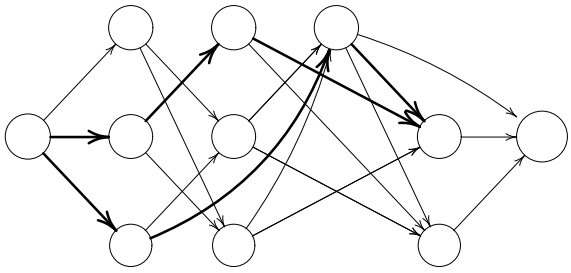
Diplomarbeit

Das dynamische Travelling Salesman Problem

Martin C. Müller



Universität Kaiserslautern
Fachbereich Mathematik
August 1996

Vorwort

Bisweilen stellt es eine anspruchsvolle Aufgabe dar, „das richtige Ding zur rechten Zeit“ zu tun. Trotzdem ist selbstverständlich, daß man zur rechten Zeit leichter Erfolg hat. Wer unnötig zur rush–hour in die Stadt fährt, weiß das.

Das Travelling Salesman Problem (TSP), eines der best untersuchten Probleme der Diskreten Optimierung und Testobjekt für neue Ideen und Methoden, vernachlässigt den zeitlichen Vorteil aber gänzlich – die Entscheidungen bleiben immer gleich teuer. Angeregt von Fortschritten auf dem Gebiet der dynamischen Flüße [8] begeisterte mich Herr Professor Hamacher dafür, das TSP unter Rücksichtnahme auf den zeitlichen Aspekt zu betrachten. Dadurch wurde das Dynamische Travelling Salesman Problem (DTSP) zum Objekt meiner Arbeit. Wie beim statischen TSP muß sich der Handelsmann an jedem Ort seiner Reise entscheiden, wohin er als nächstes gehen will. Kosten und Dauer des nächsten Reiseabschnitts hängen nun aber vom Startzeitpunkt ab.

Meist wird die Entscheidung dadurch nicht einfacher. Das DTSP war eine Herausforderung und die Arbeit daran hat viel Spaß gemacht. Die folgenden fünf Kapitel fassen das Erreichte zusammen.

Zum Anfang definiere ich im ersten Kapitel das Problem und versuche, durch drei Komplexitätsaussagen zu erhellen, worin die Schwierigkeit des DTSP im allgemeinen liegt.

Das zweite und dritte Kapitel enthalten die Ergebnisse meiner Literatur–Recherche nach zeitabhängigen TSP–Varianten. Abschnitt 2.1.1 beschreibt die Entwicklung der historisch ersten IP–Formulierung für das TDTSP. In Satz 2.3 gelingt es mir, die Dimension des TDTSP–Polyeders zu berechnen. Der folgende Abschnitt 2.1.2 folgt weitgehend einer neueren Arbeit von Gouveia und Voß [29], die einen alternativen Zugang vom Quadratic Assignment Problem aus zur selben Formulierung findet. Eine Idee von Hamacher zeigt einen dritten Weg ausgehend vom Cumulative Assignment Problem in Abschnitt 2.2.1. Eben diese Formulierung werde ich dann in Kapitel 4 auf das DTSP verallgemeinern. Die

beiden letzten Abschnitte des zweiten Kapitels beschreiben kurz Ansätze zu eng verwandten Problemen.

Der erste Abschnitt des dritten Kapitels gibt eine kurze, allgemeine Einführung in die Technik der Lagrange–Relaxation zur Lösung ganzzahliger Optimierungsprobleme und basiert auf den Arbeiten von Fisher [19] und Geoffrion [25]. Diese Technik wurde zusammen mit der IP–Formulierung aus Abschnitt 2.1.1 von Picard und Queyranne [58] zur Lösung des TDTSP verwendet. Dieses Verfahren wurde später von Lucena [46] weiterentwickelt. Ihre Ergebnisse finden sich in Abschnitt 3.2, der aber in der Darstellung etwas von den Originalarbeiten abweicht.

Einige der Argumente kann ich im Abschnitt 4.1 des vierten Kapitels direkt auf das DTSP übertragen. Anschließend variiere ich in Abschnitt 4.2.2 das zentrale Hilfsmittel des Verfahrens etwas und gebe mit Satz 4.2 einen weiteren Hinweis auf die inherente Komplexität des DTSP. Schließlich komme ich in Abschnitt 4.3 auf die insbesondere rechtentechnisch aufwendigeren Aspekte der Verallgemeinerung zu sprechen, die Berechnung billigster Pfade. In Abschnitt 4.4.1 untersuche ich daher eine naheliegende Idee zur Komplexitätreduktion. Wie sich im Abschnitt 4.4.2 zeigt, führt diese Idee auf natürliche Weise zurück zum TDTSP, aber in Abschnitt 4.4.3 zeige ich die Existenz einer darüber hinausführenden Klasse von DTSP–Instanzen auf.

Das fünfte Kapitel endlich beschreibt die Software gewordenen Ergebnisse der vorhergehenden Arbeit und dokumentiert, wie weit das DTSP meinem Angriff nachgegeben hat.

für Dennis und Andrea

Ich bedanke mich bei Herrn Professor Hamacher für sein Vertrauen in mich und dafür, daß er mir all die Unterstützung gegeben hat, die ich wollte. Ich bedanke mich ganz besonders bei Anita Schöbel für ihre großzügige fachliche und persönliche Unterstützung.

Außerdem möchte ich mich bei Donald E. Knuth für TEX bedanken, bei Leslie Lamport und dem LATEX3–Projekt für LATEX2e und bei Kristoffer H. Rose und Ross Moore für XY-pic.

Inhaltsverzeichnis

Kapitel 1

Das dynamische Travelling Salesman Problem: Definition und Komplexität

Das dynamische Travelling Salesman Problem ist eine sehr weitgehende Verallgemeinerung des Travelling Salesman Problems (TSP). Abschnitt 1.1 zeigt, daß das TSP und die beiden bekanntesten Erweiterungen, das TDTSP und und das Delivery Man Problem (DMP) natürliche Spezialfälle des DTSP sind. Der zweite Abschnitt macht dann einige komplexitätstheoretische Aussagen zur „Schwere“ des DTSP (Grundlagen der Komplexitätstheorie finden sich in [23, 40, 54]).

1.1 Problemdefinition

Definition 1.1
Eine *Instanz des Problems (DTSP)* ist ein 5-Tupel $\mathcal{I} = (V, E, d, c, v_0)$ mit

$V = \{v_0, v_1, \ldots, v_n\}$ Menge der Städte, in v_0 wird gestartet, $V_0 := V \setminus \{v_0\}$
$E = \{(i, j) \in V \times V | i \neq j\}$ Menge der Städteverbindungen
$d : E \times \mathbb{R}_+ \longrightarrow \mathbb{R}_+ \setminus [0, \epsilon], \epsilon > 0$ dynamische Wegezeit für Verbindung (i,j)
$(i, j), t \longmapsto d_{i,j}(t)$
$c : E \times \mathbb{R}_+ \longrightarrow \mathbb{R}_+$ dynamische Kosten für Verbindung (i,j)
$(i, j), t \longmapsto c_{i,j}(t)$

Mit diesen Daten soll folgendes Problem gelöst werden:

$$\text{(DTSP)} \qquad \begin{aligned} \min_{\substack{\pi\in S_{n+1}\\ \pi\text{zyklisch}}} \sum_{i\in V} c_{i,\pi(i)}(t_i) &=: z(\pi) \\ \text{s.d.} \quad t_{v_0} &= 0 \\ t_{\pi(i)} &= t_i + d_{i,\pi(i)}(t_i) \end{aligned}$$

Dabei gibt t_i die Abfahrtszeit in Stadt $i \in V$ an und S_{n+1} ist die Gruppe der Permutationen auf $n+1$ Elementen. Eine Permutation $\pi \in S_{n+1}$ wird in diesem Zusammenhang als *Tour* bezeichnet.

■

Bemerkung: Im Rahmen dieser Arbeit gilt, falls nicht speziell anders vereinbart, $V = \{0, \ldots, n\}$ und $v_i = i$. Die Verwendung der „v"–Notation dient hauptsächlich dazu, auch typographisch daran zu erinnern, daß die vorkommenden Zahlen oder Indizes Städte meinen.

Sind alle Wege zu allen Zeiten gleich lang, das heißt die Wegezeitfunktionen können identisch 1 gewählt werden, erhält man das *Time-dependent Travelling Salesman Problem* (TDTSP). Dabei hängen also die Kosten $c_{i,j}(t)$ einer Verbindung (i,j) nur von der Position $t = t_i$ von i in der Tour π ab:

$$\text{(TDTSP)} \qquad \begin{aligned} \min_{\substack{\pi\in S_{n+1}\\ \pi\text{zyklisch}}} \sum_{i\in V} c^{t_i}_{i,\pi(i)} &=: z_{\text{TDTSP}}(\pi) \\ \text{s.d.} \quad t_{v_0} &= 0 \\ t_{\pi(i)} &= t_i + 1 \end{aligned}$$

Das (TDTSP) wurde im Rahmen von Scheduling Problemen (ein–Maschinen–Probleme mit reihenfolgeabhängigen Bearbeitungszeiten: $1|\text{seq-dep}|\cdot$, siehe dazu [22, 58], und von Vehicle Routing untersucht, siehe [4]). Das (TDTSP) ist offensichtlich ein Spezialfall des (DTSP), in dem $d_{i,j}(t) \equiv 1$ und $c_{i,j}(t) = c^{\lfloor t \rfloor}_{i,j}$ gilt. Man beachte, daß hier mit „Zeitpunkt" eigentlich „Position" gemeint ist. Zur Illustration mag ein Handlungsvertreter dienen, der immer genau einen Tag in jeder Stadt verbringt und abends weiterreist. Eine Abwandlung eines klassischen Beispiels für das (TSP) (siehe [58]) behandelt eine Farbmischmaschine, die jeden Tag eine andere Farbe herstellt. Sie wird über Nacht umgerüstet, allerdings von einer Mannschaft, die hauptsächlich andere Aufgaben hat, so daß die zur Verfügung stehende Arbeitszeit abhängig vom Wochentag ist und somit auch die entstehenden Kosten in Form von Unterbeschäftigung oder Reinigungsmittel.

Bastian und Rinnooy Kan [4] verwenden das TDTSP als Modell für stochastische Vehicle Routing Probleme.

Das *Delivery Man Problem* (auch *Cumulative TSP* (CTSP) oder *Minimum Latency Tour Problem*, beziehungsweise $1|\text{seq-dep}|\sum C_i$ im Scheduling) ist ein weiterer wichtiger Spezialfall des DTSP. In diesem Fall ist die Zielfunktion die Summe der Ankunftszeiten (bei konstanten Wegezeiten):

$$\text{(DMP)} \qquad \begin{aligned} &\min_{\substack{\pi\in S_{n+1}\\ \pi\text{zyklisch}}} \sum_{i\in V} t_i =: z_{\text{DMP}}(\pi) \\ &\text{s.d.} \quad t_{v_0} = 0 \\ &\qquad t_{\pi(i)} = t_i + d_{i,\pi(i)} \end{aligned}$$

Wie man leicht sieht, kommt der Summand $d_{i,\pi(i)}$ genau $(n-t)$-mal in der Zielfunktion vor, wenn $i = \pi^t(0)$ ist, das heißt, wenn Stadt i Position t in π hat. Sind d_{ij} die Wegezeiten einer DMP-Instanz, dann läßt es sich als Spezialfall des TDTSP mit $c^t_{i,j} := (n-t)d_{i,j}$ formulieren (Beachte, daß die Wegezeiten des DMP jetzt in der Zielfunktion des TDTSP stehen und die Wegezeiten des TDTSP identisch 1 sind):

$$\text{(1.1)} \qquad \begin{aligned} &\min_{\substack{\pi\in S_{n+1}\\ \pi\text{zyklisch}}} \sum_{i\in V} (n-t_i) d_{i,\pi(i)} \\ &\text{s.d.} \quad t_{v_0} = 0 \\ &\qquad t_{\pi(i)} = t_i + 1 \end{aligned}$$

Bemerkung: Formulierung (DMP) weicht von der vorwiegend in der Literatur [6,17,51] verwendeten Definition des DMP etwas ab. In der hier angegebenen Form wird der Zeitpunkt der Rückkehr nach v_0 nicht berücksichtigt. Diese Variante wird gelegentlich als *Travelling Repairman Problem* [1] oder *Travelling Deliveryman Problem* [46] bezeichnet. Ein DMP mit n Städten läßt sich in dieser Formulierung darstellen, indem man die Städte v_0 und v_n miteinander identifiziert, das heißt $d_{v,v_n} = d_{v,v_0}$ und $d_{v_n,v} = \infty$ setzt (beachte, daß in (DMP) diese Werte niemals in der Zielfunktion auftauchen können).

Hängen die Kosten einer Verbindung nur vom „wo“ und nicht vom „wann“ ab, so erhält man das *Travelling Salesman Problem*:

$$\text{(TSP)} \qquad \min_{\substack{\pi\in S_{n+1}\\ \pi\text{zyklisch}}} \sum_{i\in V} c_{i,\pi(i)} =: z_{\text{TSP}}(\pi)$$

wobei $c_{i,j}$ = Wegezeit oder Kosten von Stadt i nach Stadt j

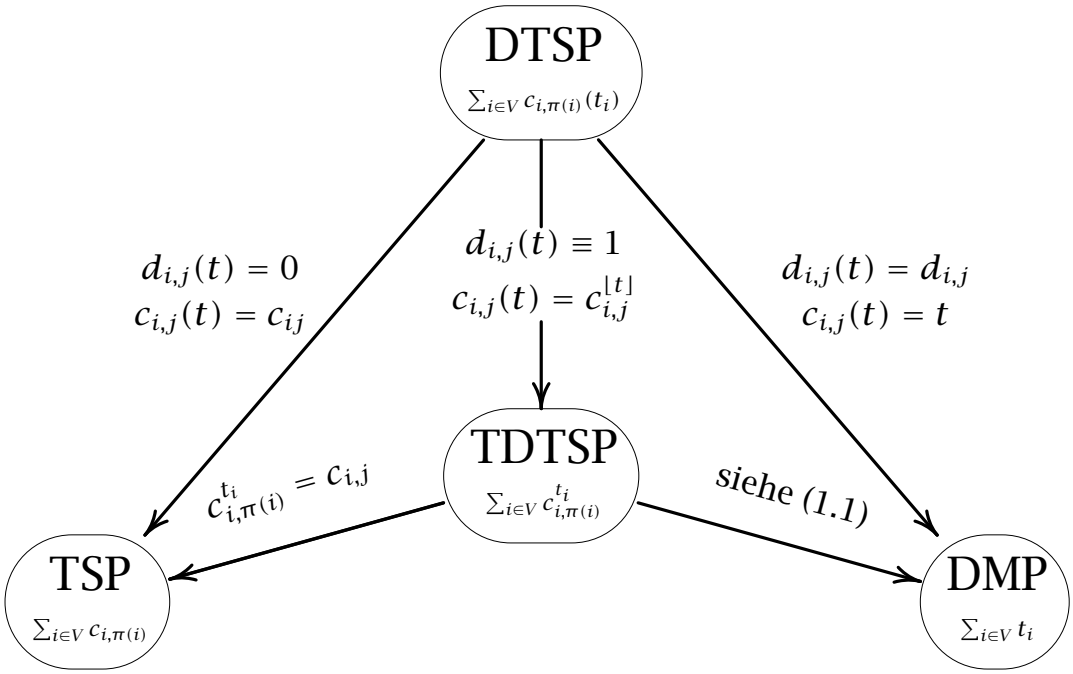


Abbildung 1.1: Eine Hierarchie von TSP–artigen Problemen

Es ist ausführlich untersucht worden [36, 44]. Das TSP ist ein Spezialfall des DTSP, bei dem die Kostenfunktionen konstant sind, d.h. es gilt $c_{i,j}(t) \equiv c_{i,j}$ (dadurch entfällt der Einfluß der Wegezeitfunktionen $d_{i,j}(t)$ auf die Zielfunktion). Die im DTSP enthaltene Festlegung der ersten zu besuchenden Stadt ist irrelevant, da die Zielfunktion für jede Wahl von v_0 gleich bleibt.

Abbildung 1.1 stellt das Verhältnis von DTSP, TDTSP, DMP und TSP graphisch dar. Die Pfeile (⟵) geben die Relation „ist Spezialisierung von“ an. In den Knoten ist die Zielfunktion des jeweiligen Problems angegeben.

1.2 Komplexitätsergebnisse

Das DTSP in seiner allgemeinen Form ist als Verallgemeinerung des TSP oder des DMP sicher NP-schwer. Besonders beachten muß man jedoch die Größe der Eingabekodierung, insbesondere die Kodierung der Wegezeit– und Kostenfunktionen. Werden sie als (endliche) Wertetabellen für $t = 0, \ldots, T$ abgelegt und der *Horizont* T ist größer als $n!$, die Gesamtzahl der Touren, so ergäbe die vollständige Enumeration aller Lösungen einen polynomialen Algorithmus

für das (DTSP). Daran ändert sich auch nichts, wenn der Horizont T neben n Modellgröße ist, da der Betrag von T nur logarithmisch in die Größe der Eingabekodierung eingeht. Selbstverständlich würde es genügen, die Wertetabellen nur für die Zeitpunkte auszufüllen, zu denen überhaupt eine Tour die betreffende Stadt erreichen kann. Aber selbst dann zeigt Satz 4.2 auf Seite 60, daß die Größe der Tabellen die Zahl der Touren übersteigen kann. Diese Überlegungen werden in Kapitel 4 noch vertieft.

Diese Arbeit geht deshalb immer davon aus, daß es eine Eingabekodierung des betrachteten DTSP gibt, die polynomial in n ist (zum Beispiel indem die Wegezeit– und Kostenfunktionen als stückweise lineare Funktionen mit polynomial vielen Knickpunkten und Steigungen durch Listen angegeben werden.).

Das TSP ist bekanntermaßen [26] genau dann konstant, wenn die Kantenkosten durch Kostenpaare (a_i^1, a_i^2) auf den Knoten in der Form $c_{ij} = a_i^1 + a_j^2$ gegeben sind. Jede Tour hat dann Kosten $\sum_{i\in V} a_i^1 + a_i^2$. Läßt man jetzt zu, daß sich die Knotenkosten alle zu einem gemeinsamen Zeitpunkt ändern, wird das Problem NP–schwer:

Satz 1.1
Das (DTSP) mit

$$c_{i,j}(t) = d_{i,j}(t) = \begin{cases} a_i & t < t^\star \\ b_i & t \geq t^\star \end{cases};\ t^\star > 0;\ a_i, b_i \in \mathbb{N}, a_i \geq b_i \tag{1.2}$$

ist streng polynomial äquivalent zum Rucksackproblem (KP).

Beweis:(nach [43])
(In diesem Beweis ist immer $V = \{0, \ldots, n\}$ und $v_0 = 0$)
Sei π eine beliebige Tour in V. Berechne zunächst $z(\pi)$, numeriere dazu so um, daß $\pi(i) = i + 1$. Im Fall $\sum_{i=0}^{n} a_i < t^\star$ gilt $z(\pi) = \sum_{i=0}^{n} a_i$, sonst:

$$\begin{aligned} z(\pi) &= \sum_{i\in V} c_{i,i+1}(t_i) \\ &= \sum_{i=0}^{n_0} a_i + \sum_{i=n_0+1}^{n} b_i && 0 = t_0 \leq \cdots \leq t_{n_0} < t^\star \leq t_{n_0+1} \leq \cdots \leq t_n \\ &= \sum_{i=0}^{n} b_i + \sum_{i=0}^{n_0} (a_i - b_i) && t_{n_0+1} = \sum_{i=0}^{n_0} a_i \geq t^\star \end{aligned} \tag{1.3}$$

Das Optimierungsziel ist es also, möglichst viel Wegezeit a_i in den Rucksack zu packen, mit möglichst wenig Extra–Gewicht vom Typ $a_i - b_i$.

Reduziere (KP) auf (DTSP):

Sei eine Instanz (KP) gegeben durch eine Menge $I = \{1, \ldots, n\}$, Preise $p_i \in \mathbb{N}, i \in I$, Gewichte $w_i \in \mathbb{N}, i \in I$, einen Mindestwert $\hat{c} \in \mathbb{N}$ und ein Maximalgewicht $\hat{b} \in \mathbb{N}$. Ohne Einschränkung der Allgemeinheit sei $p_i \geq w_i$ (sonst multipliziere alle p_i und $\hat{c}$ mit einem genügend großen Faktor). Gibt es eine Teilmenge $\hat{I} \subseteq I$, so daß

$$\sum_{i \in \hat{I}} p_i \geq \hat{c} \quad \text{und} \quad \sum_{i \in \hat{I}} w_i \leq \hat{b} \tag{1.4}$$

gilt?

Ist $\sum_{i \in I} p_i \leq \hat{c}$, so ist die Antwort trivial. Andernfalls konstruiere eine Instanz (DTSP) mit $V := I \cup \{0\}$, $a_0 = b_0 := 0$, $a_i := p_i, b_i := p_i - w_i, i \in I$ und $t^\star := \hat{c}$. Sei nun π eine Tour mit $z(\pi) \leq \sum_{i=1}^{n} b_i + \hat{b}$. Numeriere wieder so um, daß $\pi(i) = i + 1$. Wegen $\sum_{i=1}^{n} a_i > t^\star$ gilt $t_n \geq t^\star$ und es gibt ein n_0 mit $t_{n_0+1} \geq t^\star$. Dann gilt wie in (1.3):

$$\sum_{i=1}^{n} b_i + \sum_{i=1}^{n_0} w_i = z(\pi) \leq \sum_{i=1}^{n} b_i + \hat{b} \implies \sum_{i=1}^{n_0} w_i \leq \hat{b}$$

$$t_{n_0+1} = \sum_{i=1}^{n_0} p_i \geq \hat{c}$$

Also erfüllt $\hat{I} := \{1, \ldots, n_0\}$ die Bedingung (1.4).

Erfülle nun umgekehrt $\hat{I} = \{i_1, \ldots, i_{n_0}\}$ Gleichungen (1.4). Numeriere so um, daß $\hat{I} = \{1, \ldots, n_0\}$. Dann läßt sich die Tour $\pi := (0\,1 \cdots n_0\, n_0 + 1 \cdots n)$ wie oben nach (1.3) berechnen und es gilt $z(\pi) \leq \sum_{i=1}^{n} b_i + \hat{b}$.

Reduziere (DTSP) auf (KP):

Sei eine Instanz (DTSP) mit (1.2) gegeben. Gibt es eine Tour π mit $z(\pi) \leq \hat{z}$?

Ist $\sum_{i=0}^{n} a_i < t^\star$ sind alle Touren gleich teuer und die Antwort ist trivial. Andernfalls konstruiere eine Instanz (KP) mit $I := V$, $p_0 = w_0 := 0$, $p_i := a_i$, $w_i := a_i - b_i, i > 0$; $\hat{c} := t^\star - a_0$ und $\hat{b} := \hat{z} - \sum_{i=1}^{n} b_i - a_0$.

Es gebe nun ein $\hat{I} \subseteq I$, das (1.4) erfüllt. Entweder ist $0 \in \hat{I}$ oder man füge 0 hinzu, ohne (1.4) zu verletzen, da $w_0 = 0$. Numeriere wieder um, so daß $\hat{I} = \{0, 1, \ldots, n_0\}$. Betrachte $\pi := (01 \cdots n)$. Nach Konstruktion läßt sich (1.3) anwenden und es gilt:

$$t_{n_0+1} = \sum_{i=0}^{n_0} p_i + a_0 \geq \hat{c} + a_0 = t^\star$$

$$z(\pi) = \sum_{i=0}^{n} b_i + \sum_{i=1}^{n_0} w_i + (a_0 - b_0) \leq \sum_{i=1}^{n} b_i + \hat{b} + a_0 = \hat{z}$$

Umgekehrt gilt für eine (umnumerierte) Tour $\pi = (01 \cdots n)$ mit $z(\pi) \le \hat{z}$, daß $\hat{I} := \{1, \ldots, n_0\}$ die Bedingungen (1.4) erfüllt, denn:

$$\sum_{i=1}^{n_0} p_i = \sum_{i=0}^{n_0} a_i - a_0 = t_{n_0+1} - a_0 \ge t^\star - a_0 = \hat{c}$$
$$\sum_{i=1}^{n_0} w_i = \sum_{i=0}^{n_0} (a_i - b_i) - (a_0 - b_0) \le \hat{z} - \sum_{i=1}^{n} b_i - a_0 = \hat{b}$$

■

Aus Satz 1.1 ergibt sich sofort:

Korollar 1.2
Das (DTSP) mit (1.2) *ist NP-schwer, aber in pseudo-polynomialer Zeit lösbar.*

Beweis:
Das Rucksackproblem ist ein klassisches pseudo-polynomiales, NP-schweres Problem (MP9 in [23]). Das (DTSP) mit (1.2) ist sicher in NP, damit folgt die Behauptung aus Satz 1.1. ■

Das Rucksackproblem bleibt NP-schwer, falls Preise p_i und Gewichte w_i gleich sind (siehe [23]). Daraus folgt, daß das (DTSP) wie in Satz 1.1 mit identischen Verhältnissen zwischen a_i und b_i immer noch NP-schwer ist.

Satz 1.3
Das (DMP) reduziert sich polynomial auf das (DTSP) mit

$$d_{i,j}(t) \equiv d_{i,j} \in \mathbb{N}; \quad c_{i,j}(t) = d_{i,j} + \alpha t \tag{1.5}$$

Beweis:
Sei eine Instanz (DMP) gegeben mit den Städten $V = \{0, \ldots, n\}$, $v_0 = 0$ und den Entfernungen $d_{i,j}$. Gibt es eine Tour π mit $z_{\mathrm{DMP}}(\pi) \le \hat{c}$?
Betrachte dazu das (DTSP) mit (1.5), denselben Daten und $\alpha = (n+1) \max_{i,j} d_{i,j} + 1$. Berechne $z(\pi)$:

$$z(\pi) = \sum_{i=0}^{n} c_{i,\pi(i)}(t_i) = \sum_{i=0}^{n} d_{i,\pi(i)} + \alpha \sum_{i=0}^{n} t_i$$
$$\text{mit} \quad t_0 = 0$$
$$t_{\pi(i)} = t_i + d_{i,\pi(i)}$$
$$\Longrightarrow z(\pi) = z_{\mathrm{TSP}}(\pi) + \alpha z_{\mathrm{DMP}}(\pi) \tag{1.6}$$

Es gebe nun für das (DTSP) eine Tour π mit $z(\pi) < \alpha(\hat{c}+1)$. Dann gilt:

$$
\begin{aligned}
& z_{\mathrm{TSP}}(\pi) + \alpha z_{\mathrm{DMP}}(\pi) < \alpha(\hat{c}+1) \\
\Longrightarrow \quad & z_{\mathrm{DMP}}(\pi) < \hat{c} + \underbrace{\frac{\alpha - z_{\mathrm{TSP}}(\pi)}{\alpha}}_{\in(0;\,1]} \qquad \text{denn} \quad z_{\mathrm{TSP}}(\pi) < \alpha \\
\Longrightarrow \quad & z_{\mathrm{DMP}}(\pi) \le \hat{c}
\end{aligned}
$$

Gibt es nun umgekehrt eine Tour π mit $z_{\mathrm{DMP}}(\pi) \le \hat{c}$, so gilt $z(\pi) = z_{\mathrm{TSP}}(\pi) + \alpha z_{\mathrm{DMP}}(\pi) < \alpha(\hat{c}+1)$.

■

Die in Satz 1.3 untersuchte „wenig dynamische'" Variante des DTSP ist also bereits dann NP-schwer, wenn das DMP mit denselben Daten NP-schwer ist. So wird zum Beispiel vermutet, daß das DMP auf Bäumen NP-schwer ist , wo das TSP aber trivial lösbar ist (siehe [1, 52]).

Der einfachste Fall eines konstanten DTSP wäre sicher der, in dem alle Kostenfunktionen konstant sind. Läßt man allerdings zu, daß eine einzige Verbindung zu einem bestimmten Zeitpunkt billiger wird, erhält man ein NP–schweres Problem:

Satz 1.4
Das (DTSP) mit

$$
\begin{aligned}
& d_{i,j}(t) \equiv d_{i,j} \in \{0,1\} \quad \forall \qquad\qquad i,j \in V, i \neq j \\
(1.7) \qquad & c_{i,j}(t) = \begin{cases} 0 & (i,j) = (v_0, v_n) \text{ für } t \ge t^\star;\ t^\star \ge 0 \\ 1 & \text{sonst} \end{cases}
\end{aligned}
$$

ist NP-vollständig

Beweis:
Das (DSTP) mit (1.7) ist sicher in NP.
Reduktion von Hamiltonscher Kreis (HC) ([GT37] in [23]):
Gegeben sei eine Instanz (HC) im Graphen $G_{HC} = (V_{HC}, E_{HC})$. Gibt es dann in G_{HC} eine Tour?
Konstruiere zu jeder Kante $(v_n, v_0) \in E_{HC}$ eine Instanz $\mathfrak{I}_{v_n,v_0}$ von (DTSP) mit (1.7) durch:

$$
V := V_{HC};\ \ t^\star := n = |V_{HC}| - 1;\ \ d_{i,j} := \begin{cases} 1 & \text{falls } (i,j) \in E_{HC} \\ 0 & \text{sonst} \end{cases}
$$

In einer solchen Instanz hat eine Tour also genau dann Kosten n, wenn sie v_n erst zum Zeitpunkt n erreicht, das heißt, wenn sie alle anderen Städte auf Kanten aus E_{HC} vorher erreicht hat. Alle anderen Touren haben Kosten $n+1$.

Gibt es also eine Tour $\pi = (v_0 \cdots v_n)$ in G_{HC}, so gilt in der Instanz $\mathfrak{I}_{v_n,v_0}$ eben $z_{\mathrm{DTSP}}(\pi) = n$.

Gibt es umgekehrt in einer Instanz $\mathfrak{I}_{v_n,v_0}$ eine Tour π mit $z_{\mathrm{DTSP}}(\pi) \leq n$, so hat π zwischen v_0 und v_n alle anderen Städte durchlaufen. Dies muß auf Kanten mit $d_{i,j} = 1$ geschehen sein, um erst erst um $t_{v_n} = n$ in v_n anzukommen. Da $(v_n, v_0) \in E_{HC}$, ist π somit eine Tour in G_{HC}. ■

Korollar 1.5
Das (DTSP) mit

(1.8)

$$d_{i,j}(t) \equiv d_{i,j};\ c_{i,j}(t) = \begin{cases} a-d & t \geq t^\star \wedge i = i_0 \\ a & sonst \end{cases};\ a,d \in \mathbb{N}, d < a, t^\star \geq 0, i_0 \in V$$

ist polynomial äquivalent zum (TSP).

Beweis:
Das (DTSP) mit (1.7) aus dem vorhergehenden Satz ist ein Spezialfalls des (DTSP) mit (1.8). Wegen Satz 1.4 ist (DTSP) mit (1.7) also auch NP–vollständig und somit polynomial äquivalent zum (TSP), da dies auch NP–vollständig ist. ■

Satz 1.4 und Korollar 1.5 zeigen, daß bereits in der durch die Wegezeiten gegebenen Struktur die ganze Komplexität eines TSP verborgen sein kann, selbst wenn die Kostenfunktionen „beinahe“ konstant sind. Bemerkenswert ist, daß die Variante zu (1.8), in der die Kosten einer Verbindung zu genau einem Zeitpunkt $t^\star$ steigen, einfach zu lösen ist. Man braucht nur einen kürzesten Weg von v_0 nach i_0 beliebig zu einer Tour zu ergänzen, denn in kürzesten Wegen kommen Knoten höchstens einfach vor.

Kapitel 2

Formulierungen für TDTSP, DMP und DTSP

Die Art, in der das (DTSP) in der Definition 1.1 auf Seite 1 formuliert ist, eignet sich wenig als Basis für ein effizientes Lösungsverfahren. In diesem Kapitel werden eine Reihe von Formulierungen vorgestellt, die in der Literatur Verwendung finden. Man hat sich dort ([21, 22, 24, 29, 31, 46, 58, 63, 64]) hauptsächlich mit dem TDTSP und häufig nur mit einem Spezialfall, dem DMP ([6, 17, 32, 63]) beschäftigt. Abschnitte 2.1 und 2.2 stellen die verwendeten Formulierungen vor und gegenüber, Abschnitt 2.4 stellt schließlich die einzige bekannte DTSP–Formulierung (aus [47–49]) vor. Die Grundlagen zur linearen und ganzzahligen Optimierung, wie sie in diesem Kapitel verwendet werden, finden sich zum Beispiel in [54].

2.1 TDTSP-Formulierungen

Die TDTSP-Formulierungen zerfallen in zwei Klassen. Bei den kantenorientierten geben mit Start– und Zielort und Zeitpunkt dreifach indizierte Variablen an, wann eine bestimmte Verbindung genutzt wird. Der knotenorientierte Ansatz hingegen weist mittels doppelt indizierter Variablen jeder Stadt eine Position zu.

2.1.1 Kantenorientierte Formulierungen

Bereits Hadley schlägt 1964 in [31] die Verwendung von dreifach indizierten Entscheidungsvariablen x_{ijt} zur IP–Formulierung des TSP vor:

Formulierung (HAD):

$$\text{(2.1)} \qquad \min \sum_{i,j \in V} \sum_{t=0}^{n} c_{i,j}^{t} x_{ijt}$$

unter den Nebenbedingungen

$$\text{(2.2)} \qquad \sum_{i,j \in V} x_{ijt} = 1 \qquad t = 0, \ldots, n$$

$$\text{(2.3)} \qquad \sum_{j \in V} \sum_{t=0}^{n} x_{ijt} = 1 \qquad i \in V$$

$$\text{(2.4)} \qquad \sum_{i \in V} \sum_{t=0}^{n} x_{ijt} = 1 \qquad j \in V$$

$$\text{(2.5)} \qquad \sum_{i \in V} x_{ijt} = \sum_{k \in V} x_{j,k,t+1} \qquad t = 0, \ldots, n-1; \qquad j \in V$$

$$\text{(2.6)} \qquad \sum_{j \in V_0} x_{0j0} = 1$$

$$\text{(2.7)} \qquad x_{ijt} \in \{0, 1\} \qquad i, j \in V; \qquad t = 0, \ldots, n$$

x_{ijt} ist genau dann 1, wenn zum Zeitpunkt t die Verbindung $i \to j$ verwendet wird, d.h. $x_{ijt} = 1 \iff (t_i = t \wedge \pi(i) = j)$. Die „Assignment–Bedingungen“ (2.2)–(2.4) ordnen jedem Zeitpunkt eine Stadt und jeder Stadt eine Vorgängerin und eine Nachfolgerin zu. Durch die „Flußerhaltungsbedingung“ (2.5) wird sichergestellt, daß jede erreichte Stadt sofort wieder verlassen wird und (2.6) macht Stadt 0 zum Zeitpunkt 0 zum Anfang der Tour.

Klar ist, daß jede Tour π zu einer zulässigen Lösung von (2.1)–(2.6) führt. Ist nun so eine Lösung gegeben, legen (2.3) und (2.4) für jede Stadt i eine Nachfolgerin j fest und definieren dadurch eine Permutation $\pi \in S_{n+1}$. Wegen (2.6) gilt $t_0 = 0$. Bedingungen (2.2) stellen sicher, daß die linken und rechten Seiten von (2.5) nur die Werte 0 oder 1 annehmen können. Würde nun der Zykel von

π, der Stadt 0 enthält, vor Zeitpunkt n nach 0 zurückkehren, d.h. es existiert $t \leq n-1$ und i mit $x_{i0t} = 1$, so müßte wegen (2.5) $x_{0,k,t+1} = 1$ gelten für ein k. Dann wäre aber (2.3) für $i = 0$ durch $\sum_{j,t} x_{ijt} \geq x_{0,\pi(0),0} + x_{0,k,t+1} = 2$ verletzt. Somit ist π zyklisch. Zuletzt ist klar, daß durch (2.1) $z_{\text{TDTSP}}(\pi)$ berechnet wird.

Bereits Hadley merkte an, daß einige Variablen und Nebenbedingungen für Fälle mit $i = j$ und $t = 0, n$ weggelassen werden können. Hadley hatte diese Formulierung zur Lösung des TSP vorgeschlagen, d.h. mit $c_{ij}^t \equiv c_{ij}$. In [31] wird nicht von Anwendungen zur Lösung des TSP berichtet (siehe dazu auch Abschnitt 2.2.1 auf Seite 29).

1978 präsentieren Picard und Queyranne in [58] eine Vereinfachung, die von Houck und Vemuganti [37] vorgeschlagen wurde:

Formulierung (HVPQ):

$$\min \sum_{(i,j,t)\in\mathfrak{I}_{V,n}} c_{i,j}^t x_{ijt} \tag{2.1}$$

unter den Nebenbedingungen

$$x_{0j0} + \sum_{i\in V_0\setminus\{j\}} \sum_{t=1}^{n-1} x_{ijt} = 1 \qquad j \in V_0 \tag{2.8}$$

$$\sum_{j\in V_0} x_{0j0} = 1 \tag{2.9}$$

$$x_{0j0} = \sum_{i\in V_0\setminus\{j\}} x_{ji1} \qquad j \in V_0 \tag{2.10}$$

$$\sum_{i\in V_0\setminus\{j\}} x_{ijt} = \sum_{k\in V_0\setminus\{j\}} x_{j,k,t+1} \qquad j \in V_0;\ t = 1,\ldots,n-2 \tag{2.11}$$

$$\sum_{i\in V_0\setminus\{j\}} x_{i,j,n-1} = x_{j0n} \qquad j \in V_0 \tag{2.12}$$

$$x_{ijt} \in \{0,1\} \qquad (i,j,t) \in \mathfrak{I}_{V,n} \tag{2.13}$$

$$\mathfrak{I}_{V,n} := \{(0,j,0) \mid j \in V_0\} \cup \{(i,j,t) \mid i,j \in V_0, i \neq j, t = 1,\ldots,n-1\} \cup \{(i,0,n) \mid i \in V_0\}$$

Offensichtlich stellen (2.9)–(2.12) die Flußerhaltungsbedingungen für das in Abbildung 2.1 auf der folgenden Seite (für $n = 5$) dargestellte Netzwerk dar (Dieses

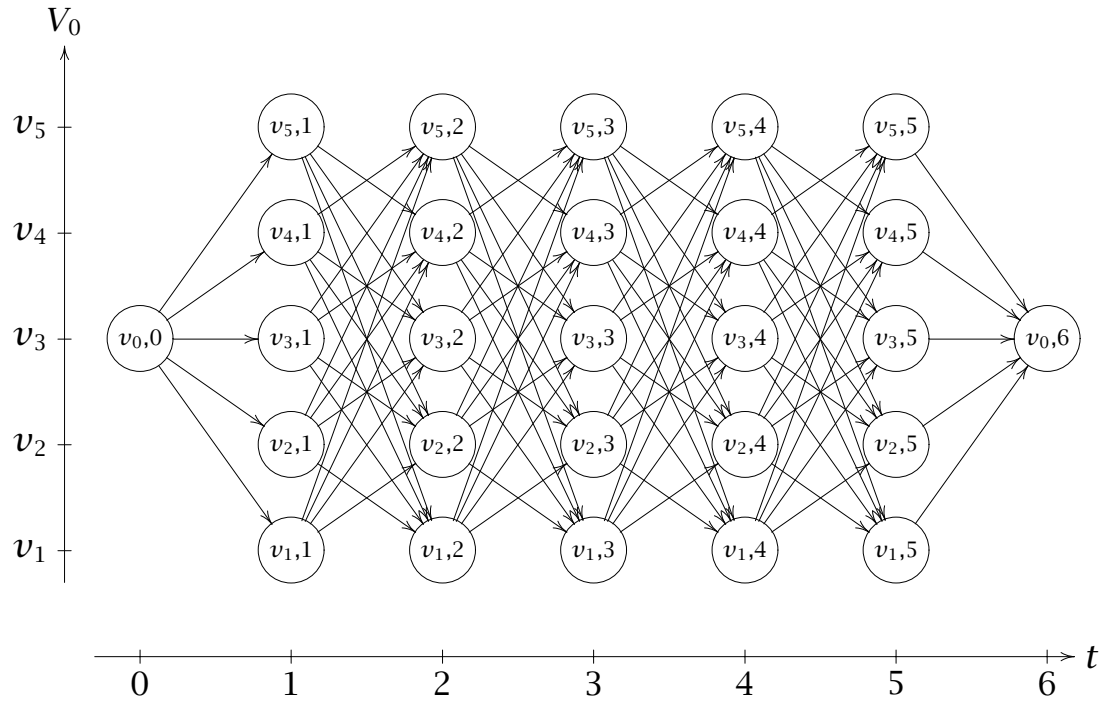


Abbildung 2.1: Netzwerk zu Formulierung (HVPQ)

Netzwerk wird später Basis für die Entwicklung eines DTSP–Algorithmus sein. Siehe dazu Definition 3.2 auf Seite 43 und Kapitel 4). Es ist im wesentlichen ein zeit–expandierter vollständiger Graph mit n Knoten. Jeder Knoten $v \in V_0$ existiert in n Kopien (v,t) für die Zeitpunkte $t = 1,\ldots,n$. Da v_0 nur zum Zeitpunkt $t = 0$ verlassen und zum Zeitpunkt $t = n+1$ wieder erreicht werden kann, augmentieren die Knoten $(v_0,0)$ und $(v_0,n+1)$ den Graphen als „Quelle“ und „Senke“. Auch die Verbindungen $v_i \to v_j$ existieren als Kanten in je einer Kopie für jeden Zeitpunkt, zu dem eine Benutzung möglich ist. Somit ist $x_{v_i v_j t}$ die Entscheidungsvariable für die Benutzung der Kante $((v_i,t),(v_j,t+1))$ und $c^t_{v_i v_j}$ sind die dadurch entstehenden Kosten.

Bedingung (2.9) macht den Knoten $(v_0,0)$ zur Quelle einer Flußeinheit, Bedingungen (2.10) erhalten den Fluß an den Knoten für $t = 1$, (2.12) an denen mit $t = n$ und (2.11) an den „mittleren“ mit $1 < t < n$. Ein Fluß mit Wert 1 ist ein Pfad und somit ist (HVPQ) eine IP–Formulierung für das Problem, einen billigsten Pfad von $(v_0,0)$ nach $(v_0,n+1)$ im Graphen von Abbildung 2.1 zu finden, der wegen (2.8) jede der Städte in genau einer Kopie besucht.

Die Formulierung hat $2n + n(n-1)^2 = n^3 - 2n^2 + 3n$ Variablen und $n(n-2) + 3n + 1 = n^2 + n + 1$ Nebenbedingungen, dabei ist allerdings (2.9) überflüssig, weil Gleichung (2.8) bereits einen Fluß größer Null erzwingt:

Lemma 2.1
(2.8),(2.10),(2.11) *und* $x_{ijt} \geq 0$ *implizieren* (2.9)*.*

Beweis:
Addiert man (2.10)–(2.12) jeweils für $j \in V_0$ auf, so erhält man

$$\sum_{\substack{i,j \in V_0 \\ i \neq j}} x_{ijt} = \sum_{j \in V_0} x_{0j0} \quad \forall t = 1, \ldots, n-1$$

Addiert man (2.8) ebenso auf, erhält man

$$\sum_{j \in V_0} x_{0j0} + \sum_{t=1}^{n-1} \sum_{\substack{i,j \in V_0 \\ i \neq j}} x_{ijt} = n$$

und weil darin also alle n Summanden gleich sind, folgt (2.9). ∎

Ansonsten ist (HVPQ) im Sinne ihrer LP-Relaxation (das durch Abschwächen von (2.13) zu $0 \leq x_{ijt} \leq 1$ entstehende LP ($HVPQ_L$)) irreduzibel, wie der folgende Satz zeigt.

Satz 2.2
Der durch (2.8),(2.10)–(2.12) *und* $0 \leq x_{ijt} \leq 1$ *definierte Polyeder hat für* $n \geq 3$ *Dimension* $n^3 - 3n^2 + 2n$*.*

Beweis:
Die angegebene Dimension ist die Differenz aus der Anzahl Variablen und der Anzahl der immer mit Gleichheit erfüllten Nebenbedingungen in (HVPQ). Da jede Tour eine zulässige Lösung darstellt und somit keine der Ungleichungen (2.13) für alle zulässigen Lösungen mit Gleichheit erfüllt sein kann, ist dazu nur noch zu zeigen, daß keine der Gleichungen überflüssig ist.
Assoziiere also Dualvariablen μ_j zu (2.8), λ_{j0} zu (2.10), λ_{jt} zu (2.11) und $\lambda_{j,n-1}$ zu (2.12) für $j \in V_0$ und $t = 1, \ldots, n-2$. Summiere die Gleichungen mit diesen Variablen multipliziert auf. In der Summengleichung haben die Variablen x_{ijt} folgende Koeffizienten:

(2.14)	x_{0j0} :	$\mu_j + \lambda_{j0}$	$j \in V_0$
(2.15)	x_{ijt} :	$\mu_j - \lambda_{i,t-1} + \lambda_{jt}$	$i, j \in V_0, i \neq j,\ t = 1, \ldots, n-1$
(2.16)	x_{j0n} :	$-\lambda_{j,n-1}$	$j \in V_0$

Die Dualvariablen seinen nun so, so daß alle Koeffizienten 0 werden. Zu zeigen ist dann, daß die Dualvariablen bereits 0 sind. Wegen (2.16) gilt

$$\lambda_{j,n-1} = 0 \quad \forall j \in V_0$$

Daraus folgt wegen (2.15) für $t = n-2$

$$\mu_j - \lambda_{i,n-2} = 0 \quad \forall i, j \in V_0, i \neq j$$

und ein Indexvergleich zeigt sofort

$$\mu_j = \lambda_{i,n-2} =: \bar{\mu} \quad \forall i, j \in V_0$$

Betrachtung von x_{0j0} liefert

$$\bar{\mu} + \lambda_{j0} = 0 \quad \forall j \in V_0$$

und von x_{ji1} liefert

$$\bar{\mu} - \lambda_{j0} + \lambda_{i1} = 0 \quad \forall i, j \in V_0, i \neq j$$

Daraus folgt sofort

$$\lambda_{i1} = -2\bar{\mu} \quad \forall i \in V_0$$

(2.15) liest sich jetzt

$$\bar{\mu} - \lambda_{j,t-1} + \lambda_{it} = 0 \quad \forall i, j \in V_0, i \neq j, t = 2, \ldots, n-2$$

und durch wiederholtes Einsetzen beginnend mit λ_{i1} erhält man

$$\lambda_{jt} = -(t+1)\bar{\mu} \quad \forall t = 0, \ldots, n-2,\ j \in V_0$$

Da nun aber bereits $\lambda_{i,n-2} = \bar{\mu}$ gilt, muß also

$$\bar{\mu} = 0$$

gelten und damit müssen alle Dualvariablen verschwinden. ■

Eine weitere Reduzierung des Dimension ist nicht mehr möglich, denn die konvexe Hülle der Inzidenzvektoren aller Touren, das heißt der zulässigen Lösungen von (HVPQ), hat für $n \geq 5$ dieselbe Dimension:

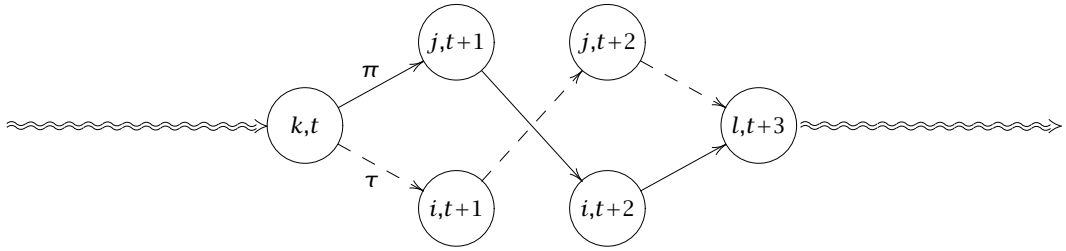
Satz 2.3
Sei $Q_{TDTSP}(n)$ die Menge der Punkte x, die (2.8)–(2.13) *erfüllen. Dann gilt für* $n \geq 5$

$$\dim(Q_{TDTSP}(n)) = n^3 - 3n^2 + 2n$$

Beweis:
Es ist zu zeigen, daß die Punkte aus $Q_{\mathrm{TDTSP}}(n)$ keine weitere, von (2.8)–(2.12) linear unabhängige, Gleichung erfüllen. Sei dazu

$$a^t x = \sum_{j \in V_0} a_{0j0} x_{0j0} + \sum_{\substack{i,j \in V_0 \\ i \neq j}} \sum_{t=1}^{n-1} a_{ijt} x_{ijt} + \sum_{j \in V_0} a_{j0n} x_{j0n} = b \tag{2.17}$$

so eine Gleichung. Zunächst ein paar Beobachtungen. Betrachte zwei Touren π und τ, die bis zur Zeit t gemeinsam verlaufen, sich dann in k trennen, danach die Städte i, j in unterschiedlicher Reihenfolge durchlaufen und sich dann in l wieder treffen und gemeinsam weiterverlaufen. Im folgenden Diagramm ist π mit durchgezogenen Pfeilen ($\longrightarrow$), τ mit unterbrochenen ($- - \rightarrow$) und der gemeinsame Verlauf mit doppelten, gewellten ($\rightsquigarrow$) eingezeichnet:



Seien x_π und x_τ die zugehörigen Inzidenzvektoren. Dann gilt:

$$0 = a^t x_\pi - a^t x_\tau = a_{kjt} - a_{kit} + a_{j,i,t+1} - a_{i,j,t+1} + a_{i,l,t+2} - a_{j,l,t+2} \tag{2.18}$$
$$\forall i, j \in V_0, k, l \in V, \quad |\{i, j, k, l\}| = 4, \quad t = 0, \ldots, n-2$$
$$k = v_0 \Longleftrightarrow t = 0, \quad l = v_0 \Longleftrightarrow t = n-2$$

Wegen $n \geq 5$ existiert eine weitere Stadt m. Substitution von k mit m zeigt:

$$a_{kjt} - a_{kit} = -(a_{j,i,t+1} - a_{i,j,t+1} + a_{i,l,t+2} - a_{j,l,t+2}) = a_{mjt} - a_{mit}$$

Somit hängt der Wert von $a_{kjt} - a_{kit}$ nicht mehr von k ab. Durch Substitution von l mit m erhält man dasselbe für $a_{i,l,t+2} - a_{j,l,t+2}$. Definiere daher:

$$C_{ijt} := a_{ilt} - a_{jlt} \qquad \forall i, j \in V_0, i \neq j \; t = 2, \ldots, n \tag{2.19}$$
$$C'_{ijt} := a_{kit} - a_{kjt} \qquad \forall i, j \in V, i \neq j \; t = 0, \ldots, n-2 \tag{2.20}$$

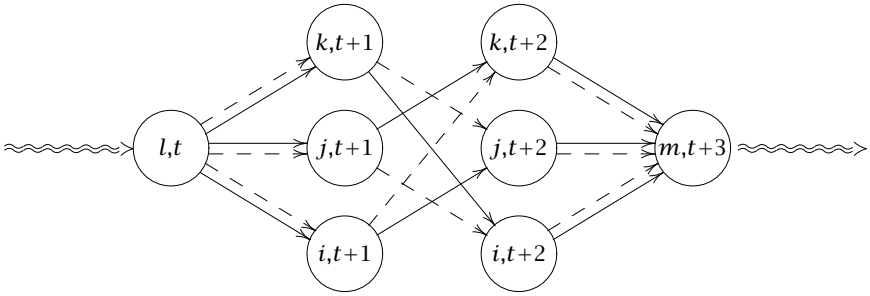
Zwar läßt (2.18) die Definition von C_{ijt} nur für $t = 2, \ldots, n$ zu, doch wegen

$$(a_{jk1} - a_{ik1}) - (a_{jl1} - a_{il1}) = C'_{kl1} - C'_{kl1} = 0$$

ist sie auch für $t = 1$ zulässig. Analog ist C'_{ijt} auch für $t = n-1$ wohldefiniert. Schließlich definiere noch $C_{iit} = C'_{iit} := 0$. Offensichtlich gelten folgende Zusammenhänge:

$$C_{ijt} = -C_{jit} \qquad C'_{ijt} = -C'_{jit} \tag{2.21}$$
$$C_{ijt} + C_{jkt} = C_{ikt} \qquad C'_{ijt} + C'_{jkt} = C'_{ikt} \tag{2.22}$$
$$C'_{jit} + a_{j,i,t+1} - a_{i,j,t+1} + C_{i,j,t+2} = 0 \quad \text{(äq. (2.18))} \tag{2.23}$$

Addiert man drei dieser Konstruktionen für fünf Städte i, j, k, l, m wie folgt:



erhält man:

$$C_{ikt} + C_{jit} + C_{kjt} = 0 \qquad t = 1, \ldots, n \tag{2.24}$$

$$C'_{ikt} + C'_{jit} + C'_{kjt} = 0 \qquad t = 0, \ldots, n-1 \tag{2.25}$$

Die Gültigkeit folgt auch direkt aus (2.21) und (2.22).

Assoziiere nun wie in Satz 2.2 zu den Gleichungen (2.8),(2.10)–(2.12) Multiplikatoren μ_j und λ_{jt} für $j \in V_0$ und $t = 0, \ldots, n-1$. Definiere:

$$\mu_2 := \begin{cases} \frac{n-1}{2n}(C'_{210} + C_{211}) + \frac{1}{n}\left(\sum_{s=0}^{\frac{n-2}{2}} a_{2,1,2s+1} + \sum_{s=1}^{\frac{n-2}{2}} a_{1,2,2s} + a_{010} + a_{20n}\right) \\ \qquad \text{falls } n \text{ gerade} \\ \frac{1}{2}(C'_{210} + C_{211}) + \frac{1}{n}\left(\sum_{s=1}^{\frac{n-1}{2}} a_{2,1,2s} + \sum_{s=1}^{\frac{n-1}{2}} a_{1,2,2s-1} + a_{020} + a_{20n}\right) \\ \qquad \text{falls } n \text{ ungerade} \end{cases} \tag{2.26}$$

$$\mu_i := \mu_2 + C'_{i20} + C_{i21} \qquad i \in V_0 \tag{2.27}$$

$$\lambda_{i0} := a_{0i0} - \mu_i \qquad i \in V_0 \tag{2.28}$$

$$\lambda_{1t} := \lambda_{2,t-1} - \mu_1 + a_{21t} \qquad t = 1, \ldots, n \tag{2.29}$$

$$\lambda_{it} := \lambda_{1,t-1} - \mu_i + a_{1it} \qquad i \in V_0 \setminus \{1\}, t = 1, \ldots, n \tag{2.30}$$

Zu Zeigen ist jetzt, daß Gleichungen (2.8),(2.10)–(2.12) mit diesen Faktoren aufaddiert gerade $a^t x = b$ ergeben, das heißt

$$a_{0j0} = \mu_j + \lambda_{j0} \qquad j \in V_0 \tag{2.31}$$

$$a_{ijt} = \mu_j - \lambda_{i,t-1} + \lambda_{jt} \qquad i, j \in V_0, i \neq j,\ t = 1, \ldots, n-1 \tag{2.32}$$

$$a_{j0n} = -\lambda_{j,n-1} \qquad j \in V_0 \tag{2.33}$$

müssen nachgeprüft werden (vergleiche (2.14)–(2.16). Offensichtlich gelten (2.31) nach den Definitionen (2.28). Rechne also (2.32) durch Induktion über $t = 1, \ldots, n-1$ nach.

Der Anfang für $t = 1$:

Für $i = 2, j = 1$ und $i = 1, j \neq i$ nach Definitionen (2.29) und (2.30) korrekt.
Für $j \neq 1, i \neq j$:

$$\begin{aligned}
\mu_j - \lambda_{i0} + \lambda_{j1} &= \mu_j - \lambda_{i0} + a_{1j1} - \mu_j + \lambda_{10} && \text{wg. (2.30)}\\
&= \mu_i - a_{0i0} + a_{1j1} + a_{010} - \mu_1 && \text{wg. (2.28)}\\
&= \mu_2 + C'_{i20} + C_{i21} + C'_{1i0} + C_{1i1} + a_{ij1} - \mu_2 + C'_{210} + C_{211} && \text{wg. (2.27)}\\
&= C'_{i20} + C'_{1i0} + C'_{210} + C_{i21} + C_{1i1} + C_{211} + a_{ij1} = a_{ij1} && \text{wg. (2.25), (2.24)}
\end{aligned}$$

Der Induktionsschritt $t-1, t-2 \to t$:

Für $j = 1, i = 2$ und $i = 1, j \neq 1$ ist (2.32) nach Definitionen (2.29),(2.30) korrekt.
Für $i \neq 1, j \neq 1, i \neq j$:

$$\begin{aligned}
\mu_j - \lambda_{i,t-1} + \lambda_{j,t} =& \mu_j - \lambda_{i,t-1} + a_{1jt} - \mu_j + \lambda_{1,t-1} && \text{wg. (2.30)}\\
&+ C'_{1,i,t-2} + a_{1,i,t-1} - a_{i,1,t-1} + a_{ijt} - a_{1jt} && \text{wg. (2.23)}\\
=& C'_{1,i,t-2} + \lambda_{1,t-1} - \lambda_{i,t-1} + a_{1,i,t-1} - a_{i,1,t-1} + a_{ijt}\\
=& C'_{1,i,t-2} + \lambda_{1,t-1} - a_{1,i,t-1} + \mu_i - \lambda_{1,t-2} + a_{1,i,t-1} - a_{i,1,t-1} + a_{ijt}\\
& && \text{wg. (2.30)} \wedge t > 1\\
=& C'_{1,i,t-2} + \lambda_{1,t-1} + \mu_i - \lambda_{1,t-2} - \mu_1 + \lambda_{i,t-2} - \lambda_{1,t-1} + a_{ijt}\\
& && \text{wg. I.V. für } a_{i,1,t-1}\\
\text{falls } t = 2 \quad =& C'_{1,i,t-2} + \mu_i - \mu_1 - a_{010} + \mu_1 + a_{0i0} - \mu_i + a_{ij2} && \text{wg. (2.28)}\\
=& a_{ijt}\\
\text{falls } t > 2 \quad =& a_{j,1,t-2} - a_{j,i,t-2} + \mu_i - \mu_1 - \lambda_{1,t-2} + \lambda_{i,t-2} + a_{ijt}\\
=& \mu_1 - \lambda_{j,t-3} + \lambda_{1,t-2} - \mu_i + \lambda_{j,t-3} - \lambda_{i,t-2} && \text{wg. I.V. für } a_{j,1,t-2}\\
&+ \mu_i - \mu_1 - \lambda_{1,t-2} + \lambda_{i,t-2} + a_{ijt} = a_{ijt} && \text{und } a_{j,i,t-2}
\end{aligned}$$

Bis jetzt wurde die Definition von μ_2 in (2.26) nicht benötigt. Sie kommt zustande durch Auflösen von (2.33) für a_{20n} nach μ_2. Berechne zunächst durch wiederholtes Einsetzen von (2.27) und (2.30) und schließlich (2.28) in (2.29) eine Darstellung von $\lambda_{1,t}$ in Abhängigkeit von μ_2 wie folgt:

$$
\begin{aligned}
\lambda_{1,t} &= a_{21t} - \mu_1 + \lambda_{2,t-1} \\
&= a_{21t} - \mu_2 + C'_{210} + C_{211} + a_{1,2,t-1} - \mu_2 + \lambda_{1,t-2} \qquad \text{wg. (2.27), (2.30)} \\
&= \dots \\
&= \begin{cases} -(t+1)\mu_2 + \frac{t+2}{2}(C'_{120} + C_{121}) + \sum_{s=1}^{\frac{t}{2}} a_{2,1,2s} + \sum_{s=1}^{\frac{t}{2}} a_{1,2,2s-1} + a_{010} & \text{t gerade} \\ -(t+1)\mu_2 + \frac{t+1}{2}(C'_{120} + C_{121}) + \sum_{s=0}^{\frac{t-1}{2}} a_{2,1,2s+1} + \sum_{s=1}^{\frac{t-1}{2}} a_{1,2,2s} + a_{020} & \text{t ungerade} \end{cases}
\end{aligned} \tag{2.34}
$$

Mit (2.32) für $a_{1,2,n-1}$ gilt dann:

$$-\lambda_{2,n-1} = \mu_2 - \lambda_{1,n-2} - a_{1,2,n-1}$$

und durch Einsetzen von (2.26) und (2.34)

$$a_{20n} = -\lambda_{2,n-1}$$

Rechne (2.33) weiter nach:
Für $j = 1$:

$$
\begin{aligned}
\lambda_{1,n-1} &= a_{2,1,n-1} - \mu_1 + \lambda_{2,n-2} && \text{wg .(2.29)} \\
&\quad + C'_{1,2,n-2} + a_{1,2,n-1} - a_{2,1,n-1} + C_{21n} && \text{wg. (2.23)} \\
&= a_{i,1,n-2} - a_{i,2,n-2} - \mu_1 + \lambda_{2,n-2} + a_{1,2,n-1} + C_{21n} \\
&= \lambda_{1,n-2} - \mu_2 + a_{1,2,n-1} + a_{20n} - a_{10n} && \text{wg. (2.32) für } a_{i,1,n-2}, a_{i,2,n-2} \\
&= -a_{10n}
\end{aligned}
$$

Für $j > 2$:

$$
\begin{aligned}
\lambda_{j,n-1} &= a_{1,j,n-1} - \mu_j + \lambda_{1,n-2} && \text{wg .(2.29)} \\
&\quad + C'_{j,1,n-2} + a_{j,1,n-1} - a_{1,j,n-1} + C_{1jn} && \text{wg. (2.23)} \\
&= a_{2,j,n-2} - a_{2,1,n-2} - \mu_j + \lambda_{1,n-2} + C_{1jn} + a_{j,1,n-1} \\
&= \lambda_{j,n-2} - \mu_1 + a_{j,1,n-1} + a_{10n} - a_{j0n} && \text{wg. (2.32) für } a_{2,j,n-2}, a_{2,1,n-2} \\
&= \lambda_{j,n-2} - \mu_1 + a_{j,1,n-1} - \lambda_{1,n-1} - a_{j0n} = -a_{j0n} && \text{wg. (2.32) für } a_{j,1,n-1}
\end{aligned}
$$

Schließlich gilt noch $b = \sum_{j \in V_0} \mu_j$. ■

Bemerkung: Wie man aufgrund der geringen Zahl an Ecken leicht nachrechnet (zum Beispiel mittels PORTA, [11]) gilt für 3 Städte $\dim(Q_{\text{TDTSP}}(3)) = 5$ bei $|Q_{\text{TDTSP}}(3)| = 6$ und für 4 Städte $\dim(Q_{\text{TDTSP}}(4)) = 22$ mit $|Q_{\text{TDTSP}}(4)| = 24$.

Eine weitere Reduzierung der Anzahl an Nebenbedingungen in (HVPQ) läßt sich durch Ausnutzen kombinatorischer Zusammenhänge, d.h. der Integralitätsbedingungen, erreichen. Fox bearbeitet 1973 in [22] Mehrmaschinen–Scheduling Probleme und verwendet dabei das TDTSP als (Einmaschinen–)Relaxation. Er gibt eine kompaktere Formulierung basierend auf denselben Variablen wie (HAD) an:

Formulierung (FOX):

$$\text{(2.1)} \qquad \min \sum_{i,j\in V} \sum_{t=0}^{n} c_{i,j}^{t} x_{ijt}$$

unter den Nebenbedingungen

$$\text{(2.2)} \qquad \sum_{i,j\in V} x_{ijt} = 1 \qquad\qquad t = 0, \ldots, n$$

$$\text{(2.3)} \qquad \sum_{j\in V} \sum_{t=0}^{n} x_{ijt} = 1 \qquad\qquad i \in V$$

$$\text{(2.4)} \qquad \sum_{i\in V} \sum_{t=0}^{n} x_{ijt} = 1 \qquad\qquad j \in V$$

$$\text{(2.35)} \qquad \sum_{k\in V} \sum_{t=1}^{n} t x_{jkt} = 1 + \sum_{k\in V} \sum_{t=0}^{n} t x_{kjt} \qquad\qquad j \in V_0$$

$$\text{(2.7)} \qquad x_{ijt} \in \{0,1\} \qquad\qquad i, j \in V; \quad t = 0, \ldots, n$$

In dieser Formulierung ersetzen also die n Bedingungen (2.35) die n^2 „Flußbedingungen“ (2.5). Zur Vereinfachung seien alle Variablen x_{ijt} mit $i = 0$ und $t \neq 0$ bzw. $j = 0$ und $t \neq n$ gleich Null (z.B. durch $c_{ij}^{t} = \infty$ für diese Indizes). Wie man leicht sieht, entsteht (2.35) als Summe von (2.5) für $t = 0, \ldots, n-1$ mit t multipliziert und (2.3) für $j \in V_0$:

$$\sum_{k\in V} \sum_{t=0}^{n-1} t x_{kjt} = \sum_{k\in V} \sum_{t=1}^{n} (t-1) x_{jkt} \qquad \text{das ist} \quad \sum_{t} t * (2.5)$$

$$1 = \sum_{k\in V} \sum_{t=1}^{n} x_{jkt} \qquad \text{das ist} \quad (2.3) \text{ mit } x_{jk0} = 0$$

$$\sum_{k\in V} \sum_{t=0}^{n-1} t x_{kjt} + 1 = \sum_{k\in V} \sum_{t=1}^{n} t x_{jkt} \qquad \text{das ist} \quad (2.35) \text{ wg. } x_{kjn} = 0, j \in V_0$$

(2.2)–(2.3) sichern, daß es jede zulässige Lösung eine Permutation der Städte definiert. Bleibt noch zu zeigen, daß (2.35) ausreicht, Untertouren zu verhindern

und den zeitlichen Zusammenhang zu sichern. Betrachte dazu eine Untertour $(i_1 i_2 \cdots i_k)$, die v_0 nicht enthält. Wegen (2.2) gibt es eine Stadt, die nicht zum Zeitpunkt 0 besucht wird. Sei also i_1 so, daß $x_{i_1,i_2,t_{i_1}} = 1$ mit $t_{i_1} > 0$. Weiterhin sei $x_{i_2,i_3,t_{i_2}} = 1$. Dann gilt in (2.35) für i_2:

$$\sum_{k\in V}\sum_{t=0}^{n} t x_{k,i_2,t} + 1 = t_{i_1} + 1 = t_{i_2} = \sum_{k\in V}\sum_{t=1}^{n} t x_{i_2,k,t}$$

Dasselbe Argument der Reihe nach für $i_2, \ldots, i_{k-1}$ angewandt liefert:

$$x_{i_1,i_2,t_{i_1}} = x_{i_2,i_3,t_{i_1}+1} = \cdots = x_{i_{k-1},i_k,t_{i_1}+k-1} = 1$$

mit $t_{i_1} > 0$. Für $j = i_1$ ergibt (2.35):

$$\sum_{k\in V}\sum_{t=1}^{n} t x_{i_1 k t} - \sum_{k\in V}\sum_{t=0}^{n} t x_{k i_1 t} = t_{i_1} - (t_{i_1} + k - 1) = 1 - k \neq 1$$

Also kann es nur eine Untertour geben, die v_0 enthält und wegen $x_{ij0} = 0$ für $i \in V_0$ zum Zeitpunkt 0 startet. Auf dieselbe Weise beweist man dann den zeitlichen Zusammenhang für diese Tour. Beachte, daß die Argumentation $x_{ijt} \in \{0, 1\}$ voraussetzt.

In [22] berichtet Fox, daß es mit einem Simplex–basierten Branch&Bound–Verfahren nicht gelang, ein 10–Städte Problem mit $c_{ij}^t \equiv c_{ij}$ in 12 Minuten zu lösen (ohne Angabe des Computers).

Fox, Gavish und Graves leiten in [21], durch Aufsummieren von (2.2)–(2.4), eine noch kompaktere Formulierung mit nur n Nebenbedingungen ab:

Formulierung (FGG):

(2.1) $$\min \sum_{i,j\in V}\sum_{t=0}^{n} c_{i,j}^t x_{ijt}$$

unter den Nebenbedingungen

(2.35),(2.7)

(2.36) $$\sum_{i,j\in V}\sum_{t=0}^{n} x_{ijt} = n$$

Die Korrektheit wird in [21] gezeigt. Gouveia und Voß [29] geben in [29] zulässige Lösungen für (FGG) an, die unzulässig sind für (FOX) und zulässige für (FOX), die unzulässig sind für (HVPQ).

Im Gegensatz zu der klassischen IP–Formulierung des TSP von Dantzig, Fulkerson und Johnson (siehe [14], auch [24]), die bei $O(n^2)$ binären Variablen exponentiell viele Nebenbedingungen zur Verhinderung von Untertouren benötigt, ist (FGG) also äußerst kompakt. Zur Lösung des TDTSP konnte diese Eigenschaft bis jetzt nicht genutzt werden. Die meisten Verfahren zur Lösung von IP-Problemen benötigen eher möglichst exakte Beschreibungen des ganzzahligen Polyeders durch viele Nebenbedingungen, wohingegen (FGG) mit seiner um etwa n^2 größeren Dimension als (HVPQ) naturgemäß eher „großzügig“ ist.

2.1.2 Knotenorientierte Formulierungen

Bereits Picard und Queyranne weisen in [58] darauf hin, daß das TDTSP ein Spezialfall des „Quadratic Assignment Problem“ (QAP) ist:

Formuierung (QAP):

$$\min \sum_{i\in V_0} \sum_{t=1}^{n} \sum_{j\in V_0} \sum_{s=1}^{n} c_{itjs} y_{it} y_{js} \tag{2.37}$$

unter den Nebenbedingungen

$$\sum_{t=1}^{n} y_{it} = 1 \qquad i \in V_0 \tag{2.38}$$

$$\sum_{i\in V_0} y_{it} = 1 \qquad t = 1,\ldots,n \tag{2.39}$$

$$y_{it} \in \{0,1\} \qquad i \in V_0, t = 1,\ldots,n \tag{2.40}$$

mit $c_{itjs} = 0$ außer:

$$\begin{aligned} c_{i1i1} &= c^0_{0i} \qquad c_{inin} = c^n_{i0} \qquad i \in V_0 \\ c_{i,t,j,t+1} &= c^t_{ij} \qquad i,j \in V_0, i \neq j, t = 1,\ldots,n \end{aligned} \tag{2.41}$$

Die Variablen y_{it} bestimmen die Zuordnung von Städten zu Zeitpunkten oder Positionen, das heißt y_{it} ist 1, genau, wenn Stadt i zum Zeitpunkt t besucht

wird, beziehungsweise an Position t der Tour steht. Diese Interpretation macht klar, daß es in der Formulierung (QAP) gar keine Untertouren geben kann. Leider wird die geringen Zahl an Variablen (n^2) und Nebenbedingungen ($2n$) durch eine quadratische Zielfunktion erkauft. Diese läßt sich, wie Lawler bereits in [45] gezeigt hat, durch Einführung zusätzlicher Variablen x_{ijt} linearisieren. Dabei hat $x_{ijt} = 1$ die Interpretation aus Abschnitt 2.1.1, nämlich daß auf Stadt i zum Zeitpunkt t Stadt j folgt:

$$x_{ijt} = 1 \iff y_{it}y_{j,t+1} = 1 \qquad i,j \in V_0, i \neq j;\ t = 1,\dots,n-1 \tag{2.42}$$

Beachte, daß die x_{ijt} für $t = 0, n$ fehlen. Gouveia und Voß geben als Linearisierung von (QAP) folgendes Modell an:

Formulierung (QAPL1):

$$\min \sum_{i\in V_0} c^0_{0i}y_{i1} + \sum_{t=1}^{n-1}\sum_{\substack{i,j\in V_0\\ i\neq j}} c^t_{ij}x_{ijt} + \sum_{i\in V_0} c^n_{i0}y_{in} \tag{2.43}$$

unter den Nebenbedingungen

(2.39), (2.38), (2.40)

$$\sum_{t=1}^{n-1}\sum_{\substack{i,j\in V_0\\ i\neq j}} x_{ijt} = n-1 \tag{2.44}$$

$$y_{it} + y_{j,t+1} - 2x_{ijt} \geq 0 \qquad i,j \in V_0, i \neq j;\ t = 1,\dots,n \tag{2.45}$$

$$x_{ijt} \in \{0,1\} \qquad i,j \in V_0, i \neq j;\ t = 1,\dots,n-1 \tag{2.46}$$

Um die Formulierung zu verschärfen, verwenden sie zur stärkeren Koppelung der x_{ijt} und y_{it} die folgenden, zusätzlichen Nebenbedingungen:

$$y_{it} = \sum_{j\in V_0\setminus\{i\}} x_{ijt} \qquad i \in V_0;\ t = 1,\dots,n-1 \tag{2.47}$$

$$y_{it} = \sum_{j\in V_0\setminus\{i\}} x_{j,i,t-1} \qquad i \in V_0;\ t = 2,\dots,n \tag{2.48}$$

Das folgende Lemma zeigt, daß die Koppelungsgleichungen (2.47) und (2.48) sowohl die Zuordungsbedingung (2.39) als auch die schwächeren Koppelungsbedingungen (2.44) und (2.45) ersetzen.

Lemma 2.4
In (QAPL1) *mit* $y_{it}, x_{ijt} \geq 0$ *anstatt* (2.40),(2.46) *werden* (2.39),(2.44) *und* (2.45) *durch* (2.47) *und* (2.48) *impliziert.*

Beweis:(nach [29])
Aufaddieren von (2.38) und ebenso von (2.47) und (2.48) für alle i liefert

$$\text{(2.49)} \qquad \sum_{i \in V_0} \sum_{t=1}^{n} y_{it} = n$$

$$\text{(2.50)} \qquad \sum_{i \in V_0} y_{it} = \sum_{\substack{i,j \in V_0 \\ i \neq j}} x_{ijt} = \sum_{\substack{i,j \in V_0 \\ i \neq j}} x_{jit} = \sum_{i \in V_0} y_{i,t+1} \qquad t = 1, \ldots, n-1$$

$$\Rightarrow \qquad \sum_{i \in V_0} y_{it} = 1 \qquad t = 1, \ldots, n$$

$$\Longleftrightarrow \quad (2.39)$$

, da wegen (2.50) alle Summanden in (2.49) gleich sind. Setze nun (2.47) in (2.49) ein

$$\sum_{i \in V_0} \sum_{t=1}^{n-1} \sum_{j \in V_0 \setminus \{i\}} x_{ijt} = n - \sum_{i \in V_0} y_{in}$$

und mit (2.39) für $t = n$ folgt (2.44). Weiter gilt:

$$\text{(2.47)} \quad \Rightarrow \quad y_{it} - x_{ijt} \geq 0 \qquad i, j \in V_0, i \neq j,\ t = 1, \ldots, n-1$$
$$\text{(2.48)} \quad \Rightarrow \quad y_{i,t+1} - x_{jit} \geq 0 \qquad i, j \in V_0, i \neq j,\ t = 1, \ldots, n-1$$

und (2.45) folgt als Summe der letzten beiden Gleichungen. ■

Gouveia und Voß, erhalten mit diesen Ergebnissen eine verbesserte Linearisierung (QAPL2) von der sie zeigen, daß sie tatsächlich schärfer ist als (QAPL1), das heißt, die Zulässigkeitspolyeder der LP-Relaxationen sind echt ineinander enthalten. (QAPL2) sieht dann so aus:

Formulierung (QAPL2):

$$\min \sum_{i \in V_0} c^0_{0i} y_{i1} + \sum_{t=1}^{n-1} \sum_{\substack{i,j \in V_0 \\ i \neq j}} c^t_{ij} x_{ijt} + \sum_{i \in V_0} c^n_{i0} y_{in} \tag{2.43}$$

unter den Nebenbedingungen

$$\sum_{t=1}^{n} y_{it} = 1 \qquad i \in V_0 \tag{2.38}$$

$$y_{it} = \sum_{j \in V_0 \setminus \{i\}} x_{ijt} \qquad i \in V_0;\ t = 1, \ldots, n-1 \tag{2.47}$$

$$y_{it} = \sum_{j \in V_0 \setminus \{i\}} x_{j,i,t-1} \qquad j \in V_0;\ t = 2, \ldots, n \tag{2.48}$$

$$x_{ijt} \in \{0,1\} \qquad i,j \in V_0, i \neq j;\ t = 1, \ldots, n-1 \tag{2.46}$$

$$y_{it} \in \{0,1\} \qquad i \in V_0, t = 1, \ldots, n \tag{2.40}$$

Leider ist durch die Linearisierung die Attraktivität von (QAP) stark gesunken, weil die Zahl der binären Variablen auf $O(n^3)$ gestiegen ist. Einen Ausweg weist Satz 2.4, da er die Integralitätsbedingungen nicht voraussetzt. Wie Vander Wiel und Sahinidis in [64] zeigen, läßt sich (2.46) zu $x_{ijt} \geq 0$ abschwächen. Dadurch bleibt die Zahl der binären Variablen y_{it} bei n^2. Vander Wiel und Sahinidis nutzen dies zur approximativen Lösung des TDTSP aus, indem sie die kontinuierlichen Variablen durch eine Zerlegung nach Benders (siehe [54] für eine ausführliche Beschreibung dieser Methode) behandeln.

Der folgende Satz zeigt nun, daß (HVPQ) und (QAPL2) „gleichwertig“ sind. Es scheint in der Literatur keine standardisierte Definition für die Äquivalenz von IP–Problemen zu geben. Beachte, daß die hier verwendete Definition mittels eines lienearen Isomorphismus nichts über eventuelle Verzerrungen oder numerische Eigenchaftenm aussagt, obwohl in diesem speziellen Fall keine anomales Verhalten erwartet wird.

Satz 2.5
(HVPQ) *und* (QAPL2) *sind äquivalent, das heißt, es gibt einen linearen Isomorphismus* φ *zwischen den Zulässigkeitspolyedern* $F(QAPL2))$ *und* $F(HVPQ)$ *der LP–Relaxationen mit*

$$\varphi(F(QAPL2)) = F(HVPQ)$$
$$z_{QAPL2} = z_{HVPQ} \circ \varphi$$

Beweis:

Die Abbildung φ ist zu verstehen als ein Mittel zur Identifizierung der formal unterschiedlichen Variablenräume miteinander. Da sich die Dimensionen der Variablenräume aber unterscheiden, benennt φ nicht nur einfach die Variablen um, sondern „erzeugt" auch noch die fehlenden und läßt die überflüssigen weg. Sei $F(HVPQ)$ der Zulässigkeitspolyeder der LP-Relaxation von (HVPQ) und sei $F(QAPL2)$ der von (QAPL2). Definiere eine lineare Abbildung φ wie folgt:

(2.51)
$$\begin{aligned}
&\varphi : F(QAPL2) \to F(HVPQ) \\
&\varphi_{0i0}(x,y) = y_{i1} && i \in V_0 \\
&\varphi_{ijt}(x,y) = x_{ijt} && i,j \in V_0, i \neq j;\ t = 1,\ldots,n-1 \\
&\varphi_{i0n}(x,y) = y_{in} && i \in V_0
\end{aligned}$$

Wenn φ bijektiv und neutral bezüglich des Zielfunktionswerts ist, ist die Behauptung des Satzes bewiesen. Die Umkehrfunktion φ^{-1} lautet:

(2.52)
$$\begin{aligned}
&\varphi^{-1} : F(HVPQ) \to F(QAPL2) \\
&\varphi^{-1}_{i1}(x) = x_{0i0} && i \in V_0 \\
&\varphi^{-1}_{ijt}(x) = x_{ijt} && i,j \in V_0, i \neq j;\ t = 1,\ldots,n-1 \\
&\varphi^{-1}_{in}(x) = x_{i0n} && i \in V_0 \\
&\varphi^{-1}_{it}(x) = \sum_{j \in V_0 \setminus \{i\}} x_{ijt} && i \in V_0;\ t = 2,\ldots,n-1
\end{aligned}$$

Offensichtlich gilt $\varphi \circ \varphi^{-1} = \mathrm{id}$. Rechne $\varphi^{-1} \circ \varphi$ nach:

$$(\varphi^{-1} \circ \varphi)_{it}(x,y) = \sum_{j \in V_0 \setminus \{i\}} \varphi_{ijt}(x,y) = \sum_{j \in V_0 \setminus \{i\}} x_{ijt} = y_{it} \text{ wg. (2.47)}$$

Jetzt ist noch $\varphi(F(QAPL2)) \subseteq F(HVPQ)$ und $\varphi^{-1}(F(HVPQ)) \subseteq F(QAPL2)$ zu zeigen. Sei $(x,y) \in F(QAPL2)$. Rechne die Nebenbedingungen nach:

$$\begin{aligned}
(2.8):\quad \varphi_{0j0}(x,y) + \sum_{i \in V_0 \setminus \{j\}} \sum_{t=1}^{n-1} \varphi_{ijt}(x,y) &= y_{j1} + \sum_{i \in V_0 \setminus \{j\}} \sum_{t=1}^{n-1} x_{ijt} && j \in V_0 \\
&= y_{j1} + \sum_{t=2}^{n} y_{jt} = 1 && \text{wg. (2.48), (2.38)}
\end{aligned}$$

$$(2.10): \quad \sum_{j\in V_0\setminus\{i\}} \varphi_{ij1}(x,y) = \sum_{j\in V_0\setminus\{i\}} x_{ij1} \qquad i \in V_0$$
$$= y_{i1} = \varphi_{0i0}(x,y) \qquad \text{wg. (2.47)}$$
$$(2.48): \quad \sum_{i\in V_0\setminus\{j\}} \varphi_{ijt}(x,y) = y_{j,t+1} \qquad j \in v_0;\ t = 1,\dots,n-2$$
$$= \sum_{k\in V_0\setminus\{j\}} x_{j,k,t+1} \qquad \text{wg. (2.48), (2.38)}$$
$$= \sum_{k\in V_0\setminus\{j\}} \varphi_{j,k,t+1}(x,y)$$
$$(2.12): \quad \sum_{i\in V_0\setminus\{j\}} \varphi_{i,j,n-1}(x,y) = \sum_{i\in V_0\setminus\{j\}} x_{i,j,n-1} \qquad j \in V_0$$
$$= y_{jn} = \varphi_{j0n}(x,y) \qquad \text{wg. (2.48)}$$

also folgt $\varphi(x,y) \in F(HVPQ)$. Sei nun umgekehrt $x \in F(HVPQ)$. Rechne die Nebenbedingungen von (QAPL2) nach:

$$(2.38): \quad \sum_{t=1}^{n} \varphi_{it}^{-1}(x) = x_{0i0} + \sum_{s=2}^{n-1} \sum_{j\in V_0\setminus\{i\}} x_{ijs} + x_{i0n} \qquad i \in V_0$$
$$= x_{0i0} + \sum_{s=2}^{n-1} \sum_{k\in V_0\setminus\{i\}} x_{k,i,s-1} + \sum_{k\in V_0\setminus\{i\}} x_{k,i,n-1} \qquad \text{wg. (2.11), (2.12)}$$
$$= 1 \qquad \text{wg. (2.8)}$$
$$(2.47): \quad \sum_{j\in V_0\setminus\{i\}} \varphi_{ijt}^{-1}(x) = \sum_{j\in V_0\setminus\{i\}} x_{ijt} \qquad i \in V_0;\ t = 1,\dots,n-1$$
$$= \varphi_{it}^{-1}(x) \qquad \text{wg.(2.52)}$$
$$(2.48): \quad \sum_{j\in V_0\setminus\{i\}} \varphi_{j,i,t-1}^{-1}(x) = \sum_{j\in V_0\setminus\{i\}} x_{j,i,t-1} \qquad i \in V_0;\ t = 2,\dots,n$$
$$= \sum_{j\in V_0\setminus\{i\}} x_{ijt} = \varphi_{it}^{-1}(x) \qquad \text{wg.(2.11)}$$

Weil auch noch wegen (2.38)

$$\sum_{t=1}^{n} \varphi_{it}^{-1}(x) = 1 \Rightarrow \varphi_{it}^{-1}(x) \le 1 \qquad i \in V_0;\ t = 2,\dots,n-1$$

gilt, folgt $\varphi^{-1}(x) \in F(QAPL2)$. Einsetzen zeigt sofort, daß φ den Zielfunktionswert nicht verändert. ■

2.2 DMP–Formulierungen

2.2.1 Spezialisierung des CAP

Das Zuordnungsproblem (Assignment Problem, AP) ist eine der bekanntesten Relaxationen für das asymmetrische TSP (siehe [44]). In Abschnitt 2.1.2 wurde aus dem quadratischen Zuordnungsproblem eine TDTSP–Formulierung hergeleitet. In der Arbeitsgruppe von Professor Hamacher wurde eine kumulative Version des Zuordnungsproblems untersucht [30, 32]. Das kumulative Zuordnungsproblem ist wie folgt definiert:

Definition 2.1
Eine Instanz des *kumulativen Zuordnungsproblems, (CAP)* ist gegeben durch eine Menge $I = \{1, \ldots, n\}$ mit n Elementen, n *kumulativen Faktoren* $a_1 \geq a_2 \geq \cdots \geq a_n \geq 0$ und eine $n \times n$–Kostenmatrix (c_{ij}). Mit diesen Daten soll das folgende Problem gelöst werden:

$$\min_{\psi,\pi \in S_n} \sum_{i \in I} a_{\psi(i)} c_{i,\pi(i)} \tag{2.53}$$

■

Im Gegensatz zum nicht–kumulativen Zuordnungsproblem müssen also neben der optimalen Permutation π der Elemente von I auch noch gleichzeitig die kumulativen Faktoren a_i optimal verteilt werden. Hamacher, Maffioli und Dell'Amico geben in [32] die folgende, zweistufige ganzzahlige Formulierung an:

Formulierung (CAP):

$$\min_{y_{ij}} \min_{\psi \in S_n} \sum_{i,j \in I} c_{ij} y_{ij} a_{\psi(i)} \tag{2.54}$$

unter den Nebenbedingungen

$$\sum_{i \in I} y_{ij} = 1 \qquad j \in I \tag{2.55}$$

$$\sum_{j \in I} y_{ij} = 1 \qquad i \in I \tag{2.56}$$

$$y_{ij} \in \{0, 1\} \qquad i, j \in I \tag{2.57}$$

Die Formulierung ist offensichtlich korrekt, da bekannt ist, daß (2.55) und (2.56) eine Permutation π auf I definieren, für die

$$\sum_{i,j\in I} c_{ij}y_{ij} = \sum_{i\in I} c_{i,\pi(i)}$$

gilt. Die innere Minimierung ist trivial, wenn die Zuordnung π bereits festgelegt ist, denn natürlich verteilt man am besten den kleinsten Faktor auf die größten Kosten, das heißt, das optimale ψ erfüllt

$$c_{i,\pi(i)} \le c_{j,\pi(j)} \iff a_{\psi(i)} \ge a_{\psi(j)} \iff \psi(i) \le \psi(j)$$

Anders formuliert sortiert ψ^{-1} die Kosten $c_{i,\pi(i)}$ aufsteigend. Damit eine Lösung ψ, π ist eine DMP–Tour darstellt (vergleiche (DMP) auf Seite 3), muß zunächst π zyklisch sein. Die kumulativen Gewichte beim DMP haben die Werte

$$a_0 = n > a_1 = n-1 > \cdots a_i = n-i \cdots > a_0 = 0$$

und dürfen nicht mehr beliebig verteilt werden. Die Permutation ψ muß mit der Reihenfolge in der Tour verträglich sein. Es ergibt sich folgende DMP–Formulierung (in diesem Abschnitt sei $V = \{0, \ldots, n\}$):

Formulierung (HMD):

$$\text{(2.54)} \qquad \min_{y_{ij}} \min_{\psi\in S_n} \sum_{i,j\in V} c_{ij}y_{ij}a_{\psi(i)}$$

unter den Nebenbedingungen

$$\text{(2.55)} \qquad \sum_{i\in V} y_{ij} = 1 \qquad\qquad j \in V$$

$$\text{(2.56)} \qquad \sum_{j\in V} y_{ij} = 1 \qquad\qquad i \in V$$

$$\text{(2.58)} \qquad y_{ij} = 1 \Rightarrow \psi(i) < \psi(j) \qquad\qquad i,j \in V, j \neq 0$$

$$\text{(2.57)} \qquad y_{ij} \in \{0,1\} \qquad\qquad i,j \in V$$

$$\text{(2.59)} \qquad a_i := n-i \qquad\qquad i \in V$$

Bedingungen (2.58) bewirken zunächst, daß den Elementen (Städten) von V, die früher in der Tour stehen, die größeren Faktoren zugeordnet werden. Außerdem verhindern sie Untertouren $(i_0 i_1 \cdots i_k)$, die 0 nicht enthalten, denn:

$$y_{i_0i_1} = y_{i_1i_2} = \cdots = y_{i_{k-1}i_k} = y_{i_ki_0} = 1$$
$$\Rightarrow \psi(i_0) < \psi(i_1) < \cdots < \psi(i_{k-1}) < \psi(i_k) < \psi(i_0)$$

und das ist sicher nicht möglich. Also gibt es nur eine Untertour in π, etwa $(0i_1 \cdots i_n)$. Dann gilt oben:

$$\psi(0) < \psi(i_1) < \cdots < \psi(i_{n-1}) < \psi(i_n)$$

und da nun $\psi(0)$ kleiner als alle anderen n möglichen Werte ist, muß

$$\psi(0) = 0$$

gelten. Damit ist (HMD) eine korrekte DMP–Formulierung, die nun wie in [32] durch Einführung von Variablen x_{ijt} mit

$$x_{ijt} = 1 \iff \psi(i) = t \wedge y_{ij} = 1$$

linearisiert wird:

Formulierung (HMDL):

$$\text{(2.60)} \qquad \min \sum_{i,j \in V} \sum_{t=0}^{n} c_{ij} a_t x_{ijt}$$

unter den Nebenbedingungen

$$\text{(2.61)} \qquad \sum_{t=0}^{n} \sum_{i \in V} x_{ijt} = 1 \qquad j \in V$$

$$\text{(2.62)} \qquad \sum_{t=0}^{n} \sum_{j \in V} x_{ijt} = 1 \qquad i \in V$$

$$\text{(2.63)} \qquad \sum_{i \in V} \sum_{j \in V} x_{ijt} = 1 \qquad t = 0, \ldots, n$$

$$\text{(2.64)} \qquad \sum_{i \in V} x_{ijt} = \sum_{k \in V} x_{j,k,t+1} \qquad t = 0, \ldots, n-1; \quad j \in V_0$$

$$\text{(2.57)} \qquad x_{ijt} \in \{0,1\} \qquad i,j \in V; \quad t = 0, \ldots, n$$

$$\text{(2.59)} \qquad a_i := n - i \qquad i \in V$$

Bedingung (2.63) sorgt dafür, daß für jeden Wert von t genau ein $x_{ijt} = 1$ ist und definiert dadurch die Permutation ψ mittels

$$\psi(i) = t \iff x_{ijt} = 1$$

Bedingungen (2.61) und (2.62) sind einfach (2.55) und (2.56) für $y_{ij} = \sum_{t=0}^{n} x_{ijt}$ und erzeugen so wie gehabt eine Permutation π von V. Bedingungen (2.64) schließlich garantieren, daß aus $x_{ijt} = 1$ auch $x_{j,k,t+1} = 1$ folgt, stellen also (2.58) sicher. Umgekehrt ist durch $x_{i,j,\psi(i)} = y_{ij}$ und $x_{ijt} = 0$ für $\psi(i) \neq t$ sicher eine zulässige Lösung gegeben.

Bis auf (2.6) und die Kantenkosten ist (HMD) identisch zur Formulierung (HAD) auf Seite 12. Die Kosten $c_{ij}a_t = (n-t)c_{ij}$ sind die eines DMPs formuliert als TDTSP–Spezialfall (siehe Seite 3) und (2.6) läßt sich genauso wie $\psi(0) = 0$ für (HMD) folgern. Bei der Spezialisierung des kumulativen Zuordnungsproblems zum DMP entsteht also eine bereits bekannte TDTSP–Formulierung.

Bianco, Mingozzi und Ricciardelli benutzen 1990 in [6] genau diese Formulierung zur exakten Lösung des DMP. Sie entwickeln daraus ähnlich wie Picard und Queyranne [58] 1978 und Lucena [46] einen Branch&Bound–Algorithmus basierend auf einer n–Pfad Lagrange–Relaxation zur Berechnung unterer Schranken. Dieser Ansatz wird in Kapitel 3 ausführlich beschrieben. Zusätzlich gewinnen sie durch Dynamische Programmierung mit beschränktem Zustandsraum einen heuristischen Algorithmus für das DMP. Sie lösen damit auf einem PC zufällige Beispiele mit bis zu 35 Städten exakt und mit bis zu 60 Städten innerhalb von 15% der Optimallösung.

2.3 Netzwerkfluß und kumulative Matroide

Fischetti, Laporte und Martello stellen 1993 in [17] eine DMP–Formulierung vor, die wie (HVPQ) ebenfalls Eigenschaften von Netzwerkfluß– und Zuordnungsproblem kombiniert. Da das darauf aufgebaute Verfahren einige spezielle, hier noch nicht eingeführte theoretische Ergebnisse benötigt und sich eher nicht auf das DTSP verallgemeinern läßt, würde eine ausführliche Besprechung den Rahmen dieser Arbeit überschreiten. Dieser Abschnitt gibt also nur die Grundzüge wieder:

Formulierung (FLM):

$$\min_{i,j\in V} d_{ij} z_{ij} \tag{2.65}$$

unter den Nebenbedingungen

$$\sum_{j\in V} y_{ij} = 1 \qquad i \in V \tag{2.66}$$

$$\sum_{i\in V} y_{ij} = 1 \qquad j \in V \tag{2.67}$$

$$\sum_{i\in V_0} z_{iv_0} = 1 \tag{2.68}$$

$$\sum_{i\in V} z_{ij} - \sum_{k\in V} z_{jk} = \begin{cases} -n & j = v_0 \\ +1 & j \in V_0 \end{cases} \tag{2.69}$$

$$z_{ij} \le \begin{cases} y_{ij} & j = v_0 \\ (n+1)y_{ij} & i = v_0 \\ n y_{ij} & \text{sonst} \end{cases} \tag{2.70}$$

$$y_{ij} \in \{0,1\};\ z_{ij} \in \mathbb{N} \qquad i,j \in V \tag{2.71}$$

Die Variablen y_{ij} definieren wegen (2.66) und (2.67) eine Permutation π auf V mit Untertouren. die man an der Zielfunktion (2.65) ablesen kann, erfüllen die z_{ij} die Aufgabe des Produkts $a_{\psi(i)}x_{ij}$ in (2.54) auf Seite 29, kombinieren also Tour und kumulative Faktoren. Sei K_V der vollständige Digraph mit der Knotenmenge V. Die z_{ij} definieren nun einen speziellen Kreisfluß auf K_V. Dieser Fluß verliert an jedem Knoten in V_0 wegen (2.69) eine weitere Einheit, bis er mit nur noch einer Einheit in v_0 ankommt (wegen (2.68)). Bedingung (2.69)

macht v_0 dann zur Quelle für n Flußeinheiten. Bedingungen (2.70) stellen sicher, daß der Fluß nur in den Untertouren von π verlaufen kann. Da aber v_0 die einzige Quelle ist, gibt es keine von v_0 getrennten Untertouren. Ist nun $\pi = (v_0 v_1 \cdots v_n)$ so gilt $z_{v_0 v_1} = n + 1$, denn wegen $y_{v_0 v_1} = 1$ kann der Fluß v_0 auf keiner anderen Kante verlassen. Entlang der Kanten von π verliert z_{ij} dann immer eine Einheit, das heißt es gilt

$$z_{v_i v_j} = n + 1 - i$$

Also ist (FLM) korrekt.

Relaxiere nun die Flußerhaltungsbedingungen (2.69) teilweise dergestalt, daß der Fluß zwar nicht mehr an den Knoten zusammenhängt, aber immer noch auf genau $n+1$ Kanten die Werte von 1 bis $n+1$ annimmt. So wie beim CAP die kumulativen Gewichte beliebig auf die Elemente verteilt werden durften, dürfen sie hier nun beliebig auf die Kanten verteilt werden. So betrachtet ist (FLM) eine andere Art (CAP) zu einer DMP–Formulierung zu machen, indem die Knotenfaktoren durch Kantenfaktoren ersetzt werden.

Das relaxierte Problem bedeutet, zu jedem Knoten in V eine dort endende Kante so auszuwählen, so daß die Summe der kumulativen Kantenkosten minimal wird. Das ist für $z_{ij} = 1$ das head–partition Matroid und läßt sich durch den Greedy–Algorithmus mit Komplexität $O(n^2)$ lösen (siehe zum Beispiel [41, Satz 5.1.3]). Fischetti, Laporte und Martello zeigen, daß Matroid–Probleme mit kumulativen Faktoren ebenfalls durch den Greedy–Algorithmus mit Komplexität $O(n^2)$ lösbar sind. Statt einer Tour liefert diese Relaxation also einen Digraph mit $n+1$ Kanten, dessen Knoten alle indegree eins haben. Die kumulativen Faktoren werden nicht mehr „der Reihe nach“, sondern beliebig über die Kanten dieses Graphen verteilt.

Das ist eine recht weitgehende Relaxation, die in [17] nach der Lagrange–Methode (siehe Abschnitt 3.1 ab Seite 39) und durch Einführen zusätzlicher, ebenfalls Lagrange–artig relaxierter, gültiger Nebenbedingungen verschärft wird. Außerdem entwickeln die Autoren für den Spezialfall symmetrischer Kosten eine angepaßte Relaxation. Trotz einer recht spezialisierten Aufstiegsprozedur bleiben die berechneten unteren Schranken, insbesondere bei nicht–euklidischen Beispielen, deutlich hinter den n–Pfad–basierten Schranken in [6, 46] zurück. Der auf diesen Schranken aufbauende Branch&Bound–Algorithmus erzeugt zur Lösung von Instanzen mit bis zu 55 Städten bis zu 300000 Knoten. Da die unteren Schranken aber mit sehr wenig Aufwand berechnet werden, scheint es sich hierbei um das schnellste zur Zeit bekannte Verfahren zur exakten Lösung des DMP zu handeln. Es löst auf einer IBM RS6000/320 Workstation Instanzen mit bis zu 50 Städten in unter einer Stunde.

2.4 DTSP–Formulierungen

Kapitel 4 verallgemeinert die bereits vorgestellte TDTSP–Formulierung (HVPQ) von Seite 13 auf das DTSP. Dieser Abschnitt stellt kurz die einzige dem Autor bekannte Formulierung in der Literatur vor, die eine nicht durch Einheitszeiten definierte zeitliche Struktur zuläßt. Malandraki und Daskin untersuchen 1992 in [47] eine Variante des *Vehicle Routing Problems*, das sie als TDVRP bezeichnen, in dem für die Wegezeit Treppenfunktionen mit m Stufen zugelassen sind. Die Funktionen $d_{ij}(t)$ werden dabei durch eine Unterteilung des Zeitintervalls $[0;\ T_{ijm}]$ in m Teilintervalle $[T_{ijl};\ T_{i,j,l+1}]$ definiert, so daß $d_{ij}(t) \geq 0$ innerhalb jedes Intervalls konstant ist:

$$d_{ij}(t) := \begin{cases} d_{ijl} & \text{falls } t \in (T_{ijl}, T_{i,j,l+1}) \\ \min\{d_{ijl}, d_{i,j,l-1}\} & \text{falls } t = T_{ijl} \end{cases} \tag{2.72}$$

Die Endpunkte gehören also immer zu dem Intervall mit den geringeren Wegezeiten. Die implizite Forderung nach derselben Anzahl Intervalle für alle Verbindungen stellt keine Einschränkung dar, wenn m die maximal gewünschte Zahl an Intervallen ist und erlaubt eine vereinfachte Notation. Außerdem soll T_{ijm} immer genügend groß sein, daß keine Tour die Kante zu einem späteren Zeitpunkt benutzt.

Die IP–Formulierung von Malandraki und Dial verwendet die Variablen x_{ijt}. Allerdings haben sie im Gegensatz zu bisher eine etwas geänderte Interpretation:

$$x_{ijt} = 1 \iff \pi(i) = j \wedge t_i \in [T_{ijt};\ T_{i,j,t+1}] \tag{2.73}$$

Der dritte Index gibt jetzt also nicht mehr den genauen Zeitpunkt t_i des Besuchs in i an, sondern nur noch in welchem Zeitintervall er stattfindet. Aus diesem Grund müssen auch Variablen t_i zur Angabe der Besuchszeitpunkz explizit mitgeführt werden. Die Originalformulierung in [47] wurde fürs TDVRP aufgestellt und beinhaltet also mehrere Transportfahrzeuge mit zusätzlichen Kapazitätsbeschränkungen. Ziel ist die Minimierung der Gesamtreisezeit. Für die Zwecke dieser Arbeit wird die Formulierung unter Beschränkung auf einen Reisenden ohne Transportfunktion dargestellt. Ersetzt man die Wegezeiten d_{ijt} in der Zielfunktion durch Kosten c_{ijt}, die analog zu (2.73) und (2.72) zu verstehen sind so entsteht eine IP–Formulierung für ein DTSP mit Treppenfunktionen als Wegezeit– und Kostenfunktion. Malandraki und Daskin behandeln in [47] ausdrücklich diesen von ihnen als TDTSP bezeichneten Fall, geben die folgende Formulierung aber nicht explizit an.

Formulierung (MD):

$$\min \sum_{i,j \in V} \sum_{t=0}^{m} c_{ijt} x_{ijt} \tag{2.74}$$

unter den Nebenbedingungen

$$\sum_{i \in V} \sum_{t=0}^{m-1} x_{ijt} = 1 \qquad j \in V \tag{2.75}$$

$$\sum_{j \in V} \sum_{t=0}^{m-1} x_{ijt} = 1 \qquad i \in V \tag{2.76}$$

$$t_j - t_i - L_1 x_{ijt} \geq d_{ijt} - L_1 \qquad i \in V, j \in V_0, i \neq j;\ t = 0, \ldots, m-1 \tag{2.77}$$

$$t_i + L_2 x_{ijt} \leq T_{ijt} + L_2 \qquad i \in V, j \in V_0, i \neq j;\ t = 0, \ldots, m-1 \tag{2.78}$$

$$t_i \geq T_{i,j,t-1} x_{ijt} \qquad i \in V, j \in V_0, i \neq j;\ t = 0, \ldots, m-1 \tag{2.79}$$

$$t_{v_0} = 0 \tag{2.80}$$

$$t_i \geq 0 \qquad i \in V \tag{2.81}$$

$$x_{ijt} \in \{0, 1\} \qquad i, j \in V;\ \ t = 0, \ldots, m-1 \tag{2.82}$$

$$L_1, L_2 \gg 0 \text{ const}$$

Bedingungen (2.75) und (2.76) sind die bekannten Zuordnungsbedingungen und deshalb definieren die x_{ijt} eine Familie von Untertouren ohne zeitlichen Zusammenhang zwischen den Verbindungen. Ist $x_{ijt} = 0$, so stellt (2.77) keine Bedingung dar (für genügend großes L_1). Gilt hingegen $x_{ijt} = 1$, so liest sich (2.77) folgendermaßen:

$$t_j - t_i \geq d_{ijt}$$

Zusammen mit $t_{v_0} = 0$ aus (2.80) sorgt (2.77) also dafür, daß t_i tatsächlich die Abfahrtszeit aus Stadt i angibt. Ungleichungen (2.78) und (2.79) garantieren nun die Einhaltung der Wegezeitfunktionen. Ist $x_{ijt} = 0$, so stellt (2.78) für genügend großes L_2 keine Einschränkung dar und im umgekehrten Fall wird $t_i \leq T_{ijt}$ erzwungen. In diesem Fall erzwingt (2.79) dann auch $t_i \geq T_{i,j,t-1}$, so daß die Abfahrtszeit tatsächlich im richtigen, zu d_{ijt} gehörenden Intervall liegen muß. Im Fall von $t_i = T_{ijt}$ kann entweder x_{ijt} oder $x_{i,j,t+1}$ eins sein, aber die Zielfunktion (2.74) stellt sicher, daß die Verbindung mit den geringeren Wegezeiten in der Optimallösung verwendet wird. Schließlich erzwingen für $x_{ijt} = 1$ Gleichungen (2.77) $t_i < t_j$, so daß Untertouren ohne v_0 ausgeschlossen sind.

Der Einsatz von „großen Zahlen“ in IP–Formulierungen führt zu eher losen LP–Relaxationen. Bedingungen (2.77) sind eine Verallgemeinerung der von Miller,

Tucker und Zemlin 1960 zur Verhinderung von Untertouren beim TSP eingeführten Ungleichungen (siehe [24]):

$$u_i - u_j + n y_{ij} \leq n - 1 \qquad i, j \in V_0, i \neq j$$

die ebenfalls eher lose sind (siehe [65]). Malandraki verallgemeinert nun zusätzlich die stärkeren Eliminationsbedingungen von Dantzig, Fulkerson und Johnson [14]

$$\sum_{i,j \in S} \sum_{t=0}^{m-1} x_{ijt} \leq |S| - 1 \qquad S \subset V \tag{2.83}$$

und gibt auch stärkere Bedingungen zur Koppelung der t_i an. Malandraki und Daskin bauen eine Branch&Cut–Heuristik zur Lösung des DTSP auf diesen und weiteren, nicht genannten, gültigen Nebenbedingungen auf. Sie berichten, daß die Heuristik nur in einem Drittel der Fälle in der Lage war, bessere Ergebnisse zu liefern als eine Reihe von Nearest–Neighbor Heuristiken und dazu noch wesentlich rechenzeitaufwendiger war. Malandraki und Dial [48,49] entwickeln dann eine recht effektive DP–Heuristik für das TDVRP durch Beschränkung des Zustandsraums (siehe auch [6] für einen ähnlichen Ansatz). Dieser letzte Algorithmus läßt sich als eine Nearest–Neighbor Methode beschreiben, in der die k nächsten Nachbarn für variierendes k berücksichtigt werden.

Kapitel 3

Lösungsverfahren

3.1 Lagrange-Relaxation für IP–Probleme

Die Lagrange–Relaxation ist ein inzwischen klassisches Verfahren zur Berechnung von unteren Schranken für lineare IP–Probleme. Sie wurde in den 70er Jahren entwickelt. Die erste, höchst erfolgreiche Anwendung fand sie 1970 bei Held und Karp zur Lösung des TSP in [33, 34]. Die Bezeichnung „Lagrange-Relaxation“ wurde 1974 von Geoffrion in [25] geprägt. Hervorragende Übersichtsartikel finden sich auch in [19, 62].

Die Idee hinter der Lagrange-Relaxation ist es, ein schweres kombinatorisches Problem als eine eigentlich einfache Aufgabe zu betrachten, die durch einige zusätzliche Bedingungen erschwert wird (in [33, 34] zum Beispiel die Berechnung eines aufspannenden Baumes zuzüglich einer weiteren Kante, dessen Knoten alle Grad zwei haben sollen).

Ein ganzzahliges lineares Programm läßt sich wie folgt schreiben:

$$
\text{(P)} \qquad \begin{aligned} \min\ & c^t x =: z(x) \\ \text{s.d.}\quad & Ax = b \\ & Dx = e \\ & x \in \mathbb{N}^n \end{aligned}
$$

wobei x Dimension $n \times 1$, b $m \times 1$, e $k \times 1$ und c, A, D passende Dimensionen haben. Für die Zwecke dieser Arbeit sei (P) zulässig mit endlichem Optimalwert. Mit (P^L) sei das LP gemeint, das durch Abschwächen von $x \in \mathbb{N}$ zu $x \geq 0$ entsteht.

Die Lagrange-Relaxation (PR_λ) von (P) bezüglich $Ax = b$ entsteht durch teilweise Dualisierung:

$$
\begin{aligned}
& & \min\ & c^t x + \lambda^t (b - Ax) =: z_\lambda(x) \\
(PR_\lambda) & & \text{s.d.}\quad & Dx = e \\
& & & x \in \mathbb{N}^n
\end{aligned}
$$

wobei $\lambda \in \mathbb{R}^m$ ein Vektor von *Lagrange-Faktoren* ist. In einer erfolgversprechenden Anwendung der Methode sollte (PR_λ) im Vergleich zu (P) einfach zu lösen sein. Zur vereinfachten Darstellung sei die Menge aller zulässigen Lösungen von (PR_λ) $F(PR_\lambda) = \{x \in \mathbb{N} \mid Dx = e\}$ endlich. Unter den angegebenen Voraussetzungen ist dann $z_\lambda^\star$, der optimale Zielfunktionswert von (PR_λ), für jedes λ endlich. Sei $x^\star$ eine Optimallösung von (P) und $z^\star := z(x^\star)$ der optimale Zielfunktionswert, dann gilt

$$
z_\lambda^\star \le c^t x^\star + \lambda^t (b - Ax^\star) = z^\star \tag{3.1}
$$

nach Definition von $z_\lambda^\star$ und wegen $b - Ax^\star = 0$. Allerdings hat die Dualitätslücke zwischen rechter und linker Seite von (3.1) im allgemeinen einen Minimalwert größer 0. Die bestmögliche auf diese Weise erreichbare untere Schranke für $z^\star$ ist die Lösung des folgenden Optimierungsproblems:

$$
z_D^\star := \max_{\lambda \in \mathbb{R}} z_\lambda^\star \tag{D}
$$

Wie die Bezeichnung (D) schon andeutet, bilden (P) und (D) ein (schwach) duales Paar von Optimierungsproblemen, denn es gilt $z_D^\star \le z^\star$, und falls $z_\lambda^\star = z(x)$ für ein Paar x, λ, dann sind x und λ optimal für (P) beziehungsweise (D). Aufgrund der Annahme, daß $F(PR_\lambda)$ endlich ist, läßt sich (D) als LP schreiben:

$$
\begin{aligned}
& & z_D^\star = \max\ & w \\
(D') & & \text{s.d.}\quad & w \le c^t x + \lambda^t (b - Ax) \qquad x \in F(PR_\lambda) \\
& & & w \in \mathbb{R}, \quad \lambda \in \mathbb{R}^m
\end{aligned}
$$

(D') ist korrekt, denn

$$\begin{aligned} z_\lambda^\star &= \min\{c^t x + \lambda(Ax - b) \mid x \in F(PR_\lambda)\} \\ &= \max\{w \mid w \leq c^t x + \lambda(Ax - b) \forall x \in F(PR_\lambda)\} \end{aligned}$$

Damit ist $z_\lambda^\star$ als Funktion von λ das Minimum einer Menge linearer Funktionen und somit stückweise linear und konkav. Die Knickpunkte befinden sich an den Stellen λ, an denen (PR_λ) mehrfache Optimallösungen hat. Da sich (D') natürlich nur theoretisch aufstellen läßt und $z_\lambda^\star$ nicht differenzierbar ist, bietet sich das Subgradienten–Verfahren zur Optimierung nach λ in (D) an (siehe zum Beispiel [35]). Das Verfahren benötigt als Aufstiegsrichtung in einem Punkt λ_i ein Element d_i des Subgradienten $\partial z_{\lambda_i}^\star$. Offensichtlich ist wegen (D') die konvexe Hülle von $\{b - Ax \mid x$ Optimallösung von $(PR_{\lambda_i})\}$ gerade der Subgradient. Der nächste Punkt λ_{i+1} wird dann als

$$\lambda_{i+1} := \lambda_i + \delta_i d_i$$

bestimmt, wobei δ_i ein Parameter zur Schrittweitenbeschränkung ist. Grundsätzlich ist bekannt (siehe [27,35,59]), daß das Verfahren gegen $z_D^\star$ konvergiert, solange

$$\delta_i \overset{i\to\infty}{\longrightarrow} 0 \text{ und } \sum_{i=1}^{\infty} \delta_i = \infty$$

gilt. In der Praxis wird meist die auch von Held und Karp benutzte Formel

$$\delta_i = \frac{\tau_i(\bar{z} - z_{\lambda_i}^\star)}{\|d_i\|^2} \qquad \tau_i \in (0;\ 2] \text{ Kontrollparameter; } \bar{z} \in \mathbb{R} \tag{3.2}$$

angewendet. Werden dabei die Parameter τ_i und $\bar{z}$ so gewählt, daß $\bar{z}$ eine untere Schranke für $z_D^\star$ und $\epsilon < \tau_i \leq 2$ für ein $\epsilon > 0$ gilt, so konvergiert das Verfahren entweder gegen $z_D^\star$ oder ein Punkt λ_i mit $z_{\lambda_i}^\star > \bar{z}$ wird erreicht (siehe [35]). Im Fall von $\bar{z} = z_D^\star$ konvergiert das Verfahren sogar geometrisch. Da es aber häufig tatsächlich nicht trivial ist, eine geeignete untere Schranke $\bar{z}$ zu bestimmen, kommen mit gutem Erfolg meist obere Schranken zur Verwendung, die sich leicht als Zielfunktionswerte $\bar{z} = z(x)$ von zulässigen Lösungen x von (P) gewinnen lassen. Das Subgradientenverfahren garantiert auch im Falle der Konvergenz gegen $z_D^\star$ keine monoton steigenden Iterationswerte $z_{\lambda_i}^\star$. Ist die Schrittweite zu lang, „schießt der Schritt über den Berg hinaus“ und ein λ_{i+1} mit $z_{\lambda_{i+1}}^\star < z_{\lambda_i}^\star$ wird berechnet. Kommt das zu häufig vor, wird der Kontrollparameter τ_i halbiert.

Zunächst einmal ist $\lambda_0 = 0$ ein geeigneter Startwert für die Iteration, manchmal jedoch bieten sich bessere Werte an, zum Beispiel wenn die Relaxation im Rahmen eine Branch&Bound–Verfahrens gelöst wird. Dann kann man gegebenenfalls bessere Startwerte aus den optimalen Lagrange–Faktoren des Vater–Knotens im Baum bestimmen (siehe [58]).

A–priori Aussagen über die Güte der unteren Schranke $z_D^\star$ für $z^\star$ sind sicher kaum möglich, allerdings zeigt Geoffrion [25], wie man die Dualitätstheorie der Linearen Programmierung anwenden kann. Er definiert

Definition 3.1 (nach [25]**)**
Die Formulierung (PR_λ) hat die *Integralitätseigenschaft*, falls sich für alle λ der optimale Zielfunktionswert $z_\lambda^\star$ nicht ändert, wenn $x \in \mathbb{N}$ zu $x \geq 0$ abgeschwächt wird. ■

Die Integralitätseigenschaft haben zum Beispiel alle Probleme, in denen D unimodular ist. In diesem Fall ist $z_D^\star$ nicht besser als die LP-Relaxation von (P), also der optimale Zielfunktionswert $z_L^\star$ von (P^L).

Satz 3.1
Es gilt

$$z_L^\star \leq z_D^\star$$

mit Gleichheit, falls (PR_λ) *die Integralitätsbedingung erfüllt. In diesem Fall gilt für die optimalen Dualvariablen* $\lambda^\star$ *zu den Nebenbedingungen* $Ax = b$ *in* (P)

$$z_L^\star = z_{\lambda^\star}^\star = z_D^\star$$

Beweis:(nach [19])

$$\begin{aligned}
& z_D^\star = \max_\lambda\{\min_x\{c^t x + \lambda^t(b - Ax) \mid Dx = e, x \in \mathbb{N}\}\} \\
(*) \quad & \geq \max_\lambda\{\min_x\{c^t x + \lambda^t(b - Ax) \mid Dx = e, x \geq 0\}\} \\
& = \max_\lambda\{\max_\kappa\{\lambda^t b + \kappa^t e \mid D^t\kappa \leq c - A^t\lambda, \kappa \lneqq 0\}\} && \text{wg. LP-Dualität} \\
(**) \quad & = \min_x\{c^t x \mid Ax = b, Dx = e, x \geq 0\} && \text{wg. LP-Dualität} \\
& = z_L^\star
\end{aligned}$$

Wobei für die LPs (*) und (**) κ der Vektor der Dualvariablen zu $Dx = e$ und λ der zu $Ax = b$ ist. Offensichtlich gilt im Fall der Integralitätsbedingung das Gleichheitszeichen überall. ■

In solchen Fällen ist also zu prüfen, ob sich (PR_λ) effizienter lösen läßt als (P^L). Die Ergebnisse dieses Abschnitts lassen auch auf den Fall von Ungleichheitsbedingungen in (P) und Unzulässigkeit von (P) beziehungsweise von unendlich vielen Punkten in $F(PR_\lambda)$ verallgemeinern (siehe [19, 25]).

3.2 Die Verfahren von Lucena und von Picard und Queyranne zur Lösung des TDTSP

Lucena entwickelt 1990 in [46] ein bereits 1978 von Picard und Queyranne in [58] veröffentlichtes Verfahren zur Lösung des TDTSP weiter. Einen ähnlichen Ansatz verfolgen ebenfalls 1990 auch Bianco, Mingozzi und Ricciardelli in [6], allerdings spezialisiert auf das DMP. Lucenas Darstellung basiert auf der Formulierung (QAP) auf Seite 23. Wie in Satz 2.5 bewiesen, ist die Linearisierung von (QAP) äquivalent zu (HVPQ) auf Seite 13. Es ist daher möglich und zulässig, Lucenas Verfahren aufbauend auf (HVPQ) zu beschreiben.

3.2.1 Der zeitexpandierte Graph G^T

Um die Beschreibung zu vereinfachen, wird zunächst der in Abbildung 2.1 auf Seite 14 beispielhaft gezeigte und durch die Formulierung (HVPQ) implizit erklärte Graph explizit definiert. Dies geschieht hier bereits etwas allgemeiner als notwendig, nämlich nicht nur für TDTSPs, sondern gleich für allgemeine DTSP–Instanzen.

Definition 3.2
Sei durch $\mathcal{I} = (V, E, d, c, v_0)$ eine DTSP–Instanz gegeben, $n := |V_0|$. Sei weiterhin $d_{i,j}(t) \in \mathbb{N}$. Definiere zunächst die Knotenmenge V^T und die *Menge der inneren Kanten* E_0^T:

$$V^T := \{(v,t) \mid v \in V_0, t = 1, \ldots, T\} \cup \{(v_0, 0), (v_0, \infty)\} \tag{3.3}$$

$$E_0^T := \{((v,s),(w,t)) \mid (v,s),(w,t) \in V^T, t = s + d_{v,w}(s)\} \tag{3.4}$$

Im Graphen (V^T, E_0^T) sei mit $r((v,s))$ die maximale Kantenzahl eines Pfads von $(v_0, 0)$ nach $(v,s) \in V^T$ bezeichnet.

Dann ist die *Menge der Endkanten* E_Ω^T gegeben durch:

$$E_\Omega^T := \{((v,s),(v_0,\infty)) \mid (v,s) \in V^T, r((v,s)) \geq n\} \tag{3.5}$$

Dann ist $G^T = (V^T, E^T) = (V^T, E_0^T \cup E_\Omega^T$ der zu $\mathcal{I}$ gehörende, gerichtete *zeitexpandierte Graph mit Horizont* $T \in \mathbb{N}$. Auf G^T werden Kantenkosten $c(e), e \in E^T$ erklärt durch:

$$c(e) := c_{i,j}(s) \text{ falls } e = ((v,s),(w,t)) \in E^T \tag{3.6}$$

■

Bemerkung: In dieser Arbeit ist ein „Pfad“ dasselbe wie ein „Weg“ bei Jungnickel (Abschnitt 1.2 in [41]), das heißt ein Kantenzug ohne Kantenwiederholung. Die Wiederholung von Knoten ist explizit nicht ausgeschlossen. Die Berechnung der Rangfunktion $r((v,s))$ wie in der Definition verwendet ist im allgemeinen NP–schwer (siehe [23]). Der Graph (V^T, E_0^T) ist jedoch ein azyklischer Digraph, da Kanten nur in Richtung zunehmender Zeit vorkommen und dadurch gerichtete Kreise ausgeschlossen sind. In diesem Fall läßt sich r mit Komplexität $O(|E^T|)$ berechnen (siehe [41, Kap. 3.5]) und somit ist die Berechnung von E_Ω^T auch nicht aufwendiger als die von E_0^T. Im Fall eines TDTSPs enthält E_Ω^T einfach die Kanten $((v,T),(v_0,\infty))$ (vergleiche Abbildung 2.1 auf Seite 14).

Weiterer Erläuterung bedarf sicher die Definition der Menge der Endkanten in (3.5). Durch E_Ω^T wird sozusagen der „Ausgang“ aus dem Graphen definiert. Dafür würde es reichen, einfach alle Knoten mit (v_0,∞) zu verbinden und die Anzahl an Kanten würde sich dadurch auch größenordnungsmäßig nicht ändern. Andererseits sind viele dieser Kanten unnötig und G^T wäre nicht mehr der Graph von Formulierung (HVPQ) der in Abbildung 2.1 dargestellt ist. Der eigentlichen Zweck von G^T ist es, eine Tour für V darin zu suchen. Daher ist es ausreichend, nur die Kanten einzuschließen, die notwendig sind, damit jeder Pfad, der in v_0 gestartet ist und bereits n Städte besucht hat, auch wieder nach v_0 zurückkehren kann. Abbildung 2.1 stellt also einen G^T für ein TDTSP mit $|V| = 6$ und $n = T = 5$ dar. In Abbildung 4.1 auf Seite 54 ist ein G^T für ein DTSP, dessen Wegezeiten nicht konstant und identisch 1 sind, dargestellt.

Definition 3.3
Sei K_V der vollständige Graph mit Knotenmenge V. Ein Pfad

$$P = i_0 \to i_1 \to \cdots \to i_k$$

in K_V heißt *k-schrittiger Pfad in V*. Ist $i_0 = v_0$, so heißt P *Anfangspfad*. Ist $k = n + 1$ und $i_0 = i_k = v_0$ so heißt P *Pfad durch V_0*. Sind zusätzlich alle anderen i_j paarweise verschieden, so heißt P *Tour in V*. Sei $t_0 = 0$ und $t_{j+1} = t_j + d_{i_j,i_{j+1}}(t_j)$. Dann heißt der Graph mit der Knotenmenge $\{(i_j, t_j) \mid j = 0, \ldots, k\}$ und den Kanten $\{((i_j, t_j), (i_{j+1}, t_{j+1})) \mid j = 0, \ldots, k-1\}$ der zu P gehörende *zeitexpandierte Pfad*. ■

Bemerkung: Beachte, daß durch diese Definition ausgeschlossen ist, daß ein Pfad in V dieselbe Stadt zweimal *direkt hintereinander* besucht.

Zur Beschreibung des Verfahrens wird noch die nächste Definition benötigt:

Definition 3.4
In einem zeitexpandierten Graph $G^T = (V^T, E^T)$ wird durch

$$\vec{\delta}_{G^T}(v) := \{e \in E^T \mid \exists w \in V;\ s, t \in \mathbb{N} : e = ((w, s), (v, t))\} \qquad v \in V \tag{3.7}$$

der *zeitexpandierte In-Stern $\vec{\delta}_{G^T}(v)$ von v* definiert, das heißt die Menge aller Kanten in E^T, die ihren Endpunkt in einer der Kopien (v, t) von v in V^T haben. Schließlich ist noch

$$A_{E^T} := (a_{ve})_{\substack{v \in V^T \\ e \in E^T}} \text{ mit } a_{ve} = \begin{cases} -1 & \text{falls } v \text{ Startpunkt von } e \\ +1 & \text{falls } v \text{ Endpunkt von } e \\ 0 & \text{sonst} \end{cases} \tag{3.8}$$

die *Knoten-Kanten-Inzidenzmatrix* (vgl. [41, Kap. 4.2]) zu G^T. ■

Der Rest des Kapitels verläßt nun das DTSP und beschäftigt sich nur noch mit dem TDTSP. Daher ist ab jetzt G^T der zeitexpandierte Graph zu einem TDTSP mit Horizont $T = n$.

3.2.2 Relaxation durch billigste Pfade

Mit den Begriffen des letzten Abschnitts läßt sich (HVPQ) jetzt kompakter angeben:

Alternative Formulierung (HVPQ):

$$(2.1)\quad \min \sum_{e\in E^T} c_e x_e := z(x)$$

unter den Nebenbedingungen

$$(2.8)\quad \sum_{e\in\vec{\delta}_{G^T}(v)} x_e = 1 \qquad v \in V_0$$

$$(3.9)\quad A_{E^T} x = (-1, 0, \ldots, 0, 1)^t$$

$$(2.13)\quad x_e \in \{0,1\} \qquad e \in E^T$$

Dabei entsprechen für eine Kante $e = ((i,t),(j,s)) \in E^T$ die Variablen c_e und x_e in der neuen Formulierung den Variablen $c^t_{i,j}$ und x_{ijt} in der Formulierung auf Seite 13. Wie in Satz 2.2 auf Seite 15 gezeigt, können in (3.9) die erste und letzte Zeile, die den Knoten $(v_0, 0)$ und $(v_0, T+1)$ entsprechen, weggelassen werden. Lucena dualisiert jetzt die Zuordnungsbedingungen (2.8) im Sinne einer Lagrange-Relaxation:

$$(3.10)\quad \begin{aligned} &\min \sum_{e\in E^T} c_e x_e + \sum_{v\in V_0} \lambda_v \Big(\sum_{e\in\vec{\delta}_{G^T}(v)} x_e - 1 \Big) \\ &\text{s.d.} \quad A_{E^T} x = (-1, 0, \ldots, 0, 1)^t \\ &\qquad x_e \in \{0,1\} \qquad e \in E^T \end{aligned}$$

Formulierung (3.10) beschreibt nun offensichtlich das Problem, in G^T einen billigsten Pfad durch V_0 zu finden (siehe Definition 3.3 auf der vorigen Seite), wobei jeder außer dem ersten Besuch einer Stadt v „Strafkosten“ λ_v verursacht.

Bemerkung: Beachte, daß die Zielfunktion in 3.10 noch die Konstante $-\sum_{v\in V_0} \lambda_v$ enthält. Allerdings ändert sich der Zielfunktionswert nicht, wenn auf alle λ_v derselbe

Wert aufaddiert wird, da genau n der $x_e = 1$ sind. Anschaulich betrachtet ändert sich einfach nichts, wenn dieselben Strafkosten für jede Stadt zu vergeben werden. Deshalb kann man ohne Einschränkung der Allgemeinheit immer $\sum_{v \in V_0} \lambda_v = 0$ annehmen.

Es ist wohlbekannt, daß A_{E^T} total unimodular ist (siehe z.B. [41, Satz 4.2.5]), daher erfüllt (3.10) die Integralitätseigenschaft laut Definition 3.1 auf Seite 42. G^T hat nun aber Eigenschaften, die die Lösung von (3.10) besonders einfach machen: er ist n-partit und azyklisch, deshalb lassen sich die gewünschten billigsten Pfade unproblematisch durch Dynamische Programmierung bestimmen (siehe auch [12]). Definiere $c^\star_{ik}$ als die Kosten des billigsten, genau k-schrittigen Pfads von $(v_0, 0)$ nach (i, k) (in G^T kommen alle k-schrittigen Pfade nach i in (i, k) an). Die Werte $c^\star_{ik}$ erfüllen die folgenden Rekursionsgleichungen:

$$
\begin{cases}
c^\star_{i,1} = c^0_{v_0 i} + \lambda_i & i \in V_0 \\
c^\star_{i,k} = \min\limits_{j \in V_0 \setminus \{i\}} \{c^\star_{j,k-1} + c^{k-1}_{ji} + \lambda_i\} & i \in V_0, k = 1, \ldots, n \\
c^\star_{v_0,n+1} = \min\limits_{j \in V_0} \{c^\star_{j,n} + c^n_{j0}\}
\end{cases}
\tag{3.11}
$$

Der optimale Zielfunktionswert von (3.10) läßt sich dann als $c^\star_{v_0,n+1} - \sum_{i \in V_0} \lambda_i$ berechnen. Zur Bestimmung des optimalen Pfads verwende $c^\star_{ik}$ als Marken an den Knoten und verfolge den Ablauf mit Hilfe der Kantenkosten zurück. Offensichtlich kann (3.11) mit $O(n^3)$ Additionen und Vergleichen gelöst werden. Dieselben Rekursionsgleichungen wurden bereits von Picard und Queyranne [58] verwendet und bis hierher sind die beiden Verfahren identisch. Picard und Queyranne maximieren nun direkt die unteren Schranken mittels der Subgradientenmethode wie in Abschnitt 3.1 beschrieben. Auf diesen unteren Schranken bauen sie dann einen Branch&Bound–Algorithmus (für eine allgemeine Beschreibung dieser Technik siehe [3, 41, 54]). In ihrem Algorithmus wird in jedem Knoten des Branch&Bound–Baums ein Anfangspfad als partielle Tour festgelegt und dann nach allen noch nicht besuchten Städten verzweigt. Zusätzlich kommt ein Dominanztest zum Einsatz, der versucht, durch Vertauschen von zwei aufeinanderfolgenden Städten in der partiellen Tour eine Verbesserung zu erzielen. Ist dies möglich, kann dieser Zweig des Baumes vernachlässigt werden. Im folgenden wird beschrieben, wie Lucena das Verfahren durch Verschärfung der Relaxation weiterentwickelt hat.

3.2.3 Eine Folge monoton steigender unterer Schranken

Der Aufwand von $O(n^3)$ zur Lösung des Lagrange-Problems ist sicher recht hoch und so versucht Lucena, die Güte der Schranke zu verbessern. Zu diesem Zweck eignen sich sogenannte „through–circuits“ (siehe [10]).

Definition 3.5
Ein (i,k)*–through–circuit* ist ein billigster unter allen Pfaden durch V_0, die im k-ten Schritt Stadt i besuchen. Eine (i,k)*–through–tour* ist eine billigste Tour unter allen Touren π mit $\pi^k(v_0) = i$.

In G^T ist also ein (i,k)–through–circuit ein billigster $(v_0,0) \to (v_0,n+1)$–Pfad, der den Knoten (i,k) enthält. Bezeichnet nun $b^\star_{i,k}$ die Lagrange–Kosten eines billigsten, k–schrittigen Pfades von $(i,n+1-k)$ nach $(v_0,n+1)$, so gilt offenbar für die Kosten ψ_{ik} eines i,k—through–circuits:

$$\psi_{ik} = c^\star_{ik} + b^\star_{i,n+1-k} - \sum_{i\in V_0} \lambda_i \tag{3.12}$$

Die $b^\star_{ik}$ lassen sich auf dieselbe Weise wie in (3.11) berechnen, nur „von hinten“:

$$\begin{cases} b^\star_{i,1} = c^n_{iv_0} & i \in V_0 \\ b^\star_{i,k} = \min\limits_{j\in V_0\setminus\{i\}} \{b^\star_{j,k-1} + c^k_{ij} + \lambda_j\} & i \in V_0, k = 1,\ldots,n \\ b^\star_{v_0,n+1} = \min\limits_{j\in V_0}\{b^\star_{j,n} + c^0_{0j} + \lambda_j\} \end{cases} \tag{3.13}$$

Auch (3.13) läßt sich mit Aufwand $O(n^3)$ lösen, so daß die Komplexität gegenüber der Berechnung eines billigsten Pfades durch V_0 nicht steigt.

Bemerkung: Die Berechnung eines i,k–through–circuits läßt sich als total unimodulares LP formulieren. Dazu muß in (3.10) die Matrix A_{E^T} durch die Inzidenzmatrix des Graphen ersetzt werden, der entsteht, wenn man aus G^T alle Knoten (j,k) mit $j \neq i$ entfernt (siehe auch Abschnitt 5.3.3 auf Seite 87).

Beachte, daß ψ_{ik} keine untere Schranke für die Kosten $z(\pi^\star)$ der optimalen Tour $\pi^\star$ darstellt, da $\pi^\star$ ja nicht unbedingt Stadt i zum Zeitpunkt k besuchen muß. Allerdings muß $\pi^\star$ die Stadt i *irgendwann einmal* besuchen, so daß sicher $z(\pi^\star) \geq \min_{k=1,\ldots,n} \psi_{ik}$ gilt. Da das aber für alle Städte $i \in V_0$ gilt, stellt also

(3.14) $$L(0,\lambda) := \max_{i \in V_0} \min_{k=1,\ldots,n} \psi_{ik}$$

eine untere Schranke für $z(\pi^\star)$ dar, die mindestens so scharf ist wie die aus der Lagrange–Relaxation (3.10). Lucena argumentiert weiter, daß $\pi^\star$ ja zu jedem Zeitpunkt irgendeine Stadt besuchen muß, allerdings erfüllt natürlich der billigste Pfad durch V_0 bereits diese Bedingung, so daß sich $L(0,\lambda)$ damit nicht mehr verschärfen läßt.

Die numerische Verbesserung der unteren Schranke durch $L(0,\lambda)$ läßt sich darauf zurückführen, daß eine „besonders schwer“ zu erreichende Stadt ihren Einfluß nicht nur auf den Zielfunktionswert sondern auch auf die untere Schranke geltend machen kann, während der billigste Pfad durch V_0 sie vielleicht einfach gar nicht berührt hätte. Diese Idee liegt der nun folgenden Methode zur weiteren Verschärfung der unteren Schranke zugrunde.

Sei $\pi^\star_{i,k}$ eine (i,k)–through–tour, dann gilt sicher:

(3.15) $$\psi_{ik} \le z(\pi^\star_{ik}) \text{ und } L(0,\lambda) \le z(\pi^\star) \le z(\pi^\star_{ik}) \qquad i \in V_0, k = 1,\ldots,n$$

Sei nun

(3.16) $$\Psi := \Big(\max\{\psi_{ik}, L(0,\lambda)\}\Big)_{\substack{i \in V_0 \\ k=1,\ldots,n}}$$

die Matrix der besten bekannten unteren Schranken für $z(\pi^\star_{ik})$. Es gilt

(3.17)
$$\begin{aligned} \pi^\star &= \pi^\star_{\pi^{\star k}(v_0),k} && k = 1,\ldots,n \\ \Longrightarrow \quad z(\pi^\star) &= z(\pi^\star_{\pi^{\star k}(v_0),k}) \\ \Longrightarrow \quad z(\pi^\star) &= \frac{1}{n}\sum_{k=1}^{n} z(\pi^\star_{\pi^{\star k}(v_0),k}) \end{aligned}$$

(3.18)
$$\begin{aligned} &\ge \frac{1}{n}\sum_{k=1}^{n} \Psi_{\pi^{\star k}(v_0),k} \\ &\ge \min_{\pi} \frac{1}{n}\sum_{k=1}^{n} \Psi_{\pi^{k}(v_0),k} \end{aligned}$$

Die erste Zeile ist dabei so zu verstehen, daß $\pi^\star$ eine mögliche $\pi^{\star k}(v_0), k$–through–tour ist. Die letzte Zeile ist nur eine etwas komplizierte Schreibweise für das Problem, jeder Stadt $i \in V_0$ genau eine Position k zuzuordnen, so daß die angegebene Summe minimal wird. Also stellt die Lösung $L(1,\lambda)$ des folgenden Zuordnungsproblems eine untere Schranke für $z(\pi^\star)$ dar:

$$L(1,\lambda) = \min \frac{1}{n} \sum_{i \in V_0} \sum_{k=1}^{n} \Psi_{ik} y_{ik} \tag{3.19}$$

unter den Nebenbedingungen

$$\sum_{i \in V_0} y_{ik} = 1 \qquad k = 1, \ldots, n \tag{3.20}$$

$$\sum_{k=1}^{n} y_{ik} = 1 \qquad i \in V_0 \tag{3.21}$$

$$y_{ik} \in \{0,1\} \qquad i \in V_0,\ t = 1, \ldots, n \tag{3.22}$$

Wegen (3.16) gilt $\Psi_{ik} \geq L(0,\lambda)$ und somit auch $L(1,\lambda) \geq \frac{1}{n} n L(0,\lambda)$. Für $L(1,\lambda)$ läßt sich nun wieder (3.15) anwenden, so daß durch

$$\Psi := \Big(\max\{\Psi_{ik}, L(1,\lambda)\} \Big)_{\substack{i \in V_0 \\ k=1,\ldots,n}}$$

eine Matrix verbesserter unterer Schranken für $z(\pi^\star_{ik})$ definiert wird. Aus Ψ läßt sich durch Lösen des entsprechenden Zuordnungsproblems eine neue untere Schranke $L(2,\lambda)$ gewinnen. Diese Iteration wird fortgesetzt, bis sich keine Verbesserung mehr ergibt (in der Praxis ist das recht bald der Fall, allerdings macht [46] keine Konvergenzaussagen, siehe auch Abschnitt 5.3.2).

3.2.4 Aufstiegs–Verfahren und Branch&Bound

Stagniert das Verfahren für die aktuellen Lagrange–Faktoren λ^s, das heißt

$$L(t,\lambda^s) = L(t+1,\lambda^s) =: L(\infty,\lambda^s)$$

ist erreicht, werden mittels (3.2) von Seite 41 neue Lagrange-Faktoren λ^{s+1} berechnet. Als Aufstiegsrichtung kommt dabei die gemittelte Verletzung von (2.8)

der durch das letzte gelöste Zuordnungsproblem ausgewählten through–circuits zum Einsatz — in Formeln: Sei y die Lösung des Zuordnungsproblems zu $L(t+1, \lambda^s)$ und sei x^{ik} der Inzidenzvektor des through–circuits zu ψ_{ik}, so bestimme d_s durch:

$$d_s = \frac{1}{n} \sum_{i \in V_0} \sum_{k=1}^{n} y_{ik} (A_{\bar{\delta}_{G^T}} x^{ik} - \mathbf{1}) \tag{3.23}$$

wobei $A_{\bar{\delta}_{G^T}} x = \mathbf{1}$ die Matrixschreibweise für (2.8) ist und $\mathbf{1}$ ein geeignet großer Vektor aus Einsen. Nun werden neue through–circuits berechnet und die Einträge der Matrix Ψ aktualisiert, falls verbesserte Schranken berechnet wurden. Da also die Einträge von Ψ nur steigen können, ist die neue Schranke $L(\infty, \lambda^{s+1})$ auf jeden Fall nicht schlechter als die alte $L(\infty, \lambda^s)$.

Der Kontrollparameter τ ist zunächst 2.0 und wird nach fünf Schritten ohne Verbesserung der Schranke halbiert. Nach spätestens 100 Schritten insgesamt oder 25 Schritten ohne Schrankenverbesserung, wird die Iteration abgebrochen, falls nicht bereits vorher d_s verschwindet. (Dies gilt für den Wurzel–Knoten des Branch&Bound–Baumes, siehe unten. An den tieferen Knoten wird bereits nach maximal 20 Schritten oder nach 8maliger Stagnation abgebrochen.)

Lucena verwendet dieses Verfahren zur Berechnung unterer Schranken zur exakten Lösung des TDTSP mittels Branch&Bound. Diese Standard–Technik wird hier nicht beschrieben, siehe dazu zum Beispiel [3,41]. Die wichtigste Zutat zum B&B–Algorithmus, die Relaxation, wurde gerade ausführlich besprochen. Es fehlen jetzt noch Separierungs– und Auswahlregel und obere Schranken. Wie bei Picard und Queyranne [58] wird das aktuelle Problem durch Festlegung einer noch nicht besuchten Stadt als nächster Stadt der Tour in Teilprobleme partitioniert. Ein Knotenproblem auf Stufe m des B&B–Baumes ist also ein TDTSP mit $n - m$ Städten, das aus dem Ausgangsproblem durch Festlegung der ersten m Kanten der Tour entstanden ist. Der nächste zu untersuchende Zweig wird nach der „weakest–bound–first"–Regel ausgewählt, das heißt, das Teilproblem mit der bis jetzt schlechtesten Schranke wird als nächstes separiert (vergleiche auch Seite 47).

Obere Schranken gewinnt Lucena auf drei Weisen:

- Wann immer eine Zuordnungsproblem (3.19)–(3.22) gelöst wurde, werden die Kosten der dadurch angegeben Tour wie in (2.37) auf Seite 23 berechnet.
- In einer gegebenen Tour wird versucht, durch Vertauschen zweier Städte ein billigere Tour zu erzeugen (2-Opt, siehe [28])

- In einer gegebenen Tour wird jede Stadt probeweise zwischen die beiden folgenden eingesetzt (eingeschränktes 3-Opt, siehe [28])

Die beiden Tour–Verbesserungs Heuristiken sind vom TSP her wohlbekannt, lassen sich allerdings nur schwer auf das TDTSP verallgemeinern. In der Regel müssen nach einer Vertauschung die Kosten eines Großteils der Tour neu berechnet werden, so daß der Rechenaufwand signifikant höher ist als beim TSP (siehe [17] für eine geschickte Implementierung im Fall des DMP und [64] für eine Verallgemeinerung der r–opt Heuristik von Lin und Kernigham [42] auf das TDTSP).

Lucena hat seinen Algorithmus für zwölf zufällig erzeugte DMPs getestet. Dazu wurden aus einem Quadrat mit Seitenlänge 100 zwischen 15 und 30 Punkte als Städte gleichverteilt ausgewählt. Als Kosten wurden die gerundeten euklidischen Entfernungen gewählt. Er berichtet, daß der Algorithmus die größten Probleme in 4006 bis 4663 CPU-Sekunden auf einer Data General MC15000 gelöst hat. Die Qualität der Schranken war sehr gut (<5% auch für die größten Beispiele) und Lucena führt die lange Rechenzeit darauf zurück, daß das DMP schwieriger zu lösen sei, als das TSP. Er berichtet, daß die oberen Schranken schlechter wurden für zufällige TDTSP–Instanzen, die unteren Schranken aber scharf blieben.

Kapitel 4

Verallgemeinerung des Verfahrens von Lucena und von Picard und Queyranne auf das DTSP

Diese Kapitel beschäftigt sich nun damit, das im letzten Kapitel beschriebene Verfahren zur Lösung des TDTSP zu einem Verfahren zur Lösung des DTSP zu verallgemeinern. Die Idee dazu ist, in Formulierung (HVPQ) auf Seite 46 den zeitexpandierten Graphen zum TDTSP durch den zu einem allgemeinen DTSP gehörenden zeitexpandierten Graphen zu ersetzen. Dadurch erhält man eine ILP–Formulierung für das DTSP. Führt man diese Ersetzung auch in Lucenas Algorithmus durch, so erhält man eine Methode zur Lösung allgemeiner DTSPs. Was dabei im allgemeinen zu beachten ist, ist Thema der Abschnitte 4.1 und 4.2, Abschnitt 4.3 geht ausführlich auf das Problem der Pfadberechnung ein. Kapitel 5 wird schließlich die Implementation des so gewonnenen Algorithmus beschreiben.

4.1 Erste Analyse der modifizierten Methode

Den Kern von Lucenas Methode bildet die Berechnung (spezieller) billigster $n+1$–schrittiger Pfade durch V_0. Dies ist im Fall des TDTSP recht einfach möglich mittels der Rekursionsgleichungen (3.11) und (3.13). Die Struktur der Gleichungen ist die des multipartiten Graphen G^T (siehe Abbildung 2.1 auf Seite 14) und tatsächlich stellt Lucena in seiner Implementation den Graphen G^T niemals explizit auf, sondern nur implizit bei der Lösung der Rekursionsgleichungen.

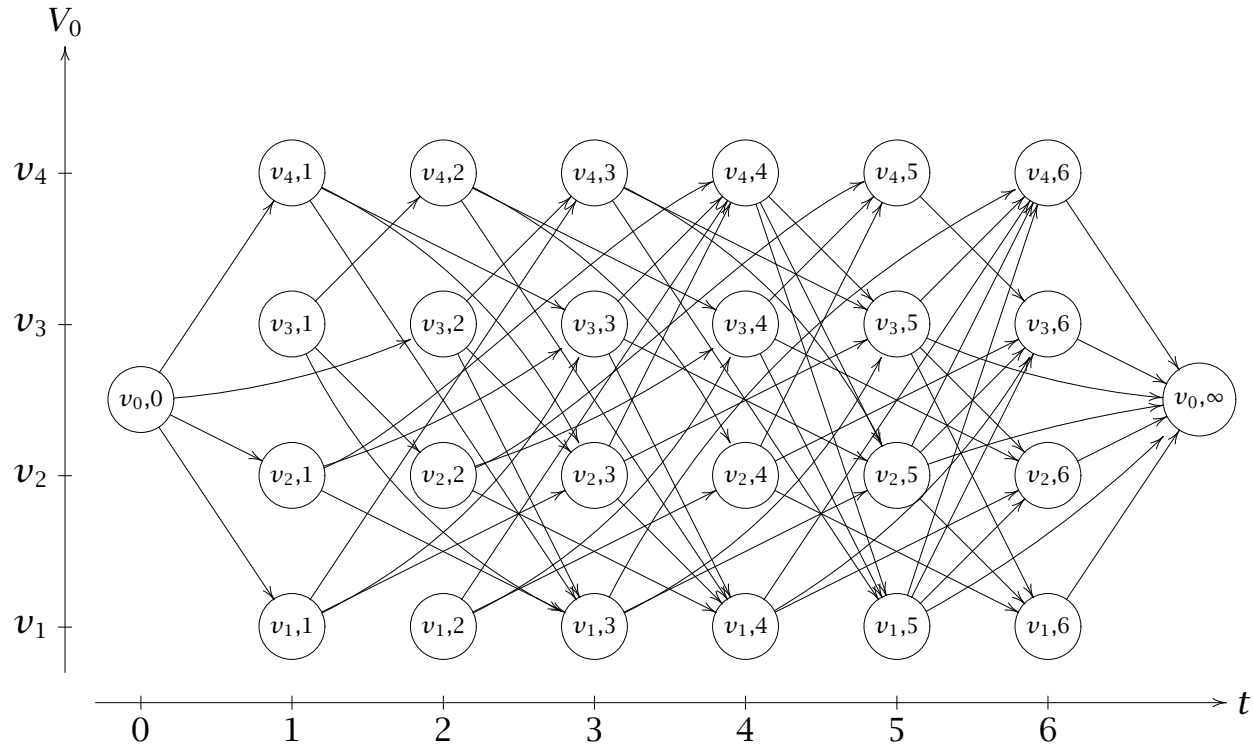


Abbildung 4.1: Ein Beispiel für den G^T eines nicht-TDTSP

Betrachte nun statt dessen den G^T eines allgemeinen DTSP, dessen Wegezeiten nicht alle identisch 1 sind. Abbildung 4.1 zeigt so einen Graphen für $n = 4$ und $T = 6$. In diesem Beispiel werden $d_{v_1,\cdot}(t)$ und $d_{v_3,\cdot}(t)$ durch die folgenden Wertetabellen angeben:

$d_{v_3,\cdot}(t)$	$t=1$	2	3	4	5	6
v_1	2	1	1	1	1	
v_2	1	1	2	2		
v_4	1	1	1	1		
v_0						1

$d_{v_1,\cdot}(t)$	$t=1$	2	3	4	5	6
v_2	2	2	2	2		
v_3	2	2	2	2		
v_4	2	2	3			
v_0					2	1

Die Wertetabellen für die restlichen $d_{ij}(t)$ lassen sich leicht aus der Abbildung ablesen.

Wie man sieht ist dieser G^T nicht mehr multipartit und wesentlich „unübersichtlicher“. Die Begriffe des k–schrittigen Pfades und des Pfades durch V_0 haben keinen Schaden genommen, sie lassen sich ja durchaus auch in einem vollständigen Graphen K_V darstellen (siehe Abbildung 4.2). Leider ist es nicht aber

möglich, an den Kanten von K_V Gewichte so zu definieren, daß die Kosten der Pfade in V einfach durch Aufaddieren der Kantengewichte berechnet werden können. Dieselbe Kante verursacht nämlich in unterschiedlichen Pfaden nicht notwendig dieselben Kosten wenn sie zu unterschiedlichen Zeiten benutzt wird.

Aber genauso wie im Fall $d_{i,j}(t) \equiv 1$ lassen sich die Pfadkosten einfach durch Aufaddieren der Kantenkosten im zeitexpandierten Graphen bestimmen. So wurde in Abschnitt 3.2 nur in der Formulierung der Gleichungen (3.11),(3.13) und (3.12) zur Berechnung billigster Pfade von den Einheitszeiten Gebrauch gemacht. Die wichtigste fehlende Eigenschaft im allgemeinen Fall ist die Gleichheit der Position einer Stadt in einem Pfad und der Ankunftszeit dortselbst. Beim TDTSP kamen alle k–schrittigen Pfade nach Stadt i im Knoten (i,k) des G^T an. Daher war die Bestimmung der billigsten k–schrittigen Pfade gleichbedeutend mit dem Aufstellen des Baums kürzester Wege, einmal mit Wurzel in $(v_0,0)$ und einmal in (v_0,∞). Bei nicht–identischen Wegezeiten jedoch sind die ersten $k-1$–Schritte eines billigsten k–schrittigen Pfades nicht notwendigerweise ein billigster k–schrittiger Pfad.

Die Begriffe des through–circuits und der through–tour sind wieder von den Wegezeiten unabhängig, da sie in Definition 3.5 auf Seite 48 mittels Positionen eingeführt wurden. Allerdings kann im allgemeinen Fall nicht mehr ein einzelner Knoten in V^T identifiziert werden, den alle Kandidaten für einen i,k–through–circuit durchlaufen müssen. Statt dessen kann irgendeine Kopie der Stadt i an k-ter Position auf einem Pfad durch V_0 stehen.

Die Argumente des Abschnitts 3.2.3, insbesondere (3.14),(3.18) und (3.19), behalten Gültigkeit, da sie sich auf Positionen und nicht auf Knoten im G^T beziehen. Abschnitt 4.3 wird sich damit beschäftigen, wie man Ψ im Fall allgemeiner Wegezeitfunktionen trotzdem berechnen kann.

Offensichtlich kommt nicht nur G^T als Grundlage für den neuen wie den alten Algorithmus in Frage, er ist jedoch prototypisch:

Definition 4.1
Sei $\mathcal{I} = (V,E,d,c,v_0)$ eine DTSP–Instanz. Dann heißt ein Untergraph $G^X = (E^X,V^X)$ eines zeitexpandierten Graphen G^T mit Horizont T zu $\mathcal{I}$ und Kantenkosten $c(e)$ *allgemeiner zeitexpandierter Graph zu $\mathcal{I}$ mit Kantenkosten $c(e)$* falls er mindestens $(v_0,0)$, (v_0,∞) und einen Pfad zwischen diesen beiden Knoten enthält. ■

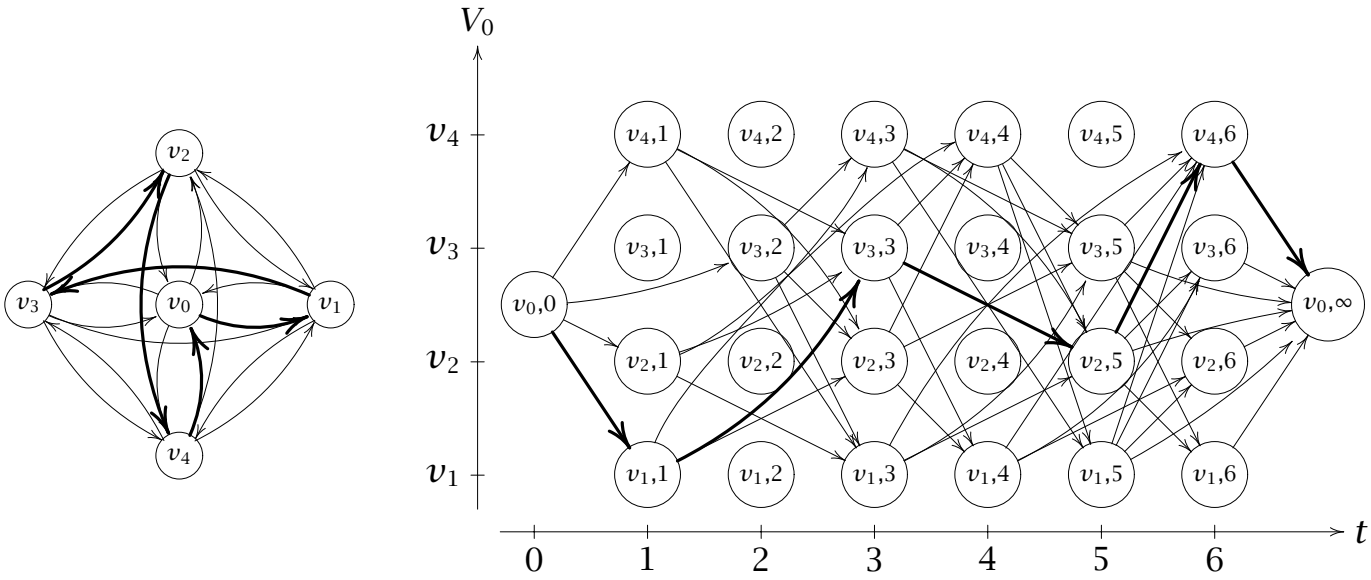


Abbildung 4.2: Der Pfad $v_0 \to v_1 \to v_3 \to v_2 \to v_4 \to v_0$ im K_5 und im G^T

Wie klein darf nun so ein G^X sein, damit Lucenas Methode noch funktioniert? Als hinreichende Voraussetzung für die Korrektheit des Algorithmus bleibt:

(4.1) **Der zeitexpandierte Graph G^X muß eine Optimaltour $\pi^\star$ enthalten.**

Solange diese Bedingung gilt, gibt es auch einen billigsten Pfad durch V_0. In einem zeitexpandierten Graphen, der nicht mehr alle through–touren $\pi^\star_{ik}$ enthält, sind die darin berechneten ψ_{ik} nicht notwendigerweise untere Schranken für $z(\pi^\star_{ik})$. Betrachte die (i,k)–through–touren, die mit $\pi^\star$ zusammenfallen, das heißt, die mit $i = \pi^{\star k}(v_0)$. Weil ja $\pi^\star$ im Graphen enthalten ist, gilt sicher

$$\psi_{\pi^{\star k}(v_0),k} \le z(\pi^\star) = z(\pi^\star_{\pi^{\star k}(v_0),k})$$

und (3.18) bleibt richtig. Weiterhin ist

$$y_{ik} = 1 \iff \pi^{\star k}(v_0) = i$$

eine zulässige Lösung des Zuordnungsproblems (3.19)–(3.22). Diese Überlegung wird später wichtig sein beim Fixieren von Variablen in Abschnitt 5.3.2 auf Seite 86 (beachte, daß Bedingung (4.1) von den Wegekosten abhängig ist).

4.2 Mehr zeitexpandierte Graphen

Bis jetzt wurde nur ein zeitexpandierter Graph eingeführt, der G^T. Wie eben angedeutet entstehen im Verlauf des Algorithmus eventuell durch die Fixierung von Variablen und die daraus resultierende Entfernung von Kanten aus dem Graphen weitere Beispiele. Eine Alternative zu G^T als Grundlage des Verfahrens wird in diesem Abschnitt eingeführt.

4.2.1 Kritik von G^T

Sei $v_n := \pi^{\star -1}(v_0)$ die letzte Stadt, die $\pi^\star$ vor der Rückkehr nach v_0 besucht. Dann erfüllt G^T für $T \geq t_{v_n}$ die Bedingung (4.1), da alle Kanten von $\pi^\star$ enthalten sein müssen. Allerdings kann G^T im allgemeinen viele offensichtlich unnötige Kanten enthalten, wie Abbildung (4.1) zeigt. Nachteilig an G^T ist unter anderem:

- G^T hängt im allgemeinen nicht zusammen — für den Algorithmus sind aber von $(v_0, 0)$ aus nicht erreichbare Knoten offensichtlich unnötig (Abbildung (4.2) zeigt deshalb nur die Kanten der Zusammenhangskomponente von $(v_0, 0)$).
- Zum Zeitpunkt der Aufstellung muß ein genügend großer Horizont T bekannt sein.
- Die Wegezeiten müssen ganzzahlig sein ($d_{i,j}(t) \in \mathbb{N}$).

Besonders die Bestimmung eines geeigneten Wertes für T erscheint schwierig, da ohne einschränkende Voraussetzungen an die Kostenfunktionen (wie zum Beispiel Monotonizität in t) keine offensichtliche Abschätzung für die Dauer von $\pi^\star$ zur Verfügung steht. Insbesondere ist die billigste Tour nicht unbedingt auch die kürzeste, so daß Konstruktionsheuristiken zur Bestimmung von T ausscheiden (dies gilt zum Beispiel in dem Fall, wo Umwege zur Kostenreduzierung führen oder wo ein langsameres Verkehrsmittel billiger ist). Variieren die Größenordnungen der Wegezeiten, ist zum Beispiel eine Stadt „weit weg“ von allen anderen, so muß T sehr groß sein und somit auch G^T. Ist T etwa größer als $n!$, so ist das Aufstellen von G^T bereits aufwendiger als die Enumeration aller Touren. Für die Anwendung stellt es keine Einschränkung dar, nur rationale Wegezeiten zuzulassen. Dann ließe sich die Ganzzahligkeit der Wegezeiten durch Multiplikation mit dem Hauptnenner erreichen, allerdings würde auch das wieder zu einem sehr großen T führen.

4.2.2 Eine Alternative zu G^T

Definition 4.2
Sei durch $\mathcal{I} = (V, E, d, c, v_0)$ eine DTSP–Instanz gegeben. Sei $\mathfrak{V}$ die Menge aller $n+1$–schrittigen Pfade durch V_0. Sei $\mathfrak{V}^T$ die Menge aller zeitexpandierten Pfade zu den Elementen von $\mathfrak{V}$. Identifiziere alle Endknoten (v_0, t_{n+1}) mit dem Knoten (v_0, ∞). Dann heißt die Vereinigung aller Graphen aus $\mathfrak{V}^T$ der *schrittzahlbeschränkte, zeitexpandierte Graph* zu $\mathcal{I}$ und wird mit

$$G^{\mathbb{P}_n} = (V^{\mathbb{P}_n}, E^{\mathbb{P}_n})$$

bezeichnet. ■

Anders formuliert ist $G^{\mathbb{P}_n}$ der durch alle $n+1$–schrittigen $(v_0, 0) \rightarrow (v_0, \infty)$–Pfade induzierte Untergraph eines genügend großen G^T.

Bemerkung: Die Dauer des langsamsten $n+1$–schrittigen Pfades durch V_0 ist sicher ein genügend großer Horizont T. Solange keine bessere Abschätzung für die Dauer der optimalen Tour bekannt ist, müßte dieser Wert auch zum Aufstellen eines G^T als Grundlage des DTSP–Algorithmus verwendet werden.

Offensichtlich erfüllt auch $G^{\mathbb{P}_n}$ Bedingung (4.1). $G^{\mathbb{P}_n}$ läßt sich durch den Algorithmus 4.1 berechnen.

Satz 4.1
Algorithmus (4.1) *berechnet* $G^{\mathbb{P}_n}$ *und hat Komplexität* $O(|E^{\mathbb{P}_n}| \ln |E^{\mathbb{P}_n}|)$

Beweis:
Bis auf die **compute**–Anweisung im zweiten Teil des Algorithmus sind alle Operationen Standard. Der Rang $r(v, w)$ eines Knotens w in einem Graphen ist die Länge eines längsten Weges von v nach w mit Kantengewichten 1. Ein Algorithmus zur Berechnung des Rangs findet sich zum Beispiel in [41, Kap. 3.5].

Offensichtlich terminiert der Algorithmus, da alle Schleifen und die Rangberechnung endlich sind. Zur Korrektheit:
Zu Beginn der k–Schleife gilt

$$V^{\mathbb{P}_n} = \bigcup_{j=0}^{k} O_k \tag{4.2}$$

da jeder im innersten Schritt neu zu $V^{\mathbb{P}_n}$ hinzugefügte Knoten auch in O_{k+1} eingefügt wird.

Algorithmus 4.1 Berechnung $G^{\mathbb{P}_n}$

input : $V, v_0 \in V;\ d_{i,j}(t)$
output : $G^{\mathbb{P}_n} = (V^{\mathbb{P}_n}, E^{\mathbb{P}_n})$
$n := |V| - 1$
$V^{\mathbb{P}_n} := \{(v_0, 0)\}$
$O_0 := \{(v_0, 0)\};\ O_k := \emptyset, k = 1, \ldots, n$
Berechne die inneren Kanten:
for $k := 0$ **to** n **do**
foreach $(v, s) \in O_k$ **do**
foreach $w \in V \setminus \{v_0, v\}$ **do**
$t := s + d_{vw}(s)$
if $(w, t) \notin V^{\mathbb{P}_n}$
then $V^{\mathbb{P}_n} := V^{\mathbb{P}_n} \cup \{(w, t)\}$
$O_{k+1} := O_{k+1} \cup \{(w, t)\}$
fi
$E^{\mathbb{P}_n} := E^{\mathbb{P}_n} \cup \{((v, s), (w, t))\}$
od
od
od
Berechne die Kanten nach (v_0, ∞) *:*
$V^{\mathbb{P}_n} := V^{\mathbb{P}_n} \cup \{(v_0, \infty)\}$
compute *rang* : $V^{\mathbb{P}_n} \to \mathbb{N};\ (v, t) \mapsto r((v_0, 0), (v, t))$
foreach $(v, t) \in V^{\mathbb{P}_n}$ **do**
if $rang((v, t)) \geq n$ **then** $E^{\mathbb{P}_n} := E^{\mathbb{P}_n} \cup \{((v, s), (v_0, \infty))\}$ **fi**
od

Sei $(i_0, 0) \to (i_1, t_1) \to \cdots \to (i_n, t_n) \to (i_{n+1}, t_{n+1})$ ein zeitexpandierter Pfad in V mit $i_0 = i_{n+1} = v_0$. Identifiziere (i_{n+1}, t_{n+1}) mit (v_0, ∞). Dann gibt es für $j \leq n$ ein $k \leq j$, so daß $(i_j, t_j) \in O_k$. Beweis durch Induktion über j:

Die Aussage ist klar für $j = 0$, denn $(v_0, 0) \in O_0$. Sei nun $(i_j, t_j) \in O_k$. Da $k \leq n$ werden in der innersten Schleife also alle von (i_j, t_j) aus erreichbaren Knoten, insbesondere also auch (i_{j+1}, t_{j+1}) erzeugt. Entweder ist dann bereits $(i_{j+1}, t_{j+1}) \in V^{\mathbb{P}_n}$ und die Behauptung gilt nach (4.2) für $k \leq j$ oder sie gilt für $k + 1 = j + 1$.

Schließlich sind also wegen 4.2 nach Abschluß der k-Schleife alle Knoten des Pfades bis auf den letzten in $V^{\mathbb{P}_n}$ enthalten, analog sind alle Kanten bis auf die letzte in $E^{\mathbb{P}_n}$ enthalten. Da es also mindestens einen n–schrittigen Pfad nach (i_n, t_n) gibt, gilt $r((i_n, t_n)) \geq n$ und die letzte Kante wird im zweiten Teil auch noch hinzugefügt.

Zur Komplexität:

Offensichtlich werden die Anweisungen der Schleifen im ersten Teil für jede innere Kante in $E^{\mathbb{P}_n}$ genau einmal ausgeführt. Verwendet man zur Darstellung von $V^{\mathbb{P}_n}$ eine geeignete Suchstruktur (zum Beispiel balancierte Bäume, siehe [57, Kap.5]), so lassen sich Einfügeoperationen und der Test auf Enthaltensein in $O(\ln|V^{\mathbb{P}_n}|)$ ausführen. Wegen $|V^{\mathbb{P}_n}| \leq |E^{\mathbb{P}_n}|$ hat der erste Teil also Komplexität $O(|E^{\mathbb{P}_n}| \ln |E^{\mathbb{P}_n}|)$.
Da für alle Kanten $((v,s),(w,t)) \in E^{\mathbb{P}_n}$ gilt $s < t$, ist $G^{\mathbb{P}_n}$ sicher azyklisch und der zu Beginn des zweiten Teils des Algorithmus bereits erzeugte Teil davon somit auch. Der eingangs zitierte Algorithmus zur Berechnung des Rangs benötigt in diesem Fall Zeit $O(|E^{\mathbb{P}_n}|)$ (bei Verwendung von nach Endknoten sortierten Adjazenzlisten zur Darstellung von $E^{\mathbb{P}_n}$). Da schließlich die letzte Schleife genau $|V^{\mathbb{P}_n}|$–mal ausgeführt wird, folgt die Behauptung.

■

Da sich $G^{\mathbb{P}_n}$ sicher nicht schneller als in $|E^{\mathbb{P}_n}|$ Schritten aufstellen läßt, da nun mal jede Kante mindestens einmal erzeugt werden muß, scheint der zusätzliche Term $\ln|E^{\mathbb{P}_n}|$ kein allzu hoher Preis zu sein. Wie groß $G^{\mathbb{P}_n}$ in Abhängigkeit von n werden kann, untersucht der nächste Abschnitt. Für den Fall des TDTSP läßt sich aus Algorithmus 4.1 eine $O(|E^T|)$–Methode für den G^T gewinnen. Dazu wird $V^{\mathbb{P}_n}$ durch ein zweidimensionales Feld dargestellt. Dann benötigt die Entscheidung $(v,t) \in V^{\mathbb{P}_n}$ nur noch konstante Zeit. Für diesen Fall gilt $G^{\mathbb{P}_n} = G^T$ und $|E^T| = O(n^3)$. Einer der Hauptgründe, $G^{\mathbb{P}_n}$ aufzustellen, war ja die Hoffnung, er sei im Falle allgemeiner DTSPs kleiner als G^T. Diese Hoffnung wird im nächsten Abschnitt relativiert.

4.2.3 Die Größe von $G^{\mathbb{P}_n}$

Der folgenden Satz zeigt, daß auch $G^{\mathbb{P}_n}$ schlimmstenfalls sehr groß wird:

Satz 4.2
Es gibt für jedes n DTSP–Instanzen, so daß $V^{\mathbb{P}_n}$ so viele Knoten enthält, wie es unterschiedliche, maximal $n+1$–schrittige Pfade in V gibt, die frühestens nach $n+1$ Schritten nach v_0 zurückkehren.

Beweis:
Die Beweisidee ist, die Information über den bis zu einem Knoten zurückgelegten Weg in der Ankunftszeit zu kodieren. Sei $V = \{0,\ldots,n\}$. Definiere die Wegezeiten wie folgt:

$$d_{ij}(t) = tn + j \tag{4.3}$$

Die Kosten sind beliebig. Sei nun $P = 0 \to i_1 \to i_2 \to \cdots \to i_k$ ein k-schrittiger Pfad in V. Dann berechnet sich die Ankunftszeit $d(P)$ in i_k so

$$d(P) = (\cdots(i_1(n+1)+i_2)(n+1)+\cdots+i_{k-1})(n+1)+i_k = \sum_{j=0}^{k}(n+1)^{k-j}i_j \tag{4.4}$$

das heißt, $d(P)$ ist gerade die Zahl $i_1 i_2 \cdots i_k$ in $n+1$–adischer Darstellung. Also ist die Abbildung, die jedem Knoten $(v,t) \in V^{\mathbb{P}_n}$ den Pfad, der durch die $n+1$–adischen Darstellung von t gegeben ist, zuordnet, eine Bijektion zwischen $V^{\mathbb{P}_n}$ und der Menge der maximal n–schrittigen Pfade in V, die frühestens nach $n+1$ Schritten nach v_0 zurückkehren. ■

Bemerkung: Wie man leicht nachrechnet, gibt es $n\frac{(n-1)^n-1}{n-3}+2$ Pfade des Typs von Satz 4.2. Da von jedem Knoten maximal $n+1$ Kanten ausgehen, hat Algorithmus 4.1 dann die Komplexität $O(n^2(n-1)^n \ln n)$.

Da insbesondere alle Touren auch Pfade sind, kann also auch die Aufstellung von $G^{\mathbb{P}_n}$ bereits aufwendiger sein als die Enumeration aller Touren.

Satz 4.3
Sei $w : S_{n+1} \to \mathbb{R}$ eine Funktion, die jeder Tour einen Wert zuordnet. Dann gibt es eine DTSP–Instanz mit

$$z_{DTSP}(\pi) = w(\pi)\,\forall \pi \in S_{n+1}, \pi \text{ zyklisch}$$

Beweis:
Definiere die Wegezeiten wie in Satz 4.2. Sei $t_{\pi^{-1}(v_0)}$ der Zeitpunkt, zu dem die Tour π in der letzten Stadt vor v_0 ankommt. Definiere die Kostenfunktionen

$$c_{i,j}(t) := \begin{cases} 1 & \\ w(\pi)-n+1 & j = v_0 \wedge \exists \text{Tour } \pi \text{ mit } i = \pi^{-1}(v_0) \wedge t_i = t \end{cases} \tag{4.5}$$

Jede Tour ist bereits durch den Pfad, auf dem sie durch V_0 verläuft, eindeutig bestimmt. Somit gibt es nach Satz 4.2 auch keine andere Tour, die zum Zeitpunkt $t_{\pi^{-1}(v_0)}$ in $\pi^{-1}(v_0)$ ankommt. Also sind die $c_{i,j}(t)$ wohldefiniert. Es gilt nach Konstruktion

$$z_{\mathrm{DTSP}}(\pi) = (n-1) + c_{\pi^{-1}(v_0),v_0}(t_{\pi^{-1}(v_0)}) = (n-1) + w(\pi) - n + 1 = w(\pi)$$

■

Satz 4.3 besagt also, daß ein Algorithmus zum Lösen allgemeiner DTSPs rein gar keine Struktur in den Kosten der Touren erwarten kann und sozusagen „beliebig pathologische“ Instanzen existieren. Am Anfang von Abschnitt 1.2 auf

Seite 4 wurde die Generalvoraussetzung getroffen, daß alle betrachteten DTSP–Instanzen polynomiale Eingabekodierungen besitzen. Für die Wegezeitfunktionen, definiert wie in Satz 4.2, trifft das sicher zu, die Bedingung wird aber potentiell von der Klasse der Kostenfunktionen aus Satz 4.3 verletzt. Insofern ist Satz 4.3 lediglich die theoretische Rechtfertigung dafür, die Allgemeinheit der untersuchten DTSP–Instanzen einzuschränken.

4.3 Bestimmung billigster Pfade mit einer festen Anzahl Schritte in G^X

Um die Methode aus Abschnitt 3.2 auf das DTSP anwenden zu können, fehlen noch Verfahren zur Berechnung der billigsten Pfade. Abschnitt 4.3.2 wird die Gleichungen (3.11),(3.12) und (3.13) von den Seiten 47 bis 48 für allgemeine, zeitexpandierte Graphen G^X anpassen, deren Paradigma der $G^{\mathbb{P}_n}$ ist. Da diese Verfahren nicht polynomial sind, versucht Abschnitt 4.4, Spezialfälle zu klassifizieren, in denen die Berechnung genauso einfach wie beim TDTSP ist.

4.3.1 Pfade

Zunächst ein paar Definitionen:

Definition 4.3
Sei $P = i_0 \to i_1 \to \cdots \to i_k$ ein Pfad in K_V wie in Definition 3.3 auf Seite 45. Dann bezeichnet $P(j) := i_j$ die Stadt an Position j in P. Ein Pfad P in V heißt also Anfangspfad, wenn gilt $P(0) = v_0$ und *Endpfad*, wenn gilt $P(k) = v_0$. ■

(vergleiche Definition 3.3 auf Seite 45) Die Notation $P(i)$ interpretiert den Pfad als eine Abbildung, die jeder Position eine Stadt zuordnet. Man beachte dies im Unterschied zur Einführung von Touren in Definition 1.1 auf Seite 1, wo durch eine Permutation jeder Stadt ihre Nachfolgerin in der Tour zugeordnet wurde. Wenn nicht anders vermerkt, wird im folgenden immer von Anfangspfaden ausgegangen.

Definition 4.4
Sei $\mathcal{I} = (V, E, d, c, v_0)$ eine Instanz von DTSP. Dann wird durch

$$\mathbb{P}_{i,k} = \mathbb{P}_{i,k}(V) := \{P \text{ Pfad in } K_V \mid P = i_0 \to i_1 \to \cdots \to i_k, i_0 = v_0, i_k = i\} \tag{4.6}$$

$$\mathbb{O}_{i,k} = \mathbb{O}_{i,k}(V) := \{P \text{ Pfad in } K_V \mid P = i_0 \to i_1 \to \cdots \to i_k, i_0 = i, i_k = v_0\} \tag{4.7}$$

für jede Stadt $i \in V$ und Schrittzahl $k \in \mathbb{N}$ die *Menge aller k-schrittigen Anfangspfade nach i* und *Menge aller k-schrittigen Endpfade von i aus* definiert. $\mathbb{P} = \mathbb{P}(V)$ ist die Menge *aller endlichen Pfade in V* überhaupt.

■

Definitionen 4.3 und 4.4 führen Begriffe für den „statischen" Aspekt von Pfaden ein. Die nächste Definition verläßt jetzt den K_V und betrachtet Pfade zusammen mit dem *Zeitpunkt*, zu dem sie starten. Da Anfangspfade nur zum Zeitpunkt 0 starten können, wird für diese im folgenden zur Vereinfachung die Angabe des Startzeitpunkts in der Regel weggelassen.

Definition 4.5
Sei $\mathcal{I} = (V, E, d, c, v_0)$ eine DTSP–Instanz und $G^X = (V^X, E^X)$ ein zeitexpandierter Graph dazu. Die *Länge* $l(P)$ eines Pfades $P = i_0 \to i_1 \to \cdots \to i_k$ ist k. *Der Pfad P ist für Startzeitpunkt* t_0 *in* G^X *enthalten*, wenn es Kanten $((i_j, t_j), (i_{j+1}, t_{j+1})) \in E^X$ für alle $j = 0, \ldots, k-1$ gibt. Der durch diese Kanten induzierte Untergraph von G^X heißt *in* G^X *zum Zeitpunkt* t_0 *zeitexpandierter Pfad*, kurz P_{t_0}. In Formeln:

$$P_{t_0} \in G^X :\Longleftrightarrow \exists t_1, \ldots, t_{l(P)} : \{((P(i), t_i), (P(i+1), t_{i+1})\} \subseteq E^X$$

Die *Dauer* $d(P, t_0)$ *eines Pfades P* und die *Kosten* $c(P, t_0)$ *eines Pfades P* berechnen sich durch:

$$d(P, t_0) := \begin{cases} t_0 & l(P) = 0 \\ d(Q) + d_{P(l(P)-1),i}(d(Q)) & l(P) > 0 \wedge P = Q \to i \end{cases} \tag{4.8}$$

$$c(P, t_0) := \begin{cases} t_0 & l(P) = 0 \\ c(Q) + c_{P(l(P)-1),i}(d(Q)) & l(P) > 0 \wedge P = Q \to i \end{cases} \tag{4.9}$$

Zur Berechnung von Pfadkosten in G^X sei

$$c_{G^X}(P_{t_0}) = c(P_{t_0}) := \begin{cases} \infty & \text{falls } P_{t_0} \notin G^X \\ c(P, t_0) & \text{sonst} \end{cases} \tag{4.10}$$

vereinbart.

$$c^\star_{i,k} := \min_{P \in \mathbb{P}_{i,k}} c(P) \tag{4.11}$$

$$\mathbb{P}^\star_{i,k} := \{P \in \mathbb{P}_{i,k} | c(P) = c^\star_{i,k}\} \tag{4.12}$$

$$\mathbb{P}^{\star\star}_{i,k} := \{P \in \mathbb{P}^\star_{i,k} | d(P) \leq d(P') \forall P' \in \mathbb{P}^\star_{i,k}\} \tag{4.13}$$

sind die *Kosten des billigsten k-schrittigen Anfangspfades nach i* und die *Mengen der billigsten und schnellsten billigsten, k-schrittigen Anfangspfade nach i*. Die Einschränkungen auf Pfade in G^X lauten:

$$\mathbb{P}_{i,k}(G^X) := \{P \in \mathbb{P}_{i,k} \mid P \in G^X\} \tag{4.14}$$

$$c^\star_{i,k}(G^X) := \min_{P \in \mathbb{P}_{i,k}(G^X)} c(P) \tag{4.15}$$

$$\mathbb{P}^\star_{i,k}(G^X) := \{P \in \mathbb{P}_{i,k}(G^X) | c(P) = c^\star_{i,k}(G^X)\} \tag{4.16}$$

$$\mathbb{P}^{\star\star}_{i,k}(G^X) := \{P \in \mathbb{P}^\star_{i,k}(G^X) | d(P) \leq d(P') \forall P' \in \mathbb{P}^\star_{i,k}(G^X)\} \tag{4.17}$$

Für eine endliche Menge $T_0 \subset \mathbb{R}$ von Startzeitpunkten sei

$$b^\star_{i,k}(T_0) := \min_{\substack{P \in \mathbb{O}_{i,k} \\ t \in T_0}} c(P,t) \tag{4.18}$$

$$\mathbb{O}^\star_{i,k}(T_0) := \{(P,t) \mid c(P,t) = b^\star_{i,k}(T_0), P \in \mathbb{O}_{i,k}, t \in T_0\} \tag{4.19}$$

die *Kosten der billigsten k-schrittigen Endpfade von i aus zu den Zeitpunkten* T_0 und die *Mengen eben dieser Endpfade*. $b^\star_{i,k}(G^X)$ und $\mathbb{O}^\star_{i,k}(G^X)$ sind die Einschränkungen auf die Endpfade in G^X. ■

Bemerkung: Der zeitexpandierte Graph muß nicht notwendigerweise alle möglichen Pfade enthalten, deshalb gilt wegen 4.10 $c_{G^X}(P) \geq c(P)$ und somit $c^\star_{i,k} \leq c^\star_{i,k}(G^X)$.

Die Begriffsbildung für Endpfade ist aufwendiger, da es keine Entsprechung mehr zwischen den Pfaden in K_V und den zeitexpandierten Versionen gibt. Vielmehr existiert zu jedem beliebigen Zeitpunkt eine Expansion. In einem zeitexpandierten Graphen G^X schränkt sich diese Vielfalt auf eine endliche Anzahl ein.

4.3.2 Pfadberechnung in zeitexpandierten Graphen

Ein zeitexpandierter Graph G^X ist ein dünner, azyklischer Digraph. Jeder Knoten ist Anfangsknoten von maximal $n+1$ Kanten und somit gilt $O(|E^X|) = O(n|V^X|)$. Weil Kanten immer später enden als sie beginnen, kann G^X keine Kreise enthalten.

Sei $P(i,t,k)$ ein billigster Pfad unter den k-schrittigen Pfaden nach i, die zum Zeitpunkt t ankommen. Mittels dynamischer Programmierung lassen sich Pfade $P(i,t,k)$ einfach berechnen. Die folgenden Gleichungen sind die bekannten

Bellmann–Gleichungen [5] für einen beliebigen zeitexpandierten Graphen G^X. Sie sind analog zu (3.11) auf Seite 47.

$$P(v_0, 0, 0) = (v_0) \tag{4.20}$$

$$P(i, t, k+1) = P(j, s, k) \to i \quad \text{so daß } s + d_{j,i}(s) = t \wedge c(P(j, s, k) \to i) \leq c(P(j', s', k) \to i) \; \forall j', s' : s' + d_{j',i}(s \tag{4.21}$$

Satz 4.4
Für die durch (4.20) und (4.21) beschriebene Rekursion gilt:

i) $P(i, t, k) \in \mathbb{P}_{i,k}$
ii) $d(P(i, t, k)) = t$
iii) $\exists P \in \mathbb{P}_{i,k} : d(P) = t \Longrightarrow P(i, t, k)$ *existiert* $\wedge\, P(i, t, k) \in \mathbb{P}^{\star}_{i,k}$
iv) $c^{\star}_{i,k} = \min_t c(P(i, t, k))$

Beweis:
zu i): Induktion über k

I.A. : $P(v_0, 0, 0) = v_0 \in \mathbb{P}_{v_0,0}$
I.S. : $k \to k+1$
$\exists j, s : P(i, t, k+1) = P(j, s, k) \to i$ nach (4.21)
$\Longrightarrow (P(i, t, k+1))(k+1) = i$
$\wedge \quad l(P(i, t, k+1)) = l(P(j, s, k)) + 1 = k+1$
$\Longrightarrow P(i, t, k+1) \in \mathbb{P}_{i,k+1}$

zu ii):

$k = 0 : d(P(v_0, 0, 0)) = 0$ nach (4.20)
$k \geq 1 : d(P(i, t, k+1)) = d(P(j, s, k) \to i) = t$ nach (4.21)

zu iii): Induktion über k

I.A. : $\mathbb{P}_{i,0} = \mathbb{P}_{v_0,0} = \{(v_0)\}$
I.S. : $k \to k+1$
Sei $P \in \mathbb{P}_{i,k+1}$; $d(P) = t$; $P = Q \to i$; $Q \in \mathbb{P}_{j,k}$
$\Longrightarrow P(j, d(Q), k)$ existiert nach I.V.

$$
\begin{aligned}
&\Longrightarrow (j, d(Q)) \in \{(j', s') | d(P(j', s', k) \to i) = t\} \\
&\Longrightarrow P(i, t, k+1) \text{ existiert} \\
c(P) &\geq \min_{j,R} \{c(R \to i) | R \in \mathbb{P}_{j,k};\ d(R \to i) = t\} \\
&= \min_{s',j,R} \{c(R) + c_{j,i}(s') | R \in \mathbb{P}_{j,k};\ d(R) = s';\ s' + d_{j,i}(s') = t\} \\
&= \min_{s',j} \{c_{j,i}(s') + \min_{R} \underbrace{\{c(R) | R \in \mathbb{P}_{j,k};\ d(R) = s'\}}_{\text{i),ii),I.V. } \Rightarrow c(R) \geq c(P(j,s',k))} | s' + d_{j,i}(s') = t\} \\
&= \min_{j,s'} \{c(P(j, s', k)) + c_{j,i}(s') | s' + d_{j,i}(s') = t\} \\
&= \min_{j,s'} \{c(P(j, s', k) \to i) | d(P(j, s', k) \to i) = t\} \\
&= c(P(i, t, k+1)) \quad \text{nach 4.21}
\end{aligned}
$$

zu iv): $P(i,t,k) \in \mathbb{P}_{i,k}$ nach i)

$$
\begin{aligned}
&\Longrightarrow c^\star_{i,k} \leq c(P(i,t,k)) \quad \forall t \\
&\Longrightarrow c^\star_{i,k} \leq \min_t c(P(i,t,k)) \\
&\exists P \in \mathbb{P}_{i,k} : c^\star_{i,k} = c(P) \\
&\Longrightarrow c^\star_{i,k} = c(P) \geq c(P(i, d(P), k)) \text{ wg. iii)} \\
&\qquad\qquad \geq \min_t c(P(i,t,k))
\end{aligned}
$$

■

Lemma 4.5
Algorithmus 4.2 berechnet $P(i,t,k)$ mit Komplexität $O(n|E^X|)$.

Beweis:

Offensichtlich terminiert der Algorithmus, da die innerste Schleife für jede Kante in E^X genau $n+1$-mal ausgeführt wird. V^X wird in topologischer Reihenfolge durchlaufen, das heißt, alle Kanten mit Endknoten $v \in V^X$ haben einen Anfangsknoten, der in der Sortierung vor v liegt. Nach jeder Ausführung der inneren Schleife gilt:

$$cost((i,t),k) = c(P(i,t,k))$$

da alle Knoten, über die die Minimierung in (4.20) läuft, als Anfangsknoten der untersuchten Kanten vorkamen. Die Pfade $P(i,t,k)$ lassen sich anhand der Kostenwerte zurückverfolgen. Nach Satz 4.4,iv) ist auch *bestcost* korrekt.
Wird *cost* zum Beispiel so implementiert, daß die k Werte $cost((i,t),\cdot)$ direkt am Knoten (i,t) notiert werden, so benötigen alle Zugriffe konstante Zeit und die Initialisierung somit $O(n|V^X|) = O(|E^X|)$. *bestcost* ist ein zweidimensionales Feld und die Initialisierung gelingt in Zeit $O(n^2)$. Wird G^X durch nach Endknoten sortierte Adjazenzlisten dargestellt, benötigt die Schleifenkontrolle konstante Zeit. Der zentrale Schritt benötigt auch nur konstante Zeit und wird $O(n|E^X|)$-mal ausgeführt.

■

Algorithmus 4.2 Berechnung von $P(i,t,k)$ in G^X

input : $G^X = (V^X, E^X)$ topologisch sortiert, Kantenkosten $c(e)$
$V, v_0 \in V$
output : $cost : V^X, k \to \mathbb{R};\ ((i,t),k)) \mapsto c_{G^X}(P(i,t,k))$
$bestcost : V, k \to \mathbb{R};\ i,k \mapsto c^\star_{ik}(G^X)$

```
n:= |V| −1
cost(v,k):=∞   ∀v ∈ V^X, k = 0,...,n+1
bestcost(i,k):=∞   ∀i ∈ V, k = 0,...,n+1
cost((v_0,0),0):=0
bestcost(v_0,0):=0
foreach (i,t) ∈ V^X in topologischer Reihenfolge do
  foreach k:=0 to n do
    foreach e = ((j,s),(i,t)) ∈ E^X do
      cost((i,t),k+1):= min{cost((j,s),k)+c(e), cost((i,t),k+1)}
    od
    bestcost(i,k+1) = min{bestcost(i,k+1), cost((i,t),k+1)}
  od
od
```

Bemerkung: Da G^X ein azyklischer Digraph ist, läßt er sich in $O(|E^X|)$ topologisch sortieren (siehe [41]) und erhöht als Vorverarbeitungsschritt die Komplexität nicht.

Im Fall des TDTSP ist $|E^T| = O(n^3)$. Die Berechnung von $c^\star_{ik}$ mittels (3.11) hat Komplexität $O(n^3)$, mittels Algorithmus 4.2 jedoch $O(n^4)$. Da $|E^X|$ nicht polynomial in n beschränkt ist (siehe Abschnitt 4.2.2), ist Algorithmus 4.2 im allgemeinen nicht polynomial in n.

Ähnlich wie im TDTSP–Fall lassen sich auch through–circuits berechnen. Der Unterschied liegt darin, daß nicht a–priori bekannt ist, durch welchen Knoten (i,t) der i,k–through-circuit hindurchläuft. Sei $O(i,t,k)$ ein billigster, k–schrittiger Endpfad von i aus zum Zeitpunkt t. Dann gilt:

$$O(v_0,t,0) = v_0 \tag{4.22}$$

$$O(i,t,k+1) = i \to O(j,s,k) \tag{4.23}$$
so daß $t + d_{i,j}(t) = s \wedge$
$c(i \to O(j,s,k)) \le c(i \to O(j',s',k))\ \forall j',s' : t + d_{i,j'}(t) = s'$

Satz 4.6
Für die durch (4.22) und (4.23) beschriebene Rekursion gilt:

$$\begin{aligned} &i) && O(i,t,k) \in \mathbb{O}_{i,k} \\ &ii) && O(i,t,k) \text{ hat Startzeitpunkt } t \\ &iii) && (O(i,t,k),t) \in \mathbb{O}^{\star}_{i,k}(t) \\ &iv) && b^{\star}_{i,k}(T) = \min_{t \in T} c(O(i,t,k)) \qquad T \subset \mathbb{R} \text{ endlich} \end{aligned}$$

Beweis:
i), ii) und iv) sind trivial. Teil iii) folgt analog zum Beweis von iii) in Satz 4.4. Beachte, daß sich im Fall von Endpfaden die Existenzfrage nicht stellt. ■

Betrachtet man G^X in umgekehrter topologischer Reihenfolge und die Kanten von E^X in umgekehrter Richtung so berechnet Algorithmus 4.2 $O(i,t,k)$. Die Anwendung von (4.22),(4.23) beginnt dabei mit den $O(v_0,t,0)$, zu denen es eine Kante $((v,s),(v_0,\infty))$ mit $t = s + d_{v,v_0}(s)$ in E^X gibt.

Betrachte nun einen i,k—through–circuit P, der Stadt i an Position k zum Zeitpunkt t durchläuft. Dann ist der Anfang von P bis i ein billigster k–schrittigen Anfangspfad nach i und der Rest von P ab i ein billigster $n+1-k$–schrittiger Endpfad von i aus. Deshalb gilt:

$$\psi_{ik} = \psi_{ik}(G^X) = \min_t \{c_{G^X}(P(i,t,k)) + c_{G^X}(O(i,t,n+1-k))\} \tag{4.24}$$

wobei die Minimierung wegen (4.10) natürlich tatsächlich nur über endlich viele t läuft.

4.4 Berechnung billigster, fortgesetzter Anfangspfade

Der hohe Aufwand zur Berechnung von $c^{\star}_{i,k}$ mittels Algorithmus 4.2 kommt daher, daß er in G^X arbeitet, aber eigentlich „zeitlose“, positionsabhängige Werte berechnen soll. Da es im allgemeinen nicht wie im statischen und im TDTSP–Fall möglich ist, billigste Wege aus kürzeren billigsten Wegen zusammenzusetzen, können die Informationen, die bei der Berechnung $c^{\star}_{\cdot,k}$ gefunden wurden, nicht zur Berechnung von $c^{\star}_{\cdot,k+1}$ verwendet werden. Dieser Abschnitt befaßt

sich mit DTSP Instanzen, in denen so etwas dennoch möglich ist. Leider wird sich in Abschnitt 4.4.3 herausstellen, daß sich die Ergebnisse nicht numerisch nutzen lassen. Aus diesem Grund sind alle Aussagen in diesem Abschnitt für DTSP–Instanzen formuliert und nicht für deren algorithmische Inkarnationen, die zeitexpandierten Graphen. Für Graphen G^X ließen sich die Voraussetzungen der meisten Ergebnisse etwas abschwächen und die Aussagen ein wenig verschärfen. Darauf wurde verzichtet, um die Formulierung nicht unnötig zu komplizieren.

4.4.1 DTSP–Instanzen mit Fortsetzungseigenschaft

Um billigsten k–schrittige Pfade mittels einer auf Positionen beruhenden Rekursion zu berechnen, reicht es nicht aus, nur die Existenz von aus billigsten Pfaden zusammengesetzten billigsten Pfaden zu verlangen. Damit die Rekursion die Pfade auch tatsächlich findet, muß noch der zeitlicher Aspekt eingeschlossen werden:

Definition 4.6
Sei $\mathcal{I} = (V, E, d, c, v_0)$ eine DTSP Instanz.

- ein schnellster billigster k–schrittiger Anfangspfad P in V heißt *fortgesetzt*, wenn seine ersten $k-1$ Schritte einen schnellsten billigsten $k-1$–schrittigen Pfad bilden:

$$P \in \mathbb{P}^{\star\star}_{i,k} \text{ heißt fortgesetzt} :\Longleftrightarrow \exists Q \in \mathbb{P}^{\star\star}_{j,k-1} : P = Q \to i$$

- $\mathcal{I}$ erfüllt *die Fortsetzungseigenschaft für schnellste billigste Pfade mit fester Schrittzahl*, wenn für jede Stadt und jede Schrittzahl ein fortgesetzter, schnellster billigster Anfangspfad existiert:

$$\forall i \in V, k \in \mathbb{N}, k > 0 \exists P \in \mathbb{P}^{\star\star}_{i,k} : P = Q \to i \text{ mit } Q \in \mathbb{P}^{\star\star}_{j,k-1}$$

■

Definiere nun durch folgende Variante von (4.20),(4.21) k–schrittige Anfangspfade $P(i,k)$:

$$P(v_0, 0) = v_0 \tag{4.25}$$

$$P(i, k+1) = P(j,k) \to i \tag{4.26}$$

$$\text{so daß} \quad c(P(j,k) \to i) \leq c(P(j',k) \to i) \quad \forall j' \tag{4.26a}$$

$$d(P(j,k) \to i) \leq d(P(j',k) \to i) \quad \forall j' : c(P(j',k) \to i) = c(P(j,k) \to i) \tag{4.26b}$$

Satz 4.7
Sei $\mathcal{I}$ eine DTSP–Instanz, die die Fortsetzungseigenschaft erfüllt. Dann gilt für die Pfade $P(i,k)$ aus (4.25),(4.26)*:*

$$P(i,k) \in \mathbb{P}^{\star\star}_{i,k}$$

Beweis:
Klar ist: $l(P(i,k)) = k$ und $P(i,k) \in \mathbb{P}_{i,k}$. Induktion über k. Die Behauptung ist trivial für $k = 0, 1$. Induktionsschritt $k \longrightarrow k+1$:
Nach Voraussetzung existiert $P' \in \mathbb{P}^{\star\star}_{i,k+1}$ mit $P' = Q \to i$ und $Q \in \mathbb{P}^{\star\star}_{j',k}$. Deshalb gilt:

$$\begin{aligned}
d(P(j',k)) &= d(Q) && \text{wg. I.V.:} Q, P(j',k) \in \mathbb{P}^{\star\star}_{j',k} \\
c(P(i,k+1)) &\le c(P(j',k) \to i) && \text{wg. (4.26a)} \\
&= c(P(j',k)) + c_{j',i}(d(P(j',k))) \\
&\le c(Q) + c_{j',i}(d(Q)) && \text{wg. I.V.} \\
&= c(Q \to i) = c(P') \\
\Longrightarrow \quad P(i,k+1) &\in \mathbb{P}^{\star}_{i,k+1}
\end{aligned}$$

Deswegen gilt tatsächlich überall das Gleichheitszeichen. Weiter:

$$\begin{aligned}
d(P(i,k+1)) &= d(P(j,k) \to i) && \text{für geeignetes } j \\
&\le d(P(j',k) \to i) && \text{wg. (4.26b)} \\
&= d(P(j',k)) + d_{j',i}(d(P(j',k))) \\
&= d(Q) + d_{j',i}(d(Q)) && \text{wg. I.V.} \\
&= d(Q \to i) = d(P') \\
\Longrightarrow \quad P(i,k+1) &\in \mathbb{P}^{\star\star}_{i,k+1}
\end{aligned}$$

■

Wann gilt nun die Fortsetzungseigenschaft? Mit den beiden folgenden Korollaren gelingt es, eines der Sternchen in der Definition der Fortsetzungseigenschaft durch andere Bedingungen zu ersetzen. Genauer gesagt, es genügt, wenn billigste, kürzeste Pfade Fortsetzungen von billigsten Pfaden sind.

Korollar 4.8
Falls in $\mathcal{I}$ gilt:

$$\begin{aligned}
&\textit{i)}\ \exists P \in \mathbb{P}^{\star\star}_{i,k} \textit{ mit } Q \in \mathbb{P}^{\star}_{j,k-1} : P = Q \to i \\
&\textit{ii)}\ c_{i,j}(t) \textit{ monoton steigend} \\
&\textit{iii)}\ d_{i,j}(t) \le \tau + d_{i,j}(t+\tau) \qquad \textit{„Überholverbot“}
\end{aligned}$$

dann gilt für $P(i,k)$ aus (4.25),(4.26)*: $P(i,k) \in \mathbb{P}^{\star\star}_{i,k}$*

Beweis:

Zu zeigen ist die Fortsetzungseigenschaft für *1*. Sei also $P \in \mathbb{P}^{\star\star}_{i,k}$ mit Q wie in i) und sei $Q' \in \mathbb{P}^{\star\star}_{j,k-1}$, dann gilt

$$\begin{aligned}
c(Q' \to i) &= c(Q') + c_{j,i}(d(Q')) \\
&= c(Q) + c_{j,i}(d(Q) - \tau), \quad \tau = d(Q) - d(Q') \geq 0 \qquad \text{wg.} \quad Q' \in \mathbb{P}^{\star\star}_{j,k-1} \\
&\leq c(Q) + c_{j,i}(d(Q)) \\
&= c(Q \to i) = c(P) \\
d(Q' \to i) &= d(Q') + d_{j,i}(d(Q')) \\
&= d(Q) - \tau + d_{j,i}(d(Q) - \tau), \quad \tau = d(Q) - d(Q') \geq 0 \\
&\leq d(Q) + d_{j,i}(d(Q)) \\
&= d(Q \to i) = d(P) \\
\Longrightarrow \qquad Q' \to i &\in \mathbb{P}^{\star\star}_{i,k}
\end{aligned}$$

■

Korollar 4.9
Falls in 1 gilt:

$$\begin{aligned}
&i)\ \exists P \in \mathbb{P}^{\star\star}_{i,k}\ \textit{mit}\ Q \in \mathbb{P}^{\star}_{j,k-1} : P = Q \to i \\
&ii)\ c_{i,j}(t)\ \textit{streng monoton steigend}
\end{aligned}$$

dann gilt für $P(i,k)$ *aus* (4.25),(4.26)*:* $P(i,k) \in \mathbb{P}^{\star\star}_{i,k}$

Beweis:

Zu zeigen ist wieder die Fortsetzungseigenschaft für *1*. Sei also $P \in \mathbb{P}^{\star\star}_{i,k}$ mit Q wie in i) und sei $Q' \in \mathbb{P}^{\star\star}_{j,k-1}$, dann gilt

$$\begin{aligned}
& c(Q) + c_{ji}(d(Q)) = c(P) \leq c(Q' \to i) = c(Q') + c_{ji}(d(Q')) && \text{wg.} Q' \in \mathbb{P}_{i,k} \\
\Longrightarrow \quad & c_{ji}(d(Q)) \leq c_{ji}(d(Q')) && \text{wg.} Q, Q' \in \mathbb{P}^{\star}_{j,k-1} \\
\Longrightarrow \quad & d(Q) \leq d(Q') && \text{wg. ii)} \\
\Longrightarrow \quad & Q \in \mathbb{P}^{\star\star}_{j,k-1}
\end{aligned}$$

■

Bemerkung: Man kann sich natürlich des zweiten Sternchens in „$P \in \mathbb{P}^{\star\star}_{i,k}$“ dadurch entledigen, daß man die Bedingung zu allen fraglichen Zeitpunkten fordert, das heißt indem man i) in den Korollaren durch i*) ersetzt:

i*) $\forall i,k,t : P \in \mathbb{P}^{\star}_{i,k}, d(P) = t \Rightarrow \exists P' \in \mathbb{P}^{\star}_{i,k} : d(P') = t \wedge \exists j, Q \in \mathbb{P}^{\star}_{j,k-1} : P' = Q \to i$

denn dann gilt sie sicher auch für die schnellsten, billigsten Pfade. Tatsächlich kann man i) so in einer Instanz eines G^X auch algorithmisch prüfen. Alternativ kann man auch noch die Bedingung an t in i*) weglassen, das heißt fordern, daß alle billigsten Pfade Fortsetzungen billigster Pfade sind.

Die Rekursionsgleichungen (4.25),(4.26) lassen sich in Zeit $O(n^3)$ lösen, denn zur Lösung von (4.26) müssen jeweils n Kandidaten untersucht werden und es werden $O(n^2)$ Pfade $P(i,k)$ berechnet. Im Fall des TDTSP sind (4.25),(4.26) äquivalent zu (3.11), da dann $d(P(i,k)) = k$ gilt und somit (4.26b) trivialerweise erfüllt ist. Auffällig ist, daß der G^T eines TDTSPs die Fortsetzungseigenschaft für beliebige Kantengewichte erfüllt.

4.4.2 Partitionierte DTSP-Instanzen

Wie die folgenden Überlegungen zeigen werden, ist das TDTSP (bis auf Äquivalenz) auch der einzige Spezialfall des DTSP, der die Fortsetzungseigenschaft für beliebige Kantengewichte erfüllt.

Definition 4.7
Eine DTSP–Instanz heißt *partitioniert*, falls alle k–schrittigen Anfangspfade in eine Stadt $i \in V_0$ gleichzeitig ankommen:

$$\forall i \in V_0, k > 0 : d(\mathbb{P}_{i,k}) = \{d_{i,k}\}$$

Offensichtlich sind TDTSP–Instanzen partitioniert. Außerdem gilt:

Lemma 4.10
Der $G^{\mathbb{P}_n}$ zu einer partitionierten DTSP–Instanz mit mehr als zwei Städten ist $n+2$–partit.

Beweis:
Sei $\mathfrak{M}_k$ die Menge der Knoten, die nach genau k Schritten von $(v_0, 0)$ aus erreichbar sind. Dann bildet die Familie $(\mathfrak{M}_k)_{k=0}^{n+1}$ eine Partition der Knotenmenge $V^{\mathbb{P}_n}$. Beweis:

$$V^{\mathbb{P}_n} = \bigcup_{k=0}^{n+1} \mathfrak{M}_k$$

nach Definition 4.2 auf Seite 58 des $G^{\mathbb{P}_n}$. Die Inklusion nach rechts gilt, da $V^{\mathbb{P}_n}$ die Vereinigung aller Knoten aller $n+1$–schrittigen Pfade ist und sich jeder k–schrittige Pfad zu einem $n+1$–schrittigen fortsetzen läßt. Die umgekehrte Inklusion gilt, da insbesondere $V^{\mathbb{P}_n} = \mathfrak{M}_{n+1}$.
Zu zeigen ist also noch, daß die $\mathfrak{M}_k$ paarweise disjunkt sind. Definiere die Zeitpunkte $s_{v,k}$ für alle $v \in V$ und $k \in \mathbb{N}$ durch $d(\mathbb{P}_{v,k}) = \{s_{v,k}\}$. Sei nun $(v, s_{v,k}) \in \mathfrak{M}_k$ und $(w, s_{w,l}) \in \mathfrak{M}_l$ mit $l > k$ und zeige $(v, s_{v,k}) \neq (w, s_{w,l})$:

Falls $v \neq w$: trivial

Falls $v = w \wedge l > k+1$:

Sei $P \in \mathbb{P}_{v,k} \Longrightarrow \exists Q : P \to Q \in \mathbb{P}_{v,l}$
$\Longrightarrow s_{v,l} = d(P \to Q) \geq d(P) + \underbrace{d_{v,Q(0)}(d(P))}_{>0 \text{ wg. Def. 1.1}} > s_{v,k}$

Falls $v = w \wedge l = k+1$: Indirekt. Annahme: $s_{v,k} = s_{v,k+1}$

Sei $P \in \mathbb{P}_{v,k}, P' \in \mathbb{P}_{v,k+1};\ u, u' \in V \setminus \{v\}, u \neq u'$
$\Longrightarrow \quad s_{u,k+1} = d(P \to u) = d(P' \to u) = s_{u,k+2}$
$\wedge\ s_{u',k+1} = d(P \to u') = d(P' \to u') = s_{u',k+2}$ denn $d(P) = d(P') = s_{v,k}$
Sei $Q \in \mathbb{P}_{u,k+1}, Q' \in \mathbb{P}_{u',k+1}$
$\Longrightarrow \quad s_{u,k+2} = d(Q' \to u) = s_{u',k+1} + d_{u',u}(d(Q')) > s_{u',k+1}$
$\wedge\ s_{u',k+2} = d(Q \to u') = s_{u,k+1} + d_{u,u'}(d(Q)) > s_{u,k+1}$
$\Longrightarrow s_{u,k+1} = s_{u,k+2} > s_{u',k+1} = s_{u',k+2} > s_{u,k+1}$ Widerspruch!

Da nun alle Kanten in $E^{\mathbb{P}_n}$ zu Pfaden gehören, gibt es nur Kanten mit Anfangsknoten in $\mathfrak{M}_k$ und Endknoten in $\mathfrak{M}_{k+1}$ ■

Die Vermutung drängt sich auf, daß die partitionierten Instanzen gerade die TDTSP–Instanzen sind.

Satz 4.11
Sei $\mathcal{I} = (V, E, d, c, v_0)$ eine partitionierte DTSP–Instanz. Dann existiert ein DTSP–Instanz $\mathcal{I}' = (V, E, d', c', v_0)$ mit

$$d'_{i,j}(t) \equiv 1 \qquad \forall\, i, j$$
$$c'(P) = c(P) \qquad \forall P \in \mathbb{P}(V)$$

das heißt, $\mathcal{I}$ ist äquivalent zu einem TDTSP.

Beweis:
Sei $\{s_{i,k}\} = d(\mathbb{P}_{i,k})$, wobei $d(P)$ die Wegezeitfunktion (4.8) für Pfade zur Instanz $\mathcal{I}$ ist. Definiere $c'_{i,j}(k) := c_{i,j}(s_{i,k})$ (da $\mathcal{I}'$ ein TDTSP ist, genügt das).

Induktion über die Pfadlänge $k = l(P)$:
Induktionsanfang: $k = 0 \Longrightarrow c(P) = c'(P) = 0$
Induktionsschritt: $k \to k+1$, $P = Q \to i, Q \in \mathbb{P}_{j,k}$

$$\begin{aligned} c'(P) &= c'(Q) + c'_{j,i}(d'(Q)) \\ &= c'(Q) + c'_{j,i}(k) && \text{wg. } \mathcal{I}' \text{ TDTSP} \\ &= c(Q) + c_{j,i}(s_{j,k}) && \text{wg. I.V. und Definition } c'_{i,j}(k) \\ &= c(Q) + c_{j,i}(d(Q)) = c(P) \end{aligned}$$ ■

Die partitionierten Instanzen sind weiterhin genau die, die unabhängig von den Wegekosten die Fortsetzungseigenschaft erfüllen:

Lemma 4.12
Eine partitionierte Instanz erfüllt die Fortsetzungseigenschaft

Beweis:
Sei $P = Q \to i \in \mathbb{P}^{\star\star}_{i,k}$ mit $Q \in \mathbb{P}_{j,k-1}$. Sei $Q' \in \mathbb{P}^{\star\star}_{j,k-1}$ Dann gilt:

$$c(Q' \to i) = c(Q') + c_{j,i}(d(Q')) \leq c(Q) + c_{j,i}(d(Q)) = c(P) \quad \text{weil } d(Q) = d(Q')$$
$$d(Q' \to i) = d(Q') + d_{j,i}(d(Q')) = d(Q) + d_{j,i}(d(Q)) = d(P)$$
$$\Longrightarrow \quad P' = Q' \to i \in \mathbb{P}^{\star\star}_{i,k} \text{ ist fortgesetzt}$$

■

Satz 4.13
Sei $V = \{v_0\} \cup V_0$ eine Menge von mehr als zwei Städten und seien $d_{i,j}(t)$ Wegezeitfunktionen auf V. Wenn dann alle DTSP–Instanzen $\mathcal{I}_c = (V, E, d, c, v_0)$ mit beliebigen Kostenfunktionen $c_{ij}(t)$ die Fortsetzungseigenschaft erfüllen, sind alle $\mathcal{I}_c$ partitioniert.

Beweis:
Zur Notation: In diesem Beweis sei $V = \{v_0, u, v, w, x_1, \ldots, x_{n-3}\}$. Pfade werden als Worte über V notiert, zum Beispiel

$$uvwx_1 \text{ ist kurz für } u \to v \to w \to x_1$$

Außerdem sei zusätzlich vereinbart, daß, wenn nicht anders angegeben, „x“ für irgendein x_i steht.
Induktion über die Pfadlänge $k = l(P)$. Induktionsbehauptung:

$$d(\mathbb{P}_{i,l}) = \{s_{i,l}\} \forall\, i \in V_0, l \leq k$$

Induktionsanfang: klar für $k = 1$, wegen $\mathbb{P}_{i,1} = \{v_0 \to i\}$

Induktionsschritt: $k \to k + 1$
Nimm zum indirekten Beweis an, es gebe $u \in V_0$ mit $P \in \mathbb{P}_{v,k}$ und $P' \in \mathbb{P}_{w,k}$, so daß

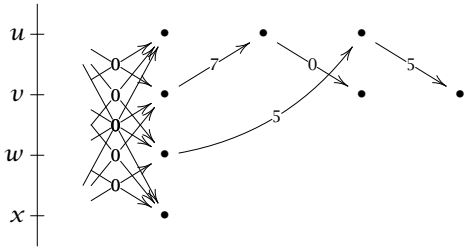
$$d(P \to u) =: s \neq s' := d(P' \to u)$$

gilt. Wäre $v = w$, so wäre wegen der Induktionsvoraussetzung auch $d(P) = d(P')$ und damit $s = s'$. Also folgt $v \neq w$.

Gelingt es nun, die Wegekosten c so zu definieren, daß

$$\begin{aligned}\mathbb{P}^{\star}_{v,k+2} &= \{Q \to vuv \mid Q \in \bigcup_{i \in V_0 \setminus \{v\}} \mathbb{P}_{i,k-1} \setminus \overline{\mathbb{P}}\} \text{ und} \\ \mathbb{P}^{\star}_{u,k+1} &= \{Q \to wu \mid Q \in \bigcup_{i \in V_0 \setminus \{w\}} \mathbb{P}_{i,k-1} \setminus \overline{\mathbb{P}}\}\end{aligned} \tag{4.27}$$

folgt, ist wegen $v \neq w$ ein Widerspruch zur Fortsetzungseigenschaft konstruiert. $\overline{\mathbb{P}}$ ist eine Ausschlußmenge, die aus technischen Gründen später noch gebraucht wird. Man kann sich zunächst $\overline{\mathbb{P}} = \emptyset$ vorstellen. Da im Fall von (4.27) also alle billigsten $k+1$–schrittigen Anfangspfade nach u über w, alle $k+2$–schrittigen Anfangspfade nach v aber über v und dann u laufen, kann keiner der letzteren, insbesondere auch kein schnellster, fortgesetzt sein. Definiere dazu c wie folgt:



$c_{i,j}(t)$	u	v	w	x
$c_{u,\cdot}(s)$		0	10	10
$c_{u,\cdot}(s')$		5	10	10
$c_{v,\cdot}(s_{v,k})$	7		10	10
$c_{w,\cdot}(s_{w,k})$	5	10		10

$c_{i,j}(s_{i,l}) := 0 \quad \forall i,j \in V_0, l < k, s_{i,l} \notin \{s, s', s_{i,k}\}$

$c_{i,j}(t) := 10$ sonst

Die Idee ist, daß alle Anfangspfade bis zur ersten Reihe Knoten in der Skizze oben keine Kosten haben und daß das genau die k–schrittigen Anfangspfade sind. Schwierigkeiten macht dabei der Fall, daß $s_{i,l}$ und $s_{i,k}$ für $k \neq l$ zusammenfallen können. Ist die Behauptung des Satzes erst einmal etabliert, zeigt dann der Beweis von Lemma 4.10 auf Seite 72, daß dieser Fall gar nicht eintreten kann.

Wie dort gilt $s_{i,l} \neq s_{i,k}$ für $k < k-1$, denn:

$$s_{i,k} = s_{i,l} + d_{i,j}(s_{i,l}) + d(Q, s_{j,l+1}) > s_{i,l} \quad Q \text{ ein } j\text{–}i \text{ Pfad mit } l(Q) = k-l-1$$

aufgrund der Induktionsvoraussetzung. Weiterhin kann $s_{i,k-1} = s_{i,k}$ nur für eine Stadt i gelten: Annahme: $\exists i, j \in V_0, i \neq j : s_{i,k-1} = s_{i,k} \wedge s_{j,k-1} = s_{j,k}$ dann gilt:

$$\begin{aligned}s_{i,k} &= s_{j,k-1} + d_{j,i}(s_{j,k-1}) > s_{j,k-1} = s_{j,k} \\ \wedge s_{j,k} &= s_{i,k-1} + d_{i,j}(s_{i,k-1}) > s_{i,k-1} = s_{i,k}\end{aligned}$$

ein Widerspruch (siehe auch den dritten Fall des Beweises zu Lemma 4.10). Sei nun $Q \in \mathbb{P}_{i,k-1}$ ein beliebiger $k-1$–schrittiger Anfangspfad. Dann gilt aufgrund der Induktionsvoraussetzung und dem eben Gezeigten:

$$c(Q) = \sum_{l=0}^{k-2} c_{Q(l)Q(l+1)}(s_{Q(l),l}) = 0$$

Berechne jetzt alle relevanten Pfadkosten:

1. Fall: $s_{i,k} \neq s_{i,k-1} \forall i \in V_0$
Dann gilt $c(Q) = 0$ für alle $Q \in \mathbb{P}_{i,k}$ und alle $i \in V_0$.

$c(\mathbb{P}_{v,k+2})$	$\rightarrow uv$	$\rightarrow wv$	$\rightarrow xv$
$Q \rightarrow u$		20	20
$Q \rightarrow v$	**7**	20	20
$Q \rightarrow w$	10		20
$Q \rightarrow x$	20	20	

$c(\mathbb{P}_{u,k+1})$	$\rightarrow vu$	$\rightarrow wu$	$\rightarrow xu$
Q	7	**5**	10

Also gilt (4.27) mit $\overline{\mathbb{P}} = \emptyset$.

2. Fall: $s_{u,k} = s_{u,k-1}$
Dann gilt

$$\forall Q_u \in \mathbb{P}_{u,k-1}, i \neq u : c(Q_u \rightarrow i) = c_{u,i}(s_{u,k}) = 10$$
$$\forall Q \in \mathbb{P}_{j,k-1}, j \neq u, i \neq j : c(Q \rightarrow i) = c_{j,i}(s_{j,k-1}) = 0$$

$c(\mathbb{P}_{v,k+2})$	$\rightarrow uv$	$\rightarrow wv$	$\rightarrow xv$
$Q \rightarrow u$		20	20
$Q \rightarrow v$	**7**	20	20
$Q_u \rightarrow v$	17	30	30
$Q \rightarrow w$	10		20
$Q_u \rightarrow w$	20		30
$Q \rightarrow x$	20	20	
$Q_u \rightarrow x$	30	30	

$c(\mathbb{P}_{u,k+1})$	$\rightarrow vu$	$\rightarrow wu$	$\rightarrow xu$
Q	7	**5**	10
Q_u	17	15	20

Also gilt (4.27) mit $\overline{\mathbb{P}} = \mathbb{P}_{u,k-1}$.

3. Fall: $s_{v,k} = s_{v,k-1}$
Dann gilt $s = s_{u,k}$ und:

$$\forall Q_v \in \mathbb{P}_{v,k-1}, i \neq u : c(Q_v \rightarrow i) = c_{v,i}(s_{v,k}) = \begin{cases} 7 & i = u \\ 10 & i \neq u \end{cases}$$
$$\forall Q \in \mathbb{P}_{j,k-1}, j \neq v, i \neq j : c(Q \rightarrow i) = c_{j,i}(s_{j,k-1}) = 0$$

$c(\mathbb{P}_{v,k+2})$	$\rightarrow uv$	$\rightarrow wv$	$\rightarrow xv$
$Q \rightarrow u$		20	20
$Q_v \rightarrow u$		27	27
$Q \rightarrow v$	**7**	20	20
$Q \rightarrow w$	10		20
$Q_v \rightarrow w$	20		30
$Q \rightarrow x$	20	20	
$Q_v \rightarrow x$	30	30	

$c(\mathbb{P}_{u,k+1})$	$\rightarrow vu$	$\rightarrow wu$	$\rightarrow xu$
Q	7	**5**	10
Q_v		12	17

Also gilt (4.27) mit $\overline{\mathbb{P}} = \mathbb{P}_{v,k-1}$.

4. Fall: $s_{w,k} = s_{w,k-1}$ Identisch zum 3. Fall, wenn dort v mit w und s mit s' vertauscht wird.

5. Fall: $s_{x,k} = s_{x,k-1}$

Dann gilt

$$\forall Q_x \in \mathbb{P}_{x,k-1}, i \neq x : c(Q_x \to i) = c_{x,i}(s_{x,k}) = 10$$
$$\forall Q \in \mathbb{P}_{j,k-1}, j \neq x, i \neq j : c_{Q \to i} = c_{j,i}(s_{j,k-1}) = 0$$

$c(\mathbb{P}_{v,k+2})$	$\to uv$	$\to wv$	$\to xv$
$Q \to u$		20	20
$Q_x \to u$		30	30
$Q \to v$	**7**	20	20
$Q_x \to v$	27	30	30
$Q \to w$	10		20
$Q_x \to w$	20		30
$Q \to x$	20	20	

$c(\mathbb{P}_{u,k+1})$	$\to vu$	$\to wu$	$\to xu$
Q	7	**5**	10
Q_x	17	15	

Also gilt (4.27) mit $\overline{\mathbb{P}} = \mathbb{P}_{x,k-1}$.

■

Bemerkung: Tatsächlich ist etwas mehr bewiesen worden, nämlich daß schon wenn die DTSP–Instanzen $\mathcal{I}_c$ mit Kostenfunktionen $c_{i,j}(t)$, die nur Werte $0, a, b, c$ mit $a > b > c > 0$ annehmen, alle die Fortsetzungseigenschaft erfüllen, in allen zeitexpandierten Graphen G^T mit beliebigem Horizont die Zusammenhangskomponente, die $(v_0, 0)$ enthält, $T+1$–partit ist.

Wie dieser Abschnitt gezeigt hat, stellt also die Klasse von DTSP–Instanzen, für die man ohne Ansehen der Wegekosten die vereinfachte Wegeberechnung mittels (4.25), (4.26) anwenden kann, keine Erweiterung der bereits bekannten Spezialfälle dar.

4.4.3 λ–stabile DTSP–Instanzen

Um neue DTSP–Instanzen mit Fortsetzungseigenschaft zu finden, reicht es also nichgt aus, nur die Wegezeiten zu betrachten. Vielleicht lassen sich ja Klassen von Kostenfunktionen identifizieren, die die Fortsetzungseigenschaft sichern. Das vorgestellte Lösungsverfahren vergibt Strafkosten λ auf den Knoten für mehrfachen Besuch. Diese ändern sich in jedem Schritt des Aufstiegsverfahrens und damit ändern sich auch die Kantenkosten des Graphen, in dem die billigsten Wege gesucht werden. Also müssen die gesuchten Klassen abgeschlossen sein gegenüber der Addition von Strafkosten auf den Knoten.

Definition 4.8
Eine DTSP–Instanz heißt λ–*stabil bezüglich der Fortsetzungseigenschaft*, falls sie für alle Strafkosten $\lambda \in \mathbb{R}^{V_0}$ die Fortsetzungseigenschaft erfüllt. ■

Das einzige bis jetzt bekannte Beispiel für λ–Stabilität sind also die partitionierten Instanzen. Es gibt aber Hoffnung:

Satz 4.14
Es gibt eine nicht-partitionierte DTSP–Instanz, die λ–stabil bezüglich der Fortsetzungseigenschaft ist.

Beweis:

Beweis durch Konstruktion. Betrachte die durch Abbildung 4.3 auf der anderen Seite gegebene DTSP–Instanz mit $V = \{x, a, b, c\}$ und $v_0 = x$. In diesem Beweis werden die Pfade einfach als Worte über V geschrieben, zum Beispiel ist $xabc$ der Pfad $x \to a \to b \to c$. Wegen $d(xba) = 2 \neq 3 = d(xca)$ ist die Instanz nicht partitioniert. Überprüfe die Fortsetzungseigenschaft für alle Werte der Strafkosten λ_a, λ_b und λ_c: Für $k = 0, 1$ braucht sie nicht überprüft werden, da $|\mathbb{P}_{i,k}| = 1$ ist und für $k = 2$ auch nicht, weil wegen $|\mathbb{P}_{i,k-1}| = 1$ alle kürzesten Pfade trivialerweise fortgesetzt sind. Betrachte also $\mathbb{P}_{i,3}$, hier die Tabelle der Ankunftszeiten:

$\mathbb{P}_{\cdot,2}$	·	→a	→b	→c
xba	2		4	4
xca	3		4	4
xab	2	3		4
xcb	2	3		4
xac	2	3	4	
xbc	2	3	4	

Insbesondere ist also $\mathbb{P}^{\star\star}_{i,3} = \mathbb{P}^{\star}_{i,3}$ für $i = a, b, c$ und $\mathbb{P}^{\star\star}_{i,2} = \mathbb{P}^{\star}_{i,2}$ für $i = b, c$. Bleibt also zu prüfen, ob es immer einen Pfad aus $\mathbb{P}^{\star\star}_{\cdot,3}$ gibt, der fortgesetzt ist. Ist zum Beispiel $\mathbb{P}^{\star\star}_{a,3} = \{xaba\}$, so würde die Fortsetzungseigenschaft genau dann verletzt, falls $xab \notin \mathbb{P}^{\star\star}_{b,2}$. In diesem Fall lautet die Bedingung für die Verletzung also:

$$\exists\lambda : \mathbb{P}^{\star\star}_{a,3} = \{xaba\} \wedge xab \notin \mathbb{P}^{\star\star}_{b,2} \tag{4.28}$$

$$\iff \quad \exists\lambda : \begin{aligned} c_\lambda(xaba) &< c_\lambda(xcba) \\ c_\lambda(xaba) &< c_\lambda(xaca) \\ c_\lambda(xaba) &< c_\lambda(xbca) \\ c_\lambda(xcb) &< c_\lambda(xab) \end{aligned} \tag{4.29}$$

$$\iff \quad \begin{aligned} 0 &< \max s \qquad \text{so daß} \\ c_\lambda(xaba) + s &\leq c_\lambda(xcba) \\ c_\lambda(xaba) + s &\leq c_\lambda(xaca) \\ c_\lambda(xaba) + s &\leq c_\lambda(xbca) \\ c_\lambda(xcb) + s &\leq c_\lambda(xab) \\ s &\geq 0 \end{aligned} \tag{4.30}$$

wobei $c_\lambda(P)$ die Pfadkostenfunktion bei Strafkosten λ ist. Da $c_\lambda(P)$ linear in λ ist, stellt (4.30) eine Lineares Programm dar, das mittels der Standard–Verfahren (Simplex) gelöst werden kann. Auf diese Weise erzeuge nun eine Fallunterscheidung,

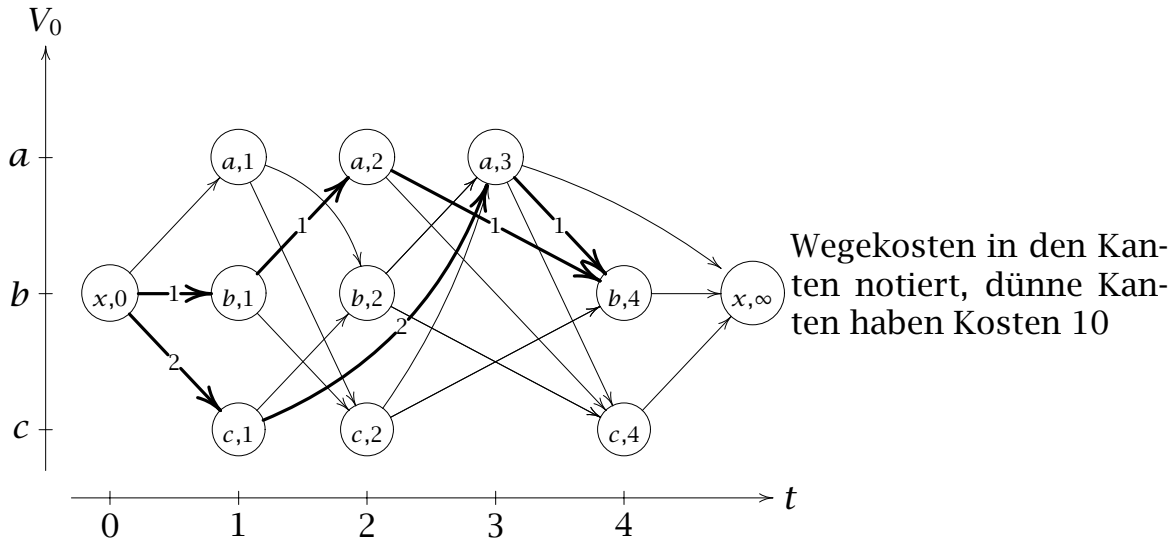


Abbildung 4.3: Das λ–stabile DTSP zu Satz 4.14

die alle möglichen Verletzungen der Fortsetzungseigenschaft erfaßt. Für den Fall $\mathbb{P}^{\star\star}_{a,3} = \{xaba, xaca\}$ liegt eine Verletzung genau dann vor, falls $xab \notin \mathbb{P}^{\star\star}_{b,2}$ und $xac \notin \mathbb{P}^{\star\star}_{c,2}$. Dies übersetzt sich in:

$$\exists \lambda : \mathbb{P}^{\star\star}_{a,3} = \{xaba, xaca\} \wedge xab \notin \mathbb{P}^{\star\star}_{b,2} \wedge xac \notin \mathbb{P}^{\star\star}_{c,2} \tag{4.31}$$

$$\iff \exists \lambda : \begin{aligned} c_\lambda(xaba) &= c_\lambda(xaca) \\ c_\lambda(xaba) &< c_\lambda(xbca) \\ c_\lambda(xaba) &< c_\lambda(xcba) \\ c_\lambda(xcb) &< c_\lambda(xab) \\ c_\lambda(xbc) &< c_\lambda(xac) \end{aligned} \tag{4.32}$$

Auch (4.32) läßt sich analog zu (4.29) als LP lösen. Betrachte als nächstes den Fall $\{xaba, xcba\} \subseteq \mathbb{P}^{\star\star}_{a,3}$. In diesem Fall kann keine Verletzung auftreten, da alle Pfade aus $\mathbb{P}^{\star\star}_{2,b}$ fortgesetzt in $\mathbb{P}^{\star\star}_{a,3}$ vorkommen. Neben den zwei bisher betrachteten Typen von Verletzungsbedingungen gibt es noch einen dritten. Im Fall $\mathbb{P}^{\star\star}_{a,3} = \{xcab\}$ liegt genau dann eine Verletzung vor, falls $c_\lambda(xba) \leq c_\lambda(xca)$ ist. In diesem Fall ist wegen $d(xba) < d(xca)$ die nicht–strikte Ungleichung hinreichend. Insgesamt ergeben sich die folgenden 24 Fälle, die Tabelle 4.1 auflistet.
Alle entsprechenden LPs wurden mit Hilfe eines Rechners aufgestellt und mittels AMPL [20] und CPLEX [13] gelöst. In allen Fällen war das LP entweder unzulässig oder die Optimallösung war 0. ■

Die Methode im Beweis von Satz 4.14 läßt sich natürlich auf beliebige DTSPs erweitern. Dabei müssen dann exponentiell viele lineare Programme gelöst werden. Auf diese Weise wurden auch weitere λ–stabile Instanzen identifiziert

$\mathbb{P}^{\star\star}_{\cdot,3} = \{\cdots\}$	Verletzung falls
$xaba$	$c_\lambda(xcb) < c_\lambda(xab)$
$xcba$	$c_\lambda(xab) < c_\lambda(xcb)$
$xaca$	$c_\lambda(xbc) < c_\lambda(xac)$
$xbca$	$c_\lambda(xac) < c_\lambda(xbc)$
$xaba, xaca$	$c_\lambda(xcb) < c_\lambda(xab) \wedge c_\lambda(xbc) < c_\lambda(xac)$
$xaba, xbca$	$c_\lambda(xcb) < c_\lambda(xab) \wedge c_\lambda(xac) < c_\lambda(xbc)$
$xcba, xaca$	$c_\lambda(xab) < c_\lambda(xcb) \wedge c_\lambda(xbc) < c_\lambda(xac)$
$xcba, xbca$	$c_\lambda(xab) < c_\lambda(xcb) \wedge c_\lambda(xac) < c_\lambda(xbc)$
$xbcb$	$c_\lambda(xac) < c_\lambda(xbc)$
$xacb$	$c_\lambda(xbc) < c_\lambda(xac)$
$xbab$	$c_\lambda(xca) < c_\lambda(xba)$
$xcab$	$c_\lambda(xba) \leq c_\lambda(xca)$
$xbcb, xbab$	$c_\lambda(xac) < c_\lambda(xbc) \wedge c_\lambda(xca) < c_\lambda(xba)$
$xbcb, xcab$	$c_\lambda(xac) < c_\lambda(xbc) \wedge c_\lambda(xba) \leq c_\lambda(xca)$
$xacb, xbab$	$c_\lambda(xbc) < c_\lambda(xac) \wedge c_\lambda(xca) < c_\lambda(xba)$
$xacb, xcab$	$c_\lambda(xbc) < c_\lambda(xac) \wedge c_\lambda(xba) \leq c_\lambda(xca)$
$xcac$	$c_\lambda(xba) \leq c_\lambda(xca)$
$xbac$	$c_\lambda(xca) < c_\lambda(xba)$
$xcbc$	$c_\lambda(xab) < c_\lambda(xcb)$
$xabc$	$c_\lambda(xcb) < c_\lambda(xab)$
$xcac, xcbc$	$c_\lambda(xba) \leq c_\lambda(xca) \wedge c_\lambda(xab) < c_\lambda(xcb)$
$xcac, xabc$	$c_\lambda(xba) \leq c_\lambda(xca) \wedge c_\lambda(xcb) < c_\lambda(xab)$
$xbac, xcbc$	$c_\lambda(xca) < c_\lambda(xba) \wedge c_\lambda(xab) < c_\lambda(xcb)$
$xbac, xabc$	$c_\lambda(xca) < c_\lambda(xba) \wedge c_\lambda(xcb) < c_\lambda(xab)$

Tabelle 4.1: Alle zu untersuchenden Fälle aus dem Beweis zu Satz 4.14

(zum Beispiel bleibt die untersuchte Instanz λ–stabil, wenn auf alle dünn gezogene Kanten in Abbildung 4.3 beliebige, aber gleiche Kosten gelegt werden). Bis jetzt ist keine polynomial überprüfbare Bedingung für λ–Stabilität bekannt.

Kapitel 5

Implementation und numerische Ergebnisse

Mit den Ergebnissen aus Kapitel 4, insbesondere Abschnitte 4.3 und 4.3.2, wurde das Programm `solve` zur exakten Lösung des DTSP implementiert. Wie Lucena's Methode benutzt es Branch&Bound auf Basis der beschriebenen n–Pfad Relaxation, mit verschärften Schranken durch die Optimierung der Lagrange–Faktoren und der Ausnutzung von through–circuits. Die Programmabschnitte von `solve` werden durch Algorithmus 5.1 auf der nächsten Seite zusammengefaßt dargestellt. Die Kommentare am rechten Rand verweisen auf die folgenden Abschnitte, in denen die Ausgestaltung der mathematischen Algorithmen beschrieben wird. Abschnitt 5.5 auf Seite 93 stellt die numerischen Ergebnisse vor und Abschnitt 5.6 auf Seite 97 weißt auf, wo sich `solve` noch verbessern ließe.

5.1 Initialisierung

Wie bereits in Abschnitt 1.2 auf Seite 4 erklärt, gibt es keine kanonische Eingabekodierung für DTSP–Instanzen. Das Programm `solve` erwartet Wegezeit– und Kostenfunktionen als stückweise lineare Funktionen mit beliebig vielen Knickpunkten, ließe sich aber einfach auf andere Darstellungen erweitern (Polynomfunktionen, Wertetabellen usw.). Als erster Schritt wird mittels Algorithmus 4.1 auf Seite 59 der zeitexpandierte Graph $G^{\mathbb{P}_n}$ berechnet. Dabei werden alle vorkommenden Fließkomma–Werte, das heißt Wegekosten und –zeiten, gerundet. Durch Multiplikation der Wegezeit– und Kostenfunktionen mit einem genügend großen Faktor läßt sich eine höhere Genauigkeit erreichen.

Algorithmus 5.1 Der Algorithmus von `solve`

input : $\mathcal{D} = (V, E, d, c, n, v_0)$ DTSP–Instanz
output : $\pi^\star$ Tour minimaler Kosten
  Erzeuge $G^{\mathbb{P}_n}$ mittels Algorithmus 4.1 (siehe 5.1)
  Konstruiere aktuelle Tour π (siehe 5.2)
  Branch&Bound: (siehe 5.3)
    Mache $G^{\mathbb{P}_n}$ zur Wurzel des B&B-Baumes $\mathbb{B}$
    while $\mathbb{B}$ hat offene Knoten **do**
      Wähle offenen Knoten P (siehe 5.3.1)
      if P dominiert **then next fi**
      Berechne untere Schranke L_P für P: (siehe 5.3.2)
        Berechne λ aus $\lambda^\star$ des Vaterknotens
        do
          Berechne $\Psi_P(\lambda)$ mittels Algorithmus 4.2
          Berechne $L_P(\lambda)$ wie in Abschnitt 3.2.3
          Verbessere π (siehe 5.2)
          Bestimme Aufstiegsrichtung $\Delta\lambda$ (siehe 5.3.3)
          $\lambda := \lambda + \Delta\lambda$
        until λ optimal (siehe 5.3.3)
      $\lambda^\star := \lambda$; $L_P := L_P(\lambda^\star)$
      Schließe Knoten P
      if $L_P < z_{\mathrm{DTSP}}(\pi)$
        then Separiere P in die Söhne von P in $\mathbb{B}$ (siehe 5.3.1)
      fi
    od
  $\pi^\star := \pi$

Im Rahmen der Machbarkeitsuntersuchung zur Implementation von `solve` wurde der $G^{\mathbb{P}_n}$ zu 450 zufälligen Beispielen vom Typ Q_2 (siehe 5.4 auf Seite 92) mit zwischen sieben und zwanzig Städten berechnet. Bei bis zu acht Städten kamen Seitenlängen zwischen 100 und 2000 vor und zwischen 100 und 700 bei mehr als acht bis zu zwanzig Städten. In Abbildung 5.1 auf der gegenüberliegenden Seite sind die über der Anzahl n der Städte in V_0 die Anzahl $|V^{\mathbb{P}_n}|$ der Knoten von $G^{\mathbb{P}_n}$ aufgetragen und zusätzlich zum Vergleich die Gesamtzahl an Pfaden, die ja nach Satz 4.2 auf Seite 60 die obere Schranke für $|V^{\mathbb{P}_n}|$ darstellt (beachte die logarithmische Skalierung der Abszisse). Man erkennt eine höchstens lineare Steigerung, obwohl $|V^{\mathbb{P}_n}|$ für alle Werte von n eine ganze Größenordnung überstreicht. Der Grund dafür liegt in der Rundung der Wegezeiten. Ist T der Horizont des berechneten $G^{\mathbb{P}_n}$, das heißt die maximale Dauer eines $n+1$–schrittigen Pfades, so können maximal nT Knoten erzeugt

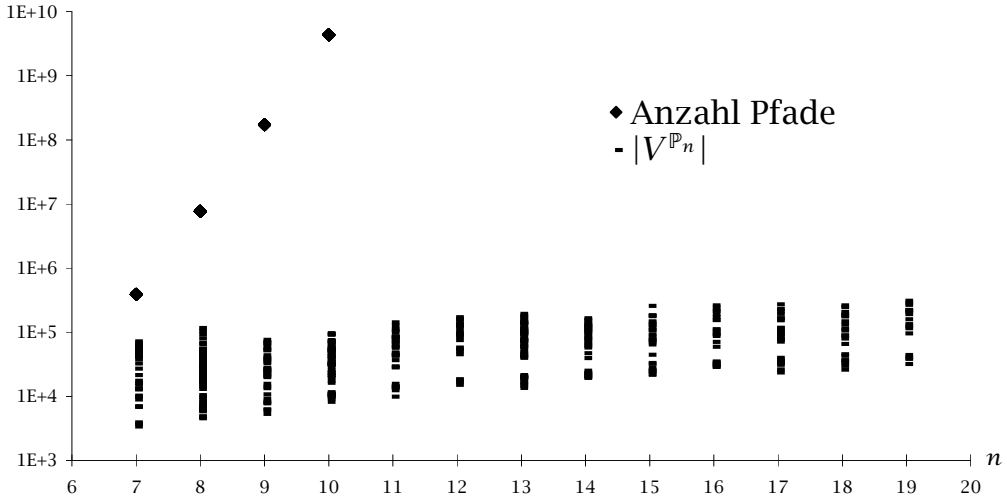


Abbildung 5.1: Anzahl an zeitexpandierten Knoten der Testbeispiele

werden, da nicht mehr ganze Zahlen im Intervall $[1, T]$ zur Verfügung stehen und pro Zeitpunkt zu nicht mehr als n Städten zeitexpandierte Knoten geben kann. In Abbildung 5.2 auf der folgenden Seite sind für die Beispiele die Punkte $(nT, |V^{\mathbb{P}_n}|)$ eingetragen, das heißt die Anzahl Städte über dem n–fachen des Horizonts des jeweiligen $G^{\mathbb{P}_n}$. Es zeigt sich, daß $|V^{\mathbb{P}_n}|$ linear von nT abhängt mit einer statischen Korrelation von 97%. Bei den untersuchten Beispielen bewirkt also die endlich genaue Berechnung, daß der $G^{\mathbb{P}_n}$ praktisch ein G^T ist mit zwischen 75% und 90% der Größe. Um also einen exakten $G^{\mathbb{P}_n}$ zu berechnen, müßte bei endlicher Genauigkeit der Horizont also exponentiell in n steigen. Anders betrachtet engt also die Bedingung $d_{i,j}(t) \in \mathbb{N}$ in Definition 3.2 auf Seite 43 die Anzahl an möglichen zeitlichen Strukturen stark ein. Die Berechnung der Beispiele benötigte über 17 CPU–Stunden (siehe auch Abschnitt 5.5 auf Seite 93) und die größten Datenstrukturen belegten mehr als 100MB Hauptspeicher.

5.2 Konstruktionsheuristiken für obere Schranken

In `solve` kommen fünf Methoden zur Erlangung von oberen Schranken, das heißt zur Konstruktion von Touren, zum Einsatz:

- Optional kann zu Beginn des Programms die beste von mehreren zufälligen Touren bestimmt werden. Dies ist aber für $n > 6$ ineffektiv.

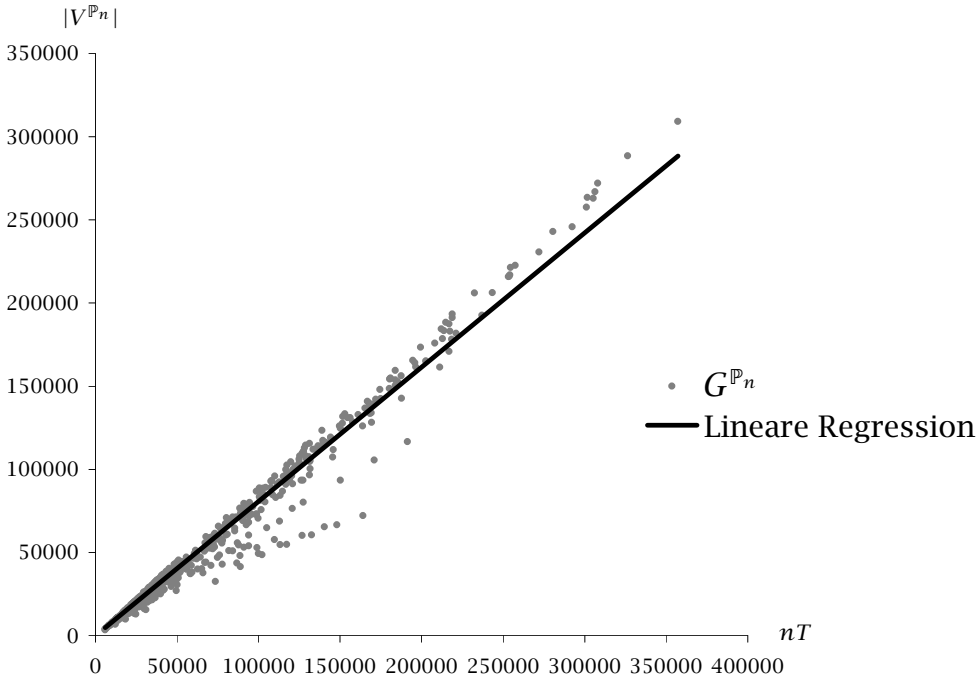


Abbildung 5.2: Abhängigkeit von Knotenzahl und Horizont

- Sowohl zu Beginn, als auch nach jedem Schritt des Aufstiegsverfahrens wird die forward–greedy Tour im aktuellen Problem bestimmt.

  Die *forward–greedy–tour* entsteht, indem in $(v_0, 0)$ beginnend immer die billigste Kante zu einer noch nicht besuchten Stadt verwendet wird. Liegen aus der Berechnung von Ψ die Kosten der billigsten k–schrittigen Endpfade vor, so wird als nächste Stadt die gewählt, deren Endpfad am billigsten ist. Da die Tour im zeitexpandierten Graphen gesucht wird, kann es vorkommen, daß aufgrund von Variablenfixierung (Abschnitt 5.3.2 auf Seite 86) keine forward–greedy Tour mehr existiert. In diesem Fall wäre sie dann aber auch teurer als die momentane obere Schranke gewesen. So zu verfahren, hat den Vorteil, daß Information aus der Schrankenberechnung genutzt wird. Experimente legen nahe, daß dadurch die Effizienz deutlich über die eines nicht–restringierten, kantenorientierten Greedy–Algorithmus steigt.

- Die *backward-greedy-tour* wird analog zur forward–greedy–tour durch Suche im zeitexpandierten Graphen von (v_0, ∞) aus rückwärts bestimmt.

- Die optimale Lösung des Zuordnungsproblems zu $L_P(\lambda)$ wird wie von Lucena vorgeschlagen als Tour ausgewertet. Dieses Verfahren ist für Probleme vom Typ D_2 effektiver als bei den Typen Q_1 und Q_2 (siehe unten).
- Die bei der Berechnung von $\Delta\lambda$ benötigten through–circuits werden daraufhin geprüft, ob es sich um through–touren handelt (siehe auch Abschnitt 5.3.2 auf der nächsten Seite).

Die folgenden Tabelle dokumentiert die relative Effektivität der Tour–Konstruktionen. Sie zeigt den Anteil der verschiedenen Heuristiken an der Auffindung verbesserter oberer Schranken bei der Berechnung der Beispiele in Abschnitt 5.5 auf Seite 93:

Typ	greedy–tour	through–tour	Zuordnung
Q_1, Q_2	68%	27%	5%
D_2	52%	35%	13%

`solve` verwendet keine Verbesserungs–Heuristiken (wie zum Beispiel in [17,64]), ihr Einsatz würde aber vermutlich die Leistungsfähigkeit des Verfahrens deutlich steigern. Insgesamt liegt der Rechenaufwand zur Konstruktionvon Touren bis jetzt unter 1% der Gesamtrechenzeit.

5.3 Branch&Bound

Um mittels der beschriebenen n–Pfad–Relaxation exakte Lösungen zu gewinnen, verwendet `solve` die Branch&Bound–Methode. Eine Beschreibung der generellen Technik findet sich zum Beispiel in [3, 41, 54]. Intern wird der B&B–Baum durch einen Graphen dargestellt, so daß leicht beliebige Verzweigungs– und Auswahlregeln implementiert werden können.

5.3.1 Verzweigungs– und Auswahlregel und Dominanztest

Das Teilproblem eines Knoten auf Stufe p ist dadurch definiert, daß der p–schrittige Anfangspfad der Tour bereits festgelegt ist (v_0 an der Wurzel). Für jede vom zeitexpandierten Ende (v, t) des Pfades erreichbare, noch nicht besuchten Stadt kann ein Sohn–Knoten erzeugt werden, indem diese Stadt an den

Anfangspfad angehängt wird. Konnten bereits Kanten, die von (v,t) ausgehen, eliminiert werden (Variablenfixierung, siehe Abschnitt 5.3.2), so wird dadurch ein ganzer Zweig des Baumes gar nicht erst erzeugt. Die Teilprobleme werden durch Entfernen der entsprechenden Kanten aus dem zeitexpandierten Graphen erzeugt. Dies geschieht so, daß immer nur eine Kopie des $G^{\mathbb{P}_n}$ existiert, da Kopieren des benötigten Teilgraphen ausgesprochen rechenzeit- und speicheraufwendig wäre. Zu diesem Zweck verwaltet `solve` an jedem Knoten eine Liste der entfernten Kanten.

Zur Auswahl des nächsten zu untersuchenden Knoten kennt `solve` die folgenden Regeln:

- **breadth–first**: Es wird einer der offenen Knoten mit kürzestem Anfangspfad gewählt.
- **depth–first–loosest**: Unter den offenen Knoten mit längstem Anfangspfad wird der mit den geringsten Kosten gewählt.
- **depth–first–tightest**: Unter den offenen Knoten mit längstem Anfangspfad wird der mit den höchsten Kosten gewählt.

In der Praxis scheint sich „depth–first–loosest“ am besten zu bewähren, da diese Regel am schnellsten zu verbesserten oberen Schranken führt und dadurch hilft, den B&B–Baum früh zu beschneiden. Die Breitensuche („breadth–first“) resultiert bereits bei mittlerer Problemgröße (12–15 Städte) in sehr hohem Hauptspeicherverbrauch.

Vor der Berechnung der unteren Schranke an einem Knoten P wird überprüft, ob schon einmal ein Knoten P', dessen Anfangspfad sich nur in der Reihenfolge der durchlaufenen Städte unterscheidet, untersucht wurde. Ist dies der Fall und der Anfangspfad von P ist nicht billiger als der von P', wird P sofort geschlossen. Andernfalls werden die unter P bereits fixierten Variablen und die optimalen Lagrange–Faktoren übernommen.

5.3.2 Untere Schranken,Through–circuits und Variablenfixierung

Den Kern von `solve` bilden die beiden Routinen zur Berechnung von $c^\star_{ik}$ und $b^\star_{ik}$. Sie führen Algorithmus 4.2 auf Seite 67 einmal vorwärts zur Bestimmung der

billigsten k–schrittigen Anfangspfade $P(v,t,k)$ und einmal rückwärts zur Bestimmung der billigsten k–schrittigen Endpfade $O(v,t,k)$ durch. Bereits während der Berechnung werden Kanten eliminiert, die nicht zur Optimaltour $\pi^\star$ gehören können. Erfüllt eine Kante $e = ((v,s),(w,t))$ die Bedingung

$$\min_k c^*_{v,k} + c_{v,w}(s) + b^*_{w,n-k} > z_{\mathrm{DTSP}}(\pi) \tag{5.1}$$

so wird $x_e = 0$ fixiert, wobei π die zu der Zeit beste bekannte

e läuft, hat mindestens die Kosten $c_e = c_{v,w}(s)$ der Kante e zuzüglich der billigsten daran anschließenden Anfangs– und Endpfade mit geeigneter Schrittzahl. Dementsprechend ist eine Verletzung von (4.1) auf Seite 56 ausgeschlossen. In [6] wird ein ähnliches Kriterium zur Reduzierung des Zustandsraums einer DP–Methode verwendet. Da `solve` wie in der Bemerkung in Abschnitt 3.2.2 auf Seite 46 ausgeführt nur Lagrange–Faktoren λ mit $\sum_{v\in V_0}\lambda_v = 0$ verwendet (siehe Abschnitt 5.3.3), brauchen die Faktoren nicht explizit berücksichtigt werden. Das Fixieren einer Variablen x_e auf Null geschieht durch Entfernen der Kante e aus dem zeitexpandierten Graphen. Dadurch wird auch ganz natürlich erreicht, daß die Kante im gesamten nachfolgenden Zweig des B&B–Baumes fehlt.

Der $G^{\mathbb{P}_n}$ enthält für die berechneten Testbeispiele mit bis zu zwanzig Städten üblicherweise mehrere hunderttausend bis zu einer Million Kanten. Die Durchführung von `solve` entspricht nun dem Lösen eines ganzzahligen Linearen Programms mit genauso vielen Variablen. Deshalb wird die Effizienz des Verfahrend entscheidend von der Anzahl der fixierten Variablen beeinflußt (siehe auch Abschnitt 5.6 auf Seite 97). `solve` eliminiert üblicherweise bereits an der Wurzel des B&B–Baumes etwa 80% aller Kanten.

Aus den Ergebnissen der Pfadberechnungen bestimmt `solve` nach (4.24) auf Seite 68 die Matrix der through–circuit Kosten Ψ. Aus dieser Matrix wird dann durch Lösen einer Reihe von Zuordnungproblemen (3.19) wie in Abschnitt 3.2.3 auf Seite 48 die untere Schranke $L_P(\lambda)$ zum Knoten P berechnet.

5.3.3 Bestimmung der Aufstiegsrichtung

Bereits in Abschnitt 3.2.4 auf Seite 50 wurde Lucenas Vorgehensweise zur Optimierung der Strafkosten λ trotz der auffälligen Ähnlichkeit nicht als Subgradienten–Verfahren bezeichnet, da die maximierte Funktion weder konkav ist, noch die verwendeten Aufstiegsrichtungen Subgradienten sind. Wie schon

in Bemerkung 3.2.3 auf Seite 48 dargelegt, kann man die through–circuit Kosten $\psi_{ik} = \psi_{ik}(\lambda)$ als Lösung eines Lagrange–relaxierten IP–Problems nach Abschnitt 3.1 auf Seite 39 betrachten. Auch im Fall eines allgemeinen zeitexpandierten Graphen ist es möglich, ψ_{ik} durch eine IP–Formulierung auszudrücken. Allerdings reicht es nicht, einfach einige Kanten wegzulassen wie im TDTSP–Fall, sondern es müssen zusätzliche Variablen eingeführt werden, die die Bedingung „*i steht an k–ter Stelle im Pfad*“ erzwingen. Dadurch ist die Formulierung nicht mehr total unimodular und die Integralitätseigenschaft gilt nicht mehr. An dieser Stelle sei darauf hingewiesen, daß dies in allgemeinen, nicht–partitionierten Graphen auch bereits für die Bestimmung des billigsten $n+1$–schrittigen $(v_0, 0) \to (v_0, \infty)$–Pfades gilt. Ein solcher Graph enthält dann nämlich durchaus Pfade mit mehr als n Schritten, so daß die Schrittzahl nicht mehr automatisch sicherstellt ist. Das durch Algorithmus 4.2 gelöste Problem lautet als IP also:

$$
\begin{aligned}
& \min \sum_{e \in E^T} c_e x_e + \sum_{v \in V_0} \lambda_v \Big(\sum_{e \in \vec{\delta}_{G^T}(v)} x_e - 1 \Big) \\
\text{s.d.} \quad & A_{E^T} x = (-1, 0, \ldots, 0, 1)^t \\
& \sum_{e \in E} x_e = n + 1 \\
& x_e \in \{0, 1\} \qquad\qquad e \in E^T
\end{aligned}
\tag{5.2}
$$

Im Vergleich zu (3.10) auf Seite 46 zerstört die letzte Nebenbedingung in (3.10') die Integralitätseigenschaft.

Somit sind also $\psi_{ik}(\lambda)$ und damit auch

$$\overline{\psi}_i(\lambda) := \min_{k=1,\ldots,n} \psi_{ik}(\lambda)$$

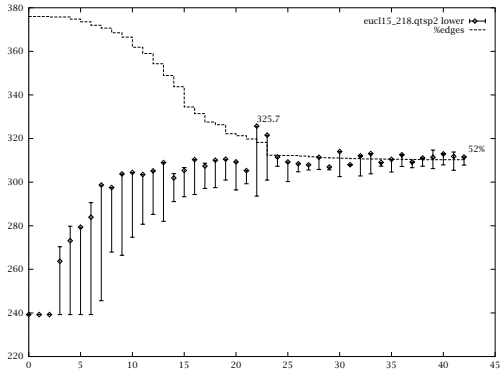
(vergleiche (3.14) auf Seite 49) stückweise linear und konkav in λ, nicht mehr jedoch

$$L(0, \lambda) := \max_{i \in V_0} \overline{\psi}_i$$

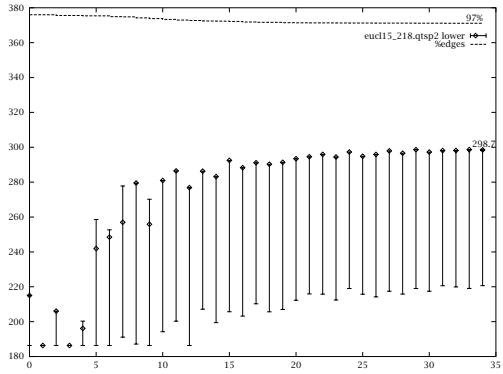
$L(0, \lambda)$ ist also die Lösung des folgenden Optimierungsproblems:

$$
\begin{aligned}
& L(0, \lambda) := \min w \\
& \quad \text{s.d.} \qquad w \geq \overline{\psi}_i(\lambda) \qquad i \in V_0
\end{aligned}
\tag{5.3}
$$

Da die $\overline{\psi}_i(\lambda)$ stückweise linear und konkav sind, handelt es sich dabei um ein LP mit sehr vielen Nebenbedingungen, nämlich für jeden through–circuit eine.



Lucenas Aufstiegsrichtung



Richtung eines bindenden ψ_{ik}

Abbildung 5.3: Verlauf der Schrankenberechnung

Eine geschlossene Darstellung für $L(\infty, \lambda)$, die Lösung der von Lucena vorgeschlagenen Reihe an Zuordnungsproblemen, ist nicht bekannt. Im folgenden werden zwei Regeln zur Bestimmung einer Aufstiegsrichtung vorgestellt.

Regel 1: Lucenas Aufstiegsrichtung

Ist x^{ik} der Inzidenzvektor eines i,k–through–circuit zu den Lagrange–Faktoren λ, so stellt nach Abschnitt 3.1 die Verletzung der Tour–Bedingung $A_{\delta_{GT}} x^{ik} - \mathbf{1}$ einen Subgradienten in λ an $\psi_{ik}(\lambda)$ dar. Lucena mittelt jetzt einfach diese Subgradienten für die i,k, für die im letzten gelösten Zuordnungsproblem $y_{ik} = 1$ galt. In [46] wird keine Begründung für diese Regel gegeben. Bei Testläufen von `solve` mit dieser Aufstiegsrichtung kam es mehrfach vor, daß $\Delta\lambda$ verschwand und die Iteration abgebrochen werden mußte, obwohl offensichtlich noch nicht das Maximum erreicht worden war.

Regel 2: Bindender through–circuit

Formulierung (5.3) legt es nahe, als Verbesserungsrichtung die durch eine aktive Nebenbedingung gegebene Richtung zu verwenden, das heißt einen Subgradienten an ein $\overline{\psi}_i$ mit $\overline{\psi}_i = L(\infty, \lambda)$. Der Subgradient an ein Minimum einer Familie von Funktionen ist der Schnitt der Subgradienten der an dieser Stelle minimierenden Funktionen. Zur Bestimmung des gesamten Subgradienten an $\psi_{ik}(\lambda)$ müssen aber *alle* i,k–through–circuits berechnet werden – dies wäre sehr aufwendig. Statt dessen wählt `solve` unter den bekannten Subgradienten der bindenden ψ_{ik} den, dessen Richtung am wenigsten von der vorhergehenden abweicht. Bereits zur Berechnung dieser vereinfachten Regel benötigt `solve` bis zu 5% der Gesamtausführungszeit.

Abbildung 5.3 auf der vorhergehenden Seite stellt den typische Verlauf einer Schrankenberechnung unter den beiden Aufstiegsrichtung gegenüber, in diesem Fall für ein Beispiel vom Typ Q_2 mit 15 Städten. Über dem Iterationsschritt auf der Ordinate sind als Balken die unteren Schranken aufgetragen. Das untere Ende jedes Balkens sind die Kosten des billigsten $n+1$–schrittigen $(v_0, 0) \to (v_0, \infty)$ Pfades, das obere Ende das Ergebnis der Zuordnungsprobleme, also $L(\infty, \lambda)$. Die Markierung innerhalb der Balken zeigt die Höhe von $L(0, \lambda)$. Am linken Rand sind zur besseren Skalierung einige der Balken unten abgeschnitten. Die gestrichelte Linie schließlich gibt die Anzahl der noch im Graphen verbliebenen Kanten an, in Prozent der ursprünglichen Anzahl und ist somit ein Maß für die Effektivität der Variablenfixierung.

Die Diagramme illustrieren die folgenden, vergleichenden Aussagen, die durch die Beobachtung vieler Beispiele gewonnen wurden:

Regel 1	**Regel 2**
Das Maximum wird lange vor Ende der Iteration erreicht	Die untere Schranke steigt fast monoton und erreicht erst am Ende ihr Maximum
Abbruch erfolgt wegen zu vieler, nicht–verbessernder Schritte.	Abbruch, weil Konvergenzkriterium erfüllt
Der Abstand zwischen dem billigstem $n+1$–Pfad und $L(\infty, \lambda)$ wird minimiert.	Beide Werte steigen ungefähr gleich
Die Schrittweite muß stark begrenzt werden, um Divergenz zu verhindern	Gleichmäßige Reduzierung der Schrittweite möglich
Viele Kanten werden eliminiert	Wenige Kanten werden eliminiert

Bemerkung: Im Gegensatz zu Lucena schreibt `solve` die Einträge der Matrix Ψ nicht zwischen zwei λ–Schritten fort. Die Fortschreibung sichert zwar monoton steigende untere Schranken, allerdings scheint sie das Maximum nicht zu verbessern. Die scheinbar monotone Konvergenz macht es schwerer, geeignete Abbruchkriterien zu etablieren. Außerdem ist unklar, welcher Subgradient zur Berechnung der Aufstiegsrichtung herangezogen werden soll: Betrachte den Fall, in dem ein ψ_{ik} mit $y_{ik} = 1$ aus einem früheren Iterationsschritt, etwa zu λ', entstammt. Der Subgradient an den i,k–through–circuit zum aktuellen λ, hat wenig mit dem Wert von

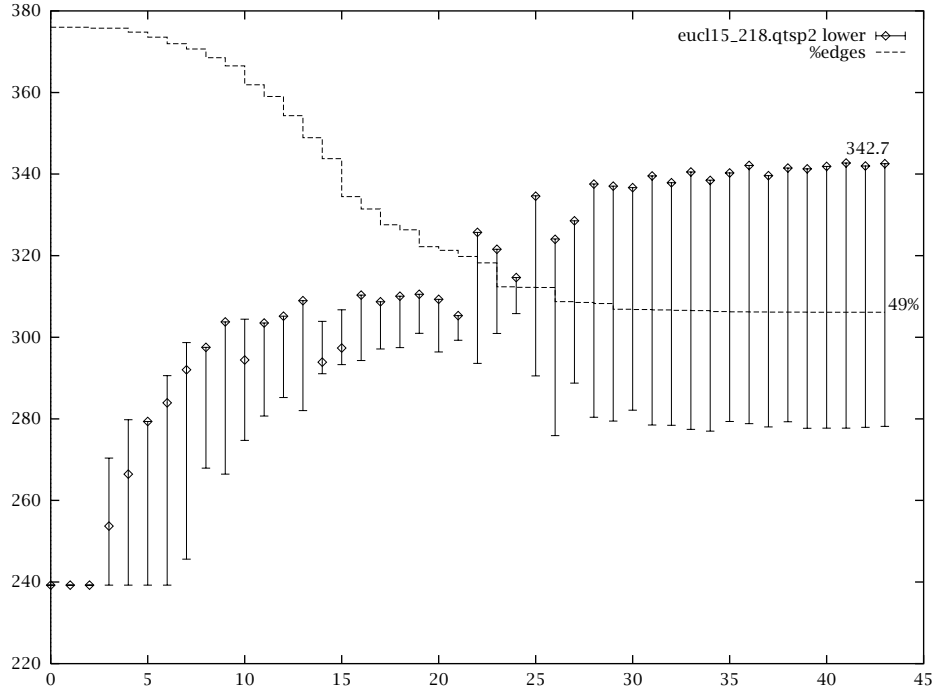


Abbildung 5.4: Die Hybrid–Strategie

ψ_{ik} zu tun. Kommt andererseits der gespeicherte Subgradient an den i,k–through–circuit zu λ' zur Verwendung, so hat er wenig mit den aktuellen Strafkosten zu tun – gegebenenfalls existiert dieser through–circuit gar nicht mehr.

Die „schöne“ Konvergenz durch die zweite Regel scheint sich nicht in verbesserten Schranken niederzuschlagen. Dies hat eine Ursache darin, daß viel weniger Variablen fixiert werden können (vermutlich aufgrund der großen Abstands zwischen billigstem Pfad und $L(\infty,\lambda)$). In Abbildung 5.3 auf Seite 89 erreicht Lucenas Variante zwar die bessere Schranke allerdings in einem Graphen, der nur noch die Hälfte der ursprünglichen Kanten enthält. Eingehendere Beobachtungen zeigen, daß die zweite Regel im Fall gleichgroßer Graphen die schärferen Schranken liefert. Um die Vorteile beider Verfahren nutzen zu können, wurden für die in Abschnitt 5.5 auf Seite 93 berichteten Ergebnisse eine Hybrid–Strategie angewendet: Zunächst folgt sie Lucenas Strategie bis sich die Zahl der Kanten nicht mehr stark reduziert. In diesem Moment wird die Schrittweite nochmal erhöht und nach der zweiten Regel fortgefahren. Einen Iterationsverlauf nach der Hybrid–Strategie stellt Abbildung 5.4 dar. Typisch ist die „Taille“ an der Stelle, wo auf die zweite Regel umgeschaltet wird.

Bemerkung: Wie in den Abbildungen 5.3 und 5.4 zu sehen ist, gilt in den seltensten Fällen $L(0,\lambda) < L(1,\lambda)$. Bei Beispielen des Typs D_2, also DMPs, kommt das et-

was häufiger als dargestellt vor. Allerdings wurde kein Fall mit $L(0,\lambda^\star) < L(1,\lambda^\star)$ beobachtet. Es wäre aber falsch, die Verbesserung der Schranken durch die Zuordnungsprobleme deswegen als ineffektiv zu unterlassen. Da in den ersten Schritten des Aufstiegsverfahrens durchaus Schrankenverbesserung erreicht werden, können tiefer im B&B–Baum dadurch häufig Knoten schneller geschlossen werden (bei den Beispielen >10%).

5.3.4 Schrittweitenkontrolle und Abbruchkriterien

Der tatsächlich durchgeführte Schritt $\Delta\lambda$ wird aus dem Subgradienten nach der Formel (3.2) auf Seite 41 berechnet. Dabei wird $\bar{z} = z_{\mathrm{DTSP}}(\pi)$ mit der aktuell billigsten, bekannten Tour verwendet. Die Schrittweitenkontrolle τ ist zu Beginn an der Wurzel des B&B–Baumes 0.5 und 0.6 an den inneren Knoten. Bei mehr als zehn Städten halbiert sich der Wert. Im Verlauf der Iteration wird τ nach minimal 5 Schritten halbiert.

`solve` bricht die λ–Iteration ab, falls:

1. mehr als $2(n+1)$ Aufstiegsschritte an der Wurzel oder mehr als $n+1-p$ Schritte bei einem Knoten der Tiefe p durchgeführt wurden,
2. falls die Schrittweitenkontrolle an der Wurzel 2^{-6} oder im Baum 2^{-5} unterschreitet,
3. falls an der Wurzel über 25 Schritte hinweg keine Verbesserung eingetreten ist oder sich die untere Schranke innerhalb der letzten 10 Schritte um weniger als 0.5% geändert hat, (Konvergenzkriterium) (im Inneren des Baums lauten diese Parameter 10 Schritte und 1% Änderung),
4. falls die Aufstiegsrichtung verschwindet, das heißt wenn ein stationärer Punkt erreicht wird

   oder
5. falls die untere Schranke die obere erreicht.

5.4 Erzeugung der Testbeispiele

Zur Erzeugung der Testbeispiele werden n Punkte als Städte pseudo–zufällig und gleichverteilt aus einem Quadrat mit beliebiger Seitenlänge r ausgewählt.

Die Startstadt liegt in der Mitte des Quadrats. Als Seitenlänge r wurde 100 gewählt, außer für die Beispiele aus Abschnitt 5.1 auf Seite 81. Aus diesen Daten werden vier DTSP–Typen gewonnen:

Typ	$d_{v,w}(t)$ stückweise linear ergänzt	$c_{v,w}(t)$ stückweise linear ergänzt
Q_1	$\lfloor \|v-w\| \rfloor$ falls $t \le 0$ $1.5\lfloor \|v-w\| \rfloor$ bei $t = 1.5r$ $\lfloor \|v-w\| \rfloor$ falls $t \ge 5r$	$= d_{i,j}(t)$
Q_2	$\lfloor \|v-w\| \rfloor$ falls $t \le 0$ $\mathcal{L}\lfloor \|v-w\| \rfloor$ bei $t = \mathcal{T}rn$ $\lfloor \|v-w\| \rfloor$ falls $t \ge 5rn$ $\mathcal{L}$ gleichverteilt in $[0,25;\ 0,6]$ $\mathcal{T}$ gleichverteilt in $[0,75;\ 2,0]$	$= d_{i,j}(t)$
D_1	$\lfloor \|v-w\| \rfloor$	t
D_2	1	$(n+1-t)\lfloor \|v-w\| \rfloor$

Bei Typen Q_1&Q_2 wird eine Tour von minimaler Dauer gesucht. Die Wegezeiten sind bei Q_1 konkav und gleichartig zeitabhängig über alle Verbindungen und bei Q_2 unimodal und pro Verbindung einzeln mit Hilfe der gleichverteilten Zufallsvariablen $\mathcal{L}$ und $\mathcal{T}$ „dynamisiert“. Typen D_1 und D_2 modellieren beide ein euklidisches DMP. Typ D_1 verwendet dazu die Formulierung (DMP) von Seite 3, D_2 stellt das DMP dagegen als TDTSP dar. Der Typ Q_1 wurde fast ausschließlich zur Programmverifikation verwendet, da er sich für kleine Instanzen noch mit dem IP–Solver von CPLEX lösen läßt. Auch die Typen D_1 und D_2 wurden zur Überprüfung der Korrektheit von `solve` verwendet, da auf zwei unterschiedlichen Wegen dasselbe Ergebnis berechnet wird. Typ Q_2 schließlich soll ein „allgemeines euklidisches DTSP “ mit variierenden Wegezeiten modellieren, wie es zum Beispiel im Individualverkehr durch den tageszeitabhängigen Verkehrsfluß in Innenstädten oder auf Autobahnen entsteht.

5.5 Numerische Ergebnisse

Das Programm `solve` und die Hilfsprogramme zur Erzeugung von Testbeispielen und zur Datenauswertung wurden in circa 6000 Codezeilen in C++ und PERL5 auf einer IBM RS6000/360 unter AIX 3.2 mit 64MB Hauptspeicher implementiert und getestet. Zur Lösung der auftretenden Zuordnungsprobleme und

	Laufzeit		% US/Opt.		% OS/Optimum		B&B Knoten		% Kanten		
n	∅	max	min	∅	∅	max	∅	max	∅	max	N
8	6.2	12	84.6	97.0	100.6	103.4	7.2	11	3.3	9.9	19
9	11.7	22	82.1	96.1	100.9	104.9	8.8	31	3.1	9.5	20
10	27.0	70	78.0	94.4	103.4	112.6	19.8	94	6.6	19.4	20
11	30.8	58	87.4	95.6	101.2	106.1	14.8	84	4.7	11.7	20
12	71.3	299	83.4	94.1	102.5	113.8	49.3	322	7.1	17.8	58
13	137.5	586	85.2	93.6	103.2	114.0	101.8	437	7.8	19.0	20
14	491.3	1769	82.5	91.6	105.4	114.0	352.1	1300	13.4	22.5	20
15	689.1	6540	75.9	91.7	103.2	115.5	574.1	6793	9.2	19.3	20
16	776.2	3126	86.0	91.9	104.4	113.5	435.8	2742	11.2	20.0	19
17	1556.0	8838	84.9	92.5	105.6	117.0	767.9	4155	11.1	18.5	21
18	2152.8	5285	87.9	91.4	105.7	114.0	915.0	2124	12.0	15.7	11
19	2615.4	4765	89.1	91.9	103.9	106.5	1018.4	2226	10.1	13.9	8

Tabelle 5.1: Beispiele Typ Q_2, Durchschnitts– und schlechteste Werte

	Laufzeit		% US/Opt.		% OS/Optimum		B&B Knoten		% Kanten		
n	∅	max	min	∅	∅	max	∅	max	∅	max	N
7	3.0	4	99.6	97.7	100.0	100.0	3.0	6	0.4	6.9	20
8	5.0	11	98.5	96.4	100.0	103.4	3.0	13	0.2	7.5	19
9	10.5	40	94.9	93.3	100.0	107.1	6.0	23	1.3	28.8	19
10	13.0	34	95.6	93.5	100.0	108.5	5.0	44	0.6	25.5	20
11	40.0	103	93.3	91.0	101.6	109.5	11.0	45	11.4	28.3	20
12	83.5	309	88.6	88.6	101.5	110.8	24.5	174	18.1	33.9	20
13	121.5	472	90.2	90.4	101.6	114.2	26.5	148	18.4	28.8	20
14	378.0	855	87.6	87.5	104.5	110.4	134.0	324	24.9	32.4	20
15	562.5	2251	82.4	83.6	102.5	109.4	193.5	744	25.1	30.5	19
16	775.0	6269	86.4	85.4	107.4	113.7	174.5	1697	26.4	32.3	20
17	1215.0	4154	81.3	83.4	105.4	109.5	205.5	1117	24.1	29.8	8
18	1385.5	4559	82.7	83.0	102.7	105.2	271.0	963	20.3	23.5	3
20	3777.0	3829	82.3	82.4	102.1	107.7	349.0	352	20.7	25.6	2

Tabelle 5.2: Beispiele Typ D_1, Durchschnitts– und schlechteste Werte

von LP–Relaxationen (siehe unten) wurde der Netzwerk–Simplex–Algorithmus von CPLEX [13] benutzt (zur Lösung von APs verbraucht `solve` weniger als 5% der CPU–Zeit). Höhere Datenstrukturen für Graphen, Listen, Dictionaries, dynamische Felder und balancierte Bäume wurden der Klassenbibliothek LEDA [50] entnommen. Insbesondere wird der zeitexpandierte Graph intern als LEDA–Graph dargestellt.

Das Programm solve wurde an etwa 800 Beispielen der Typen Q_2, D_1 und D_2 getestet. Tabelle 5.1 gibt zur Städtezahl n in vier Kategorien jeweils Durchschnittswert und schlechtestes Ergebnis an. Spalten zwei und drei enthalten die Laufzeit in CPU–Sekunden. Spalten vier bis sieben führen die Werte der oberen und unteren Schranke am Wurzelknoten des Branch&Bound–Baumes relativ zum Optimalwert auf und die Spalten acht und neun die Anzahl der nach der

	Laufzeit		% US/Opt.		% OS/Optimum		B&B Knoten		% Kanten		
n	∅	max	min	∅	∅	max	∅	max	∅	max	N
8	0.0	4	96.8	94.0	100.0	103.3	4.0	32	17.5	69.4	21
9	1.0	4	96.5	94.5	100.0	107.0	5.0	43	17.2	74.1	20
10	3.0	7	89.9	89.9	100.4	106.9	21.0	82	60.0	82.3	20
11	2.5	10	88.0	89.7	100.7	107.9	17.0	103	52.0	76.2	20
12	3.0	14	89.0	90.0	101.3	105.7	18.5	197	54.9	82.7	20
13	7.0	20	87.3	87.1	100.0	112.6	40.5	188	57.2	82.5	20
14	10.5	35	86.1	86.4	101.8	108.1	69.0	303	65.4	78.3	20
15	12.5	64	85.8	86.5	102.7	109.2	74.5	590	67.1	85.4	20
16	26.0	203	84.7	85.0	104.0	112.7	162.0	1106	72.0	89.6	20
17	44.0	401	85.4	84.8	103.3	110.7	272.0	2371	70.9	90.9	39
18	64.0	697	84.0	84.8	103.0	113.4	412.0	5464	77.5	93.8	39
19	147.5	491	83.8	83.0	105.9	114.8	885.0	2989	80.4	93.3	20
20	100.0	1304	81.7	82.4	104.4	109.5	455.5	7801	76.4	92.7	20
21	250.5	2001	82.0	81.3	105.8	110.3	1056.0	9401	86.2	96.7	20
22	225.0	2609	83.9	82.7	104.5	119.7	865.0	14152	77.5	98.7	21
23	447.0	3752	82.8	81.7	105.4	122.7	1565.0	12317	80.8	97.0	25
24	1098.0	5802	79.5	79.7	108.2	116.1	3009.5	25951	90.7	97.9	12
25	898.5	7230	78.0	80.6	105.3	116.0	2491.5	27487	87.8	97.6	11
26	725.0	9876	69.5	78.0	101.6	105.9	2338.0	26319	81.1	96.8	2

Tabelle 5.3: Beispiele Typ D_2, Durchschnitts– und schlechteste Werte

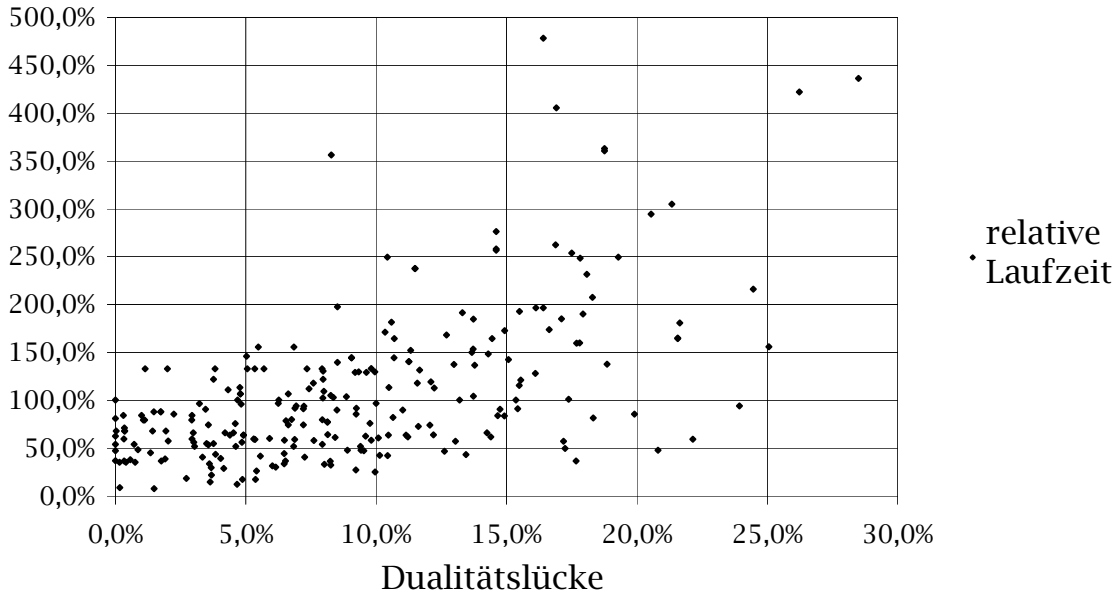


Abbildung 5.5: Laufzeitdiagramm für Typ Q_2–Beispiele

Variablenfixierung am Wurzelknoten noch verbliebenen Kanten relativ zur Gesamtzahl $|E^{\mathbb{P}_n}|$. Die Spalte ganz rechts gibt die Anzahl an Beispielläufen an, aus denen die Daten stammen. Die Tabellen 5.2 und 5.3 listen dieselben Daten für die Beispiele der anderen beiden Typen auf. Offensichtlich sind die Probleme

des Typs D_2 am einfachsten zu lösen. Beim Vergleich mit den Ergebnissen für die Typen Q_2 und D_1 ist zu beachten, daß zum Beispiel die TDTSP–Instanzen mit 17 Städten aus Tabelle 5.3 nach dem Wurzelknoten noch typischerweise 3000 Kanten, die Instanzen aus Tabelle 5.2 aber etwa noch 100000 und die aus Tabelle 5.1 etwa noch 70000 Variablen bei derselben Städtezahl haben.

Es ist nicht gelungen, die Ergebnisse von Lucena [46] zu verifizieren, der für zwölf Beispiele mit bis zu 30 Städten deutlich schärfere Schranken berichtet. Die von Lucena verwendete Austauschheuristik könnte die besseren oberen Schranken erklären, außerdem wird aus [46] nicht klar, wie effektiv seine Implementation Variablen fixiert. Schließlich erwähnt Lucena noch eine Methode zur Verhinderung von Untertouren aus drei Städten, die nicht auf das DTSP verallgemeinert werden konnte. Um die Qualität der von `solve` berechneten Schranke zu testen, wurde sie mit der Schranke aus der LP–Relaxation von (HVPQ) auf Seite 46 verglichen. Dieser Wert wurde meist weit übertroffen. Der Optimalwert dieses LPs ist wegen der Integralitätseigenschaft gleich den maximalen Kosten des billigsten Pfades durch V_0 unter Strafkosten. Wurde statt der in Abschnitt 5.3.3 auf Seite 87 angegeben Aufstiegsrichtung eine zur Maximierung dieser Kosten geeignete gewählt, so erreichte die Iteration in typischen Fällen über 96% des Optimalwertes. Auch die Schranke $L(0,\lambda)$ läßt sich im TDTSP–Fall über ein LP theoretisch exakt berechnen. Da dazu aber parallel n^2 Flußprobleme in G^T zur Berechnung der through–circuits gelöst werden müssen, enthält die Formulierung bereits für 15 Städte über 700000 Variablen, so daß dieser Weg nicht gangbar erscheint. Eventuell lassen sich mittels Formulierung 5.3 durch Spaltengenerierung exakte Lösungen des Lagrange–Problems berechnen.

Die Bedeutung scharfer Schranken für den Lösungsaufwand macht noch einmal Abbildung 5.5 klar. Für die berechneten Beispiele vom Typ Q_2 (Tabelle 5.1) sind im Diagramm als Punkte eingetragen mit der Dualitätslücke relativ zum Optimalwert an der Wurzel des Branch&Bound–Baumes als Ordinate und der Laufzeit relativ zum Mittelwert für dieselbe Städtezahl als Abszisse. Das Bild deutet an, daß man bis etwa zu einer Dualitätslücke von 15% durchschnittliche Laufzeiten erwarten kann und ab 25% mit hohen Laufzeiten gerechnet werden muß.

Zusammenfassend ist zu sagen, daß `solve` für DMPs deutlich dem spezialisierten Verfahren von Fischetti, Laporte und Martello [17] unterlegen ist. Es gelingt `solve` jedoch mit vertretbarem Aufwand, die als deutlich schwieriger einzuschätzenden DTSP–Instanzen mit variierenden Wegezeiten zu lösen.

5.6 Möglichkeiten zur Weiterentwicklung

Der Autor sieht drei Hauptpunkte, in denen `solve` noch weiterentwickelt werden sollte:

- **Verbesserungsheuristiken**

  Ein Schwachpunkt von `solve` sind die oberen Schranken. Durch den Einsatz von Kantentausch–Verfahren (siehe [17, 28, 64]), die noch vom DMP oder dem TDTSP auf das DTSP verallgemeinert werden müßten, ließe sich sicher viel gewinnen.

- **Variablenfixierung**

  Die Effizienz von `solve` hängt entscheidend davon ab, schnell viele Kanten aus dem zeitexpandierten Graphen zu eliminieren. Eine weitere Methode zur Variablenfixierung kann aus der Berechnung teuerster statt billigster $n+1$–Pfade gewonnen werden. Die vorhandenen Algorithmen können dies durch Vorzeichenumkehr der Kosten leisten. Ist dann der teuerste $n+1$–schrittige Pfad über einen Kante immer noch billiger als die momentane untere Schranke, kann die Kante eliminiert werden, da es offensichtlich keine Tour gibt, die darüber verläuft.

- **Effizientere Datenstrukturen**

  Es wäre sicher effizienter, den zeitexpandierten Graphen durch Adjazenzfelder statt durch Adjazenzlisten wie in LEDA darzustellen, da die Felder von beschränkter Größe wären (n Kanten pro Knoten).

Nachwort

Die Niederschrift ist zu Ende, aber Arbeit ist noch genügend übrig. Die folgende Punkte würde ich gerne weiter verfolgen:

- **Fortsetzungseigenschaft**

  Die Suche nach einer größeren Klasse von DTSP–Instanzen mit der Fortsetzungseigenschaft ist theoretisch und praktisch ein lohnendes Unterfangen. Auch die Entwicklung eines polynomialen Verfahrens zur Erkennung von *lambda*–stabilen zeitexpandierten Graphen würde mich reizen. Notwendig ist dazu ein vertieftes Verständnis der Lagrange–Relaxation.

- **Spezialfälle**

  Das DTSP ist ein sehr allgemeines Modell. Vielleicht gibt es Spezialfälle, die sich einfacher behandeln lassen. Rechentechnisch wären Verfahren zur Schätzung des Horizonts von großem Interesse

- **Alternative Konzepte für zeitexpandierte Graphen**

  Die Arbeit an `solve` hat gezeigt, wie bedeutsam die Rechengenauigkeit und das Konzept des Zeitpunkts für die Repräsentation des zeitexpandierten Graphen sind. Die Idee ist, Zeitpunkte durch Zeitintervalle zu ersetzen. Dadurch würde der gesamte Zeitstrahl abgedeckt anstatt nur einer abzählbaren Teilmenge an Zeitpunkten. Durch das wiederholte Unterteilen der Intervalle ließe sich eine Folge von sich verbessernden Näherungen erreichen.

It's a magical world ol'buddy, let's explore!

Bill Watterson

Literaturverzeichnis

Das Literaturverzeichnis umfaßt alle Quellen, die zur Erstellung dieser Arbeit verwendet wurden. Nicht alle Einträge sind auch im Text zitiert.

[1] AFRATI, Foto ; COSMADAKIS, Stavros ; PAPADIMITRIOU, Christos H. ; PAPAGEORGIOU, George ; PAPAKOSTANTINOU, Nadia: The Complexity of the Travelling Repairman Problem. In: *Informatique théorique et Applications* 20 (1986), Nr. 1, S. 79–87. – ISSN 0399–0540

[2] AVERBAKH, Igor ; BERMAN, Oded: Routing and Location–Routing p–Delivery Men Problems on a Path. In: *Transportation Science* 28 (1994), Mai, Nr. 2, S. 162–166. – ISSN 0041–1655

[3] BALAS, E. ; TOTH, P.: Branch and bound methods. In: Lawler (Hrsg.) [u. a.] (siehe [44]). – ISBN 0–471–90413–9, Kapitel 10, S. 361–401

[4] BASTIAN, Cock ; RINNOOY KAN, Alexander H. G.: The stochastic vehicle routing problem revisited. In: *European Journal Of Operational Research* 56 (1992), S. 407–412. – ISSN 0377–2217

[5] BELLMANN, R.: On a routing problem. In: *Quarterly Applied Mathematics* 16 (1958), S. 87–90

[6] BIANCO, Lucio ; MINGOZZI, Aristide ; RICCIARDELLI, Salvatore: The Traveling Salesman Problem with Cumulative Costs. In: *Networks* 23 (1993), S. 81–91. – ISSN 0028–3045

[7] BIENSTOCK, Daniel ; GOEMANS, Michel X. ; SIMCHI-LEVI, David ; WILLIAMSON, David: A note on the prize collecting traveling salesman problem. In: *Mathematical Programming* 59 (1993), S. 413–420. – ISSN 0025–5610

[8] BURKARD, R. E. ; DLASKA, K. ; KLINZ, B.: The Quickest Flow Problem / Institut für Mathematik, Technische Universität Graz. 1991. – Forschungsbericht

[9] CHEN, Y. L. ; CHIN, Y. H.: The Quickest Path Problem. In: *Computers Operations Research* 2 (1989), S. 153–161. – ISSN 0305–0548

[10] CHRISTOFIDES, Nicos ; MINGOZZI, A. ; TOTH, P.: State Space Relaxation Procedures for the Computation of Bounds to Routing Problems. In: *Networks* 11 (1981), S. 145–164. – ISSN 0028–3045

[11] CHRISTOF, Thomas ; LÖBEL, Andreas: PORTA. Internet `http://www.iwr.uni-heidelberg.de/iwr/comopt/soft/PORTA/readme.html`. Februar 1994. – Software des Instituts für wissenschaftliches Rechnen an der Universität Heidelberg und des Konrad-Zuse-Zentrums für Informationstechnik Berlin

[12] COOKE, Kenneth L. ; HALSEY, Eric: The Shortest Route Through a Network with Time-Dependent Internodal Transit Times. In: *Journal of Mathematical Analysis and Applications* 14 (1966), S. 493–498. – ISSN 0022–247x

[13] CPLEX OPTIMIZATION, Inc.: CPLEX Version 2.1. CPLEX Optimization, Inc., 930 Tahoe Blvd., Bldg. 802, Incline Village, NV. 1993. – Softwarepaket zur linearen und ganzzahligen Programmierung

[14] DANTZIG, G. B. ; FULKERSON, D. R. ; JOHNSON, S. M.: Solution of a Large Scale Traveling Salesman Problem. In: *Operations Research* 2 (1954), S. 393–410. – ISSN 0001–3226

[15] DEO, Narsingh ; PANG, Chi-yin: Shortest-Path Algorithms: Taxonomy and Annotation. In: *Networks* 14 (1984), S. 275–323. – ISSN 0028–3045

[16] DUMAS, Yvan ; DESROSIERS, Jacques ; GELINAS, Eric ; SOLOMON, Marius M.: An Optimal Algorithm for the Traveling Salesman Problem with Time Windows. In: *Operations Research* 43 (1995), März–April, Nr. 2, S. 367–374. – ISSN 0001–3226

[17] FISCHETTI, Matteo ; LAPORTE, Gilbert ; MARTELLO, Silvano: The Delivery Man Problem and Cumulative Matroids. In: *Operations Research* 41 (1993), November–Dezember, Nr. 6, S. 1055–1064. – ISSN 0001–3226

[18] FISHER, Marshall L. ; JAIKUMAR, R. ; VAN WASSENHOVE, Luk N.: A Multiplier Adjustment Method for the Generalized Assignment Problem. In: *Management Science* 37 (1986), September, Nr. 9, S. 1095–1103. – ISSN 0025–1909

[19] FISHER, Marshall L.: The Lagrangean Relaxation Method for Solving Integer Programming Problems. In: *Management Science* 27 (1981), Januar, Nr. 1, S. 1–18. – ISSN 0025–1909

[20] FOURER, Robert ; GAY, David M. ; KERNIGHAM, Brian W.: *AMPL: A Modeling Language for Mathematical Programming*. The Scientific Press, 1993. – ISBN O–89426–234–3

[21] FOX, Kenneth R. ; GAVISH, Bezalel ; GRAVES, Stephen C.: An n-Constraint Formulation of the (Time-Dependent) Traveling Salesman Problem. In: *Operations Research* 28 (1980), Juli–August, Nr. 4, S. 1018–1021. – ISSN 0001–3226

[22] FOX, Kenneth R.: *Production Scheduling on Parallel Lines with Dependencies*, The John Hopkins University, Baltimore, Diplomarbeit, 1973

[23] GAREY, Michael R. ; JOHNSON, David S.: *Computers and Intractability*. Bell Labratories, 1979

[24] GARFINKEL, R. S.: Motivation and modeling. In: Lawler (Hrsg.) [u. a.] (siehe [44]). – ISBN 0–471–90413–9, Kapitel 2, S. 17–36

[25] GEOFFRION, A. M.: Lagrangean Relaxation for Integer Programming. In: *Mathematical Programming Study* 2 (1974), S. 82–114. – ISSN 0303–3929

[26] GILMORE, P. C. ; LAWLER, E. L. ; SHMOYS, D. B.: Well solved special cases. In: Lawler (Hrsg.) [u. a.] (siehe [44]). – ISBN 0–471–90413–9, Kapitel 4, S. 87–143

[27] GOFFIN, J. L.: On the Convergence Rates of Subgradient Optimization Methods. In: *Mathematical Programming* 13 (1977), S. 329–377. – ISSN 0025–5610

[28] GOLDEN, B. L. ; STEWART, W. R.: Empirical analysis of heuristics. In: Lawler (Hrsg.) [u. a.] (siehe [44]). – ISBN 0–471–90413–9, Kapitel 7, S. 207–249

[29] GOUVEIA, Luis ; VOSS, Stefan: A classification of formulations for the (time-dependent) traveling salesman problem. In: *European Journal Of Operational Research* 83 (1995), S. 69–82. – ISSN 0377–2217

[30] HAAS, Oliver: *The Cumulative Assignment Problem – Local Search and Heuristics*, Universität Kaiserslautern, Diplomarbeit, Juni 1995. – am Fachbereich Mathematik, AG Prof. Hamacher

[31] HADLEY, George: *Nonlinear and Dynamic Programming*. Reading, Mass., USA : Addison-Wesley Publishing Company, Inc., 1964

[32] HAMACHER, Horst W. ; MAFFIOLI, Francesco ; DELL'AMICO, Mauro: Cumulative Matroid Intersections. Januar 1996. – Arbeitsvorlage

[33] HELD, Michael ; KARP, R. M.: The Traveling Salesman Problem and Minimum Spanning Trees. In: *Operations Research* 18 (1970), S. 1138–1162. – ISSN 0001–3226

[34] HELD, Michael ; KARP, R. M.: The Traveling Salesman Problem and Minimum Spanning Trees, Part II. In: *Mathematical Programming* 1 (1971), S. 6–25. – ISSN 0025–5610

[35] HELD, Michael ; WOLFE, Philip ; CROWDER, Harlan P.: Validation of Subgradient Optimization. In: *Mathematical Programming* 6 (1974), S. 62–88. – ISSN 0025–5610

[36] HOFFMANN, A. J. ; WOLFE, P.: History. In: Lawler (Hrsg.) [u. a.] (siehe [44]). – ISBN 0–471–90413–9, Kapitel 1, S. 1–15

[37] HOUCK, D. J. ; VEMUGANTI, R. R.: The Traveling Salesman Problem and Shortest n–Paths. Mai 1976. – vorgestellt bei der ORSA–TIMS Konferenz, Philadelphia, Pa.

[38] JEROMIN, Bernd ; KÖRNER, Frank: On the Refinement of Bounds of Heuristic Algorithms for the Traveling Salesman Problem. In: *Mathematical Programming* 32 (1985), S. 114–117. – ISSN 0025–5610

[39] JEROMIN, Bernd: Untersuchungen zu einer Dualisierung des Rundreiseproblems. In: *Wissenschaftliche Zeitschrift der Universität Dresden* 1 (1990), Nr. 39, S. 179–182. – ISSN 0043–6925

[40] JOHNSON, D. S. ; PAPADIMITRIOU, Christos H.: The Traveling Salesman Problem. In: Lawler (Hrsg.) [u. a.] (siehe [44]). – ISBN 0–471–90413–9, Kapitel 3, S. 37–85

[41] JUNGNICKEL, Dieter: *Graphen, Netzwerke und Algorithmen*. 3., vollständig überarbeitete und erweiterte Auflage. Mannheim, Leipzig, Wien, Zürich : BI–Wissenschafts–Verlag, 1994. – ISBN 3–411–14263–4

[42] KERNIGHAM, B. W. ; LIN, S.: An Effective Heuristic Algorithm for the Traveling Salesman Problem. In: *Operations Research* 21 (1973), S. 498–516. – ISSN 0001–3226

[43] KLINZ, Bettina. Persönliche Gespräche. 1995

[44] LAWLER, E. L. (Hrsg.) ; LENSTRA, J. K. (Hrsg.) ; RINNOY KAN, Alexander H. G. (Hrsg.) ; SHMOYS, D. B. (Hrsg.): *The Traveling Salesman Problem*. John Wiley & Sons Ltd., 1985. – ISBN 0–471–90413–9

[45] LAWLER, E. L.: The Quadratic Assignment Problem. In: *Management Science* 19 (1963), S. 586–599

[46] LUCENA, Abilio: Time-Dependent Traveling Salesman Problem – The Deliveryman Case. In: *Networks* 20 (1990), S. 753–763. – ISSN 0028–3045

[47] MALANDRAKI, Chryssi ; DASKIN, Mark S.: Time Dependent Vehicle Routing Problems: Formulations, Properties and Heuristic Algorithms. In: *Transportation Science* 26 (1992), August, Nr. 3, S. 185–200. – ISSN 0041–1655

[48] MALANDRAKI, Chryssi ; DIAL, Robert B.: A restricted dynamic programming heuristic for the time dependent traveling salesman problem / Roadnet Technologies, Inc. 2311 York Road, Timonium, MD 21093, USA, Dezember 1993. – Forschungsbericht

[49] MALANDRAKI, Chryssi ; DIAL, Robert B.: A restricted dynamic programming heuristic for the time dependent traveling salesman problem. In: *European Journal Of Operational Research* 90 (1996), S. 45–55. – ISSN 0377–2217

[50] MEHLHORN, Kurt ; NÄHER, Stefan: LEDA-R-3.3 – Library of Efficient Data types and Algorithms. Internet `ftp://ftp.mpi-sb.mpg.de/pub/LEDA`. Februar 1996. – Max-Planck-Institut für Informatik, Saarbrücken und Fachbereich Mathematik und Informatik der Martin-Luther Universität Halle-Wittenberg

[51] MINIEKA, Edward. The Delivery Man Problem on a Tree Network. Paper presented at the 1984 ORSA/TIMS Meeting San Francisco. 1984

[52] MINIEKA, Edward: The Delivery Man Problem on a Tree Network. In: *Annals of Operations Research* 18 (1989), S. 261–266. – (siehe auch [51] für eine frühere Veröffentlichung). – ISSN 0254–5330

[53] MURTHY, Ishwar: Solving the Multiperiod Assignment Problem with Start–Up Costs Using Dual Ascent. In: *Naval Research Logistics Quarterly* 40 (1993), S. 325–344. – ISSN 0028–1441

[54] NEMHAUSER, George L. ; WOLSEY, Laurence A.: *Integer and Combinatorial Optimization*. New York : John Wiley& Sons, Inc., 1988. – ISBN 0–471–82819–X

[55] ORDA, Ariel ; ROM, Raphael: Minimum Weight Paths in Time–Dependent Networks. In: *Networks* 21 (1991), S. 235–319. – ISSN 0028–3045

[56] ORDA, Ariel ; ROM, Raphael: Shortest–Path and Minimum–Delay Algorithms in Networks with Time–Dependent Edge–Length. In: *Journal of the Association for Computing Machinery* 37 (1990), Juli, Nr. 3, S. 607–625. – ISSN 0004–5411

[57] OTTMANN, T. ; WIDMAYER, P.: *Algorithmen und Datenstrukturen*. Mannheim, Wien, Zürich : BI–Wissenschaftsverlag, 1990 (Informatik, Band 70). – ISBN 3–411–03161–1

[58] PICARD, Jean-Claude ; QUEYRANNE, Maurice: The Time–Dependent Traveling Salesman Problem and Its Application to the Tardiness Problem in One–Machine Scheduling. In: *Operations Research* 26 (1978), Jan–Feb, Nr. 1, S. 86–110. – ISSN 0001–3226

[59] POLJAK, B. T.: A general Method of Solving Extremum Problems. In: *Soviet Mathematics Doklady* 8 (1967), S. 593–5997. – Übersetzung der Doklady Akademii Nauk SSSR 174. – ISSN 0038–5573

[60] ROSEN, J. B. ; SUN, S.-Z. ; XUE, G.-L.: Algorithms for the Quickest Path Problem and the Enumeration of Quickest Paths. In: *Computers Operations Research* 18 (1991), Nr. 6, S. 579–584. – ISSN 0305–0548

[61] SAKSENA, J. P. ; KUMAR, Santosh: The Routing Problem with 'K' Specified Nodes. In: *Operations Research* 14 (1966), S. 909–913. – ISSN 0001–3226

[62] SHAPIRO, Jeremy F.: A Survey of Lagrangean Techniques for Discrete Optimization. In: *Annals of Discrete Mathematics* 5 (1979), S. 113–138. – ISSN 0167–5060

[63] SIMCHI-LEVI, David ; BERMAN, Oded: Minimizing the Total Flow Time of n Jobs on a Network. In: *IIE Transactions* 23 (1991), Nr. 3, S. 236–244. – ISSN 0740–817X

[64] VANDER WIEL, Russ J. ; SAHINIDIS, Nikolaos V.: Heuristic Bounds and Test Problem Generation for the Time–Dependent Traveling Salesman Problem. In: *Transportation Science* 29 (1995), Mai, Nr. 2, S. 167–183. – ISSN 0041–1655

[65] WONG, R. T.: Integer Programming Formulations of the Travelling Salesman Problem. In: *Proceedings of the International Conference on Circuits and Computers*, 1980, S. 149–152

Hiermit bestätige ich, daß ich die vorliegende Diplomarbeit alleine erstellt habe und keine weiteren außer den angegebenen Hilfsmitteln dazu verwendet habe.