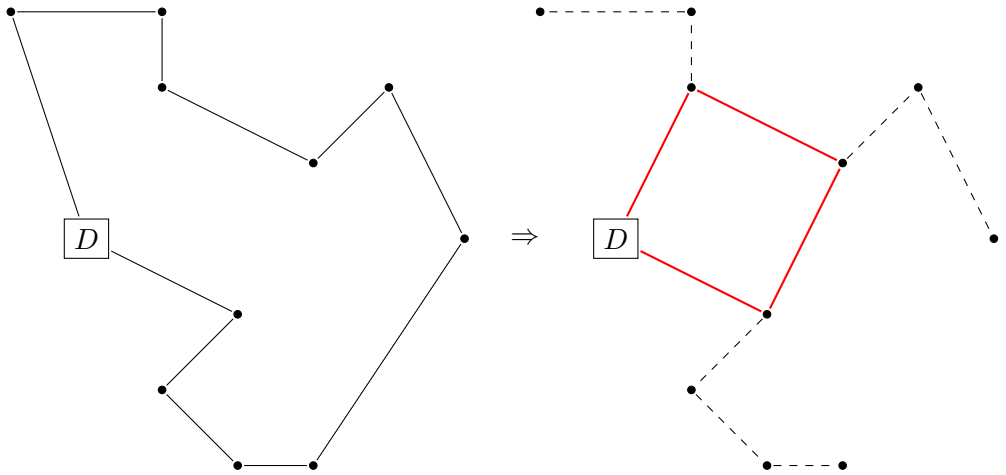



Abbildung 2.2: Beispiel für eine Subtour Partition mit 10 Knoten.

SUBTOUR PARTITIONIERUNG
**Instanz:** Gegeben sei eine Pre-Haupttour $H$.
**Aufgabe:** Finde eine Subtour Partition von $H$ mit minimalen Kosten.

Um dieses Problem zu lösen, definieren wir zunächst für eine Pre-Haupttour $C = Dv_1 \dots v_n D$ einen Hilfsgraphen $C'$ mit Kantengewichten $c'$. Dabei wählen wir zunächst für $C$ eine Orientierung und verwenden die Konvention $v_0 = v_{n+1} = D$. Wir bezeichnen mit $b_{i,j}$ den Gesamtbedarf und mit $s_{i,j}$ die Subtour-Länge entlang des $v_i$-$v_j$-Weges in $C$. Es seien

$$\begin{aligned} V(C') &:= \{v_i^o \mid 1 \le i \le n+1\} \cup \{v_i^c \mid 0 \le i \le n\}, \\ E(C') &:= \{(v_i^o, v_j^o) \mid 1 \le i < j \le n+1,\, b_{i,j-1} \le K_s\} \\ &\quad \cup \{(v_i^o, v_j^c) \mid 1 \le i < j \le n,\, \exists k \in [i,j).\, b_{i,k} \le K_s \text{ und } b_{k+1,j} \le K_s\} \\ &\quad \cup \{(v_i^c, v_j^c) \mid 0 \le i < j \le n,\, b_{i+1,j} \le K_s\} \cup \{(v_i^c, v_{i+1}^o) \mid 0 \le i \le n\}. \end{aligned}$$

Die Kantengewichte $c' : E(C') \to \mathbb{R}_{\ge 0}$ seien gegeben durch:

$$c'(v_i^o, v_j^o) := c_{sv} + d_m(v_i, v_j) + s_{i,j-1} \tag{2.1}$$

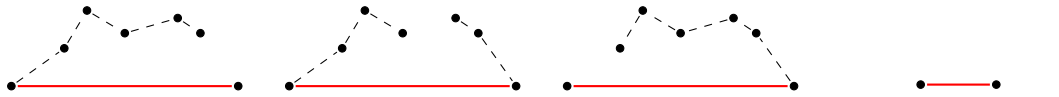
$$\begin{aligned} c'(v_i^o, v_j^c) &:= 2c_{sv} + d_m(v_i, v_j) \\ &\quad + \min\{s_{i,k} + s_{k+1,j} \mid i \le k < j,\, b_{i,k} \le K_s,\, b_{k+1,j} \le K_s\} \end{aligned} \tag{2.2}$$

$$c'(v_i^c, v_j^c) := c_{sv} + d_m(v_i, v_j) + s_{i+1,j} \tag{2.3}$$

$$c'(v_i^c, v_{i+1}^o) := d_m(v_i, v_{i+1}) \tag{2.4}$$

Die Knoten $v_i^c$ und $v_i^o$ sind dabei als Stopps der Haupttour zu interpretieren, an denen wir entweder bereits eine Subtour abgespaltet haben (*closed*) oder noch nicht (*open*).

Damit entsprechen die vier Arten von Kanten in $C'$ den vier möglichen Maneuvern, die wir ausführen können, um eine bestehende Haupttour um einen Stopp zu erweitern. Diese sind in Abbildung 2.3 illustriert.



(a) $e = (v_i^o, v_j^o)$ (b) $e = (v_i^o, v_j^c)$ (c) $e = (v_i^c, v_j^c)$ (d) $e = (v_i^c, v_{i+1}^o)$

Abbildung 2.3

**Lemma 11.** *Jeder $v_0^c$-$v_{n+1}^o$-Weg in $C'$ entspricht einer Subtour Partition von $C$ und umgekehrt. Zudem entspricht die Länge eines solchen Weges den Kosten der zugehörigen Partion ohne die Fixkosten der Haupttour.*

*Beweis.* Sei $P$ ein Weg in $C'$, der mit $v_0^c$ beginnt. Wir definieren rekursiv eine Menge $S(P)$ von Subtouren und eine Haupttour $H(P)$. Falls $P$ nur aus $v_0^c$ besteht, so sei $S(P) \coloneqq \emptyset$ und $H(P) \coloneqq \{D\}$. Ansonsten sei $e$ die letzte Kante auf $P$ und $Q$ das Anfangsstück von $P$ bis zu dieser Kante. Wir unterscheiden die vier Arten von Kanten:

- Maneuver 2.3a bzw. $e = (v_i^o, v_j^o)$:

$$\begin{aligned} S(P) &\coloneqq S(Q) \cup \{C_{v_i, v_{j-1}}\} \\ H(P) &\coloneqq H(Q) \cup \{v_j\}. \end{aligned}$$

- Maneuver 2.3b bzw. $e = (v_i^o, v_j^c)$ — sei $k$ so gewählt, dass das Minimum in (2.2) angenommen wird:

$$\begin{aligned} S(P) &\coloneqq S(Q) \cup \{C_{v_i, v_k}, C_{v_{k+1}, v_j}\} \\ H(P) &\coloneqq H(Q) \cup \{v_j\}. \end{aligned}$$

- Maneuver 2.3c bzw. $e = (v_i^c, v_j^c)$:

$$\begin{aligned} S(P) &\coloneqq S(Q) \cup \{C_{v_{i+1}, v_j}\} \\ H(P) &\coloneqq H(Q) \cup \{v_j\}. \end{aligned}$$

- Maneuver 2.3d bzw. $e = (v_i^c, v_{i+1}^o)$:

$$\begin{aligned} S(P) &\coloneqq S(Q) \\ H(P) &\coloneqq H(Q) \cup \{v_{i+1}\}. \end{aligned}$$

Offenbar sind die Kanten von $C'$ gerade so gewählt worden, dass alle resultierenden Subtouren tatsächlich die Fahrzeugkapazitäten einhalten. Außerdem wurden die Kosten $c'$ so definiert, dass diese mit den Gesamtkosten von $H(P)$ und $S(P)$ übereinstimmen.

Dass sich zudem alle Subtour Partitionen auf diese Weise darstellen lassen, sehen wir leicht: Zwischen zwei Stopps der Haupttour können 0, 1 oder 2 Subtouren liegen. Diese entsprechen den vier Maneuvern aus Abbildung 2.3 und jedes dieser Maneuver erhält eine entsprechende Kante in dem Graphen $C'$. □

**Satz 12.** *Das Subtour Partitionierungs Problem lässt sich in $O(n^2)$ Laufzeit lösen.*

*Beweis.* Da $C'$ ein gerichteter azyklischer Graph ist, lässt sich das kürzeste Wege Problem in $C'$ in linearer Laufzeit lösen. Hieraus ergibt sich also $O(|V(C')|+|E(C')|) = O(n+n^2) = O(n^2)$ Laufzeit. Danach können wir offenbar den gefundenen Weg wie in Lemma 11 zu einer Subtour Partition umbauen.

Allerdings ist nicht unbedingt klar, ob sich der Graph selber und insbesondere die Kostenfunktion $c'$ ebenfalls in geeigneter Laufzeit bestimmen lassen. Alle Werte $b_{i,j}$ und $s_{i,j}$ lassen sich in einer Präprozessierung in $O(n^2)$ Laufzeit berechnen. Die Frage ist, wie wir mit Kanten der Art $(v_i^o, v_j^c)$ — also Maneuver 2.3b — und deren Kosten umgehen können.

Es genügt offenbar — sowohl für das Bestimmen der Kosten als auch für die Rekonstruktion der Subtour Partition — für jedes Paar $(v_i, v_j)$ mit $i < j$ die schwerste Kante auf dem $v_i$-$v_j$-Weg in $C$ zu kennen, an der wir diesen Weg in zwei zulässige Subtouren teilen können, oder zu entscheiden, dass es keine solche Kante gibt. Für festes $i$ verwenden wir Algorithmus 1, um dies zu bewerkstelligen.

**Algorithmus 1** Präprozessierung für Maneuver 2.3b

① Bestimme die schwerste Kante für alle $j$ mit $\sum_{l=i}^{j} b(v_l) \leq K_s$.

② Setze $j \leftarrow n$.

③ Bestimme

$$k_{min} \coloneqq \min\{k \geq i \mid \sum_{l=k+1}^{j} b(v_l) \leq K_s\},$$
$$k_{max} \coloneqq \max\{k \leq j \mid \sum_{l=i}^{k} b(v_l) \leq K_s\}.$$

④ Falls $k_{min} = i$ bzw. $k_{max} = j$, so sind wir fertig.

⑤ Falls $k_{min} > k_{max}$, so lässt sich der $v_i$-$v_j$-Weg nicht in zwei zulässige Subtouren teilen. Ansonsten bestimme eine schwerste Kante des $v_{k_{min}}$-$v_{k_{max}}$-Weges in $C$.

⑥ Setze $j \leftarrow j - 1$ und gehe zu Schritt ③.

Schritt ① ist einfach in $O(n)$ Laufzeit zu implementieren: Wir erhöhen stets $j$ und

merken uns die bisher schwerste gefundene Kante. Falls die Kante $(v_j, v_{j+1})$ schwerer ist, so wird diese zu unserer neuen schwersten Kante.

Weiterhin bemerke man, dass $k_{max}$ sich in Schritt (3) nie ändern wird. Dagegen kann $k_{min}$ in jeder Iteration (nur) kleiner werden und diese Veränderungen können in insgesamt linearer Laufzeit berechnet werden. Schritt (5) lässt sich ebenfalls in linearer Laufzeit implementieren, da in jeder Iteration nur die neuen Kanten zwischen $k_{min}$ und dem $k_{min}$ der letzten Iteration betrachtet werden müssen (analog zu Schritt (1)). Also lassen sich die Maneuver 2.3b entsprechenden Kanten und ihre Kosten in $O(n^2)$ Laufzeit berechnen. □

### 2.2.2 Varianten der Partitionerung

Das Verfahren zur Partitionierung von Subtouren lässt sich auf verschiedene Varianten des VRPS erweitern. Von besonderem Interesse sind hierbei das Einhalten einer Deadline sowie eine Variante, bei der wir Subtouren an dedizierten Übergabepunkten abspalten müssen.

Bei dem VRPS mit einer *Deadline* fallen für jede Aktion (Fahrten, Übergaben und Auslieferungen) Zeitkosten an. Außerdem gibt es einen Zeitpunkt $u \in \mathbb{R}$, sodass kein Paket später als $u$ ausgeliefert werden darf. Dieses Problem lässt sich ebenfalls als ein kürzeste Wege-Problem darstellen, allerdings als eine Variante des Problems mit Ressourcen-Beschränkungen. Zwar ist dieses selbst auf azyklischen Graphen NP-schwer — denn sie enthält das Knapsack Problem (siehe [Jaf84]) —, es kann allerdings ein Labelling-Verfahren verwendet werden, welches in Algorithmus 2 implementiert ist. Damit sind praktische Instanzen mit $n \leq 250$ trotzdem in unter 100 Millisekunden lösbar.

**Algorithmus 2** Subtour Partitionierung mit Deadlines

$\mathcal{O}_0 \leftarrow \{(\emptyset, 0, 0)\}, \quad \mathcal{C}_0 \leftarrow \emptyset$
**for** $i \leftarrow 1, \ldots, n+1$ **do**
  **for** $s \in \bigcup_{j<i} \mathcal{O}_j$ **do**
    $\mathcal{O}_i \leftarrow \mathcal{O}_i \cup \{extend_{o,o}(s, i)\}$
    $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup \{extend_{o,c}(s, i)\}$
  **for** $s \in \bigcup_{j<i} \mathcal{C}_j$ **do**
    $\mathcal{O}_i \leftarrow \mathcal{O}_i \cup \{extend_{c,o}(s, i)\}$
    $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup \{extend_{c,c}(s, i)\}$
  $\mathcal{O}_i \leftarrow filter(\mathcal{O}_i), \quad \mathcal{C}_i \leftarrow filter(\mathcal{C}_i)$
**return** $\mathcal{O}_{n+1}$

Wir behalten stets *offene* Label $\mathcal{O}_i$ und *abgeschlossene* Label $\mathcal{C}_i$. Dabei handelt es sich um Tripel $(h, t, c)$ mit $h$ einer partiellen Haupttour mit Subtouren, $t$ der Zeit, die das Haupttour Fahrzeug bisher benötigt hat, sowie $c$ den Kosten der Lösung, sodass der letzte Haupttour-Stopp bei $v_i$ liegt. Ein Label heißt dann abgeschlossen, wenn der Punkt $v_i$ bereits beliefert wurde und sonst heißt es offen. Die Funktionen $extend_{o/c,o/c}(s, i)$ erweiterten eine gegeben partielle Lösung $s$ optimal zu einer Lösung, bei der das Haupttour Fahrzeug bis $v_i$ gefahren ist. Dazu werden die Maneuver aus Abbildung 2.3 verwendet.

Insbesondere muss für Maneuver 2.3b abhängig von der Ankunftszeit der Haupttour eine optimale Aufteilung des übersprungenen Segments gefunden werden. Schließlich wird in jeder Iteration eine Funktion $filter$ auf die Label-Mengen angewandt, welche dominierte oder nicht zulässige Label entfernt.

**Satz 13.** SUBTOUR PARTITIONIERUNG *lässt sich unter Betrachtung einer Deadline in* $O(n^3k)$ *Laufzeit lösen, wobei* $k$ *die maximale Anzahl an nicht-dominierten, partiellen Lösungen ist.*

*Beweis.* Die Korrektheit von Algorithmus 2 folgt daraus, dass partielle Subtour Partitionen die Bellman-Bedingung erfüllen: jede Pareto-optimale Subtour Partition bis zu einem Punkt $v_i$ in der Tour setzt sich aus einer Pareto-optimalen Subtour Partition bis $v_j$ und einer optimalen Partition des $v_i$-$v_j$-Weges in Subtouren zusammen. Für die Laufzeit ist zu bemerken, dass sich die Funktionen $extend_{o/c,o/c}$ in $O(n)$ implementieren lassen. Diese werden $O(n^2k)$ mal aufgerufen, sodass sich eine Gesamtlaufzeit von $O(n^3k)$ ergibt. □

Es ist zu bemerken, dass nur Maneuver 2.3b tatsächlich lineare Laufzeit benötigt, da wir die optimalen Partitionen der entsprechenden Intervalle nicht wie in Satz 12 durch eine Präprozessierung bestimmen können. Dies liegt daran, dass die genaue Partition von der Startzeit abhängen kann. Allerdings können in der Praxis die gefundenen Partitionen zwischengespeichert und wiederverwendet werden, sodass in der Regel nur 3–6 „echte“ Aufrufe von $extend_{o,c}$ pro Iteration auftreten. Zudem wächst die Zahl der nicht-dominierten Lösungen $k$ auf realen Instanzen etwa linear (bis sie irgendwann beschränkt ist), sodass insgesamt eine Laufzeit von $O(n^3)$ beobachtet wird.

Eine andere ebenfalls für die Praxis relevante Variante der Subtour Partitionierung wird für das VRPS mit Übergabepunkten benötigt. Bei diesem Problem besteht der Graph $G$ aus

- dem Depot $D \in V(G)$,
- einer Menge $U \subseteq V(G)$ von *Übergabepunkten* und
- einer Menge $P \subseteq V(G)$ von *Lieferpunkten*.

Die Haupttouren bewegen sich dann nur noch in $G[\{D\} \cup U]$ und jede Subtour $S$ ist ein Weg in $G[U \cup P]$ mit genau einem Endpunkt in $U$. Für einen Knoten $v \in P$ bezeichnen wir mit $\mathcal{N}_m(v) \subseteq U$ die Menge der $m$ nächsten Übergabepunkte zu $v$. Außerdem sei $\mathcal{N}_m(D) \coloneqq \{D\}$. Wir verändern unsere Definition einer Subtour Partition wie folgt:

**Definition 14.** Sei $H$ eine Pre-Haupttour — das heißt eine Tour in der VRP Instanz $(G[\{D\} \cup P], b, d_m, K_m)$ — und $H'$ eine Haupttour mit Subtouren $S_1, \ldots, S_n$. Dann heißt $H'$ eine $m$-Subtour-Partition von $H$, falls

- jedes $S_i[P[P]]$ ein Weg in $H$ ist und
- für jede Kante $(v, w) \in E(H')$ gibt es Knoten $v', w' \in V(H)$ mit $v \in \mathcal{N}_m(v')$ und $w \in \mathcal{N}_m(w')$, sodass $v'$ in der Tour $H$ vor $w'$ liegt.

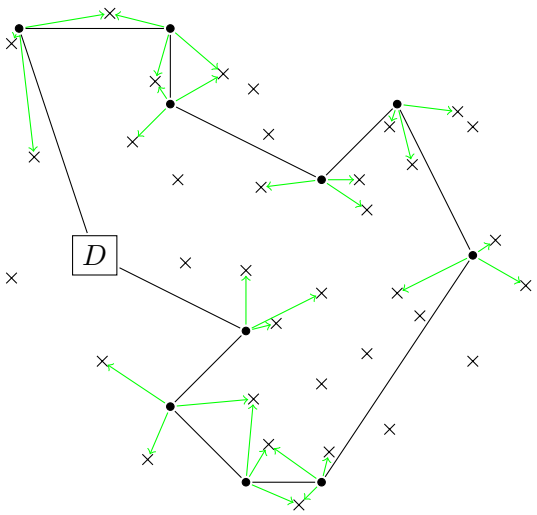



Abbildung 2.4: Gewählte Übergabepunkte für eine Pre-Haupttour mit $m = 3$.

Für eine gegebene Pre-Haupttour sollen die Übergabepunkte einer entsprechenden Subtour Partition also in einer Art Mantel um die Lieferpunkte der Pre-Haupttour liegen. In Abbildung 2.4 ist eine solche Wahl von Übergabepunkten verdeutlicht. Eine Subtour Partition mit diesen ist in Abbildung 2.5 dargestellt. In der Praxis hat sich gezeigt, dass bereits $m = 3$ zu sehr guten Ergebnissen führt. Zur Lösung des Partitionierungsproblems konstruieren wir einen Hilfsgraphen $C'$ wie folgt:

$$\begin{aligned} V(C') &:= \{v_i^{(j,u)} \mid 1 \le i \le n,\ 1 \le j \le n,\ u \in \mathcal{N}_m(v_j)\} \cup \{v_0^{(0,D)}, v_n^{(n+1,D)}\}, \\ E(C') &:= \{(v_i^{(j,u)}, v_i^{(j',u')}) \mid 0 \le i \le n,\ 0 \le j < j' \le n+1\} \\ &\quad \cup \{(v_i^{(j,u)}, v_{i'}^{(j,u)}) \mid 0 \le i < i' \le n,\ 1 \le j \le n,\ b_{i+1,i'} \le K_s\} \end{aligned}$$

mit der Kostenfunktion $c' : E(C') \to \mathbb{R}$ gegeben durch

$$\begin{aligned} c'(v_i^{(j,u)}, v_i^{(j',u')}) &:= d_m(u, u'), \\ c'(v_i^{(j,u)}, v_{i'}^{(j,u)}) &:= c_{sv} + \sum_{l=i+1}^{i'-2} d_s(v_l, v_{l+1}) + \min\{d_m(u, v_{i+1}), d_m(u, v_{i'})\}. \end{aligned}$$

**Satz 15.** *Gegeben eine Pre-Haupttour $H$, so lässt sich in $O(n^3 m)$ Laufzeit eine optimale $m$-Subtour-Partition von $H$ berechnen.*

*Beweis.* Analog zu Lemma 11 entsprechen $v_0^{(0,D)}$-$v_n^{(n+1,D)}$-Wege in $C'$ gerade den $m$-Subtour-Partitionen von $H$. Da $|V(C')| = O(n^2 m)$ und $|E(C')| = O(n^3 m)$ sowie $C'$ azyklisch ist, lässt sich ein solcher Weg in $O(n^3 m)$ Zeit mittels dynamischer Programmierung bestimmen. □

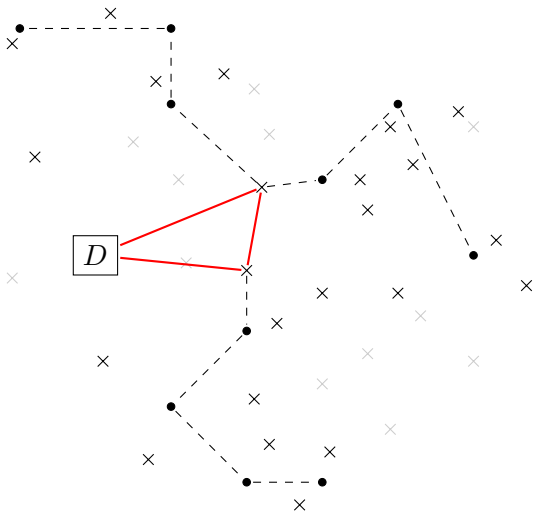



Abbildung 2.5: Subtour Partition für die Tour aus Abbildung 2.4.

## 2.3 Ein paralleles Konstruktionsverfahren für das VRPS

Mit Hilfe des Verfahrens für Subtour Partitionierung aus den letzten Abschnitten definieren wir nun ein einfaches paralleles Konstruktionsverfahren. Die Idee ist hierbei, dass wir eine Menge von Haupttouren gleichzeitig aufbauen und dabei stets nach neuen Subtour Partitionen suchen, die unsere derzeitige Lösung verbessern. Implementiert ist dieser Ansatz in Algorithmus 3.

Schritt ① ist kritisch für die Qualität des Algorithmus und zudem auch nicht trivial. In der Regel werden wir den gesamten Algorithmus mehrfach neu starten und dabei stets ein neues $k$ und neue Starttouren wählen. Wir werden hier allerdings nicht genauer auf dieses Verfahren eingehen — eine Exposition findet sich in [Erl17].

Die Wahl eines Pakets in Schritt ② kann auf verschiedene Weisen geschehen. So könnte man zum Beispiel stets greedy das Paket wählen, welches sich am besten in eine bestehende Tour einfügen lässt. Allerdings hat diese Vorgehensweise den Effekt, dass die letzten Touren nur noch die „schwierigen“ Pakete bekommen und daher sehr schlecht werden. Wir werden stattdessen stets das Paket wählen, das die Differenz zwischen den Kosten des Einfügens in die beste und in die zweitbeste Tour maximiert. Dabei verwenden wir zugunsten der Laufzeit nicht die echten Kosten, sondern lediglich eine obere Schranke mittels einer Cheapest-Insert Heuristik.

In dem Fall, dass wir in Schritt ③ keine Haupttour finden, in die sich das Paket einfügen lässt, so erstellen wir eine neue Haupttour, indem wir ein Paket wählen, welches am weitesten von den bestehenden Touren entfernt ist. Damit maximieren wir heuristisch die Anzahl der Pakete, die danach wieder in eine Tour eingefügt werden können. Ansonsten treffen wir in Schritt ④ die offensichtliche Wahl und nehmen stets die beste Tour, in die das Paket eingefügt werden kann.

Mit „optimal“ ist in Schritt ⑤ gemeint, dass wir das Paket per Cheapest-Insert

**Algorithmus 3** VRPS Parallele Konstruktion

① Bestimme $k$ Starttouren $\mathcal{R}$ und setze $P \leftarrow V(G) \setminus (\{D\} \cup \bigcup \mathcal{R})$.

② Wähle ein noch nicht geclustertes Paket $p \in P$.

③ Falls sich $p$ in keine bestehende Haupttour einfügen lässt, so erstelle eine neue Haupttour und gehe zu Schritt ⑧.

④ Wähle eine bestehende Haupttour $R \in \mathcal{R}$, sodass sich $p$ in $R$ einfügen lässt.

⑤ Sei $R'$ die Tour die aus $R$ hervorgeht, wenn man $p$ optimal in $R$ einfügt.

⑥ Bestimme eine Tour $R''$ die dadurch entsteht, dass zunächst auf $V(R')$ ein TSP gelöst wird und die resultierende TSP-Tour dann optimal in Subtouren partitioniert wird.

⑦ Ersetze $R$ durch die bessere der beiden Touren $R'$ und $R''$ und setze $P \leftarrow P \setminus \{p\}$.

⑧ Falls $P \neq \emptyset$, gehe zu Schritt ②.

an die günstigste Position einfügen. Diese haben wir bereits bestimmt, um die Wahlen in Schritt ② und ④ zu tätigen. Danach wenden wir in Schritt ⑥ eine Lin-Kernighan Heuristik sowie unsere Subroutine zur Partitionierung von Subtouren aus Abschnitt 2.2 an. Gegebenenfalls betrachten wir dabei Deadlines oder Übergabepunkte wie in Abschnitt 2.2.2 beschrieben.

Wenn wir in Schritt ⑧ schließlich keine noch nicht geclusterten Pakete mehr übrig haben, so iterieren wir gegebenfalls mit einem größeren $k$ und neuen Starttouren (siehe oben). Ansonsten sind wir fertig — aus den verschiedenen Iterationen haben wir unterschiedliche Lösungen $\mathcal{R}$ erhalten, von denen wir die beste zurückgeben.

# Kapitel 3

# Zeitfenster

In den vorherigen Kapiteln haben wir sowohl das klassische Vehicle Routing Problem, als auch eine neue Variante mit Subtouren betrachtet. Diese Probleme haben gemeinsam, dass die Zulässigkeit einer potentiellen Lösung nur von dem Einhalten der Fahrzeugkapazitäten abhängig ist. Weiterhin haben wir eng verwandte Probleme untersucht, die zusätzliche Einschränkungen, wie eine globale Deadline oder maximale Fahrzeiten, mit sich bringen. Dabei haben wir gesehen, dass sich diese zusätzlichen Bedingungen in der Regel leicht in bestehende Algorithmen einführen lassen.

Realistische Routing-Modelle, wie sie in der Logistik-Industrie zum Einsatz kommen, haben im Vergleich zu den stark abstrahierten Modellen aus der theoretischen Literatur allerdings noch viele weitere Variablen und Einschränkungen, die berücksichtigt werden müssen. Eine besonders wichige Erweiterung ist das Einführen von individuellen *Zeitfenstern*. Diese geben bereits dem Vehicle Routing Problem eine völlig neue Komponente der *Zeitplanung* (*Scheduling*), welche zu den bestehenden Komponenten des *Assignment* und des *Routing* hinzukommt.

Tatsächlich verändert das Hinzufügen von Zeitfenstern die Problemstellung so stark, dass in der Regel völlig andere Algorithmen zum Einsatz kommen müssen. In diesem Kapitel werden wir das VRPS aus Kapitel 2 zu dem VRPSTW (Vehicle Routing Problem with Subtours and Time Windows) erweitern. Unser Ziel wird es sein, eine zwei-phasige Heuristik vorzustellen, welche dieses Problem effizient lösen kann.

## 3.1 Das Vehicle Routing Problem mit Subtouren und Zeitfenstern

Im Folgenden wollen wir eine Formulierung des VRPSTW als Optimierungsproblem angeben. Da das Einhalten von Zeitfenstern von der Orientierung der Touren abhängt, werden wir nun stets davon ausgehen, dass unsere Graphen *gerichtet* sind. Ansonsten erweitern wir im Grunde nur die Problemstellung aus Abschnitt 2.1.

**Definition 16.** Eine *Instanz des VRPSTW* besteht aus einer Instanz $(G, b, d, K, c)$ des Vehicle Routing Problems mit Subtouren und zusätzlich

- Kostenfaktoren $c_{sf}$ und $c_{mf}$ für die von den Subtour und Haupttouren verbrachten Arbeitszeiten,
- Zeitfenster $[l_v, u_v] \subseteq \mathbb{R}$ für alle Pakete $v \in V(G) \setminus \{D\}$ und
- einer Übergabezeit $\theta \in \mathbb{R}_{\geq 0}$ pro Paket.

Es lassen sich zusätzlich auch individuelle Lieferzeiten für jedes Paket betrachten. Diese können jedoch auch in die Fahrzeiten der Subtouren sowie die Zeitfenster eingerechnet werden. Weiterhin kann man die Varianten aus Abschnitt 2.1.1 — insbesondere Subtour Depots, gemischte Lieferungen und Übergabepunkte — ebenfalls im Kontext von Zeitfenstern studieren. Wir werden uns allerdings nur mit dem folgenden Problem beschäftigen:

Vehicle Routing mit Subtouren und Zeitfenstern (VRPSTW)

**Instanz:** Gegeben sei eine Instanz des Vehicle Routing Problems mit Subtouren und Zeitfenstern.

**Aufgabe:** Finde

- eine gerichtete $k$-Route für die zugrundeliegende VRPS Instanz mit Haupttouren $H_1, \ldots, H_k$ und Subtouren $S_1, \ldots, S_l$ für ein $k \in \mathbb{N}_+$,
- Zeitpläne $x_i \in S(S_i)$ für alle Subtouren (siehe Definition 17) und
- Zeitpläne $y_i \in S(H_i)$ für die Haupttouren, wobei die Zeitfenster an den Übergaben jeweils durch $[-\infty, x_{j,1} - \theta|V(S_j)|]$ gegeben sind,

sodass die Gesamtkosten gegeben durch

$$\sum_{i=1}^{k} (c_{mf} + \delta(y_i)c_{mf}) + \sum_{i=1}^{l} (c_{sv} + \delta(x_i)c_{sf})$$

minimiert werden.

Bevor wir eine Heuristik für das VRPSTW angeben können, werden wir uns in den nächsten beiden Abschnitten mit den zusätzlichen Schwierigkeiten beschäftigen, die das Einhalten von Zeitfenstern mit sich bringt.

## 3.2 Effiziente Kostenberechnung mit Zeitfenstern

Für zahlreiche Heuristiken ist es notwendig, dass wir die durch lokale Änderungen verursachten Kosten effizient berechnen können. Anders als bei dem klassichen VRP und

dem VRPS, haben solche lokalen Änderungen unter dem Einfluss von Zeitfenstern oft globale Auswirkungen. So entstehen etwa zusätzliche Wartezeiten oder ein späteres Paket kann nun nicht mehr rechtzeitig ausgeliefert werden. Mit ein wenig zusätzlichem Aufwand und Speicherverbrauch lassen sich diese Effekte allerdings trotzdem effizient berechnen.

Im Folgenden betrachten wir stets einen gerichteten Graphen $G$ mit Fahrzeiten $c : E(G) \to \mathbb{R}_{\geq 0}$ und Zeitfenstern $[l_v, u_v] \subseteq \mathbb{R}$ für jeden Knoten $v \in V(G)$.

**Definition 17.** Sei $P = v_1 e_1 \dots e_{n-1} v_n$ ein Weg in $G$, dann besteht ein *Zeitplan* für $P$ aus einer Folge $x_1, \dots, x_n$ von *Ankunftszeiten*. Einen solchen Zeitplan nennen wir zulässig, falls

1. $x_i$ für alle $i$ in $[l_i, u_i]$ liegt und
2. $x_i + c(v_i, v_{i+1}) \leq x_{i+1}$ für $i < n$.

Die Differenz $\delta(x) \coloneqq x_n - x_1$ heißt das *Delay* des Zeitplans $x$. Schließlich bezeichnen wir die Menge aller zulässigen Zeitpläne für $P$ mit $S(P)$.

Offenbar ist $S(P)$ für jeden Weg ein Polyeder, sodass wir über diesen optimieren können. Das Bestimmen von optimalen Zeitplänen bezüglich Delay und Startzeit ist das Problem der Zeitplanung:

ZEITPLANUNG

**Instanz:** Gegeben sei ein Weg $P$ in $G$.
**Aufgabe:** Finde einen zulässigen Zeitplan $x$ mit minimalem Delay und (unter diesen) minimaler Startzeit $x_1$ oder entscheide, dass es keinen solchen gibt.

Betrachtet man einen Weg $P$ als „black box“, so ist die genaue Verteilung von Ankunftszeiten nicht sonderlich relevant. Wichtig ist lediglich, wie viel Delay man für $P$ aufwenden muss, wenn man zu einem bestimmten Zeitpunkt beginnt. Dies wird durch die folgende Funktion modelliert:

$$\delta_P(t) \coloneqq \min\{x_n - t \mid x \in S(P),\, t \leq x_1\}.$$

Dabei setzen wir $\delta_P(t) = \infty$, wenn $\{x \in S(P) \mid t \leq x_1\}$ leer ist. Ansonsten ist die Existenz des Minimums garantiert, da $S(P)$ ein Polyeder ist. Mit Hilfe der Funktion $\delta_P$ lässt sich eine andere Formulierung des Zeitplanungs-Problems geben:

ZEITPLANUNG (2)

**Instanz:** Gegeben sei ein Weg $P$ in $G$.
**Aufgabe:** Bestimmte $\min_{t \in \mathbb{R}}(\delta_P(t), t)$ nach lexikographischer Ordnung.

**Lemma 18.** *Sei $P$ ein Weg und $t \in \mathbb{R}$, dann hat der durch*

$$\begin{aligned} x_1 &\coloneqq \max\{t, l_1\} \\ x_{i+1} &\coloneqq \max\{x_i + c(v_i, v_{i+1}), l_{i+1}\} \end{aligned}$$

*rekursiv definierte Zeitplan folgende Eigenschaften:*

- *$x$ ist genau dann zulässig, wenn es ein $y \in S(P)$ mit $t \leq y_1$ gibt.*
- *$x_i = \min\{y_i \mid y \in S(P),\, t \leq y_1\}$ falls $x$ zulässig ist.*

*Beweis.* Sei $y \in S(P)$ mit $t \leq y_1$. Um die beiden Aussagen zu beweisen, genügt es zu zeigen, dass $x_i \leq y_i$ gilt für alle $i$. Wir beweisen dies per Induktion über $i$.

Fall $i = 1$: Es gilt $t \leq y_1$ nach Vorraussetzung und $l_1 \leq y_1$ damit $y$ zulässig ist. Demnach folgt $x_1 = \max\{t, y_1\} \leq y_1$.

Fall $i \rightsquigarrow i + 1$: Da $x_i \leq y_i$ und da $y$ Eigenschaften 1 und 2 aus Definition 17 erfüllt gilt:

$$\begin{aligned} x_{i+1} &= \max\{x_i + c(v_i, v_{i+1}), l_{i+1}\} \\ &\leq \max\{y_i + c(v_i, v_{i+1}), l_{i+1}\} \\ &\leq y_{i+1}. \end{aligned}$$

□

Man bemerke, dass sich demnach aus einer Lösung $(\delta_P(t), t)$ von ZEITPLANUNG (2) sehr einfach eine Lösung $x$ von ZEITPLANUNG berechnen lässt. Falls $\delta_P(t) = \infty$ ist, so gibt es keinen zulässigen Zeitplan für $P$. Ansonsten definieren wir $x$ nach Lemma 18. Dieser Zeitplan nimmt dann maximal das Delay $\delta_P(t)$ an und minimiert dabei $x_1$. Da $(\delta_P(t), t)$ eine Lösung von ZEITPLANUNG (2) ist, nimmt $x$ das global minimale Delay an.

**Lemma 19.** *Für jeden Weg $P$ gibt es Zahlen $l_P, u_P, d_P \in \mathbb{R}$ mit $l_P \leq u_P$ und $d_P \geq 0$, sodass*

$$\delta_P(t) = \max\{l_P - t, 0\} + d_P$$

*für $t \leq u_P$ und $\delta_P(t) = \infty$ sonst. Insbesondere wird das gesuchte Minimum in* ZEITPLANUNG (2) *stets bei $(d_P, l_P)$ angenommen.*

*Beweis.* Wähle

$$\begin{aligned} d_P &\coloneqq \min\{x_n - x_1 \mid x \in S(P)\}, \\ l_P &\coloneqq \min\{x_1 \mid x \in S(P),\, x_n - x_1 = d_P\}, \\ u_P &\coloneqq \max\{x_1 \mid x \in S(P)\}. \end{aligned}$$

Offenbar gilt $\delta_P(t) = \infty$ genau dann, wenn $t > u_P$. Weiterhin gilt $[l_P, u_P] \subseteq [l_1, u_1]$. Sei nun also $t \leq u_P$, dann betrachten wir die Fälle $t \leq l_P$ und $t > l_P$ getrennt.

Fall $t \leq l_P$: Nach Definition von $l_P$ gibt es einen Zeitplan $y$ mit $y_1 = l_P$, sodass $y$ das minimale Delay $d_P$ annimmt. Dieser erfüllt auch $y_1 \geq t$, sodass

$$\delta_P(t) \leq y_n - t = (y_n - y_1) + (y_1 - t) = d_P + l_P - t.$$

Wegen Minimalität von $l_P$ muss es ein $i \in [n]$ mit $y_i = l_i$ geben. Nun definieren wir einen für die Startzeit $t$ optimalen Zeitplan $x$ wie in Lemma 18. Dann gilt nach eben diesem Lemma $x_i \leq y_i$ und damit $x_i = y_i$. Demnach haben wir

$$\begin{aligned}\delta_P(t) &= x_n - t = (x_n - x_i) + (x_i - t) \geq (y_n - y_i) + (y_i - t) \\ &= y_n - t = d_P + l_P - t,\end{aligned}$$

wobei $x_n - x_i \geq y_n - y_i$ daraus folgt, dass $(y_1, \ldots, y_{i-1}, x_i, \ldots, x_n)$ sonst ein Zeitplan von Delay geringer als $d_P$ wäre.

Fall $t > l_P$: Sei $x$ ein optimaler Zeitplan mit $x_1 = u_P$ und $y$ ein optimaler Zeitplan mit $y_1 = l_P$ (von diesem wissen wir, dass $y_n - y_1 = d_P$). Wir wollen per Induktion beweisen, dass $x_i - x_1 \leq y_i - y_1$. Für den Induktionsanfang gilt offenbar $x_1 - x_1 = 0 = y_1 - y_1$. Ansonsten haben wir

$$\begin{aligned}x_{i+1} - x_1 &= \max\{x_i + d(v_i, v_{i+1}), l_{i+1}\} - x_1 \\ &= \max\{x_i - x_1 + d(v_i, v_{i+1}), l_{i+1} - x_1\} \\ &\leq \max\{y_i - y_1 + d(v_i, v_{i+1}), l_{i+1} - y_1\} \\ &= \max\{y_i + d(v_i, v_{i+1}), l_{i+1}\} - y_1 \\ &= y_{i+1} - y_1.\end{aligned}$$

Dementsprechend nimmt der Zeitplan $x$ ebenfalls ein Delay von $d_P$ an. Sei nun $t \in [l_P, u_P]$, dann wählen wir zunächst $\theta \in [0, 1]$, sodass $t = \theta l_P + (1 - \theta)u_P$. Die Konvexkombination $z := \theta y + (1 - \theta)x$ ist dann ebenfalls ein zulässiger Zeitplan, da $S(P)$ ein Polyeder ist und

$$\begin{aligned}z_n - z_1 &= (\theta y_n + (1 - \theta)x_n) - (\theta y_1 + (1 - \theta)x_1) \\ &= \theta(y_n - y_1) + (1 - \theta)(x_n - x_1) \\ &= d_P.\end{aligned}$$

Also gilt $\delta_P(t) \leq d_P$ und damit $\delta_P(t) = d_P$ wie zu zeigen war. □

Es genügt also für unsere Zwecke (die Berechnung der Kosten von Wegen), die Zahlen $l_P$, $u_P$ und $d_P$ ausrechnen zu können. Für Wege, die nur aus einem einzigen Knoten $v$ bestehen, gilt $l_P = l_v$, $u_P = u_v$ und $d_P = 0$. Alle anderen Wege lassen sich als Konkatenationen von kürzeren Wegen darstellen, was das folgende Lemma nützlich macht:

**Lemma 20.** *Seien $P$ und $Q$ Wege die durch eine Kante $e$ verbunden werden. Dann besitzt $P +\!\!+ Q$ genau dann einen zulässigen Zeitplan, wenn $l_P + d_P + c(e) \leq u_Q$. In diesem Fall gilt:*

$$u_{P+\!\!+Q} = \min\{u_P, u_Q - (d_P + c(e))\}, \tag{3.1}$$

$$d_{P+\!\!+Q} = d_P + c(e) + \max\{l_Q - (l_P + d_P + c(e)), 0\} + d_Q, \tag{3.2}$$

$$l_{P+\!\!+Q} = \min\{u_{P+\!\!+Q}, \max\{l_P, l_Q - (d_P + c(e))\}\}. \tag{3.3}$$

*Beweis.* Aus Lemma 19 ergibt sich, dass $l_P + d_P$ der frühstmögliche Zeitpunkt ist, bei dem wir mit $P$ fertig sein können. Demnach besitzt $P +\!\!+ Q$ genau dann einen zulässigen Zeitplan, wenn $l_P + d_P + c(e) \leq u_Q$ (Lemma 19 auf $Q$). Nun gehen wir davon aus, dass $S(P +\!\!+ Q)$ nicht leer ist. Außerdem sei $|V(P +\!\!+ Q)| = n$ und $|V(P)| = l$.

Für Gleichung (3.1) betrachten wir einen zulässigen Zeitplan $x$ für $P +\!\!+ Q$. Offenbar ist $(x_1, \ldots, x_l) \in S(P)$ und damit gilt $x_1 \leq u_P$. Außerdem folgt

$$x_1 + d_P + c(e) \leq x_l + c(e) \leq x_{l+1} \leq u_Q.$$

Demnach haben wir $x_1 \leq \min\{u_P, u_Q - (d_P + c(e))\}$. Man sieht leicht, dass der Zeitplan mit gerade diesem Startzeitpunkt zulässig ist, sodass Gleichung (3.1) folgt.

Für die anderen beiden Aussagen untersuchen wir die Delay-Funktion $\delta_{P+\!\!+Q}$. Ein optimaler Zeitplan $x \in S(P +\!\!+ Q)$ mit $t \leq x_1$ hat offenbar folgende Eigenschaften:

- $(x_1, \ldots, x_l) \in S(P)$ ist optimal mit $t \leq x_1$.
- $(x_{l+1}, \ldots, x_n) \in S(Q)$ ist optimal mit $x_l + c(e) \leq x_{l+1}$.

Daraus ergibt sich zunächst:

$$\begin{aligned} \delta_{P+\!\!+Q}(t) &= \delta_P(t) + c(e) + \delta_Q(t + \delta_P(t) + c(e)) && (3.4)\\ &= \max\{l_P - t, 0\} + d_P + c(e) \\ &\quad + \max\{l_Q - (t + \max\{l_P - t, 0\} + d_P + c(e)), 0\} + d_Q. && (3.5) \end{aligned}$$

Diesen Ausdruck vergleichen wir nun mit Lemma 19, welches uns sagt, dass $d_{P+\!\!+Q}$ gerade das Minimum dieser Funktion ist, welches bei $u_{P+\!\!+Q}$ angenommen wird. Daher gilt:

$$\begin{aligned} d_{P+\!\!+Q} &= \delta_{P+\!\!+Q}(u_{P+\!\!+Q}) = \delta_{P+\!\!+Q}(\min\{u_P, u_Q - (d_P + c(e))\}) \\ &= d_P + c(e) + d_Q + \max\{l_Q - (\min\{u_P, u_Q - (d_P + c(e))\} + d_P + c(e)), 0\} \\ &= d_P + c(e) + d_Q + \max\{l_Q - \min\{u_P + d_P + c(e), u_Q\}, 0\} \\ &= d_P + c(e) + d_Q + \max\{l_Q - (u_P + d_P + c(e)), 0\}. \end{aligned}$$

Damit ist bereits Aussage (3.2) gezeigt. Für Gleichung (3.3) müssen wir nach Lemma 19 das kleinste $t$ finden, sodass $\delta_{P+\!\!+Q}$ sein Minimum annimmt. Dafür müssen die drei Maxima in Gleichung (3.5) verschwinden oder $t$ muss maximal — also auf $u_{P+\!\!+Q}$ — gesetzt werden. Demnach haben wir

$$l_{P+\!\!+Q} = \min\{u_{P+\!\!+Q}, \max\{l_P, l_Q - (d_P + c(e))\}\}. \qquad \square$$

**Korollar 21.** ZEITPLANUNG (2) *lässt sich in $O(|V(P)|)$ — also linearer Laufzeit — lösen. Insbesondere können die Kosten eines Weges unter Berücksichtigung von Wartezeiten in linearer Zeit berechnet werden.*

*Beweis.* Wir zeigen per Induktion, dass sich $l_P$, $u_P$ und $d_P$ in linearer Zeit berechnen lassen. Falls $P$ nur aus einem einzigen Knoten $v$ besteht, so gilt $l_P = l_v$, $u_P = u_v$ und $d_P = 0$. Ansonsten zerteilen wir $P$ in einen Knoten $v$ und ein Endstück $Q$ von $P$ mit $P = v +\!\!+ Q$. Dann können wir nach Lemma 20 die Werte $l_P$, $u_P$ und $d_P$ in konstanter Zeit aus $l_Q$, $u_Q$ und $d_Q$ berechnen. Diese lassen sich per Induktionsvoraussetzung in $O(|V(Q)|)$ bestimmen. $\square$

**Korollar 22.** *Sei $P$ ein Weg, $v \in V(G)$ und $P'$ ein Weg der durch das Einfügen von $v$ in $P$ ensteht. Dann können wir $O(1)$ entscheiden, ob $P$ zulässig ist und in diesem Fall die Kosten berechnen, die durch das Einfügen entstehen.*

*Beweis.* Es gibt es Wege $P_{<v}$ und $P_{>v}$ mit $P' = P_{<v} + v + P_{>v}$. Dementsprechend können wir $l_{P'}$, $u_{P'}$ und $d_{P'}$ in $O(1)$ Zeit berechnen, falls wir die entsprechenden Werte für $P_{<v}$ und $P_{>v}$ kennen, sowie nach Lemma 20 auf Zulässigkeit testen. Diese können für alle Knoten auf $P$ in $O(|P|)$ Gesamtzeit berechnet und im Laufe unserer Algorithmen stets zwischengespeichert werden. □

Analog zu den vorherigen beiden Ergebnissen lassen sich eine Vielzahl von lokalen Veränderungen (etwa das Austauschen ganzer Segmente) ähnlich effizient berechnen, wie es ohne jegliche Zeitfenster möglich wäre. Wir können — und werden — also in der Regel davon ausgehen, dass das Berücksichtigen von Zeitfenstern keinen zusätzlichen Rechenaufwand beansprucht.

## 3.3 Zeitplanung mit schwachen Zeitfenstern

Im letzten Abschnitt haben wir einzelne Touren mit *starken* Zeitfenstern, also Zeitfenstern die auf keinen Fall verletzt werden dürfen, betrachtet. Nun wollen wir uns mit *schwachen* Zeitfenstern beschäftigen, welche — unter entsprechenden Strafkosten — verletzt werden dürfen. Es gibt dabei eine Vielzahl von gängigen Kostenmodellen (siehe Abbildung 3.1). In vielen Anwendungen wird Modell (a) verwendet, welches wir hier jedoch nicht betrachten werden, da wir auf keinen Fall Pakete nach Ablauf ihres Zeitfensters ausliefern wollen. Im Falle des Depots werden wir stets von harten Zeitfenstern ausgehen.

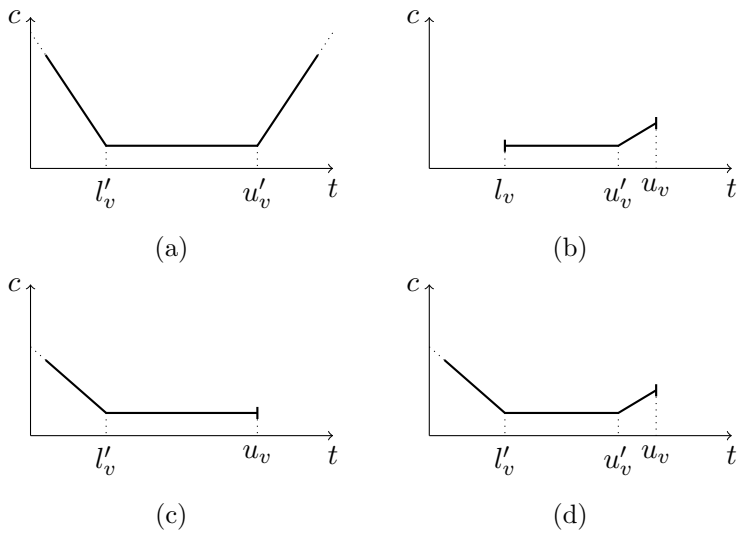



Abbildung 3.1: Dargestellt sind verschiedene Kostenfunktionen für schwache Zeitfenster.

Für die Zeitfenster an den Lieferpunkten im VRPSTW ist besonders Modell (b) interessant, da wir hiermit die Robustheit der Lösung gegenüber externen Störfaktoren (Stau, Umleitungen, Verzögerungen bei der Paket-Auslieferung etc.) steigern können. Schließt man mehrere Pakete zu einer Subtour zusammen, so entsteht dabei allerdings auch wieder ein Zeitfenster, nämlich an der Übergabe. Dieses kann von der Haupttour nach unten beliebig verletzt werden, indem das dort abgespaltene Subtour Fahrzeug entsprechend vor Beginn seiner Fahrt wartet. Dafür fallen dann also Wartekosten bei der Subtour an, welche durch Modell (c) erfasst werden können. Schließlich kann man auch hier Strafkosten betrachten, was zu Modell (d) führt. Insgesamt ergibt sich folgendes Optimierungsproblem:

Schwache Zeitplanung

**Instanz:** Gegeben seien jeweils Wegzeiten $t_1, \ldots, t_{n-1} \in \mathbb{R}_{\geq 0}$, Handlungszeiten $s_1, \ldots, s_n \in \mathbb{R}_{\geq 0}$ sowie konvexe schwache Zeitfenster $c_i : [l_i, u_i] \to \mathbb{R}_{\geq 0}$.

**Aufgabe:** Finde eine Lösung für das folgende Optimierungsproblem unter lexikographischer Ordnung:

$$\min_{x_1,\ldots,x_n} \quad \left( \sum_{i=1}^{n} c_i(x_i + s_i) + x_n - x_1, x_1 \right) \tag{3.6a}$$

$$\text{s.t.} \quad x_i + s_i + t_i \leq x_{i+1} \quad \forall i < n, \tag{3.6b}$$

$$l_i \leq x_i \quad \forall i, \tag{3.6c}$$

$$x_i + s_i \leq u_i \quad \forall i. \tag{3.6d}$$

Ein Ansatz für dieses Problem findet sich bei Sexton und Choi (siehe [SC86]), welche ein Problem mit *symmetrischen* Strafkosten nach Modell (a) in linearer Zeit lösen. Dafür wird das Problem zunächst als ein LP formalisiert, dessen Duales einem Netzwerk-Fluss Problem auf einem speziellen Graphen entspricht. Dieses kann dann entsprechend in $O(n)$ gelöst werden.

Dumas et al. reduzieren in [DSD90] die Zeitplanung mit beliebigen konvexen Funktionen auf $n$ eindimensionale Optimierungsprobleme. Sie lösen dabei zunächst eine relaxierte Variante des Problems ohne die Nebenbedinungen (3.6b). Diese werden dann sukzessive als Gleichungen wieder in das Problem eingefügt. Für lineare und quadratische Strafkosten ergibt sich daraus ebenfalls ein Algorithmus mit linearer Gesamtlaufzeit.

Die quadratischen und stückweise linearen Fälle treten zudem auch unter einem anderen Namen im VLSI-Design auf. Dort ist das entsprechende Problem als das Single Row Placement Problem mit vorgegebener Zell-Reihenfolge bekannt. Der Algorithmus von Dumas et al. ist in diesem Kontext als Clumping Algorithmus bekannt und wurde in [KTZ99] von Kahng, Tucker und Zelikovsky wiederentdeckt. Eine effiziente Implementierung dieses Algorithmus wird in [BV00] beschrieben.

Weiterhin lässt sich das Problem auch mittels einer Diskretisierung der möglichen Startzeiten an jedem Knoten lösen. So geht zum Beispiel Fagerholt in [Fag01] vor. Dadurch können allgemeine Kostenfunktionen berücksichtigt werden und das Problem reduziert sich auf ein kürzeste Wege Problem.

### 3.3.1 Präprozessierung

Das Problem Schwache Zeitplanung enthält einige überflüssige Parameter, die wir zunächst eliminieren wollen. Mit Hilfe einer linearen Präprozessierung sollen folgende Vereinfachungen bewerkstelligt werden:

1. $t_1 = \ldots = t_{n-1} = 0$,
2. $s_1 = \ldots = s_n = 0$,
3. $u_i \leq u_{i+1}$ für $i = 1, \ldots, n-1$.

Sei $(t, s, c)$ also eine Instanz der schwachen Zeitplanung, dann definieren wir zunächst Ankunftszeiten $a_1, \ldots, a_n$ durch die Rekursion:

$$\begin{aligned} a_1 &:= 0 \\ a_{i+1} &:= a_i + s_i + t_i. \end{aligned}$$

Nun definieren wir eine modifizierte Instanz $(t', s', c')$ durch

- $t'_1 := \ldots := t'_{n-1} := 0$ und $s'_1 := \ldots := s'_n := 0$,
- $l'_i := l_i - a_i$ und $u'_i := u_i - (a_i + s_i)$ sowie
- $c'_i : [l'_i, u'_i] \to \mathbb{R}_{\geq 0}$ mit $c'_i(t) := c'_i(t + (a_i + s_i))$.

**Lemma 23.** *Sei $x \in \mathbb{R}^n$ dann ist $x$ genau dann ein optimaler schwacher Zeitplan für $(t, s, c)$, wenn $x - a$ ein optimaler schwacher Zeitplan für $(t', s', c')$ ist.*

*Beweis.* Sei $x \in \mathbb{R}^n$ eine zulässige Lösung des Optimierungsproblems (3.6). Zunächst sehen wir, dass das modifizierte Problem bis auf einen konstanten Term die selbe Zielfunktion optimiert, wenn man um $-a$ verschiebt:

$$\sum_{i=1}^{n} c_i(x_i + s_i) + x_n - x_1 = \sum_{i=1}^{n} c'_i(x_i - a_i) + (x_n - a_n) - (x_1 - a_1) + (a_n - a_1).$$

Es genügt also zu zeigen, dass der Lösungsraum ebenfalls nur um $-a$ verschoben ist. Für die Nebenbedinungen (3.6b) gilt

$$\begin{aligned} x_i + s_i + t_i \leq x_{i+1} &\Leftrightarrow x_i + s_i + t_i - a_{i+1} \leq x_{i+1} - a_{i+1} \\ &\Leftrightarrow x_i + s_i + t_i - (a_i + s_i + t_i) \leq x_{i+1} - a_{i+1} \\ &\Leftrightarrow x_i - a_i \leq x_{i+1} - a_{i+1} \end{aligned}$$

und für das Einhalten der Zeitfenster gilt

$$x_i \in [l_i, u_i - s_i] \Leftrightarrow x_i - a_i \in [l_i - a_i, u_i - (a_i + s_i)]. \qquad \square$$

Die modifizierte Instanz $(t', s', c')$ erfüllt Eigenschaften 1 und 2, aber möglicherweise noch nicht Eigenschaft 3. Um dies zu bewerkstelligen, itererieren wir noch einmal rückwärts über die Instanz, ohne dabei den Lösungsraum zu verändern:

$$u''_n \coloneqq u'_n$$
$$u''_{i-1} \coloneqq \min\{u'_{i-1}, u''_i\}$$

Damit ist im Folgenden nur noch folgendes Problem zu betrachten:

SCHWACHE ZEITPLANUNG (2)

**Instanz:** Gegeben seien konvexe schwache Zeitfenster $c_i : [l_i, u_i] \to \mathbb{R}_{\geq 0}$.

**Aufgabe:** Finde eine Lösung für das folgende Optimierungsproblem unter lexikographischer Ordnung:

$$\min_{x_1,\dots,x_n} \quad \left(\sum_{i=1}^{n} c_i(x_i) + x_n - x_1, x_1\right) \tag{3.7a}$$
$$\text{s.t.} \quad x_i \leq x_{i+1} \quad \forall i < n, \tag{3.7b}$$
$$x_i \geq l_i \quad \forall i, \tag{3.7c}$$
$$x_i \leq u_i \quad \forall i. \tag{3.7d}$$

### 3.3.2 Kostenmodell (c)

Im Kostenmodell (c) darf jedes Paket beliebig früh ausgeliefert werden. Es sind daher keine Wartezeiten *notwendig*, um einen zulässigen Zeitplan zu konstruieren. Allerdings können möglichst frühe Wartezeiten die Gesamtkosten der Tour senken, wenn dadurch spätere Zeitfenster besser eingehalten werden.

Wir fixieren im Folgenden eine Instanz von SCHWACHE ZEITPLANUNG (2), welche nun nur noch durch $c_i : [l_i, u_i] \to \mathbb{R}_{\geq 0}$ für $i = 0, \dots, n$ gegeben ist. Dabei sei $c_0$ konstant 0 (dies entspricht dem Depot) und $l_i = -\infty$ für $i > 0$. Jedes $c_i$ habe die Form $c_i(t) = \max\{\alpha_i(t - l'_i), 0\}$ mit $\alpha_i < 0$.

Zur Lösung des Problems verwenden wir Algorithmus 4, welcher aus zwei Phasen besteht. Zunächst berechnen wir für jeden Punkt $i \in [n]$ in der Tour den spätesten Zeitpunkt $\pi_i$ bis zu dem es sich lohnen würde, zu warten. Hierfür verwenden wir einen Min-Heap $H$ mit der Ordnung $i \prec j \Leftrightarrow l'_i < l'_j$. Schließlich erzeugen wir einen Zeitplan, indem wir stets so lange wie möglich warten, oder bis wir $\pi_i$ erreichen.

**Satz 24.** *Algorithmus 4 liefert in $O(n \log(k))$ Laufzeit einen optimalen Zeitplan, wobei $k$ die maximale Größe des Heaps $H$ ist.*

*Beweis.* Zunächst stellen wir fest, dass wir — da $c_0$ konstant 0 ist — $x_0$ so spät wie möglich wählen sollten. Allerdings wollen wir unter den Lösungen mit den geringsten

**Algorithmus 4** Zeitplanung (c)

1: $H \leftarrow \emptyset$ ▷ 1. Phase
2: **for** $i \leftarrow n, \dots, 1$ **do**
3: push$(H, i)$
4: **if** $\sum_{j \in H} \alpha_j < -1$ **then**
5: **while** $\sum_{j \in H} \alpha_j < -1 + \alpha_{\text{peek}(H)}$ **do**
6: pop$(H)$
7: $\pi_i \leftarrow l'_{\text{peek}(H)}$
8: **else**
9: $\pi_i \leftarrow -\infty$
10: $x_0 \leftarrow \min\{u_0, \max\{l'_1, \dots, l'_n\}\}$ ▷ 2. Phase
11: **for** $i \leftarrow 1, \dots, n$ **do**
12: $x_i \leftarrow \min\{u_i, \max\{x_{i-1}, \pi_i\}\}$
13: **return** $x_0, \dots, x_n$

Kosten stets den frühsten Zeitplan finden. Daher setzen wir $x_0$ in Zeile 10 korrekterweise nur so spät wie das späteste $l'_i$, denn danach machen wir in Modell (c) keinen Gewinn mehr.

Nun übelegen wir uns induktiv, wie spät wir $x_i$ wählen sollten. Angenommen wir haben $x_{i-1}$ bereits gewählt. Offenbar sollten wir $x_i$ genau dann später wählen als $x_{i-1}$, wenn

$$\sum_{\substack{j \geq i \\ l'_j > x_{i-1}}} \alpha_j < -1.$$

Nur dann können wir an der Stelle $i$ Gewinn durch unser Warten erzielen, welches uns im Endeffekt Einheitskosten kosten wird. Die Werte $\pi_i$ werden in der ersten Phase des Algorithmus auf folgende Werte gesetzt:

$$\pi_i = \sup\{t \in \mathbb{R} \mid \sum_{\substack{j \geq i \\ l'_j > t}} \alpha_j < -1\}.$$

Nach der obigen Überlegung wählen wir also $x_i$ in Zeile 12 korrekt, indem wir es so lange erhöhen bis wir $\pi_i$ oder den Rand des Zeitfensters erreichen.

Für die Laufzeit stellen wir fest, dass Algorithmus 4 bis auf die Heap-Operationen in Zeile 3 und Zeile 6 in $O(n)$ Zeit läuft. Dabei wird Zeile 3 offenbar $O(n)$ mal aufgerufen und benötigt daher $O(n \log(k))$ Laufzeit. Ein $j \in [n]$, welches in Zeile 6 aus $H$ entfernt wird, wird nie wieder in $H$ einfügt, sodass diese Operation ebenfalls nur insgesamt $O(n)$ mal aufgerufen wird. □

Algorithmus 4 kann auch als Spezialfall von Algorithmus 5 aus dem nächsten Abschnitt verstanden werden. Allerdings sehen wir durch diese getrennte Analyse besser, dass sich das Verfahren in der Praxis sehr effizient implementieren lässt. Wenn alle $\alpha_i$ einen

konstanten Wert $\alpha$ annehmen (später werden das die Zeitkosten der Subtouren sein), so gilt im Verlaufe des Algorithmus $|H| \leq \lceil 1/\alpha \rceil + 1$. In der Praxis wird der Heap $H$ daher nur 1–4 Elemente enthalten.

### 3.3.3 Allgemeine stückweise lineare Kostenmodelle

In Algorithmus 4 für Modell (c) haben wir verwendet, dass wir stets so früh wie möglich warten sollten. Diese Eigenschaft geht in den Modellen (b) und (d) verloren. Wir werden daher nun allgemeine stückweise lineare und konvexe schwache Zeitfenster betrachten und zeigen, dass wir dort trotzdem sowohl praktisch als auch theoretisch gute Laufzeiten erzielen können.

Seien $c_i : [l_i, u_i] \to \mathbb{R}_{\geq 0}$ stückweise lineare konvexe Funktionen für $i = 1, \ldots, n$ mit insgesamt $l$ Segmenten. Außerdem sei $u_1 \leq \ldots \leq u_n$. Es genügt dann folgendes Optimierungsproblem zu lösen:

$$\min_{x_1, \ldots, x_n} \quad \sum_{i=1}^{n} c_i(x_i) \tag{3.8a}$$

$$\text{s.t.} \quad x_i \leq x_{i+1} \quad \forall i < n, \tag{3.8b}$$

$$x_i \in [l_i, u_i] \quad \forall i. \tag{3.8c}$$

Die Wartekosten $x_n - x_1$ aus dem Zeitplanungs-Problem können einfach in die Funktionen $c_n$ bzw. $c_1$ eingerechnet werden.

In [KTZ99] wird eine Implementierung des Clumping Algorithmus gegeben, die ein Spezialfall dieses Problems in $O(l \log^2(l))$ Laufzeit lösen kann. Die Arbeit [BV00] von Brenner und Vygen verbessert die Laufzeit auf $O(l \log(l) \log(\log(l)))$ und zeigt zudem, wie sich der allgemeine Fall in $O(l \log^2(l))$ lösen lässt. Schließlich wird ein alternativer Algorithmus in [Suh10] angegeben, womit eine Laufzeit von $O(l \log(n))$ erzielt werden kann. Allerdings ist dieser Algorithmus recht kompliziert und verwendet eine spezielle Version von Heaps, die in der Praxis große konstante Laufzeitfaktoren mit sich bringt. Im Folgenden wollen wir ein einfaches alternatives Verfahren betrachten, welches ebenfalls in $O(l \log(n))$ implementiert werden kann.

**Definition 25.** Sei $i \in \{1, \ldots, n\}$ dann definieren wir für $t \in [l_i, u_i]$:

$$f_i(t) \coloneqq \min\{\sum_{j=i}^{n} c_j(x_j) \mid t = x_i \leq \ldots \leq x_n \text{ und } \forall j \geq i.\, x_j \in [l_j, u_j]\}.$$

Außerdem setzen wir per Konvention $f_{n+1} \equiv 0$.

Die Bedeutung dieser Funktionen ist, dass sie die Kosten des günstigsten Weiterverlaufs angeben, falls wir am Punkt $i$ zur Zeit $t$ ausliefern. Durch sukzessives Minimieren der $f_i$ werden wir in der Lage sein, einen optimalen Zeitplan zu entwerfen. Hierfür müssen wir die $f_i$ jedoch zunächst einmal konstruieren können, was durch das nächste Lemma gesichert wird.

**Lemma 26.** *Für $i \in \{1, \ldots, n\}$ gilt*

$$f_i(t) = c_i(t) + \begin{cases} f_{i+1}(\tau) & \text{falls } t \leq \tau \\ f_{i+1}(t) & \text{sonst} \end{cases} \tag{3.9}$$

*wobei $\tau \coloneqq l_{i+1}$ falls $f^r_{i+1}(l_{i+1}) \geq 0$ und sonst*

$$\tau \coloneqq \sup\{t \in [l_{i+1}, u_{i+1}] \mid f^r_{i+1}(t) < 0\}.$$

*Hierbei ist $f^r_{i+1}$ die rechtsseitige Ableitung von $f_{i+1}$. Insbesondere ist $f_i$ stets wieder eine stückweise lineare konvexe Funktion.*

*Beweis.* Wir beweisen die Aussage per Induktion über $i$, wobei wir rückwärts vorgehen. Der Induktionsanfang $i = n$ und der Induktionsschritt $i + 1 \rightsquigarrow i$ verlaufen fast identisch. In beiden Fällen wissen wir, dass $f_{i+1}$ konvex ist, entweder per Induktionsvorraussetzung oder da $f_{n+1} \equiv 0$. Demnach nimmt $f_{i+1}$ sein erstes Minimum an der Stelle $\tau$ an.

Nun stellen wir fest:

$$\begin{aligned} f_i(t) &= \min\{\sum_{j=i}^{n} c_j(x_j) \mid t = x_i \leq \ldots \leq x_n \text{ und } \forall j \geq i.\, x_j \in [l_j, u_j]\} \\ &= c_i(t) + \min\{\sum_{j=i+1}^{n} c_j(x_j) \mid t \leq x_{i+1} \leq \ldots \leq x_n \text{ und } \forall j \geq i+1.\, x_j \in [l_j, u_j]\} \\ &= c_i(t) + \min\{f_{i+1}(x_{i+1}) \mid x_{i+1} \in [\max\{t, l_{i+1}\}, u_{i+1}]\} \end{aligned} \tag{3.10}$$

Falls $x_{i+1} \leq \tau$, so wissen wir dass das Minimum in Gleichung (3.10) genau bei $\tau$ angenommen wird. Ansonsten wird das Minimum wegen Konvexität von $f_{i+1}$ am linken Intervallrand — also bei $t$ — angenommen.

Schließlich sehen wir leicht, dass $f_i$ ebenfalls konvex ist, denn $c_i$ ist per Vorraussetzung konvex, $f_{i+1}(\tau)$ ist konstant und $f_{i+1}(t)$ ist monoton steigend für $t \geq \tau$. Damit lässt sich $f_i$ als Summe von konvexen Funktionen schreiben. Da diese zudem stückweise linear sind, ist $f_i$ ebenfalls stückweise linear. □

Gerade diese rekursiven Berechnungen von $f_i$ und $\tau$ führen wir nun in Algorithmus 5 aus, um einen optimalen Zeitplan zu erhalten. Man sieht leicht, dass sich dieses Verfahren in $O(n \cdot l)$ Laufzeit implementieren lässt, indem man die vorkommenden stückweise linearen Funktionen als Liste von Bruchpunkten speichert. Um die gewünschte Laufzeit von $O(l \log(n))$ zu erreichen, müssen wir jedoch eine effizientere Darstellung finden.

**Satz 27.** *Algorithmus 5 löst das Optimierungsproblem (3.8) und kann in $O(l \log(n))$ Laufzeit implementiert werden. Damit kann das Zeitplanungs-Problem nach den Modellen (b) und (d) in $O(n \log(n))$ Laufzeit gelöst werden.*

*Beweis.* Nach Definition von $f_1$ und Lemma 26 wissen wir, dass es für jedes $t$, welches $f_1(t)$ minimiert, eine optimale Lösung für Problem (3.8) mit $t = x_1$ gibt. Demnach ist die Wahl von $x_1$ in Zeile 5 des Algorithmus korrekt. Angenommen $x_1, \ldots, x_i$ sind bereits so

**Algorithmus 5** Stückweise Lineare Zeitplanung

1: $f_n \leftarrow c_n$
2: **for** $i \leftarrow n-1, \dots, 1$ **do**
3: $\quad \tau_{i+1} \leftarrow \max\{l_{i+1}, \sup\{t \in [l_{i+1}, u_{i+1}] \mid f^r_{i+1}(t) < 0\}\}$
4: $\quad f_n \leftarrow \left( t \mapsto c_i(t) + \begin{cases} f_{i+1}(\tau_{i+1}) & \text{falls } t \leq \tau_{i+1} \\ f_{i+1}(t) & \text{sonst} \end{cases} \right)$
5: $x_1 \leftarrow \max\{l_1, \sup\{t \in [l_1, u_1] \mid f^r_1(t) < 0\}\}$
6: **for** $i \leftarrow 2, \dots, n$ **do**
7: $\quad x_i \leftarrow \max\{x_{i-1}, \tau_i\}$
8: **return** $x_1, \dots, x_n$

gewählt worden, dass sie zu einer optimalen Lösung von Problem (3.8) erweitert werden könnten. Dann sollten wir $x_{i+1} \geq x_i$ offenbar so wählen, dass $f_{i+1}(x_{i+1})$ minimiert wird, denn dies gerantiert nach Definition von $f_{i+1}$, dass sich $x_1, \dots, x_{i+1}$ ebenfalls zu einer optimalen Lösung fortsetzen lassen. In dem Beweis von Lemma 26 haben wir gesehen, dass $\tau_{i+1}$ die Funktion $f_{i+1}$ minimiert. Da diese konvex ist, muss $x_i$ möglichst nah an $\tau_{i+1}$ gewählt werden, was in Zeile 7 geschieht. Per Induktion gibt Algorithmus 5 also eine optimale Lösung $x_1, \dots, x_n$ aus und ist damit korrekt.

Für die Laufzeit werden wir zunächst eine Darstellung der Funktionen $f_i$ beschreiben, die eine Laufzeit von $O(l \log(l))$ erzielt. Dann werden wir sehen, wie sich der $\log(l)$ Faktor zu einem $\log(n)$ Faktor verbessern lässt. Wir gehen davon aus, dass die Funktionen $c_i$ jeweils durch ein Paar $(S_i, s_i)$ gegeben sind. Dabei sei $S_i = c^r_i(l_i)$ und $s_i$ sei eine endliche Folge von Steigungsänderungen.

Die Funktionen $f_i$ wollen wir ebenfalls auf diese Art speichern, wobei wir davon ausgehen, dass die Steigungsänderungen in einem Min-Heap nach ihrer Zeit-Koordinate angelegt sind. Sei also $f_{i+1}$ durch $(S_f, s_f)$ charakterisiert. Wir wollen zeigen, dass sich $(S_f, s_f)$ und $\tau_{i+1}$ effizient aktualisieren lassen.

Zeile 3 implementieren wir wie folgt: Wir setzen $a \leftarrow S_f$ und entfernen stets das kleinste Element von $s_f$, wobei wir die entsprechende Steigungsänderung auf $a$ anwenden. Falls $a > 0$ und wir über $l_{i+1}$ hinaus sind, so haben wir $\tau_{i+1}$ gefunden und setzen in Zeile 4:

$$\begin{aligned} S_f &\leftarrow S_i \\ s_f &\leftarrow s_i \cup \{(\tau_{i+1}, a)\} \cup s_f. \end{aligned}$$

Eine Iteration dieser Operation ist in Abbildung 3.2 dargestellt. In den Zeilen 3 und 4 werden also insgesamt höchstens $n + l$ Elemente auf den Heap gepusht und wieder entfernt. Die restlichen Schritte des Algorithmus lassen sich in $O(n)$ Zeit implementieren, sodass wir eine Gesamtlaufzeit von $l \log(l)$ erreicht haben.

Um schließlich auch noch $O(l \log(n))$ zu erzielen, müssen wir dafür sorgen, dass der Heap $s_f$ nicht zu groß wird. Dies kann erreicht werden, indem wir die Steigungsänderungen von $c_i$ nicht einzeln auf den Heap pushen sondern stets die ganze Folge $s_i$. Dabei finden

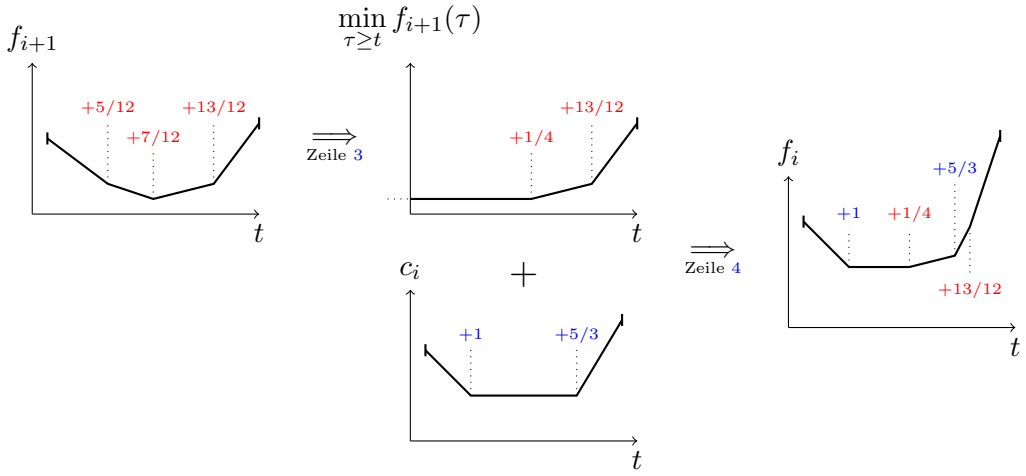



Abbildung 3.2: Dargestellt ist eine Iteration der For-Schleife in Zeile 2 von Algorithmus 5. Dabei sind die Steigungsänderungen gekennzeichnet, anhand derer wir die stückweise linearen Funktionen speichern.

die Vergleiche nach der Zeit-Koordinate des ersten Folgenglieds statt. Wenn in der Berechnung von $\tau_{i+1}$ das kleinste Element von $s_f$ entfernt werden soll, so entfernen wir nur das erste Element der ersten Folge und pushen den Rest wieder auf den Heap. Dadurch gilt stets $|s_f| \leq 2n$ und es ergibt sich die Laufzeit $O(l \log(n))$. □

Man bemerke, dass Algorithmus 5 auch in der Praxis effizient implementiert werden kann. Die verwendeten Heaps müssen nur Push und Delete-Min Operationen in logarithmischer Zeit unterstützen, was sich durch in Arrays gespeicherte Binary-Heaps erzielen lässt. Auch die $O(l \log(n))$ Variante kann in der Praxis verwendet werden, wenn man nicht wirklich ganze Folgen auf dem Heap speichert sondern lediglich Indizes in die bestehenden Arrays $s_i$. Dadurch kann sogar der zusätzliche Speicherverbrauch von $O(l)$ auf $O(n)$ reduziert werden, was interessant sein kann, wenn die Kostenfunktionen identisch sind und daher ebenfalls nur durch $O(n)$ Speicher gegeben sind.

## 3.4 Ein Zwei-Phasen Algorithmus für das VRPSTW

Mit Hilfe der Subroutinen aus den vorherigen Abschnitten lassen sich verschiedene Heuristiken und Metaheuristiken für das VRPSTW implementieren. Wir wollen nun eine Konstruktions-Heuristik mit Subtouren und Zeitfenstern angeben. Dabei könnte man so vorgehen wie in der Heuristik für das VRPS aus Kapitel 2, wo Haupttouren und Subtouren gleichzeitig aufgebaut wurden. Alternativ kann man zunächst alle Subtouren erzeugen und diese dann mit Haupttouren verbinden. Dies ist der Ansatz, den wir hier verfolgen möchten.

Zunächst verwenden wir also Algorithmus 6, um unsere Pakete in Subtouren zu clustern. Offenbar ähnelt das Verfahren sehr dem Verfahren von Algorithmus 3 aus

**Algorithmus 6** Parallele Konstruktion von Subtouren

① Bestimme Start-Subtouren $\mathcal{S}$ und setze $P \leftarrow V(G) \setminus (\{D\} \cup \bigcup \mathcal{S})$.

② Wähle ein noch nicht geclustertes Paket $p \in P$.

③ Falls sich $p$ in keine bestehende Subtour einfügen lässt, so erstelle eine neue Subtour und gehe zu Schritt ⑧.

④ Wähle eine bestehende Subtour $S \in \mathcal{S}$, sodass sich $p$ in $S$ einfügen lässt.

⑤ Sei $S'$ die Tour die aus $S$ hervorgeht, wenn man $p$ optimal in $S$ einfügt.

⑥ Bestimme eine Tour $S''$, indem ein TSP mit Zeitfenstern auf $S'$ heuristisch gelöst wird.

⑦ Ersetze $S$ durch $S''$ und setze $P \leftarrow P \setminus \{p\}$.

⑧ Falls $P \neq \emptyset$, gehe zu Schritt ②.

Kapitel 2. Für Schritt ① verweisen wir wiederum auf [Erl17]. Die Wahlen in den Schritten ② und ④ geschehen nach dem selben Fehlervermeidungs-Greedy Verfahren wie in Algorithmus 3.

Eine Besonderheit im Vergleich zu dem Konstruktionsverfahren aus Kapitel 2 besteht allerdings darin, dass die Menge $\mathcal{S}$ noch nicht die Lösung unseres Problems ist. Zum einen bedeutet dies, dass wir in Schritt ④ überprüfen müssen, ob die resultierende Subtour überhaupt von einer Haupttour beliefert werden kann. Sonst können wir nicht garantieren, dass $\mathcal{S}$ eine zulässige Instanz für die zweite Phase des Verfahrens darstellt. Außerdem verwenden wir nicht nur die reine Arbeitszeit der Subtouren — die wir effizient mit den Methoden aus Abschnitt 3.2 bestimmen — sondern auch die folgenden Strafkosten und Boni für jede Subtour $S \in \mathcal{S}$:

- Startzeitfenster-Größe: Wir berechnen einen Bonus proportional zu $u_S - l_S$, denn dies macht es in der zweiten Phase einfacher, Zeitpläne zu finden, die keine zusätzlichen Wartezeiten verursachen.
- Distanz zum Depot: Sei $s \in V(S)$ der Startpunkt von $S$ und $p \in V(S)$ der Punkt auf $S$ der minimalen Abstand zum Depot hat. Dann berechnen wir Strafkosten proportional zu $d_m(D, s) - d_m(D, p)$, damit die Haupttouren später nicht übermäßig weit fahren müssen.
- Fülle der Subtour: Um während des Algorithmus zu vermeiden, dass unsere Subtouren stets komplett gefüllt werden, berechnen wir Strafkosten proportional zu $b(S)^2/m$, wobei $m$ die Anzahl der noch nicht geclusterten Pakete ist.
- Fixkosten: Wir erhöhen die Fahrzeug Kosten $c_{sv}$, da wir die zusätzlichen Kosten

antizipieren wollen, die dadurch entstehen, dass die Haupttouren später mehr Subtouren beliefern müssen.

Nach der ersten Phase des Algorithmus (die wie bereits bei Algorithmus 3 mehrfach mit unterschiedlichen Start-Subtouren ausgeführt wird), erhalten wir nun also eine Menge $\mathcal{S}$ von Subtouren, von der wir wissen, dass diese sich von Haupttouren zu einer zulässigen Lösung verbinden lassen. Bevor wir die zweite Phase starten, führen wir noch eine Post-Optimierung auf $\mathcal{S}$ aus. Dabei werden greedy sogenannte CROSS-Austäusche durchgeführt (siehe Abbildung 3.3), welche die Gesamtkosten aller Subtouren inklusive Strafkosten verbessern. Mehr Informationen zu diesem Verfahren sind ebenfalls in [Erl17] gegeben.

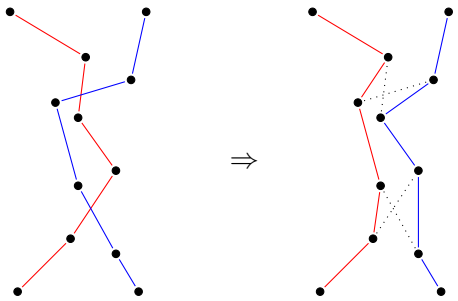


Abbildung 3.3: CROSS-Austausch zwischen zwei Subtouren, der die Gesamtlänge reduziert. Hierbei werden stets zwei zusammenhängende Segmente der beiden Touren ausgetauscht.

Schließlich führen wir die zweite Phase des Algorithmus aus, in der wir die Startpunkte der Subtouren wiederrum von Haupttouren abfahren lassen. Jede Subtour $S \in \mathcal{S}$ wird also zu ihrem Startpunkt kontrahiert. Dieser erhält dann das schwache Zeitfenster

$$c_S(t) \coloneqq c_{sf} \cdot \delta_S(t + \theta|V(S)|).$$

Dadurch ergibt sich eine Instanz des Vehicle Routing Problems mit schwachen Zeitfenstern, die wir ebenfalls mit Algorithmus 6 lösen können (wobei man überall „Subtour“ durch „Haupttour“ ersetze). Zur Berechnung der Kosten verwenden wir die Routinen aus Abschnitt 3.3.

# Kapitel 4

# Experimentelle Resultate

In den vorherigen Kapiteln wurde das Vehicle Routing Problem mit Subtouren (VRPS) sowie das Vehicle Routing Problem mit Subtouren und Zeitfenstern (VRPSTW) vorgestellt. Zum Ende der jeweiligen Kapitel haben wir dabei jeweils eine Konstruktionsheuristik betrachtet, die das entsprechende Problem effizient lösen kann. Im Folgenden wollen wir ausgewählte Ergebnisse aus Experimenten mit diesen Heuristiken untersuchen. Wir verwenden hierfür zufällig generierte Instanzen, die auf einem statistischen Modell von Berlin basieren.

## 4.1 Methodik und Parameter

Für die Testläufe werden wir zwei verschiedene Arten von Fahrzeugen — LKWs und Sprinter — betrachten. Die Spezifikationen für diese Fahrzeuge sind in Tabelle 4.1 zusammengefasst. Dabei ist zu bemerken, dass die Entladezeit der LKWs zusammen mit der Ladezeit der Sprinter die Übergabezeit $\theta$ aus Kapitel 3 ausmacht. Die Wegzeiten $d_s$ bzw. $d_m$ ergeben sich aus den Geschwindigkeiten der Fahrzeuge sowie den Distanzen bezüglich des Berliner Straßennetzes. Wir werden das Kostenmodell aus Tabelle 4.1 auch für das Verfahren aus Kapitel 2 verwenden.

| Fahrzeug | LKW | Sprinter |
|---|---|---|
| Geschwindigkeit (km/h) | 20 | 25 |
| Kosten (€/h) | 35 | 25 |
| Kapazität | 300 | 50 |
| Fixkosten (€) | 30 | 15 |
| Ladezeiten (s/Paket) | 15 | 10 |
| Entladezeit (s/Paket) | 20 | 120 |

Tabelle 4.1: Fahrzeug Daten

Die Verteilung der Pakete erfolgt nach einem statistischen Modell von Berlin, auf welches wir hier nicht detailliert eingehen werden. Pakete werden sowohl im gesamten

Stadtraum als auch vermehrt um einige Hotspots im Zentrum der Stadt verteilt. Danach wird jedes Paket auf die nächstgelegene Addresse verschoben. Schließlich erhält es noch eine Größe und ein Zeitfenster. Dabei werden wir im Folgenden stets nur Einheitsgrößen betrachten. Die genaue Verteilung der Zeitfenster wird in Abschnitt 4.2.2 festgelegt. Eine gelöste Beispiel Instanz mit 500 Paketen ist in den Abbildungen 4.1 und 4.2 dargestellt.

| Parameter | Wert |
|---|---|
| Depot Strafkosten (€/km) | 2 |
| Zeitfenster Bonus (€/h) | 15 |
| Subtour-Fülle Kosten (€) | 0 |
| Zusätzliche Fixkosten (€) | 10 |

Tabelle 4.2: Zusätzliche Parameter

Zu den Parametern für die Fahrzeuge und Pakete kommen noch die Proportionalitätskonstanten aus Abschnitt 3.4. Die Werte dieser Parameter wurden durch einige wenige Experimente ermittelt. Mit mehr Aufwand an dieser Stelle lassen sich vermutlich noch leicht bessere Ergebnisse erzielen. Außerdem werden wir im Folgenden keine Postopt-Schritte (siehe ebenfalls Abschnitt 3.4) durchführen, da die Laufzeit dieses Verfahrens noch nicht optimiert wurde.

## 4.2 Ergebnisse

Wir wenden uns nun den eigentlichen Ergebnissen zu. Weitere Läufe des Algorithmus aus Kapitel 3 sowie deren Resultate sind in [Erl17] zu finden. An dieser Stelle sei außerdem erwähnt, dass die Implementierung dieses Verfahrens noch recht jung ist und die Ergebnisse daher keinesfalls endgültig sind.

### 4.2.1 Vergleich der Algorithmen

Die Zwei-Phasen Methode aus Kapitel 3 lässt sich offenbar auch auf das VRPS aus Kapitel 2 anwenden, indem man die Zeitfenster groß genug wählt. Da das in dieser Arbeit betrachtete Problem in der Literatur bisher nicht vertreten ist und selbst für die eng verwandten Probleme aus Abschnitt 2.1.2 keine Testinstanzen vorliegen, ist dies die beste Alternative, um Aussagen über die Güte beider Verfahren machen zu können. Beide Algorithmen wurden auf Instanzen zwischen 250 und 2500 Paketen angewandt — die Ergebnisse sind in Tabelle 4.3 dargestellt.

Aus der Tabelle lässt sich entnehmen, dass der VRPS Algorithmus im Schnitt etwa 1–5% bessere Ergebnisse liefert als der VRPSTW Algorithmus. Dies war zu erwarten, da dieses Verfahren speziell für Instanzen ohne Zeitfenster entwickelt wurde. Es nutzt daher aus, dass es für das klassische TSP sehr effiziente Heuristiken gibt und dass sich die Segmente einer TSP-Lösung gut als Subtouren eignen. Für das VRPSTW scheint

| | VRPS | | VRPSTW | |
|---|---|---|---|---|
| Paketzahl | Kosten (€) | Laufzeit (s) | Kosten (€) | Laufzeit (s) |
| 250 | 988.10 | 5.73 | 1024.84 | 0.44 |
| 500 | 1818.73 | 10.42 | 1905.47 | 1.55 |
| 750 | 2634.63 | 15.08 | 2696.04 | 3.86 |
| 1000 | 3431.53 | 16.63 | 3524.86 | 12.01 |
| 1250 | 4178.21 | 25.43 | 4341.86 | 21.67 |
| 1500 | 4931.73 | 25.01 | 5070.57 | 30.47 |
| 1750 | 5645.83 | 31.91 | 5897.32 | 51.52 |
| 2000 | 6425.90 | 38.83 | 6506.41 | 80.87 |
| 2250 | 7157.22 | 36.16 | 7417.12 | 112.44 |
| 2500 | 7903.56 | 54.36 | 8207.05 | 131.96 |

Tabelle 4.3: Vergleich der Algorithmen aus Kapitel 2 und Kapitel 3.

eine Subtour Partitionierung allerdings kein sinnvoller Ansatz zu sein, sodass sich der Algorithmus aus Kapitel 2 nicht auf dieses Szenario verallgemeinern lässt.

Wir sehen jedoch durch diese Resultate, dass der VRPSTW Algorithmus noch deutlich verbessert werden kann. Die aus Gründen der Laufzeit deaktivierte Postopt-Routine für Subtouren lässt die Lücke zwischen den beiden Algorithmen auf den meisten Instanzen auf unter 1% schrumpfen. Allerdings ist nicht klar, wie effizient sich dieses Verfahren implementieren lässt. Bisher ist deutlich mehr Aufwand in die Implementierung und das Tuning des VRPS Algorithmus eingeflossen, sodass es wahrscheinlich scheint, dass sich die Lücken in Lösungsqualität und Laufzeit weiter reduzieren lassen.

### 4.2.2 Auswirkungen von Zeitfenstern

Ein großer Teil dieser Arbeit beschäftigte sich mit Zeitfenstern im Kontext von Subtouren. Demnach wollen wir in diesem Abschnitt untersuchen, welche Auswirkungen verschiedene Arten von Zeitfenstern auf die Kosten der Lösung haben.

| Modell | Zeitfenster |
|---|---|
| Normal | 14:00 – 16:00 Uhr, 16:00 – 18:00 Uhr, 18:00 – 20:00 Uhr |
| Eng | 14:00 – 15:00 Uhr, 15:00 – 16:00 Uhr, 16:00 – 17:00 Uhr, 17:00 – 18:00 Uhr, 18:00 – 19:00 Uhr, 19:00 – 20:00 Uhr |
| Relaxiert | 13:30 – 16:30 Uhr, 15:30 – 18:30 Uhr, 17:30 – 20:30 Uhr |
| Überlappend | 14:00 – 16:00 Uhr, 15:00 – 17:00 Uhr, 16:00 – 18:00 Uhr, 17:00 – 19:00 Uhr, 18:00 – 20:00 Uhr |

Tabelle 4.4: Verschiedene Zeitfenster-Modelle

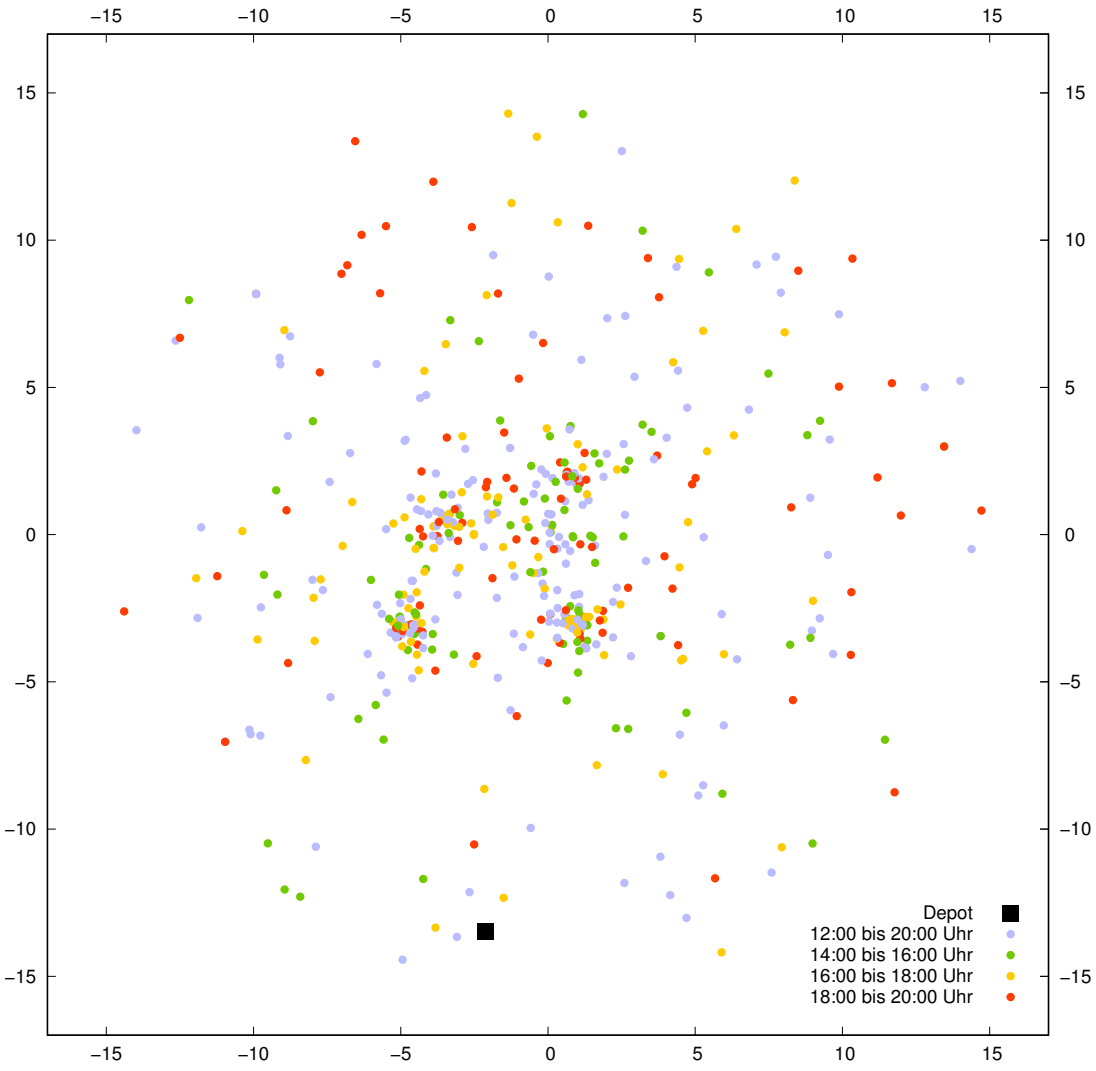



Abbildung 4.1: Abgebildet ist eine Beispiel Instanz mit 500 Paketen. Die Koordinaten in der Abbildung sind in Kilometern angegeben, wobei der Punkt $(0, 0)$ dem Berliner Fernsehturm entspricht. Im Zentrum Berlins sind mehrere Hotspots zu sehen und das Depot liegt in unseren Instanzen stets am Stadtrand.

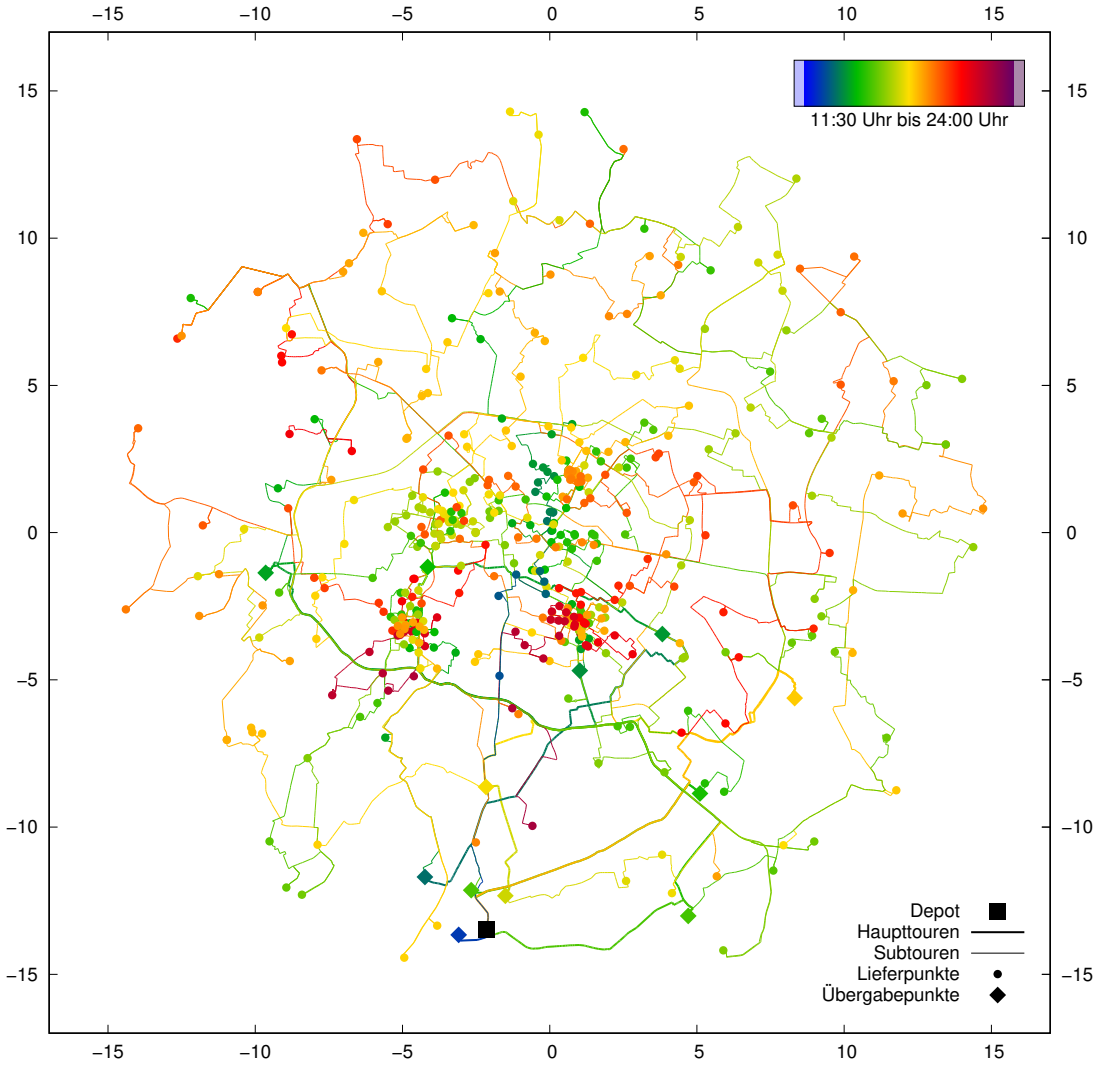



Abbildung 4.2: Zu sehen ist eine Lösung für die Instanz aus Abbildung 4.1. Sämtliche Fahrten, Übergaben und Lieferungen sind dabei farblich dem Zeitpunkt zugewiesen, zu dem sie stattfinden (siehe Farblegende).

Für den Vergleich betrachten wir vier unterschiedliche Zeitfenster-Modelle (bzw. fünf, wenn man die Instanzen ohne Zeitfenster aus dem letzten Abschnitt mitzählt). In jedem Fall werden 40% aller Pakete das „ganztägige“ Zeitfenster von 12:00 bis 20:00 Uhr erhalten. Die restlichen 60% werden gleichmäßig unter den Zeitfenstern verteilt, die in Tabelle 4.4 aufgelistet sind. Außerdem müssen alle Haupttouren zwischen 11:30 und 14:00 Uhr beginnen und dürfen zu einer beliebigen Zeit enden.

| Paketzahl | Normal (€) | Eng (€) | Relaxiert (€) | Überlappend (€) |
|---|---|---|---|---|
| 250 | 1413.24 | 1543.48 | 1316.22 | 1392.35 |
| 500 | 2400.23 | 2667.83 | 2419.16 | 2352.59 |
| 750 | 3391.62 | 3763.51 | 3419.02 | 3474.25 |
| 1000 | 4290.03 | 4868.62 | 4397.86 | 4311.82 |
| 1250 | 5495.57 | 5850.92 | 5163.61 | 5301.29 |
| 1500 | 6305.53 | 6775.02 | 6236.93 | 6078.45 |
| 1750 | 7291.30 | 7936.23 | 7080.07 | 7259.48 |
| 2000 | 8049.25 | 8639.19 | 8033.79 | 8069.95 |
| 2250 | 9261.31 | 9949.23 | 8855.52 | 8898.68 |
| 2500 | 9943.26 | 10560.80 | 9718.99 | 9836.73 |

Tabelle 4.5: Auswirkung von verschiedenen Zeitfenstern auf die Kosten der Lösung.

Wir sehen, dass Zeitfenster auf 60% der Pakete etwa 15–50% zusätzliche Kosten verursachen. Die Unterschiede zwischen den verschiedenen Modellen fallen dabei ähnlich aus, wie man es erwarten würde. So sind die relaxierten Zeitfenster in der Regel am günstigsten. Weiterhin scheinen die überlappenden Zeitfenster im Schnitt ein wenig besser zu sein, als die normalen Zeitfenster. Dies kann allerdings dadurch ausgelöst werden, dass bei den überlappenden Zeitfenstern weniger Pakete erst ab 18:00 Uhr ausgeliefert werden müssen. Schließlich verursachen solche Pakete besonders große Wartezeiten. Wenig überraschend ist, dass das Szenario mit engen Zeitfenstern mit Abstand am teuersten ausfällt.

Allerdings gibt es auch ein paar Anomalien in den Daten. Auf den Instanzen mit 500 und 1000 Paketen ist die Lösung mit normalen Zeitfenstern günstiger als die Lösung mit relaxierten Zeitfenstern. Jedoch ist die mit normalen Zeitfenstern berechnete Lösung auf jeden Fall eine zulässige Lösung unter relaxierten Zeitfenstern, sodass dies auf eine Schwäche des Algorithmus hinweist. Zumindest sehen wir daran, dass der Algorithmus auf der Instanz mit 1000 Paketen mindestens 2,5% vom Optimum entfernt ist.

## 4.3 Bemerkungen und Fazit

In dieser Arbeit haben wir zwei neue Varianten des Vehicle Routing Problems studiert und jeweils eine Konstruktionsheuristik für das Problem entwickelt. Der Hauptaufwand bestand dabei jeweils in Subroutinen der Algorithmen, nämlich der Subtour Partitionierung in Kapitel 2 und den verschiedenen Varianten der Zeitplanung in Kapitel 3. Die

Hauptergebnisse waren dabei:

- Subtour Partitionierung (ohne Deadlines) lässt sich in $O(n^2)$ Laufzeit lösen,
- Zeitplanung kann in linearer Zeit gelöst sowie in konstanter Zeit auf Veränderungen überprüft werden und
- Schwache Zeitplanung kann in $O(l \log(n))$ Zeit und mit $O(n)$ zusätzlichem Speicher gelöst werden, wobei $l$ die Gesamtzahl der linearen Segmente in den Kostenfunktionen ist.

Insbesondere das letzte Ergebnis ist interessant, da das Problem der schwachen Zeitplanung in unterschiedlichen Bereichen — wie etwa im VLSI Design — unter anderem Namen auftritt. Der angegebene Algorithmus erreicht die beste bekannte theoretische Laufzeit für dieses Problem und lässt sich deutlich effizienter (und einfacher) implementieren, als der Clumping Algorithmus aus [DSD90] bzw. [KTZ99] und als der Algorithmus, welcher in der Arbeit [Suh10] vorgeschlagen wird.

Danach haben wir in den letzten Abschnitten präliminäre Ergebnisse der Konstruktionsalgorithmen untersucht. In diesen wird deutlich, dass besonders die Heuristik für das VRPSTW noch um einiges verbessert werden kann. Geringe Verbesserungen im Rahmen von 1–5% sowie Verbesserungen in der Laufzeit sind vermutlich noch durch Tuning des Algorithmus erzielbar. Für noch bessere Ergebnisse ist vermutlich ein anderer Ansatz notwendig, was das Verfahren aus Kapitel 3 allerdings nicht obsolet macht. Denkbar sind folgende weitere Vorgehensweisen:

- Die Lösung der Konstruktionsheuristik kann als Startlösung für eine Metaheuristik (siehe Abschnitt 1.4.3) verwendet werden. Für die Implementierung einer solchen Metaheuristik können wir zudem die Subroutinen für die Zeitplanungsprobleme wiederverwenden.
- Wir können zunächst nur eine Teilmenge der Pakete in Subtouren clustern lassen. In der zweiten Phase fügen wir dann gleichzeitig Pakete in Subtouren und Subtouren in Haupttouren ein — ähnlich wie dies in Kapitel 2 geschieht.
- Völlig andere Ansätze, bei denen zum Beispiel zunächst mehrere — eventuell überlappende — Subtouren gebildet werden, auf welchen dann eine Art Set-Cover Problem gelöst wird, können ebenfalls in Betracht gezogen werden.

Schließlich kann das VRPSTW auch weiter verallgemeinert werden — zum Beispiel indem mehrere Abholpunkte anstatt eines einzigen Depots betrachten werden oder indem man kompliziertere Austäusche zwischen den Fahrzeugen erlaubt. Daher bleibt noch einiges an Potential für zukünftige Arbeit an diesen Problemen übrig.

# Literaturverzeichnis


[ABB+98] Augerat, P., J. M. Belenguer, E. Benavent, A. Corberán und D. Naddef: *Separating capacity constraints in the CVRP using tabu search.* European Journal of Operational Research, 106(2):546–557, 1998.

[BV00] Brenner, U. und J. Vygen: *Faster Optimal Single-row Placement with Fixed Ordering.* In: *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '00, Seiten 117–121, New York, NY, USA, 2000.

[CGS15] Cuda, R., G. Guastaroba und M.G. Speranza: *A Survey on Two-echelon Routing Problems.* Comput. Oper. Res., 55(C):185–199, 2015.

[CJCG+13] Coffman Jr., Edward G., János Csirik, Gábor Galambos, Silvano Martello und Daniele Vigo: *Bin Packing Approximation Algorithms: Survey and Classification*, Seiten 455–531. Springer New York, New York, NY, USA, 2013.

[CMT13] Christofides, N., A. Mingozzi und P. Toth: *The vehicle routing problem*, Seiten 315–338. Wiley, Chichester, 2013.

[CW64] Clarke, G. und J. Wright: *Scheduling of vehicles from a central depot to a number of delivery points.* Operations Research, 12(4):568–581, 1964.

[DHMM17] Dorling, K., J. Heinrichs, G. G. Messier und S. Magierowski: *Vehicle Routing Problems for Drone Delivery.* IEEE Transactions on Systems, Man, and Cybernetics: Systems, 47(1):70–85, 2017.

[DR59] Dantzig, G. B. und J. H. Ramser: *The Truck Dispatching Problem.* Manage. Sci., 6(1):80–91, 1959.

[DSD90] Dumas, Yvan, François Soumis und Jacques Desrosiers: *Technical Note—Optimizing the Schedule for a Fixed Vehicle Path with Convex Inconvenience Costs.* Transportation Science, 24(2):145–152, 1990.

[Erl17] Erlenbach, Lukas: *Vehicle Routing in Zwei Phasen.* Bachelorarbeit, Forschungsinstitut für Diskrete Mathematik, Universität Bonn, 2017.

[Fag01] Fagerholt, Kjetil: *Ship scheduling with soft time windows: An optimisation based approach.* European Journal of Operational Research, 131(3):559–571, 2001.

[GNC11] Garone, Emanuele, Roberto Naldi und Alessandro Casavola: *Traveling Salesman Problem for a Class of Carrier-Vehicle Systems.* Journal of Guidance, Control, and Dynamics, 34(4):1272–1276, 2011.

[HKV17] Held, Stephan, Jochen Könemann und Jens Vygen: *Vehicle Routing with Subtours.* 2017.

[Jaf84] Jaffe, Jeffrey M.: *Algorithms for finding paths with multiple constraints.* Networks, 14(1):95–116, 1984.

[JM76] Jameson, S. R. und R. H. Mole: *A sequential route-building algorithm employing a generalized savings criterion.* Operational Research Quaterly, 27:503–511, 1976.

[KTZ99] Kahng, A. B., P. Tucker und A. Zelikovsky: *Optimization of linear placements for wirelength minimization with free sites.* In: *Design Automation Conference, 1999. Proceedings of the ASP-DAC '99. Asia and South Pacific*, Band 1, Seiten 241–244, 1999.

[KV08] Korte, Bernhard und Jens Vygen: *Kombinatorische Optimierung: Theorie und Algorithmen.* Springer-Verlag, 2008, ISBN 978-3-540-76918-7.

[KZL17] Kafle, Nabin, Bo Zou und Jane Lin: *Design and modeling of a crowdsource-enabled system for urban parcel relay and delivery.* Transportation Research Part B: Methodological, 99:62–82, 2017.

[PR07] Pisinger, David und Stefan Ropke: *A general heuristic for vehicle routing problems.* Computers & Operations Research, 34(8):2403–2435, 2007.

[Sav85] Savelsbergh, M. W. P.: *Local search in routing problems with time windows.* Annals of Operations Research, 4(1):285–305, 1985.

[SC86] Sexton, Thomas R. und Young Mygun Choi: *Pickup and Delivery of Partial Loads with "Soft" Time Windows.* American Journal of Mathematical and Management Sciences, 6(3–4):369–398, 1986.

[Suh10] Suhl, Ulrike: *Row-Placement in VLSI Design: The Clumping Algorithm and a generalization.* Diplomarbeit, Forschungsinstitut für Diskrete Mathematik, Universität Bonn, 2010.

[TV01] Toth, Paolo und Daniele Vigo: *The Vehicle Routing Problem.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001, ISBN 0-89871-498-2.

[WPG17] Wang, Xingyin, Stefan Poikonen und Bruce Golden: *The vehicle routing problem with drones: several worst-case results.* Optimization Letters, 11(4):679–697, 2017.