Humboldt-Universität zu Berlin
Institut für Informatik
Lehrstuhl für Algorithmen und Komplexität

Simulated Annealing und seine Anwendung auf das Crew-Scheduling-Problem

Diplomarbeit

Mark Proksch

Berlin, 8. Dezember 1997
Betreuer: Prof. Dr. H. J. Prömel

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

Einleitung

Die vorliegende Arbeit entstand im Rahmen von PARALOR, einem vom Bundesministerium für Bildung, Wissenschaft, Forschung und Technologie finanzierten Verbundprojekt, an dem folgende Partner beteiligt waren:

- Universität-Gesamthochschule Paderborn, Arbeitsgruppe Professor Monien am Fachbereich Informatik
- Universität zu Köln, Arbeitsgruppen Professor Bachem und Professor Schrader am Zentrum für Paralleles Rechnen und Institut für Informatik
- Humboldt-Universität zu Berlin, Arbeitsgruppe Professor Prömel am Institut für Informatik
- Lufthansa Systems Frankfurt am Main und Berlin
- Profi.L, Remscheid

Thematisch umfaßte das Projekt kombinatorische Optimierungsprobleme aus dem Bereich Flugplanung und Logistik.

Unsere Arbeitsgruppe bestand aus Professor Prömel als deren Leiter, dem wissenschaftlichen Mitarbeiter und Doktoranden Thomas Emden-Weinert, weiterhin Peter Weisse als studentischer Hilfskraft und mir als studentischer Hilfskraft und Diplomand. Unser Schwerpunkt lag in der Pairing-Generierung, einem Teilproblem der Flugpersonalplanung (Crew-Assignment). Zur Lösung wurden verschiedene lokale Suchverfahren verwendet und hinsichtlich ihrer Eignung untersucht. Gegenstand dieser Arbeit ist hauptsächlich das Simulated Annealing, eine in Anlehnung an physikalische Vorgänge entstandene, randomisierte lokale Suchheuristik. Vergleichende Studien zu den weiteren verwendeten Verfahren wie Tabu Search, Sintflut und Record-To-Record-Travel werden in der Doktorarbeit von Thomas Emden-Weinert [Emd98] zu finden sein, deren Erscheinen für 1998 vorgesehen ist.

Kapitel 1.1 dieser Arbeit skizziert das von uns bearbeitete Problem der Pairing-Generierung im Kontext der Flugplanung. Kapitel 1.2 bietet eine Einführung zur lokalen Suche und stellt den Metropolis- und den Simulated-Annealing-Algorithmus vor.

Während des Projektes entstand ein Prototyp, an dem diverse Studien durchgeführt wurden. In Kapitel 2 geht es um die Modellierung unseres Problems in einer für die lokale Suche zugänglichen Form, eine grobe Skizze der Implementation des Prototyps, die Vorstellung unserer Testdaten und die Beurteilung einfacher Lösungsheuristiken.

Kapitel 3 beinhaltet umfangreiche Studien am Prototyp, wie die Wahl eines geeigneten Kühlschemas und dessen Parameter. Aber auch Einzelheiten der Kostenfunktion, zwei Varianten des Annealing-Algorithmus und ein simpler, aber vielversprechender Parallelisierungsansatz wurden untersucht.

An dieser Stelle möchte ich Professor Dr. Hans Jürgen Prömel und Thomas Emden-Weinert meinen Dank für die Anregung zu dieser Arbeit und deren Betreuung aussprechen. Darüber hinaus bedanke ich mich bei Peter Weisse für seinen Beitrag zur Entwicklung des Prototyps, besonders sei hier die von ihm entwickelte graphische Oberfläche hervorgehoben. Ein Wort des Dankes möchte ich auch an den Projektpartner Lufthansa Systems Berlin für die Bereitstellung der Testdatensätze richten, speziell an die dortige Projektverantwortliche, Dr. Heike Ripphausen-Lipa, für die gute Zusammenarbeit. Nicht zuletzt bin ich Henrik Pletat für seine TEXnische Unterstützung bei der Umsetzung dieser Arbeit und ein jederzeit offenes Ohr dankbar.

Berlin, den 8. Dezember 1997 Mark Proksch

Kapitel 1

Grundlagen

1.1 Das Problem Flugplanung

Unter den vielfältigen Problemstellungen, die bei der Organisation einer Fluggesellschaft zu lösen sind, kommt der Flugplanung eine zentrale Rolle zu, da die hier getroffenen Entscheidungen wesentlich für die Wirtschaftlichkeit und damit die Konkurrenzfähigkeit der Gesellschaft sind. Dabei zerfällt die Flugplanung in drei Teilprobleme: die Erstellung des Angebots, das Fleet-Assignment und das Crew-Assignment.

Zunächst wird anhand der Marktlage und einer voraussichtlichen Nachfrage ein Angebot erstellt. Zu entscheiden ist dabei, welche Flugrouten wann und mit welcher Kapazität bedient werden sollen, wie Anschlüsse zu gestalten sind und nicht zuletzt, zu welchen Konditionen die Flüge angeboten werden sollen. Dieser Prozeß bietet Raum, den zu erwartenden Erlös und das damit verbundene Risiko zu optimieren.

Beim Fleet-Assignment wird bestimmt, mit welchem Flugzeugtyp und später mit welchem konkreten Flugzeug ein bestimmter Flug durchgeführt werden soll. Bei der Auswahl des Flugzeugtyps sind Faktoren wie Kapazität, Reichweite, verursachte Kosten (Personalbedarf, Treibstoffverbrauch) und natürlich die vorhandene Anzahl von Flugzeugen eines Typs entscheidend. Die Wahl des konkreten Flugzeuges wird hauptsächlich durch die Optimierung der notwendigen Wartungspausen beeinflußt. Dieses Teilproblem bietet die Möglichkeit, sowohl Kosten als auch Erlöse zu optimieren. Möglicherweise stellt sich jedoch im Laufe des Fleet-Assignments heraus, daß das entwickelte Angebot mit dem vorhandenen Gerät gar nicht, nur mit unvertretbar hohen Kosten oder unter Verletzung zusätzlicher Nebenbedingungen realisierbar ist. In diesem Fall muß in der Planung ein Schritt zurück zur Erstellung eines neuen Angebots gegangen werden.

In der letzten Phase, dem Crew-Assignment, wird den einzelnen Flügen das notwendige Crewpersonal zugewiesen. Zu beachten sind dabei eine Vielzahl von Bedingungen, auf die im folgenden noch genauer eingegangen wird. Potential zur Optimierung bieten dabei vor allem die verursachten Personalkosten. Diese bestehen nicht nur aus Gehaltszahlungen, sondern auch aus Kosten für notwendige Hotelübernachtungen und Transfers zum oder vom Einsatzort, sogenannten Proceedings. Auch hier kann eine Rückkopplung zum Fleet-Assignment oder gar zur Angebotserstellung notwendig sein, falls keine akzeptable Lösung gefunden wird.

Ein weiteres Optimierungsziel bei allen Schritten ist immer auch die Robustheit einer gefundenen Lösung gegenüber kurzfristigen Änderungen oder Ausfällen.

1.1.1 Pairing-Generierung als Teilproblem des Crew-Assignments

Ein einzelner Flug ohne Zwischenlandung stellt für die Personalplanung die kleinste Einheit im Flugplan dar und wird als Leg bezeichnet. Jedes Leg muß mit einer Besatzung versehen werden, deren Größe und Zusammensetzung Crewkomplement genannt wird. Aufgabe des Crew-Assignments ist es nun, aus den Positionen der Crewkomplemente aller Legs die Einsatzpläne der Mitarbeiter entsprechend ihrer Qualifikation zusammenzustellen, wobei eine ganze Reihe von Nebenbedingungen zu beachten sind.

So muß der Einsatzplan eines Mitarbeiters, genannt Roster, räumlich konsistent und überlappungsfrei sein sowie die gesetzlich und tarifvertraglich vorgeschriebenen Arbeits- und Ruhezeiten einhalten. Darüber hinaus sind auch unverplanbare Zeitabschnitte im Roster zu beachten, wie z.B. Urlaub, Schulungen, Büro- und Standbydienste und ähnliches. Hat ein Mitarbeiter Lizenzen für verschiedene Flugzeugtypen, so muß er auch auf diesen Typen regelmäßig eingesetzt werden, damit die Lizenzen nicht verfallen.

Jedem Mitarbeiter ist eine sogenannte Homebase zugeordnet. Es handelt sich dabei um einen Flughafen, in dessen Nähe der Mitarbeiter wohnt und an dem alle Dienstperioden beginnen und enden müssen. Beginnt der tatsächliche Flugdienst woanders, muß auf Kosten der Fluggesellschaft und innerhalb der Arbeitszeit ein Proceeding dorthin vorgenommen werden. Gleiches gilt für das Ende des Dienstes. Neben den entstehenden Kosten und der ungenutzten Arbeitszeit gehen Proceedings, wenn sie per Flugzeug durchgeführt werden, auch in die maximal zulässige Flugzeit ein und sind daher nach Möglichkeit zu vermeiden.

Viele Fluggesellschaften formulieren außerdem eigene Forderungen an die

Crewplanung, die bestimmte, nichtmaterielle unternehmerische Ziele wie beispielsweise Mitarbeiterzufriedenheit modellieren.

Üblicherweise zerlegt man das Crew-Assignment in zwei Schritte. Zunächst werden die einzelnen Legs zu sogenannten Pairings zusammengefaßt, die als Arbeitseinheit von jeweils einem, vorerst anonymen Mitarbeiter abgearbeitet werden sollen. Diese Pairings verteilt man im zweiten Schritt, dem Rostering, auf die Roster der einzelnen Mitarbeiter. Die Unterteilung dient hauptsächlich dazu, die Gesamtkomplexität des Problems zu senken, indem man sich in den beiden Schritten auf jeweils andere Optimierungsziele konzentrieren kann. Natürlich ist trotz der Aufteilung eine enge Kopplung oder iterative Abwechslung beider Schritte wünschenswert.

In der vorliegenden Arbeit ist die Pairinggenerierung hauptsächlicher Gegenstand der Betrachtung.

1.2 Simulated Annealing

Simulated Annealing ist eine Metaheuristik zur Lösung kombinatorischer Optimierungsprobleme. Das heißt, daß es sich um eine generelle Strategie handelt, die nicht auf spezielle Probleme zugeschnitten ist. Sie wurde unabhängig entwickelt von

- Kirkpatrick, Gelatt & Vecchi [KGV83]
- Černy [Čer85].

Die Autoren machten sich dabei Erkenntnisse aus der statistischen Physik über das Wachstum von Kristallen in Schmelzen zunutze. Diesen Weg möchte ich allerdings nicht vorstellen, sondern mich auf mathematische Weise dem Annealing-Algorithmus nähern. Das physikalische Vorbild soll dabei jedoch nicht ganz aus den Augen verloren werden.

1.2.1 Kombinatorische Optimierung und lokale Suche

Für die Betrachtungen in dieser Arbeit ist eine Handhabung kombinatorischer Optimierungsprobleme sinnvoll, die von den eigentlich zu lösenden Problemstellungen abstrahiert. Dies bietet den Vorteil, allgemeine Algorithmen oder Strategien entwickeln zu können, die leicht auf verschiedene Probleme übertragbar sind. Von Nachteil ist allerdings, daß aus speziellen Eigenschaften eines Problems im allgemeinen kein Vorteil gezogen werden kann.

Es wird davon ausgegangen, daß das Finden einer Lösung trivial ist und die eigentliche Aufgabenstellung in der Suche nach einer „besten“ Lösung besteht. Ich benutze eine etwas informale Definition nach [PS82] und [AK89].

Definition 1.1
*Ein **kombinatorisches Optimierungsproblem** ist eine Menge von **Instanzen**.*

*Eine **Instanz** eines kombinatorischen Optimierungsproblems ist ein Paar $(\mathcal{S}, f)$, bestehend aus einer endlichen Menge $\mathcal{S}$ von Lösungen und einer effizient berechenbaren Kostenfunktion $f : \mathcal{S} \to \mathbb{R}$.*

*Ein $i \in \mathcal{S}$ heißt **globales Minimum**, wenn*

$$\forall j \in \mathcal{S} : f(i) \leq f(j). \tag{1.1}$$

Die Menge der globalen Minima wird mit $\mathcal{S}_{opt}$ bezeichnet. □

Diese Definition beschränkt sich auf Minimierungsprobleme, da sich auch jedes Maximierungsproblem als ein solches darstellen läßt.

Definition 1.2
*Eine **Nachbarschaft** auf einer Instanz $(\mathcal{S}, f)$ eines kombinatorischen Optimierungsproblems ist eine Funktion*

$$\mathcal{N} : \mathcal{S} \to 2^{\mathcal{S}} \text{ mit } \forall i \in \mathcal{S} : i \notin \mathcal{N}(i). \tag{1.2}$$

*$\mathcal{N}$ heißt **symmetrisch**, wenn $\forall i, j \in \mathcal{S} : j \in \mathcal{N}(i) \iff i \in \mathcal{N}(j)$.*

*$\mathcal{N}$ heißt **regulär**, wenn es ein $\Theta \in \mathbb{N}$ gibt, so daß $\forall i \in \mathcal{S} : |\mathcal{N}(i)| = \Theta$.*

*Ein $i \in \mathcal{S}$ heißt **lokales Minimum** bezüglich einer Nachbarschaft $\mathcal{N}$, wenn*

$$\forall j \in \mathcal{N}(i) : f(i) \leq f(j). \tag{1.3}$$

□

Im folgenden wird oftmals $\mathcal{S}_i$ als Kurzform von $\mathcal{N}(i)$ verwendet.

Eine solche Nachbarschaft definiert auf der Lösungsmenge $\mathcal{S}$ einen gerichteten Graphen, mit dessen Hilfe $\mathcal{S}$ abgesucht werden kann. Unter dem Begriff „lokale Suche“ faßt man Algorithmen zusammen, die nach dem folgenden Muster vorgehen:

1. Wähle eine Startlösung $i \in \mathcal{S}$.

2. Wähle einen Nachbarn $j \in \mathcal{S}_i$.

3. In Abhängigkeit von den Kostenwerten $f(i)$ und $f(j)$ und anderen Kriterien wird j zur neuen aktuellen Lösung ($i := j$).

4. Wenn das Abbruchkriterium noch nicht erfüllt ist, weiter bei 2.

Die grundlegende Idee besteht darin, daß nicht nur die Bestimmung einer Lösung, sondern auch deren Modifikation in eine andere (benachbarte) effizient durchführbar ist. Bei *NP*-vollständigen Problemen kann man jedoch nicht erwarten, in polynomieller Zeit eine Lösung aus $\mathcal{S}_{opt}$ zu finden oder gar ganz $\mathcal{S}$ absuchen zu können. Daher sind Strategien für lokale Suchalgorithmen interessant, die mit einer gewissen Wahrscheinlichkeit ein globales Minimum finden oder zumindest ein lokales Minimum, für das die Kostenfunktion f einen sehr kleinen Wert annimmt.

Verwendet man für die lokale Suche ein Greedyverfahren, das zu einer anderen Lösung nur übergeht, wenn dadurch ein Kostenvorteil entsteht, so findet man lediglich ein lokales Optimum, dessen Güte beliebig schlecht sein kann. Läuft ein solcher Algorithmus einmal in ein lokales Optimum, hat er keine Möglichkeit, es wieder zu verlassen, um eventuell bessere Bereiche des Lösungsraumes zu finden. Aus dieser Schwäche ergibt sich die Idee, während der Suche auch Erhöhungen der Kostenfunktion in Kauf zu nehmen. Andererseits müssen Kostenverschlechterungen auch sanktioniert werden, da sonst eine von der Kostenfunktion unabhängige Suche stattfindet. Deren Erfolgschancen sind minimal, weil in der Regel $|\mathcal{S}_{opt}|$ sehr klein gegenüber $|\mathcal{S}|$ ist.

1.2.2 Der Metropolis-Algorithmus

In [MRR$^+$53] stellten Metropolis et al. einen Algorithmus vor, der das Verhalten einzelner Teilchen in einer Schmelze simuliert. Er dient als Grundlage eines Verfahrens zur lokalen Suche, das heute als Metropolis-Algorithmus bekannt ist. Doch zunächst ein kleiner Exkurs in die Physik:

In einer Flüssigkeit hat jedes Teilchen die Möglichkeit, seine eigene Lage zu verändern. Es ist bestrebt, die für sich energetisch günstigste Position zu seiner Umgebung einzunehmen. Durch die dem Gesamtsystem innewohnende thermische Energie führen die Teilchen aber Bewegungen aus, die sie immer wieder in ungünstigere Positionen bringen. Verbessert sich die energetische Situation eines Teilchens, so wird diese Energiedifferenz frei, während für eine Verschlechterung Energie gebunden wird. Die Wahrscheinlichkeit, daß ihm bei einer Temperatur T genügend Energie zur Verfügung steht, eine Verschlechterung der Größe ΔE auszuführen, ist

$$e^{\left(\frac{\Delta E}{k_B \cdot T}\right)}, \tag{1.4}$$

wobei k_B für die Boltzmann-Konstante steht.

Diese Erkenntnisse werden anhand folgender Analogie auf einen Optimierungsalgorithmus übertragen: Die Menge aller Zustände, die die Schmelze in Bezug auf die Positionen ihrer Teilchen annehmen kann, bildet die Lösungsmenge $\mathcal{S}$. Die Nachbarn eines solchen Zustandes entstehen durch Positionswechsel jeweils eines Teilchens. An dieser Stelle findet eine Sequentialisierung der massiv parallelen Vorgänge in der Schmelze statt. Die Summe der energetischen Positionen der Teilchen entspricht der Kostenfunktion, deren Minimum im perfekten Kristallgitter angenommen wird.

Um das Verhalten des Gesamtsystems zu simulieren, wird ein Nachbar des aktuellen Zustandes zufällig ausgewählt und die für seine Annahme notwendige Energiedifferenz bestimmt. Im Fall einer Verschlechterung wird mit Wahrscheinlichkeit (1.4), bei einer Verbesserung mit Wahrscheinlichkeit 1 zu ihm übergegangen. Algorithmus 1 zeigt das analoge Vorgehen für Optimierungsprobleme.

Algorithmus 1 Metropolis

Eingabe: $c \in \mathbb{R}^+$
  [*wähle Startlösung* $i \in \mathcal{S}$]
  repeat
    [*wähle gleichverteilt zufällig* $j \in \mathcal{S}_i$]
    if $(f(j) \leq f(i))$ **then**
      $i := j$
    else
      [$i := j$ *mit Wahrscheinlichkeit* $e^{\left(\frac{f(i)-f(j)}{c}\right)}$]
    end if
  until ([*Abbruchbedingung*])

Der Parameter c übernimmt hier die Rolle der Temperatur und der Konstante k_B. Man kann ihn aber auch als Normalisierung der Kostenfunktion begreifen. Für $c \to 0$ ähnelt der Algorithmus einer Greedy-Heuristik, da nur Verbesserungen angenommen werden. Dies entspricht dem Einfrieren der Schmelze bei Temperatur 0. Jedes Teilchen nimmt dabei die lokal günstigste Position ein und der entstehende Festkörper ist weit von einer Kristallstruktur entfernt. Bei $c \to \infty$ hat die Kostenfunktion keinen Einfluß auf die Annahme eines Nachbarn mehr, so daß eine rein zufällige Suche im Lösungsraum stattfindet. Eine Schmelze bleibt zwar bei Temperaturen, die gegen unendlich gehen, nicht mehr flüssig, aber die Positionen einzelner Teilchen werden immer weniger von ihren Energiewerten beeinflußt, so daß man als Grenzwert von einem gänzlich zufälligen Zustand sprechen kann.

Im folgenden soll nun untersucht werden, ob der Metropolis-Algorithmus aus theoretischer Sicht als Optimierungsverfahren geeignet ist. Zu diesem Zweck wird er als Markov-Kette betrachtet. (x_t) sei die Folge zufälliger Variablen über $\mathcal{S}$, in der x_t die aktuelle Lösung i nach t Schleifendurchläufen ist. $a(t)$ ist ein $|\mathcal{S}|$-dimensionaler Vektor und beinhaltet für jede Lösung aus $\mathcal{S}$ die Wahrscheinlichkeit, nach dem t-ten Schleifendurchlauf aktuelle Lösung zu sein. $a(0)$ enthält die Verteilung nach der Wahl der Startlösung.

$$\forall t \in \mathbb{N} : a_i(t) := \mathbb{P}\{x_t = i\} \tag{1.5}$$

Die Folge der Übergangsmatrizen $(P(t))$ der Markov-Kette definiert sich demnach durch

$$\forall t \in \mathbb{N}^+ : P_{ij}(t) := \mathbb{P}\{x_t = j | x_{t-1} = i\}. \tag{1.6}$$

$\Longrightarrow$

$$\forall i \in \mathcal{S}, t \in \mathbb{N}^+ : a_i(t) = \sum_{h \in \mathcal{S}} a_h(t-1) \cdot P_{hi}(t) \tag{1.7}$$

$P_{ij}(t)$ läßt sich für $i \neq j$ zerlegen in $G_{ij}(t)$, die Wahrscheinlichkeit, daß j aus der Nachbarschaftsmenge $\mathcal{S}_i$ gewählt wird, und in die Akzeptanzwahrscheinlichkeit $A_{ij}(t)$. $G(t)$ und $A(t)$ werden auch als Generations- und Akzeptanzmatrix bezeichnet.

$\forall i, j \in \mathcal{S}, t \in \mathbb{N}^+ :$

$$G_{ij} = G_{ij}(t) = \begin{cases} \frac{1}{|\mathcal{S}_i|} & ; j \in \mathcal{S}_i \\ 0 & ; sonst \end{cases} \tag{1.8}$$

$$A_{ij} = A_{ij}(t) = e^{\left(\frac{\min(0, f(i) - f(j))}{c}\right)} \tag{1.9}$$

$$P_{ij} = P_{ij}(t) = \begin{cases} G_{ij}(t) \cdot A_{ij}(t) & ; i \neq j \\ 1 - \sum_{l \in \mathcal{S} \wedge l \neq i} P_{il}(t) & ; i = j \end{cases} \tag{1.10}$$

Da sowohl $G(t)$ als auch $A(t)$ und damit $P(t)$ konstant über t sind, ist die mit dem Metropolis-Algorithmus assoziierte Markov-Kette (x_t) homogen, weshalb im folgenden nur noch G, A und P benutzt werden.

Um zeigen zu können, daß $a(t)$ konvergiert, muß zunächst der Konvergenzbegriff für Markov-Ketten geklärt werden.

Definition 1.3 (nach Feller 1950)
*Eine Markov-Kette (x_t) **konvergiert** gegen eine **stationäre Verteilung** q, wenn q ein $|\mathcal{S}|$-dimensionaler Vektor ist und für dessen Komponenten q_i gilt:*

$$\forall j \in \mathcal{S} : q_i = \lim_{t \to \infty} \mathbb{P}(x_t = i | x_0 = j) \tag{1.11}$$

□

Zu beachten ist hierbei, daß sich der Verteilungsvektor unabhängig von der Ausgangsverteilung, oder im Fall des Metropolis-Algorithmus der Startlösung, an q annähern muß.

Definition 1.4
*(x_t) erreicht ihr **Equilibrium** (Gleichgewicht) bei einem $t_0 \in \mathbb{N}^+$, wenn für alle $t \geq t_0$ der Grenzwert aus (1.11) angenommen wird, also gilt:*

$$\forall t \geq t_0 : a(t) = q \tag{1.12}$$

*(x_t) erreicht ihr **Quasiequilibrium** für ein ε bei einem $t_0 \in \mathbb{N}^+$, wenn gilt:*

$$\forall t \geq t_0 : \|a(t) - q\| < \varepsilon \tag{1.13}$$

□

In der Literatur wird der Begriff des Quasiequilibriums häufig sehr informal verwendet. So wird meistens kein bestimmtes ε angegeben, da lediglich ausgedrückt werden soll, daß sich $a(t)$ „sehr dicht" an q angenähert hat. Auch in dieser Arbeit wird er nur zur informalen Argumentation benutzt.

Definition 1.5
*Eine homogene Markov-Kette über einer Menge $\mathcal{S}$ und mit Übergangsmatrix P heißt **irreduzibel**, falls*

$$\forall i, j \in \mathcal{S} : \exists n \in \mathbb{N}^+ : (P^n)_{ij} > 0. \tag{1.14}$$

*Sie heißt **aperiodisch**, falls*

$$\forall i \in \mathcal{S} : \mathrm{ggT}(D_i) = 1 \tag{1.15}$$

$$\textit{mit } D_i := \{n \in \mathbb{N}^+ | (P^n)_{ii} > 0\}. \tag{1.16}$$

□

Lemma 1.6
Eine irreduzible Markov-Kette ist aperiodisch, wenn

$$\exists i \in \mathcal{S} : P_{ii} > 0. \tag{1.17}$$

Beweis: Aus der Irreduzibilität folgt

$$\forall i, j \in \mathcal{S} : \exists k, l \in \mathbb{N}^+ : (P^k)_{ij} > 0 \wedge (P^l)_{ji} > 0.$$

Mit $n := k + l$ gilt:

$$\begin{aligned}(P^n)_{ii} &\geq (P^k)_{ij} \cdot (P^l)_{ji} > 0 \\ (P^{n+1})_{ii} &\geq (P^k)_{ij} \cdot P_{jj} \cdot (P^l)_{ji} > 0 \\ &\Longrightarrow n, n+1 \in D_i \\ &\Longrightarrow \mathrm{ggT}(D_i) = 1\end{aligned}$$

□

Theorem 1.7 (Feller 1950)
Eine homogene, irreduzible, aperiodische Markov-Kette mit Übergangsmatrix P konvergiert gegen eine stationäre Verteilung q, die eindeutig definiert ist durch

$$q^T = q^T P. \tag{1.18}$$

Gleichung (1.18) wird als „globale Gleichgewichtsbedingung“ bezeichnet. Sie besagt, daß q linker Eigenvektor von P mit Eigenwert 1 ist. Es ist leicht einzusehen, daß ein Grenzwert, falls er existiert, diese Bedingung erfüllen muß. Die entscheidende Aussage in Theorem 1.7 ist die Existenz dieses Grenzwertes.

Lemma 1.8
Eine homogene, irreduzible, aperiodische Markov-Kette mit Übergangsmatrix P konvergiert gegen eine gegebene Verteilung q, wenn

$$q_i P_{ij} = q_j P_{ji}. \tag{1.19}$$

Beweis:

$$\begin{aligned}
q_i P_{ij} &= q_j P_{ji} \\
\sum_{j \in \mathcal{S}} q_i P_{ij} &= \sum_{j \in \mathcal{S}} q_j P_{ji} \\
q_i &= \sum_{j \in \mathcal{S}} q_j P_{ji} \qquad (P \text{ ist stochastisch}) \\
q^T &= q^T P
\end{aligned}$$

□

Die in diesem Lemma stehende Gleichung (1.19) wird im Gegensatz zu (1.18) „lokale Gleichgewichtsbedingung“ genannt. Anschaulich gesehen fordert sie, daß eine Kante zwischen zwei Elementen der zugrunde liegenden Menge in beiden Richtungen mit gleicher Wahrscheinlichkeit benutzt wird.

Der folgende Satz formuliert die Anwendbarkeit von Lemma 1.8 bzw. Theorem 1.7 auf die Markov-Kette des Metropolis-Algorithmus.

Satz 1.9 (Aarts, Korst 1989)
Sei $(\mathcal{S}, f)$ eine Instanz eines kombinatorischen Optimierungsproblems und $\mathcal{N}$ eine symmetrische Nachbarschaftsfunktion, die einen zusammenhängenden Graphen auf $\mathcal{S}$ induziert. Sei weiterhin f nicht konstant und ein $c \in \mathbb{R}^+$

gegeben. Dann konvergiert die zum Metropolis-Algorithmus gehörende Markov-Kette gegen eine stationäre Verteilung q, definiert durch

$$\begin{aligned} q_i &:= \frac{|\mathcal{S}_i| \cdot e^{\left(\frac{-f(i)}{c}\right)}}{N_0} && (1.20) \\ \textit{mit } N_0 &:= \sum_{j \in \mathcal{S}} |\mathcal{S}_j| \cdot e^{\left(\frac{-f(j)}{c}\right)}. && (1.21) \end{aligned}$$

Beweis: (Reduktion auf Lemma 1.8)

I Irreduzibilität:
Da $\mathcal{N}$ einen zusammenhängenden Graphen auf $\mathcal{S}$ induziert, gibt es für alle $i, j \in \mathcal{S}$ einen Pfad $(l_0, \ldots, l_p) \in \mathcal{S}^p$ von i nach j. Die Wahrscheinlichkeit eines Überganges von i nach j in p Schritten ist mindestens so hoch wie die, daß genau dieser Pfad dafür genutzt wurde:

$$\begin{aligned} (P^p)_{ij} &\geq \prod_{i=0}^{p-1} P_{l_i l_{i+1}} \\ &= \prod_{i=0}^{p-1} (G_{l_i l_{i+1}} \cdot A_{l_i l_{i+1}}) \\ &> 0 \end{aligned}$$

Damit ist (1.14) erfüllt und die zu P gehörende Markov-Kette irreduzibel.

II Aperiodizität:
Seien $i, j \in \mathcal{S}$ mit $f(i) < f(j)$ und $j \in \mathcal{S}_i$. Solche i und j existieren, da f nicht konstant und der induzierte Graph zusammenhängend ist.

$$\begin{aligned} A_{ij} &= e^{\left(\frac{f(i)-f(j)}{c}\right)} < 1 \\ P_{ii} &= 1 - \sum_{l \in \mathcal{S}, l \neq i} G_{il} A_{il} \\ &= 1 - G_{ij} A_{ij} - \sum_{l \in \mathcal{S}, l \neq i,j} G_{il} A_{il} \\ &> 1 - G_{ij} - \sum_{l \in \mathcal{S}, l \neq i,j} G_{il} \\ &= 1 - \sum_{l \in \mathcal{S}, l \neq i} G_{il} \\ &= 0 \end{aligned}$$

Nach Lemma 1.6 folgt aus der Irreduzibilität die Aperiodizität der Markov-Kette.

III lokale Gleichgewichtsbedingung für q aus (1.20) und (1.21)

1. Fall $i = j$
$$q_i P_{ij} = q_j P_{ji} \text{ (trivial)}$$
2. Fall $i \neq j \wedge i \notin \mathcal{S}_j (\Longrightarrow j \notin \mathcal{S}_i)$
$$q_i P_{ij} = 0 = q_j P_{ji}$$
3. Fall $i \neq j \wedge i \in \mathcal{S}_j (\Longrightarrow j \in \mathcal{S}_i)$
   Sei o. B. d. A. $f(i) \geq f(j)$.

$$\begin{aligned}
q_i \cdot P_{ij} &= \frac{|\mathcal{S}_i| \cdot e^{\left(\frac{-f(i)}{c}\right)}}{N_0} \cdot \frac{1}{|\mathcal{S}_i|} \cdot 1 \\
&= \frac{|\mathcal{S}_j| \cdot e^{\left(\frac{-f(j)}{c}\right)}}{N_0} \cdot \frac{1}{|\mathcal{S}_j|} \cdot \frac{e^{\left(\frac{-f(i)}{c}\right)}}{e^{\left(\frac{-f(j)}{c}\right)}} \\
&= \frac{|\mathcal{S}_j| \cdot e^{\left(\frac{-f(j)}{c}\right)}}{N_0} \cdot \frac{1}{|\mathcal{S}_j|} \cdot e^{\left(\frac{f(j)-f(i)}{c}\right)} \\
&= q_j \cdot P_{ji}
\end{aligned}$$

□

Die Voraussetzungen des Satzes 1.9 stellen keine besonders starken Einschränkungen dar. Bei konstantem f verliert die Optimierung ihren Sinn, und einen zusammenhängenden Nachbarschaftsgraphen zu fordern ist durchaus vernünftig, da kein Verfahren der lokalen Suche Lösungen außerhalb der Zusammenhangskomponente der Startlösung finden kann. Allein die Forderung nach einer symmetrischen Nachbarschaft könnte sich als problematisch erweisen; sie ergibt sich allerdings bei vielen Problemen ganz natürlich.

Um den Grenzwert q bei verschiedenen Werten für c zu untersuchen, wird er im folgenden als $q(c)$ geschrieben.

$$q_i(c) = \frac{|\mathcal{S}_i| \cdot e^{\left(\frac{-f(i)}{c}\right)}}{N_0(c)} \tag{1.22}$$
$$N_0(c) = \sum_{j \in \mathcal{S}} |\mathcal{S}_j| \cdot e^{\left(\frac{-f(j)}{c}\right)} \tag{1.23}$$

Für sehr große c führt der Metropolis-Algorithmus tatsächlich die erwartete „blinde Suche“ durch:
$$\lim_{c \to \infty} q_i(c) = \frac{|\mathcal{S}_i|}{\sum_{j \in \mathcal{S}} |\mathcal{S}_j|} \tag{1.24}$$
Die Wahrscheinlichkeit einer Lösung ist nur noch von ihrer Nachbarschaftsgröße abhängig. Die Kostenfunktion hat also keinerlei Einfluß darauf.

Interessanter ist der Fall sehr kleiner c. Der Grenzwert bei $c \to 0$ wird in q^* festgehalten:

$$\begin{aligned} q_i^* &:= \lim_{c\to 0} q_i(c) && (1.25)\\ &= \lim_{c\to 0} \frac{|\mathcal{S}_i| \cdot e^{(\frac{-f(i)}{c})}}{\sum_{j\in\mathcal{S}} |\mathcal{S}_j| \cdot e^{(\frac{-f(j)}{c})}} \\ &= \lim_{c\to 0} \frac{|\mathcal{S}_i| \cdot e^{(\frac{f_{opt}-f(i)}{c})}}{\sum_{j\in\mathcal{S}} |\mathcal{S}_j| \cdot e^{(\frac{f_{opt}-f(j)}{c})}} \\ &= \frac{|\mathcal{S}_i| \cdot \chi_{\mathcal{S}_{opt}}(i)}{\sum_{j\in\mathcal{S}} |\mathcal{S}_j| \cdot \chi_{\mathcal{S}_{opt}}(j)} \\ &= \frac{|\mathcal{S}_i| \cdot \chi_{\mathcal{S}_{opt}}(i)}{\sum_{j\in\mathcal{S}_{opt}} |\mathcal{S}_j|} && (1.26) \end{aligned}$$

$\chi_{\mathcal{S}_{opt}}$ ist wie üblich definiert:

$$\forall i \in \mathcal{S} : \chi_{\mathcal{S}_{opt}}(i) := \begin{cases} 1\,;\, i \in \mathcal{S}_{opt} \\ 0\,;\, i \notin \mathcal{S}_{opt} \end{cases} \tag{1.27}$$

Nur für Lösungen aus $\mathcal{S}_{opt}$ nimmt (1.26) positive Werte an. Es gilt daher:

$$\lim_{c\to 0} \lim_{t\to\infty} \mathbb{P}\{x_t \in \mathcal{S}_{opt}\} = 1 \tag{1.28}$$

Will man den Metropolis-Algorithmus zur Optimierung verwenden, scheint sich ein sehr kleiner Wert für c anzubieten. Da allerdings die Konvergenzgeschwindigkeit mit c abnimmt, muß für die Anwendung in der Praxis zwischen einer schnelleren Konvergenz und einer höheren Wahrscheinlichkeit optimaler Lösungen ($\mathbb{P}\{x_t \in \mathcal{S}_{opt}\}$) abgewogen werden.

1.2.3 Konvergenz durch Kühlen

Aus Gleichung (1.28) folgt, daß bei einer unendlichen Folge von Metropolis-Algorithmen mit jeweils unendlicher Laufzeit die gefundenen Lösungen mit gegen 1 gehender Wahrscheinlichkeit aus $\mathcal{S}_{opt}$ sind. Für die praktische Anwendung scheint diese Aussage zunächst völlig uninteressant zu sein. Doch was geschieht, wenn man die im Metropolis-Algorithmus konstante Temperatur während der Optimierung kleiner werden läßt? Kann man so in einer Art Diagonalisierung Ausschnitte aus verschiedenen Metropolis-Läufen in einem vereinigen und trotzdem den Grenzwert aus (1.28) erhalten?

Betrachten wir dazu Algorithmus 2, das Simulated Annealing. Statt einer konstanten Temperatur wird hier eine gegen 0 konvergierende Folge (c_k) verwendet. Doch diese Temperaturänderungen werden nicht in jedem Schritt vollzogen. Vielmehr gibt es Phasen mit konstanter Temperatur, die im folgenden „Level" genannt werden und deren Längen durch die Folge (L_k) gegeben sind. Die for-Schleife entspricht einem Lauf des Metropolis-Algorithmus, während die repeat-until-Schleife über die Level läuft.

Algorithmus 2 Simulated Annealing

```
Eingabe:  Folge c_0, c_1, ... ∈ ℝ⁺     lim_{k→∞} c_k = 0
          Folge L_0, L_1, ... ∈ ℕ⁺
  [ wähle Startlösung i ∈ S ]
  k := 0
  repeat
    for l := 0 to L_k − 1 do
      [ wähle gleichverteilt zufällig j ∈ S_i ]
      if (f(j) ≤ f(i)) then
        i := j
      else
        [ i := j mit Wahrscheinlichkeit e^((f(i)−f(j))/c_k) ]
      end if
    end for
    k := k + 1
  until ( [Abbruchbedingung] )
```

Die Idee dabei ist, daß am Ende jedes Levels ein Quasiequilibrium erreicht wird. Im ersten Level soll die Temperatur hoch genug sein, damit es aufgrund der schnellen Konvergenz angenommen wird. In allen Nachfolgeleveln geht man davon aus, daß sich bei einer sehr kleinen Temperaturänderung der Verteilungsgrenzwert nur minimal von dem des vorherigen Levels unterscheidet, weshalb das neue Quasiequilibrium trotz langsamerer Konvergenz schnell gefunden wird.

Die Voraussetzung für eine Konvergenz ist also ein sehr langsames Abkühlen oder langes Verharren bei einer Temperatur. Hier spiegeln sich entsprechende Erkenntnisse aus dem physikalischen Vorbild wider, wo nur ein langsames Kühlen der Schmelze zur Ausbildung des Einkristalls führt. Eine phasenweise Temperaturreduktion ist dort aufgrund der massiven Parallelität nicht nötig.

Die Folgen (c_k) und (L_k) sowie die Abbruchbedingung werden unter dem Begriff „Kühlschema" zusammengefaßt.

In [Haj88] untersucht Hajek das Konvergenzverhalten des Simulated-Annealing-Algorithmus. Aus dieser Arbeit möchte ich ein entsprechendes Theorem vorstellen, das hier aber nicht bewiesen wird. Doch zuvor die Definition dreier wichtiger Begriffe.

Definition 1.10
*Seien $i, j \in \mathcal{S}$ und $h \in \mathbb{R}$. j ist von i **erreichbar auf Höhe** h, wenn es einen Pfad $(l_0, \ldots, l_p) \in \mathcal{S}^p$ von i nach j gibt, so daß*

$$\forall l \in \{l_0, \ldots, l_p\} : f(l) \leq h. \tag{1.29}$$

$\square$

Definition 1.11
*Die auf einer Instanz $(\mathcal{S}, f)$ eines kombinatorischen Optimierungsproblems definierte Nachbarschaft ist **schwach symmetrisch**, wenn für alle $i, j \in \mathcal{S}$ und alle $h \in \mathbb{R}$ gilt:*
i ist auf Höhe h von j erreichbar $\iff$ j ist auf Höhe h von i erreichbar. $\square$

Definition 1.12
*Sei i_l ein lokales, aber nicht globales Optimum. Die **Tiefe** von i_l ist die kleinste reelle Zahl $d(i_l)$, für die es ein $j \in \mathcal{S}$ mit $f(j) < f(i_l)$ gibt, das von i_l auf Höhe $d(i_l)$ erreichbar ist.* $\square$

Die schwache Symmetrie ist tatsächlich eine geringere Forderung an eine Nachbarschaft als die Symmetrie aus Definition 1.2, da aus letzterer die erste folgt. Für das folgende Theorem genügt die schwache Form. Die Tiefe eines lokalen Optimums gibt an, welche Verschlechterung ein Algorithmus in Kauf nehmen muß, um aus ihm zu „entkommen".

Theorem 1.13 (Hajek 1988)
Sei $(\mathcal{S}, f)$ eine Instanz eines kombinatorischen Optimierungsproblems, auf der eine schwach symmetrische Nachbarschaft $\mathcal{N}$ definiert ist. Sei weiterhin d^* das Maximum der Tiefen aller lokaler Optima, die nicht globale Optima sind. Für einen Lauf des Simulated-Annealing-Algorithmus gilt

$$\lim_{t \to \infty} \mathbb{P}\{x(t) \in \mathcal{S}_{opt}\} = 1 \iff \sum_{t=1}^{\infty} e^{\left(\frac{-d^*}{c(t)}\right)} = \infty, \tag{1.30}$$

wobei $x(t)$ die aktuelle Lösung und $c(t)$ die aktuelle Temperatur zum Zeitpunkt t (während des t-ten Laufes der for-Schleife) darstellt.

Bemerkung 1.14
Ein Kühlschema mit

$$c_k \geq \frac{d^*}{\log(k+1)} \tag{1.31}$$

erfüllt die rechte Seite der Äquivalenz (1.30) unabhängig von der Wahl der (L_k). $\square$

Man beachte, daß Theorem 1.13 nicht nur ein hinreichendes, sondern sogar ein notwendiges Konvergenzkriterium liefert. Ein beliebiges Kühlschema kann also hinsichtlich seiner Konvergenz untersucht werden.

Doch wie verhält es sich mit der Konvergenzgeschwindigkeit? Mitra, Romeo und Sangiovanni-Vincentelli zeigten in [MRS86], daß für das logarithmische Kühlschema aus (1.31) gilt:

$$k \geq \mathcal{O}\left(\left(\frac{1}{\varepsilon}\right)^{(L+1)\cdot\max(a,b)}\right) \Longrightarrow \|a(k) - q^*\| < \varepsilon \qquad (1.32)$$

mit

$$a = \Theta^{L+1} \qquad (1.33)$$

$$b = \frac{\max_{i\in\mathcal{S}} f(i) - f_{opt}}{\min_{i\in\mathcal{S}\setminus\mathcal{S}_{opt}} f(i) - f_{opt}}, \qquad (1.34)$$

wobei Θ die maximale Nachbarschaftsgröße ist.

Für die meisten interessanten Probleme führt (1.32) allerdings zu Werten für k, die die Größe des Lösungsraumes $\mathcal{S}$ weit übersteigen. Als exaktes Lösungsverfahren ist Simulated Annealing also völlig ungeeignet, da eine vollständige Enumeration schneller läuft.

Verwendet man ein nichtkonvergentes Kühlschema, das dafür aber nur polynomielle Zeit benötigt, erhält man eine randomisierte Heuristik, die sich für diverse Probleme als sehr erfolgreich erwiesen hat. Man kann zwar nicht erwarten, damit optimale Lösungen zu produzieren, aber unsere Tests zeigen, daß für unser Problem die erreichte Lösungsqualität die der Greedy-Ansätze weit übersteigt.

Üblicherweise werden noch kleinere Veränderungen am Simulated-Annealing-Algorithmus vorgenommen, um zu besseren Lösungen zu gelangen. So kann beispielsweise die während der gesamten Laufzeit beste gefundene Lösung statt der nach dem letzten Schritt des Annealing vorliegenden als Ergebnis verwendet werden, falls es möglich ist, sich eine Lösung effizient zu „merken“. Eine andere Variante besteht in einem sogenannten „Quenching“ der gefundenen Lösung; einem abschließenden Greedy-Lauf, der zumindest lokale Optimalität garantiert.

Kapitel 2

Modellierung und Implementation

2.1 Lösungsraum und Nachbarschaft

Eine Lösung der Pairing-Generierung ist eine Zerlegung der Legmenge $\mathcal{L}$ in Pairings. Als Repräsentation haben wir eine Abbildung gewählt, die jedem Leg einen Nachfolger oder ein Nullelement zuordnet.

$$succ : \mathcal{L} \to \mathcal{L} \cup \{\perp\} \tag{2.1}$$

Um den Lösungsraum nicht unnötig zu vergrößern, beschränken wir uns auf Zerlegungen in räumlich und zeitlich konsistente, also „sinnvolle" Pairings. Für jedes Leg ℓ wird eine Menge zulässiger Nachfolger $\mathcal{L}_\ell \subset \mathcal{L}$ definiert und nur noch solche *succ* betrachtet, für die gilt:

$$\forall \ell \in \mathcal{L} : succ(\ell) \in \mathcal{L}_\ell \cup \{\perp\} \tag{2.2}$$

Eine weitere Einschränkung auf tarifvertraglich zulässige Pairings wurde an dieser Stelle nicht vorgenommen (vgl. Kapitel 3.2).

Aus einer Abbildung *succ* ergibt sich die Menge der Pairings $\mathcal{P}_{succ}$. Neben den enthaltenen Legs wird jedem Pairing als zusätzliches Attribut eine Homebase zugeordnet, in der der im Rostering zuzuweisende Mitarbeiter wohnen sollte. Die Menge der Homebases wird im folgenden als $\mathcal{H}$ bezeichnet. Betrachten wir zwei Varianten der zu definierenden Lösungsmenge $\mathcal{S}$. Sei $\mathcal{S}'$ die Lösungsmenge ohne Berücksichtigung der Homebasezuordnung, also einfach die Menge der gemäß (2.2) „sinnvollen" Abbildungen *succ*. $\mathcal{S}''$ hingegen besteht aus Paaren $(succ; home)$, also je einer Nachfolgerabbildung *succ* und einer Homebasezuordnung $home : \mathcal{P}_{succ} \to \mathcal{H}$. Eine Pairingzerlegung der Legs mit zwei verschiedenen Homebaseverteilungen stellen in $\mathcal{S}'$

dieselbe und in $\mathcal{S}''$ zwei unterschiedliche Lösungen dar. Eine Lösung *succ* aus $\mathcal{S}'$ befindet sich in $|\mathcal{H}|^{|\mathcal{P}_{succ}|}$ verschiedenen Varianten in $\mathcal{S}''$.

$$|\mathcal{S}''| = \sum_{succ \in \mathcal{S}'} |\mathcal{H}|^{|\mathcal{P}_{succ}|} \gg |\mathcal{S}'| \tag{2.3}$$

Wir haben uns bei der Definition des Lösungsraumes für die kleinere Variante $\mathcal{S}'$ entschieden.

$$\mathcal{S} := \mathcal{S}' \tag{2.4}$$

Die Zuordnung der Homebases ist als Teil der Kostenfunktion $f_{\mathcal{S}} : \mathcal{S} \rightarrow \mathbb{R}$ modelliert. Im folgenden Kapitel 2.2.1 wird eine Funktion $f_{\mathcal{S}''} : \mathcal{S}'' \rightarrow \mathbb{R}$ vorgestellt, mit deren Hilfe $f_{\mathcal{S}}$ gebildet wird. Eine genaue Erläuterung hierzu findet sich in Kapitel 2.2.2; bis dahin nehmen wir an, daß eine Funktion $h : \mathcal{S} \rightarrow \mathcal{S}''$ existiert, die jeder Lösung $succ \in \mathcal{S}(= \mathcal{S}')$ eine Lösung $h(succ) \in \mathcal{S}''$ mit identischer Pairingzerlegung und bezüglich der Kosten optimaler Homebaseverteilung zuordnet. Mit Hilfe von h berechnet sich $f_{\mathcal{S}}$ wie folgt:

$$\forall succ \in \mathcal{S} : f_{\mathcal{S}}(succ) := f_{\mathcal{S}''}(h(succ)) \tag{2.5}$$

Bei der Wahl einer Nachbarschaftsfunktion haben wir auf die naheliegenste Variante zurückgegriffen, zwei Operatoren namens **Join** und **Split**. Ein Join ist die Verschmelzung zweier Pairings zu einem, während ein Split die Aufspaltung eines Pairings an einer beliebigen Stelle darstellt. Zwei Lösungen aus $\mathcal{S}$ sind benachbart, wenn sie durch Anwendung genau eines dieser Operatoren auseinander hervorgehen. Da Join und Split invers zueinander sind, ist die durch sie definierte Nachbarschaft $\mathcal{N}$ symmetrisch.

2.2 Kostenfunktion

2.2.1 Kostenarten

Im Laufe der Entwicklung des Prototyps und mit zunehmender Verfeinerung der Modellierung haben sich sieben Kostenarten herausgebildet, die sich in drei Gruppen einteilen lassen:

- monetäre Kosten (solche, die sich zumindest innerbetrieblich in Geldeinheiten ausdrücken lassen)
  - Kosten für Proceedings („PROC_COST“)
  - Übernachtungskosten („HOTEL“)

  - Arbeitszeit außerhalb der Flugzeit („XWORK")

- nichtmonetäre Kosten (zur Berücksichtigung von gewünschten Eigenschaften einer Lösung, die sich nicht in Geld messen lassen)
  - Rotationswechsel („ROT_CHANGES")
  - Bestrafung zu kurzer Pairings („SHORTIES")

- Nebenbedingungen
  - Relaxierung der Zulässigkeit („ADMISS")
  - Verteilung der Pairing-Homebases („HOME")

Bei der folgenden Erläuterung der einzelnen Kostenarten wird weiterhin von einer durch h gegebenen Homebasezuordnung der einzelnen Pairings ausgegangen.

PROC_COST Wenn ein Pairing nicht an der ihm zugewiesenen Homebase beginnt oder endet, werden Proceedings von der Homebase zum Startflughafen bzw. vom Endflughafen zur Homebase nötig. Die daraus tatsächlich entstehenden Kosten zu bestimmen ist nicht trivial. Im Idealfall müßte ein Bewertungsalgorithmus die Flugpläne und Preislisten aller Gesellschaften zur Verfügung haben sowie eine Abschätzung entstehender Kosten für andere Verkehrsmittel (Taxi, Bahn) vornehmen können. Da uns diese Daten nicht vorlagen und sich ein solcher Aufwand für einen Prototyp auch nicht rechtfertigen läßt, werden Proceedings zwischen zwei „großen" Flughäfen mit 1 und solche zu oder von einem „kleineren" mit 2 bestraft und in „PROC_COST" summiert. Die Unterteilung in „groß" und „klein" wird beim Einlesen der Daten anhand der Zahl ankommender und abgehender Flüge vorgenommen.

Im Prototyp ist bereits eine bessere Abschätzung der Proceedingkosten über eine Distanzmatrix der Flughäfen vorgesehen. Diese Matrix enthält die durchschnittlichen Proceedingkosten zwischen je zwei Flughäfen ohne Berücksichtigung der Tageszeit und kann aus einer weiteren Eingabedatei gelesen oder anhand der vorliegenden Flugdaten errechnet werden. Benutzt wird sie allerdings nicht, da beim Rostering die dort tatsächlich eingefügten Pairings derzeit konstant bewertet werden. Eine bei der Pairinggenerierung geschaffene Möglichkeit, mit besonders billigen Proceedings auszukommen, würde also beim Rostering nicht genutzt werden.

XWORK Eine untere Schranke für die Summe der Arbeitszeiten aller Mitarbeiter ist die Summe der Flugzeiten, die für die Legs notwendig werden (Länge des Legs mal Größe des Crewkomplements). Zur Arbeitszeit eines Mitarbeiters kommen aber noch die Zeiten hinzu, die zwischen zwei Legs einer Schicht liegen. Durch eine Pairingzerlegung werden Folgen von Legs und damit die Zeit zwischen ihnen fixiert. Sie stellt damit eine weitere untere Schranke der Gesamtarbeitszeit für das Rostering dar, da dort nur komplette Pairings in die Roster eingefügt werden.

Die Differenz zwischen den beiden Schranken, also die Summe aller Zeiten innerhalb der Pairings, die nicht Flugzeit sind, wird in „XWORK" (von „eXtra WORK") gemessen. Durch ihre Minimierung gibt man dem Rostering die Möglichkeit, die Gesamtarbeitszeit und damit Lohnkosten zu senken.

HOTEL Übernachtungen, die nicht an der Homebase stattfinden, werden in „HOTEL" gezählt. Eine Differenzierung der Übernachtungskosten nach Flughäfen findet hierbei nicht statt, da uns keine entsprechenden Daten zur Verfügung standen. Eine solche Erweiterung wäre aber rein technischer Natur. Da Übernachtungen auch als „XWORK" zählen, kommt dieser Kostenart eine geringere Bedeutung zu, als die tatsächlich entstehenden Kosten vermuten lassen. Mit ihr ist es aber möglich, Übernachtungen an der Homebase zu bevorzugen.

ROT_CHANGES Muß die Besatzung innerhalb einer Schicht das Flugzeug (die Flugzeugrotation) wechseln und ist das Zeitfenster zu knapp gehalten, wird der Flugplan instabil, da sich Verspätungen eines Fluges nicht mehr nur innerhalb der eigenen Rotation fortpflanzen, sondern auch auf die Flüge, die das verspätete Personal bedienen soll, und deren Anschlüsse. Solche Wechsel mit kurzen Bodenzeiten sind daher zu vermeiden. Für deren Beurteilung gibt es zwei einstellbare Zeitgrenzen, „MIN_ROT" und „MAX_ROT". Rotationswechsel, die kürzer als „MIN_ROT" sind, werden durch die Struktur des Conngraphen (vgl. Kapitel 2.3) ganz verboten, während längere als „MAX_ROT" keinen Einschränkungen unterliegen. Nur solche zwischen beiden Grenzen werden in der Kostenart „ROT_CHANGES" gezählt und damit bestraft.

Eine sinnvolle Festsetzung von „MIN_ROT" und „MAX_ROT" kann sich nur aus Erfahrungswerten der betrieblichen Praxis ergeben. Wir haben für unsere Versuche mit dem Prototyp Werte von zwei bzw. vier Stunden gewählt.

Denkbar wäre auch eine linear abfallende Bestrafung zwischen den Grenzen oder eine der Form $\frac{1}{Länge}$ ohne Benutzung irgendwelcher Begrenzungen. Die Struktur der entstehenden Lösungen hat aber eine solche Erweiterung nicht notwendig gemacht.

SHORTIES Im Lauf der Optimierung wurden meist sehr kurze Rundreisen um einen Flughafen als Pairings gebildet. Oft bestanden sie nur aus einem Hin- und Rückflug. Wenn diese Pairings auch noch an ihrer

Homebase beginnen und enden, stellt es keinen Kostengewinn dar, sie zu verknüpfen. Durch den Anstieg der „XWORK“ ergibt sich sogar ein Verlust. Für das Rostering sind aber zu kurze Pairings von Nachteil, weshalb die Kostenart „SHORTIES“ als Strafterm für zu kurze Pairings eingeführt wurde.

Bestraft wird ein Pairing, wenn es zu wenig Legs enthält oder seine Gesamtzeit zu kurz ist. Beide Grenzen sind als Parameter einstellbar. Die Bestrafung erfolgt proportional zur fehlenden Zeit oder Legzahl, wobei ein fehlendes Leg mit einer vorher errechneten durchschnittlichen Leglänge gleichgesetzt wird.

ADMISS Bei „ADMISS“ handelt es sich um die Relaxierung einer notwendigen Bedingung an die Lösung, die Zulässigkeit der Pairings. Um diese zu beurteilen, wurden die wichtigsten Regeln aus [JAA] implementiert. In „ADMISS“ werden die Längen der unzulässigen Pairings addiert und damit bestraft. Die Relaxierung erlaubt es der Optimierungsphase, auch unzulässige Lösungen zu generieren, um über diese zu guten, zulässigen Lösungen zu kommen. Es muß jedoch darauf geachtet werden, daß die endgültige Lösung nur zulässige Pairings enthält, da sie sonst unbrauchbar ist. Entsprechende Untersuchungen sind in Kapitel 3.2 zu finden.

HOME Die Pairinggenerierung kennt zwar im Gegensatz zum Rostering keine konkreten Mitarbeiter und deren Homebases, um aber die entstehenden Pairings gut rostern zu können, sollten Rundreisen vornehmlich um solche Flughäfen konstruiert werden, an denen auch genügend Mitarbeiter wohnen, um diese zu fliegen. Ungleiche Verteilungen der Homebases von Mitarbeitern und gebildeten Rundreisen stellen versteckte Proceedingkosten dar, da Personal von anderen Flughäfen hin- und zurückgebracht werden müßte.

Um dies zu vermeiden, wird aus den Mitarbeiterdaten und diversen Preassignments wie Urlaub, Weiterbildung oder Standby-Diensten eine Kapazitätsverteilung errechnet, die pro Tag und Flughafen die Anzahl der zur Verfügung stehenden Mitarbeiter angibt. Diese wird mit einer aus den jeweilig aktuellen Pairings und deren Homebases bestimmten Auslastungsverteilung verglichen. Es ergibt sich für jede Homebase und jeden Tag eine prozentuale Auslastung, die idealerweise konstant sein und unter 100% liegen sollte.

Um auch Kapazitäten für noch nicht berücksichtigte freie Tage, Standby-Dienste und andere Flotten, mit denen es Personalüberschneidungen gibt, freizuhalten, wird zusätzlich eine prozentuale Höchstauslastung festgelegt. Ihre Überschreitung wird über alle Homebases und Tage summiert und bildet die Kostenart „HOME“.

> Man sollte sich allerdings im klaren sein, daß die tatsächliche Auslastung hier nur approximiert werden kann, da es beispielsweise nicht Aufgabe der Pairinggenerierung sein sollte, vorherzusehen, ob zwei gebildete Pairings, die dicht beieinander liegen, ohne sich zu überschneiden, später von einem Mitarbeiter bedient werden können oder nicht. Es wäre also nicht sinnvoll, für die abschließende Lösung „HOME"-Kosten von null zu fordern, zumal eine ungünstige Verteilung auf die Homebases die Lösung nicht unbrauchbar macht, sondern lediglich verschlechtert. In diesem Sinne sprechen wir von einer „weichen" Nebenbedingung, im Gegensatz zur Kostenart „ADMISS", die auf null gebracht werden muß, um die Lösung verwenden zu können.

Für jede Kostenart gibt es einen sogenannten Kostenfaktor, mit dem die errechneten Werte gewichtet (multipliziert) werden. Die darüber gebildete Summe stellt die Kostenfunktion $f_{\mathcal{S}''}$ dar. Die Wahl der Kostenfaktoren bildet die unternehmerischen Ziele und die spezielle Situation einer Fluggesellschaft ab. So können für ein Unternehmen Proceedings von geringerer Bedeutung als für ein anderes sein, weil die Mitarbeiter auf freien Plätzen eigener Maschinen transportiert werden. Darüber hinaus müssen die nichtmonetären Kosten in Relation zu den monetären gesetzt werden. Vermutlich werden sich die günstigsten Kostenfaktoren in der Praxis nur iterativ im Laufe der Zeit finden lassen, indem Unzulänglichkeiten einer Lösung durch leicht veränderte Einstellungen bei der nächsten Optimierung ausgeräumt werden.

Für die mit dem Prototyp durchgeführten Testläufe mußten wir natürlich auch erst einmal Kostenfaktoren finden. Deren nicht 100%ig richtige Wahl sollte zwar auf die angestrebten Erkenntnisse über Simulated Annealing wenig Einfluß haben, aber wenn die Ergebnisse einem professionellen Handplaner vorgelegt werden sollen, müssen sie schon eine realistische Form haben. Für einen der drei in Kapitel 2.4 vorgestellten Datensätze lag uns eine Handlösung vor, an der wir uns bei der Festlegung der Kostenfaktoren orientieren konnten. Für die anderen beiden haben wir zunächst intuitiv Werte gewählt, die gewonnenen Lösungen dem Industriepartner vorgelegt und anhand dessen Beurteilung in mehreren Schritten eine günstige Einstellung gefunden.

Wenn die wichtigsten Daten über ein Pairing schon aufbereitet in ihm vorliegen und durch die Operatoren Join und Split jeweils aktualisiert werden, ist die Bestimmung der Kostenarten „PROC_COST", „XWORK", „HOTEL", „ROT_CHANGES" und „SHORTIES" für ein Pairing in konstanter und für eine Lösung $i \in \mathcal{S}$ in linearer Zeit möglich. Die Berechnung der Zulässigkeit, also der Kostenart „ADMISS", für ein Pairing benötigt lineare Zeit über der Anzahl seiner Legs, es ergibt sich also auch eine lineare Gesamtlaufzeit. Weil die Anzahl der Legs in einem zulässigen Pairing beschränkt ist und diese

Schranke auch bei unzulässigen in der Regel nicht allzusehr überschritten wird, ergibt sich auch für die Zulässigkeit eines Pairings eine konstante Laufzeit. Die Kostenart „HOME“ ist die einzige, die nicht für einzelne Pairings bestimmt und dann summiert wird. Für ihre Berechnung auf einer Lösung $i \in \mathcal{S}$ ist eine lineare Zeit notwendig, woraus sich für die gesamte Kostenfunktion eine lineare Laufzeit ergibt.

Lokale Suchverfahren, die abgesehen von der Initiallösung nur Kosten benachbarter Lösungen berechnen, profitieren vor allem von einer effizienten Berechnung von Kostendifferenzen zu Nachbarn. Bei einem Join oder Split sind jeweils drei Pairings betroffen (aus zweien werden eines oder umgekehrt). Alle Kostenarten bis auf „HOME“ können in konstanter Zeit für diese drei Pairings berechnet werden und auch für „HOME“ ist die Bestimmung der Differenz in linearer Zeit über der Anzahl der betroffenen Tage, also insgesamt konstanter Zeit möglich.

2.2.2 Die tatsächliche Bestimmung der Homebases

Gäbe es nicht die Kostenart „HOME“, wäre die Bestimmung der optimalen Homebasezuordnung, und damit die Berechnung von h, trivial, da sie für jedes Pairing lokal berechnet werden könnte. Dabei müßten auch nur die beiden Kostenarten „PROC_COST“ und „HOTEL“ betrachtet werden. Diese Berechnung für ein Pairing wäre in über den Homebases linearer Zeit möglich. Betrachtet man die Zahl der Homebases als konstant, ergäbe sich eine konstante Laufzeit und damit eine lineare für die gesamte Lösung.

Mit Einführung der Kostenart „HOME“ wird die Berechnung aufwendiger, da die ingesamt erreichte Auslastung der Homebases Einfluß auf die Wahl der Homebase eines konkreten Pairings hat. Die kostenoptimalen Homebasezuordnungen zweier benachbarter Lösungen $succ_1, succ_2 \in \mathcal{S}$ können sich in mehr als den drei betroffenen Pairings, im schlechtesten Fall sogar in allen Pairings unterscheiden. Eine konstante Laufzeit für die Bestimmung von $h(succ_2)$ bei gegebenem $h(succ_1)$ ist nicht möglich, da linear viele Änderungen nötig sein könnten. Der beste uns bekannte Algorithmus für die vollständige Berechnung von h ist die Enumeration, die bei mehr als einer Homebase exponentielle Laufzeit hat. Ob es möglich ist, h in polynomieller oder gar linearer Laufzeit zu berechnen, bleibt offen.

Aus diesem Grunde wird die bisher angenommene Funktion h aus (2.5) nicht tatsächlich berechnet, sondern nur approximiert. In jedem Pairing gibt es ein Feld, in dem die aktuell zugewiesene Homebase steht, so daß zu jedem Zeitpunkt eine Homebasezuordnung vorliegt. Für die erste betrachtete Lösung wird sie mit Hilfe einer Greedyheuristik ermittelt. Für alle weiteren wird sie iterativ bestimmt, indem bei einem Join oder Split den entstehenden ein oder zwei Pairings die lokal optimale Homebase zugewiesen wird. Dies geschieht unter Beibehaltung der Homebases aller von der Operation nicht betroffenen Pairings.

Zusätzlich wird während des Annealing in regelmäßigen Abständen eine Greedyheuristik eingeschoben, die ungünstige Zuordnungen berichtigen soll. Sie läuft über eine eingestellte Anzahl zufälliger Permutationen der Pairings und berechnet dabei für jedes Pairing die kostengünstigste Homebase anhand der aktuellen Homebasezuordnung der restlichen Lösung. Ein vorzeitiger Abbruch erfolgt, wenn während einer Permutation keine Verbesserung gefunden wurde. Dieser Algorithmus hat lineare Laufzeit unter der mehrfach verwendeten Voraussetzung einer konstanten Zahl von Homebases.

2.3 Datenstrukturen

Eines der vorrangigen Ziele bei der Implementation der Datenstrukturen bestand darin, eine möglichst flexible Schnittstelle für verschiedene Algorithmen anzubieten. Es sollte eine Testgrundlage geschaffen werden, die zwar auf Verfahren der lokalen Suche ausgerichtet ist, aber auch für diverse Heuristiken nutzbar bleibt. Im folgenden werden die wichtigsten Strukturen kurz vorgestellt.

Die grundlegenden Daten für die Pairing-Generierung sind natürlicherweise die Legs, die als Eingabe in einer Datei vorliegen und zunächst in einen binären, selbstbalancierenden Suchbaum (nach [AS89]) eingelesen werden. Zu einem Leg gehören eine eindeutige Flugnummer, Anfangs- und Endzeit, Start- und Zielflughafen, Crewcomplement und die Rotationsnummer.

Die Eingabe besteht neben den Legs noch aus einer Liste der Homebases und einer Matrix der Crewkapazitäten pro Homebase und Planungstag. In einem sinnvollen Datensatz kann man davon ausgehen, daß die Zahl der Legs größer als die der Planungstage ist. Da die Zahl der Homebases als konstant angesehen werden kann, hat die Homebaseliste konstante und die Kapazitätsmatrix höchstens lineare Größe über $|\mathcal{L}|$, der Anzahl der Legs. Daraus folgt, daß auch die Eingabegröße n, mit deren Hilfe die Laufzeiten der Algorithmen charakterisiert werden sollen, linear über $|\mathcal{L}|$ ist, weshalb man für solche Angaben mit $|\mathcal{L}|$ statt n argumentieren kann.

Nach dem Einlesen der Daten erfolgt der Aufbau eines Topologiegraphen mit den Flughäfen als Knoten und den Legs als (gerichteten) Kanten. In jedem Knoten gibt es dabei zwei chronologisch sortierte Adjazenzlisten, je eine für hinein- und hinauslaufende Kanten. Mit Hilfe dieses Graphen lassen sich Legs, die bestimmten räumlichen und zeitlichen Einschränkungen genügen, effizient finden.

Für die Bildung von Pairings ist es aber häufig notwendig, zu einem bestimmten Leg einen Anschluß oder ein Austauschleg zu finden. Zu diesem Zweck wird ein sogenannter Conngraph (von „Connection“) generiert, in dem die Legs als Knoten fungieren. Zwei Legs A und B werden genau dann mit einer gerichteten Kante (A, B) verbunden, wenn der Zielflughafen von A mit dem Startflughafen von B übereinstimmt und die Zeit zwischen Ankunft von A und Abflug von B in einem vordefinierten Intervall liegt, also wenn ein direkter Anschluß besteht. Die untere Grenze des Intervalls steht für die mindestens notwendige Zeit, die das Personal zum Ein- und Auschecken und für die Vorbereitung des folgenden Fluges benötigt. Sie hängt auch davon ab, ob die Rotation, also das physische Flugzeug, gewechselt wird oder nicht. Tarifvertragliche Einschränkungen werden hier allerdings noch nicht betrachtet. Die obere Grenze des Intervalls beinhaltet, bis wann man sinnvoller Weise noch von einem Anschluß sprechen kann bzw. bis wann eine direkte Verkettung in einer Dienstperiode sinnvoll ist. Sie soll verhindern, daß der Graph unnötig groß wird.

Ein Pairing besteht definitionsgemäß aus einer Folge von Legs, die als doppelt verkettete Liste implementiert ist. Da eine Verbindung zweier Legs im Pairing nur stattfinden kann, wenn sie auch im Conngraphen benachbart sind, kann man ein Pairing auch als Pfad in ihm auffassen. Eine Lösung aus $\mathcal{S}$ stellt eine Menge von Pairings dar, in denen alle Legs genau einmal verplant sind. Dies entspricht einer vollständigen Zerlegung des Conngraphen in paarweise kreuzungsfreie Pfade. Eine Datenstruktur namens Pool verwaltet eine variable Menge von Pairings, die genau einer solchen vollständigen Zerlegung entsprechen. In Pool liegt also zu jedem Zeitpunkt eine Lösung aus $\mathcal{S}$ vor. Initial bildet jedes Leg ein eigenes Pairing.

Die im Kapitel 2.1 definierten Operatoren Join und Split werden von Pool bereitgestellt. Mit Split kann jederzeit ein Pairing aufgespalten werden, während ein Join zweier Pairings nur möglich ist, wenn der Conngraph es vorsieht. Die Ausführung eines der beiden Operatoren, also der Übergang zu einer Nachbarlösung, wird im folgenden abkürzend als „Move“ bezeichnet.

Wenn Δ der Maximalgrad des Conngraphen ist, so läßt sich die Nachbarschaft einer Lösung in einer Zeit $\mathcal{O}(n\Delta)$ finden. Um sie nicht ständig neu berechnen zu müssen, wird eine sogenannte Moveliste angelegt, in der alle möglichen Moves und damit die Nachbarn der aktuellen Lösung gespeichert werden. Nach einem Move wird die Liste nur entsprechend modifiziert. Zwar ist der bei unserer Implementation dafür notwendige Aufwand $\mathcal{O}(n\Delta)$ mit dem für eine Neuberechnung der Nachbarschaft identisch, jedoch sinkt die tatsächliche Gesamtrechenzeit um einen nicht unwesentlichen Faktor. Das Führen der Moveliste ist abschaltbar, um Heuristiken, die ohne sie auskommen, nicht unnötig zu verlangsamen.

Darüber hinaus bietet die Moveliste die Möglichkeit, bereits berechnete Kosten der Nachbarlösungen zu speichern, um bei kleinen Akzeptanzwahrscheinlichkeiten, wenn Nachbarn mehrfach betrachtet werden, Zeit zu sparen. Diese Kosten werden zwar bei Ausführung eines Moves gelöscht, was jedoch mit Hilfe eines Zeitstempelverfahrens[1] in konstanter Zeit möglich ist.

2.4 Testdaten und einfache Heuristiken

Tabelle 2.1: Art und Größe der Testdaten

	Datensätze		
	F70	DH8	X
Legs	374	1395	4612
Rotationen	3	9	23
Zeitraum (Tage)	17	13	30
Flughäfen	10	18	45
Homebases	6	6	4
Leglänge (Minuten)			
minimale	40	25	25
maximale	215	135	165
mittlere	58.4	70.9	77.0
Handlösung	✓	—	—

Für die statistischen Aussagen über verschiedene Algorithmen oder Parametereinstellungen wurden drei Datensätze verwendet, die mit „F70“, „DH8“ und „X“ bezeichnet sind. Tabelle 2.1 zeigt einige charakteristische Werte dieser Daten. Auch wenn die Anzahl der Legs die Problemgröße bestimmt, beeinflussen die Zahl der Rotationen und der Planungszeitraum die Schwere des Problems. Weniger Rotationen oder längere Planungszeiten bei gleicher Legzahl lassen weniger Freiheitsgrade für eine Optimierung. Die Zahl der benutzten Flughäfen vermittelt einen Eindruck der Größenordnung, auch wenn sie im Gegensatz zur Zahl der Homebases keinen nennenswerten Einfluß auf die Laufzeiten der implementierten Algorithmen hat. An der Länge der Legs kann man ablesen, daß es sich bei allen drei Datensätzen um Kurzstreckendaten handelt.

Die Parameterstudien in Kapitel 3 befassen sich hauptsächlich mit den DH8- und X-Daten, da die Ergebnisse auf den recht kleinen F70-Daten stark streuen und daher nur begrenzt verwendbar sind. Leider liegt nur für F70 eine von professionellen Planern stammende Handlösung vor, an der man die erreichten Ergebnisse messen kann.

[1] Neben dem Feld d, das die Kostendifferenzen speichern soll, wird noch ein zweites Feld z gleicher Größe angelegt, in dem sogenannte „Zeitstempel“ stehen. Eine zentrale Variable T definiert, welche Zeitstempel aktuell sind. Eine Kostendifferenz $d[i]$ ist nur dann gültig, wenn der zugehörige Zeitstempel $z[i]$ gleich T ist. Will man einen neuen Wert in $d[i]$ schreiben, muß er zusätzlich mit $z[i] := T$ gültig gemacht werden und vor dem Auslesen von $d[i]$ ist ein Test auf $z[i] = T$ notwendig. Beide Operationen bleiben bei konstanter Laufzeit. Durch einfaches Erhöhen von T (konstante Zeit) können alle Einträge von d ungültig gemacht werden.

Tabelle 2.2: Kosten und Rechenzeiten einiger Heuristiken

	Datensätze		
	F70	DH8	X
Beste bekannte Lösung	69227	280423	786594
Handlösung	76086	—	—
(über Bestlösung)	(9.9%)	—	—
Handlösung + Steepest Descent	71311	—	—
(über Bestlösung)	(3.0%)	—	—
Greedy	306723	1658263	8907722
(über Bestlösung)	(343%)	(491%)	(1032%)
Streuung	8.5%	4.0%	3.7%
Zeit	< 1	5.18	57.02
Steepest Descent	123031	496857	1250714*
(über Bestlösung)	(77.7%)	(77.1%)	(59.0%)*
Streuung	7.0%	5.1%	2.7%*
Zeit	15.16	698	13355*
Linear Chainer	181277	535982	1468264
(über Bestlösung)	(161%)	(91.1%)	(86.7%)
Streuung	0.79%	0.41%	0.13%
Zeit	< 1	< 1	< 1
Linear Chainer + Greedy	102153	476567	1243285
(über Bestlösung)	(47.5%)	(69.9%)	(58.1%)
Streuung	2.1%	0.93%	0.85%
Zeit	< 1	< 1	7.64
Linear Chainer + Steepest Descent	100062	462686	1251426
(über Bestlösung)	(44.5%)	(65.0%)	(59.1%)
Streuung	2.1%	0.91%	0.63%
Zeit	≈ 1	14.1	222

Tabelle 2.2 bietet einen Überblick über die von verschiedenen Heuristiken erreichten Kostenwerte und die jeweils notwendige Rechenzeit[2]. Die in der ersten Zeile stehende „Beste bekannte Lösung“ gilt als Referenz für die qualitative Einordnung aller anderen Lösungen, da das tatsächliche Optimum nicht bekannt ist. Alle in Kapitel 3 angegebenen Graphiken stellen statt absoluter Kostenwerte „% über bester bekannter Lösung“ dar, um eine bessere Übersicht zu ermöglichen. Auch in dieser Tabelle stehen unter allen Kostenangaben in Klammern die prozentualen Verschlechterungen gegenüber der Bestlösung. Die als Ergebnis der verschiedenen Heuristiken angegebenen Werte sind Mediane, die Zeitangaben Durchschnitt über jeweils 100 Messungen. Eine Ausnahme bilden die mit einem Stern markierten Werte des Steepest Descent auf X-Daten: Hier waren aufgrund der enormen Rechenzeit nur 10 Messungen möglich. Unter der Bezeichnung „Streuung“ ist die prozentuale Standardabweichung der Kosten eingetragen. Als Heuristiken wurden verwendet:

Greedy, die simpelste Form einer lokalen Suche. Es wird jeweils zu dem ersten gefundenen Nachbarn übergegangen, der eine Kostenverbesserung darstellt. Bei einer recht guten Laufzeit werden extrem schlechte Lösungen gefunden.

Steepest Descent ist auch ein Greedyverfahren, das allerdings zum jeweils besten aller Nachbarn übergeht. Gibt es mehrere Alternativen, wird aus ihnen ein Nachbar zufällig bestimmt. Da in jedem Schritt die gesamte Nachbarschaft abgesucht werden muß, liegen die benötigten Laufzeiten weit über denen des Greedy. Die Lösungsqualität verbessert sich zwar auch deutlich, bleibt aber noch weit oberhalb der Bestlösung.

Linear Chainer stellt eine einfache Aufbauheuristik dar, die im Gegensatz zu den beiden Greedies problemspezifisch vorgeht. Es werden zulässige Pairings maximaler Länge gebildet, indem an jedes Leg der frühestmögliche Anschluß gehängt wird. In der Regel folgen die so gebildeten Pairings den physischen Flugzeugrotationen. Ursprünglich wurde der Linear Chainer entwickelt, um eine günstige Startlösung für das Simulated Annealing zu generieren (siehe dazu Kapitel 3.3). Obwohl es sich um ein deterministisches Verfahren handelt, treten leichte Streuungen der Kostenbewertung auf, da die nachgeschaltete Greedyheuristik zur Bestimmung der Homebases (s. Kapitel 2.2.2) randomisiert ist und nur ein lokales Optimum findet.

Der Linear Chainer ist ein linearer Algorithmus, dessen Laufzeiten für alle verwendeten Datensätze unter einer Sekunde liegen. Die schlech-

[2]Alle in dieser Arbeit verwendeten Angaben zu Rechenzeiten beziehen sich, soweit nicht anders kommentiert, auf verbrauchte Prozessorsekunden einer Sun UltraSPARC 1 mit 200 MHz und 256 MB Hauptspeicher.

ten Kostenwerte überraschen nicht, da die Kostenfunktion keinen Einfluß auf die Konstruktion der Lösung hat. Startet man nach dem Linear Chainer noch einen Greedy- oder Steepest-Descent-Lauf, sinken die Kosten deutlich und unterbieten bei F70- und DH8-Daten sogar den reinen Steepest Descent, während bei X-Daten qualitativ ähnliche Werte erreicht werden. An der im Vergleich zu Greedy und Steepest Descent wesentlich kleineren Laufzeit der kombinierten Verfahren kann man erkennen, daß kein völliger Umbau der Linear-Chainer-Lösung stattfindet. Es zeigt sich ihre gute Eignung als Startlösung für andere Verfahren, die in Kapitel 3.3 auch für das Simulated Annealing bestätigt werden kann.

Kapitel 3

Studien am Algorithmus

3.1 Die Wahl des Kühlschemas

Wie bereits in Kapitel 1.2.3 ausgeführt wurde, ist für eine polynomielle Laufzeit des Simulated Annealing ein nicht konvergentes Kühlschema notwendig. Wir haben für unser Problem Experimente mit 2 verschiedenen Schemata gemacht, die im folgenden skizziert werden.

Eine in diesem Zusammenhang wichtige Größe ist die Akzeptanzratio eines Levels. $acc(t)$ gibt das Verhältnis akzeptierter zu generierten Moves an und hängt damit natürlich von der Temperatur ab. In Abbildung 3.1 sieht man den Zusammenhang beider Größen. Die Darstellung zeigt zwar das Ergebnis

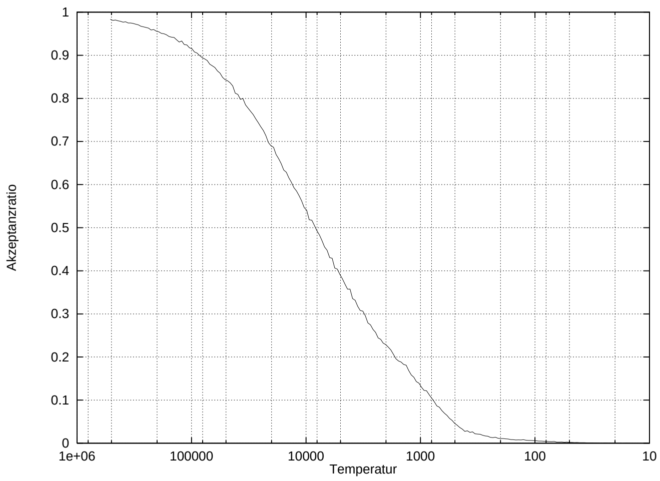


Abbildung 3.1: Der Verlauf der Akzeptanzratio über der Temperatur bei DH8-Daten

eines einzelnen Laufes, aber abgesehen von dem leichten Rauschen auf der Kurve ist der Verlauf unabhängig von Kühlschema oder Parameterwahl. Lediglich bei veränderter Kostenfunktion oder anderen Datensätzen kommt es zu einer Verschiebung des Temperaturbereiches.

Das erste hier behandelte Schema wurde von Johnson, Aragon, McGeoch und Schevon in [JAMS89] vorgestellt.

geometrisches Kühlschema

Parameter: $acc_0, acc_{end}, \alpha \in (0,1);\ L, \ell \in \mathbb{N}^+;\ T_{min} \in \mathbb{R}^+$

Die Starttemperatur c_0 wird so hoch gewählt, daß eine vorher festgesetzte Mindestakzeptanzratio $acc_0 < 1$ überschritten wird. Ursprünglich sollte acc_0 auf einen Wert nahe 1 gesetzt werden, um in der Anfangsphase des Annealing durch Annahme fast jedes Moves das angestrebte Quasiequilibrium schnell zu erreichen. Doch wie in Kapitel 3.3 gezeigt wird, genügen weitaus kleinere Werte für acc_0. Um c_0 zu bestimmen, wird empfohlen, in einer Startphase die Temperatur zu Beginn jedes Levels zu verdoppeln, bis acc_0 überschritten wird.

Die Temperaturfolge (c_k) wird mit Hilfe der Temperaturreduktion α gebildet, mit dem die Temperatur am Ende jedes Levels zu multiplizieren ist:

$$\forall k \in \mathbb{N}^+ : c_k := \alpha \cdot c_{k-1} = \alpha^k \cdot c_0 \tag{3.1}$$

Die Levellängenfolge (L_k) wird konstant auf ein Vielfaches der Nachbarschaftsgröße Θ gesetzt. Da Θ bei unserem Problem nicht konstant ist, verwenden wir statt dessen die durchschnittliche Nachbarschaftsgröße $\bar{\Theta}$. Der Levellängenfaktor wird im folgenden nur kurz als Levellänge und mit L bezeichnet.

$$\forall k \in \mathbb{N} : L_k := L \cdot \bar{\Theta} \tag{3.2}$$

Die Abbruchbedingung wird durch zwei Parameter bestimmt; die Endakzeptanzratio acc_{end} und die Kettenlänge ℓ. Der Abbruch erfolgt, wenn in den letzten ℓ Leveln die Akzeptanzratio unter acc_{end} lag und keine Verbesserung der Kosten erzielt wurde. Um die Terminierung zu garantieren, haben wir zusätzlich noch eine Minimaltemperatur T_{min} eingeführt, bei deren Unterschreitung ein sofortiger Abbruch erfolgt. Sie wird aber so klein gewählt, daß sie in der Regel keinen Einfluß hat.

Das zweite von uns untersuchte Kühlschema stammt von Aarts und van Laarhoven ([AvL85],[AK89]) und zeichnet sich vor allem dadurch aus, daß das Herabsetzen der Temperatur nicht fest vorherbestimmt ist, sondern in Abhängigkeit von der Entwicklung der Kostenfunktion gewählt wird. Aus diesem Grund wird es im folgenden als „adaptives Kühlschema“ bezeichnet.

Der Grundgedanke bei der Wahl der Temperaturen besteht darin, daß die stationären Verteilungen, gegen die zwei aufeinanderfolgende Level bei unendlicher Länge konvergieren würden, dicht beieinander liegen. Für jedes $k \in \mathbb{N}^+$ soll also gelten:

$$\|q(c_k) - q(k_{k-1})\| < \varepsilon \tag{3.3}$$

Ist ε klein genug und am Ende des $k-1$-ten Levels ein Quasiequilibrium erreicht, so soll es sich auch am Ende des k-ten Levels einstellen.

Für ein entsprechend gewähltes δ folgt (3.3) aus

$$\forall i \in \mathcal{S} : \frac{1}{1+\delta} < \frac{q_i(c_k)}{q_i(k_{k-1})} < 1 + \delta. \tag{3.4}$$

---

adaptives Kühlschema

Parameter: $acc_0 \in (0,1)$; $L \in \mathbb{N}^+$; $\delta, \varepsilon_s \in \mathbb{R}^+$
(bzw. $acc_{end} \in (0,1)$; $\ell \in \mathbb{N}^+$; $T_{min} \in \mathbb{R}^+$ statt ε_s)

Die Starttemperatur c_0 ergibt sich genau wie beim geometrischen Schema aus der Mindestakzeptanzratio acc_0.

Die Temperaturfolge (c_k) ist mit Hilfe des Parameters δ wie folgt definiert:

$$\forall k \in \mathbb{N}^+ : c_k := \frac{c_{k-1}}{1 + \frac{c_{k-1} \cdot \ln(1+\delta)}{3\sigma_{k-1}}} \tag{3.5}$$

σ_{k-1} steht dabei für die Standardabweichung der Kostenfunktion im Level $k-1$. In [AK89, Seiten 61-63] wird gezeigt, daß unter einigen vereinfachenden Annahmen (3.4) aus (3.5) folgt.

Die Levellängenfolge (L_k) wird wie auch im geometrischen Schema durch (3.2) definiert.

Als Abbruchbedingung wird die Erfüllung folgender Ungleichung vorgeschlagen:

$$\frac{c_k \cdot (\bar{f}(c_k) - \bar{f}(c_{k-1}))}{(c_k - c_{k-1}) \cdot \bar{f}(\infty)} < \varepsilon_s \tag{3.6}$$

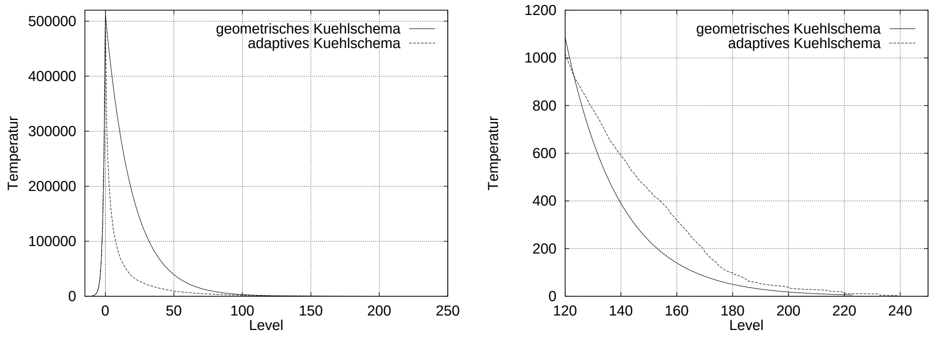
Dabei sind $\bar{f}(c)$ die durchschnittlichen Kosten eines Levels bei Temperatur c.

Dieses Kriterium hat sich jedoch für unser Problem als nicht praktikabel erwiesen. Da die mittleren Kosten besonders bei hohen Temperaturen einen lokal eher chaotischen Verlauf zeigen und (3.6) bei einem Anstieg von $\bar{f}$ erfüllt ist, kommt es zu sehr frühen Abbrüchen. Auch die Empfehlung, nicht das letzte Level $(k-1)$, sondern ein weiter zurück liegendes $(k-t)$ als Vergleich heranzuziehen, hilft nur für große t weiter, womit aber lange Laufzeiten im niedrigen Temperaturbereich verbunden sind. Daher haben wir uns entschlossen, die Abbruchbedingung des geometrischen Schemas zu verwenden, mit der wir sehr gute Erfahrungen machen konnten.

In den Abbildungen 3.2, 3.3 und 3.4 sieht man den Verlauf von Temperatur, Akzeptanzratio und Kostenfunktion bei den beiden vorgestellten Kühlschemata anhand jeweils eines repräsentativen Laufes auf DH8-Daten. Die beiden Läufe wurden auf eine etwa gleich lange Rechenzeit ausgerichtet sowie mit einer sehr hohen Mindestakzeptanzratio ($acc_{min} = 0,99$) und einer sehr niedrigen Endakzeptanzratio ($acc_{end} = 0,0001$) durchgeführt, um ein möglichst großes Temperaturintervall abzudecken. In den vergrößerten Darstellungen (b) ist der Bereich zu sehen, in dem sich normalerweise ein Lauf bewegt. Für Abbildung 3.4 gibt es zusätzlich noch eine zweite Vergrößerung (c).

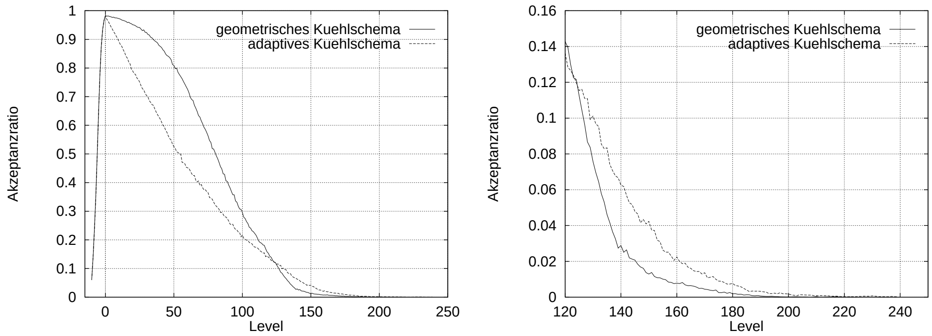
Es zeigt sich, daß das adaptive Schema im sehr hohen Temperaturbereich, der für die Lösungsqualität ohne Bedeutung ist (vgl. Kapitel 3.3), wenig Zeit verbringt und schnell abkühlt, um länger bei niederen Temperaturen zu bleiben. Wählt man das Level 123, in dem die Temperaturen beider Schemata zufällig gleich sind, als gemeinsamen Ausgangspunkt einer Optimierung im wichtigeren Temperaturbereich unter 1000, so zeigt sich ein anderes Bild. Hier kühlt das adaptive Schema beinahe linear bis etwa zum 180. Level und hält sich damit länger bei mittleren Temperaturen auf, als das geometrische. Die Akzeptanzratio spiegelt diesen Verlauf wider.

Aus dem in Abbildung 3.4 erkennbaren Kostenverlauf darf kein qualitativer Vorteil des adaptiven Kühlschemas gefolgert werden, da hier zwei einzelne Läufe gegenübergestellt werden und bei randomisierten Algorithmen gesicherte Aussagen nur aus einer größeren Testreihe gewonnen werden können. Erkennbar ist aber der qualitative Verlauf, der durchaus repräsentativ ist. So macht Abbildung 3.4(b) deutlich, daß erst ab dem 137. bzw. 147. Level überhaupt Lösungen erreicht werden, die besser als die im ersten Vorlevel (bei Temperatur 500) gefundene Bestlösung sind. Daraus folgt zwar noch nicht, daß die bei höheren Temperaturen verbrachte Zeit nutzlos war, aber



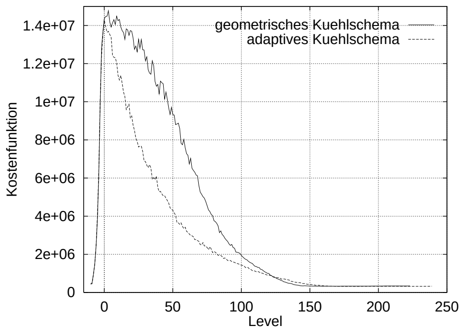
(a) Gesamtbereich **(b)** Vergrößerter Ausschnitt

Abbildung 3.2: Vergleich der Temperaturentwicklung bei geometrischem und adaptivem Kühlschema

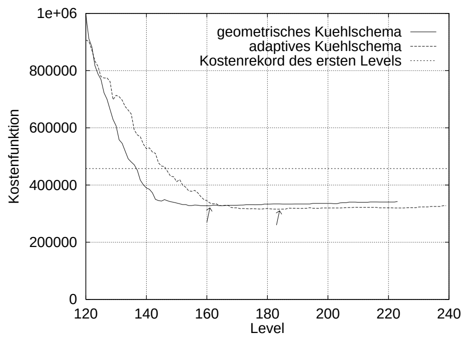


(a) Gesamtbereich **(b)** Vergrößerter Ausschnitt

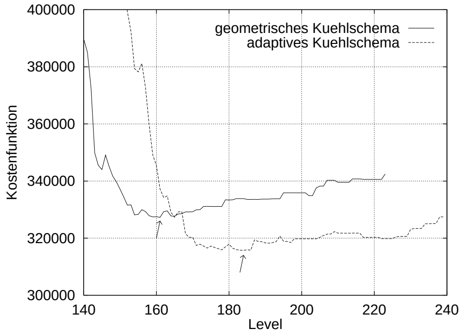
Abbildung 3.3: Die Akzeptanzratio der Level bei geometrischem und adaptivem Kühlschema



(a) Gesamtbereich



(b) 1. Vergrößerung



(c) 2. Vergrößerung

Abbildung 3.4: Die minimalen Kosten pro Level bei geometrischem und adaptivem Kühlschema. In (b) und (c) deuten die Pfeile auf den Kostenrekord des jeweiligen Laufes, in (b) sind die minimalen Kosten des ersten Levels der Aufwärmphase durch eine gepunktete Linie gekennzeichnet.

es legt die Vermutung nahe, daß auch ein „kälterer“ Start genügen könnte. Interessant ist, daß die Kosten nach ihrem Minimum bei ganz niedrigen Temperaturen wieder ansteigen. Ein konvergentes Kühlschema würde dieses Verhalten nicht zeigen. Hier erweist es sich als vorteilhaft, statt der letzten die insgesamt beste Lösung als Ausgabe zu wählen.

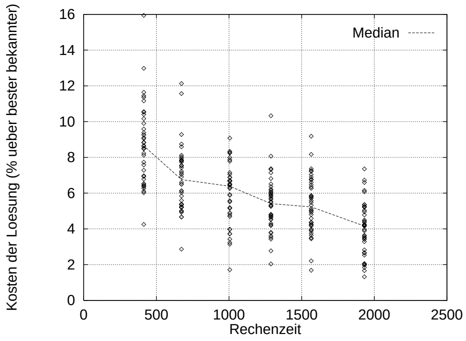
Abbildung 3.5 zeigt für die beiden Kühlschemata die erreichte Lösungsqualität bei verschiedenen zur Verfügung stehenden Rechenzeiten. Für das adaptive Schema wurde je eine Testreihe mit Levellänge 5 (3.5(a)) und 60 (3.5(b)) angelegt und die benötigte Rechenzeit mit Hilfe des Parameters δ variiert. Die zu jeder Parameterwahl durchgeführten 20 Testläufe sind jeweils durch einen Punkt in der entsprechenden Abbildung gekennzeichnet. Eine Linie verbindet die für jede Wahl von δ bestimmten Mediane. 3.5(c) ist eine analoge Darstellung für das geometrische Schema mit dem Unterschied, daß hier zur Beeinflussung der Laufzeit die Temperaturreduktion α genutzt wurde. 3.5(d) zeigt die drei Mediane im direkten Vergleich und 3.5(e) die prozentualen Standardabweichungen der drei Testreihen.

Der Median des geometrischen Kühlschemas liegt generell unter dem des adaptiven mit großer Levellänge. Obwohl die Streuung beider Testreihen so hoch ist, daß sich die Intervalle, in denen die Kosten liegen, weit überschneiden, läßt sich daraus ein gewisser Vorteil des geometrischen Schemas ableiten. Der Median des adaptiven Schemas mit kurzer Levellänge verläuft hauptsächlich zwischen den beiden anderen, auch wenn er sich stellenweise mit dem des geometrischen überdeckt. Es zeigt sich auch hier ein Vorteil des geometrischen Kühlschemas, wenn auch nicht sehr deutlich.

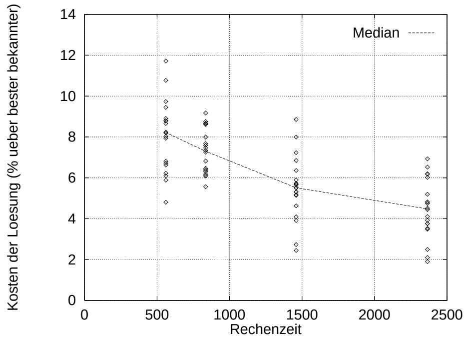
Betrachtet man die Standardabweichungen der beiden Schemata, so zeigen sich kaum Unterschiede. Der zweite Meßpunkt des adaptiven Schemas mit langer Levellänge muß als „Ausrutscher“ eingestuft werden. Ansonsten ist eine leichte Abnahme der Streuung mit steigender Rechenzeit bei allen drei Kurven erkennbar.

Für das von uns bearbeitete Problem der Pairing-Generierung scheint das geometrische Kühlschema das geeignetere zu sein. Diese Beurteilung beruht nicht allein auf dem oben dargelegten Qualitätsvorteil. Es kommt hinzu, daß sich die für einen Lauf notwendige Rechenzeit beim geometrischen Kühlschema weitaus einfacher steuern läßt, da sie logarithmisch von der Temperaturreduktion abhängt. Selbst wenn man beim adaptiven Kühlschema die Standardabweichung der Kosten innerhalb eines Levels als linear über der Kostenfunktion vereinfachen könnte, hinge die zu erwartende Rechenzeit immer noch doppelt logarithmisch vom Parameter δ ab.

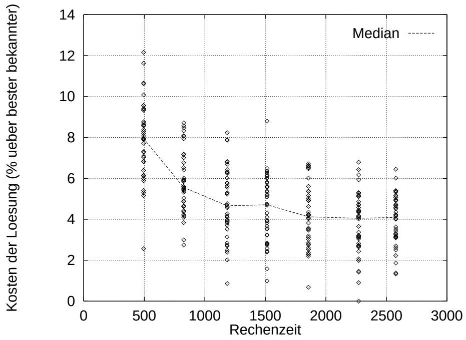
Für die weiteren Betrachtungen in diesem Kapitel wurde das geometrische Kühlschema verwendet, da das adaptive den notwendigen Mehraufwand für Berechnungen und Handhabung qualitativ nicht rechtfertigen konnte.



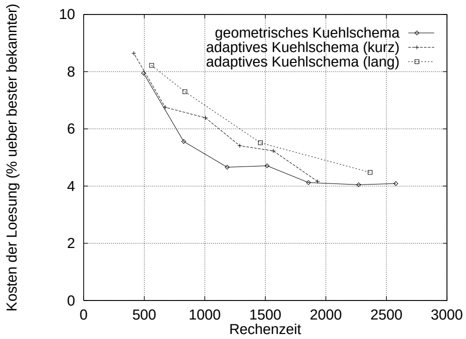
(a) Adaptives Kühlschema (kurz)



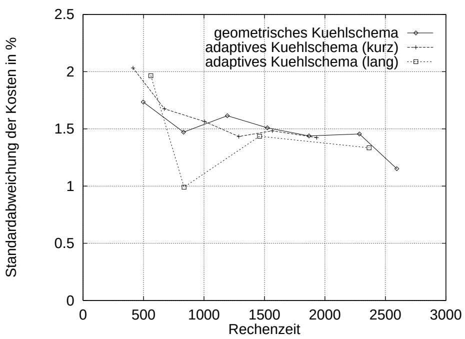
(b) Adaptives Kühlschema (lang)



(c) Geometrisches Kühlschema



(d) Mediane aus (a)–(c)



(e) Standardabweichungen

Abbildung 3.5: Lösungsqualität bei verschiedenen Rechenzeiten zum Vergleich der beiden Kühlschemata. Die verwendeten Parameter: (a) $\delta \in$\{0,1; 0,06; 0,04; 0,03; 0,025; 0,02\} (b) $\delta \in$\{10; 3; 1; 0,5\} (c) $\alpha \in \{0,95^{\frac{1}{i}} | 1 \leq i \leq 7\}$

3.2 Die Kostenart „ADMISS“ – Relaxierung einer Nebenbedingung

Bei der Implementation des Prototyps wurde die Zulässigkeit einer Lösung nicht als harte Nebenbedingung, sondern mit Hilfe der Kostenart „Admiss“ als Ziel der Optimierung formuliert. Zwischenzeitlich darf der Algorithmus also unzulässige Lösungen erzeugen in der Hoffnung, so zu besseren zulässigen Lösungen zu gelangen. Wie sinnvoll ist dieses Vorgehen?

Zunächst sind der Lösungsraum und die Nachbarschaftsmenge einer Lösung bei der relaxierten Variante entschieden größer. Um eine höhere Problemgröße handelt es sich jedoch nicht, da unzulässige Lösungen erst bei deren genauerer Betrachtung als solche erkennbar sind. Sie müssen beim Aufbau der Nachbarschaftsmenge mitgeneriert und können erst später verworfen werden. Es ließe sich zwar Zeit sparen, indem man die Berechnung der Kostenfunktion abbricht, nachdem eine Unzulässigkeit gefunden wurde, aber es stellt keine tatsächliche Reduzierung der Komplexität dar. Bei einer Implementation wären die Lösungsräume der beiden Varianten identisch, wobei bei der nichtrelaxierten Version eine Teilmenge als „verboten“ deklariert und daher zu umgehen ist.

Bei Optimierungsalgorithmen wie Simulated Annealing oder Greedyverfahren, die Verschlechterungen der Kostenfunktion nicht oder nur in Abhängigkeit von deren Maß zulassen, kommt ein sehr hoher Kostenfaktor für „Admiss“ einem Verbot unzulässiger Lösungen gleich.

Für das Simulated Annealing hat sich die Relaxierung als sehr erfolgreich erwiesen. Je kleiner der Admiss-Faktor gesetzt wird, desto besser wurden die Endergebnisse. Jedoch sind die Lösungen bei sehr kleinem Admiss-Faktor unbrauchbar, da unzulässig. Weil aber die besten Resultate im Grenzbereich zur Unzulässigkeit gefunden wurden, entstand die Idee, unzulässige Pairing-Zerlegungen durch ein simples Verfahren nachzubearbeiten. Dadurch kann man in diesem Grenzbereich optimieren, ohne das Risiko einer unzulässigen Endlösung eingehen zu müssen.

Für die Nachoptimierung wird statt des Quenching ein Algorithmus durchgeführt, den wir „Admissiblator“ genannt haben. Er besteht aus einer Folge von Greedy-Läufen bei sich langsam erhöhendem Admiss-Faktor und bricht erst ab, wenn die Lösung zulässig geworden ist. Bei diesen Korrekturschritten bleiben zulässige Pairings zunächst unberührt, da durch die gestiegenen Admiss-Kosten nur Splits in unzulässigen Pairings bevorzugt werden. Nach deren Ausführung kommt es allerdings zu Verschmelzungen der Bruchstücke mit anderen Pairings. Umfangreiche Veränderungen der Gesamtlösung sind bei einer geringen Zahl unzulässiger Pairings jedoch nicht zu erwarten. Unsere Testläufe zeigen, daß die für den Admissiblator verbrauchte Rechenzeit gegenüber der für das Simulated Annealing notwendigen kaum ins Gewicht fällt.

In den Abbildungen 3.6 und 3.7 sieht man je 20 Testläufe bei verschiedenen Admiss-Faktoren. Dabei sind Läufe mit zulässigem Ergebnis und solche mit nachträglich zulässig gemachtem Ergebnis verschieden gekennzeichnet. Es ist erkennbar, daß knapp unterhalb des Grenzbereiches noch bessere Lösungen zu finden sind als im Grenzbereich selbst, um dessen Erschließung es ja ursprünglich ging.

Dieser Effekt läßt sich dadurch erklären, daß das Annealing bis in seine letzten Level von der Möglichkeit profitiert, durch unzulässiges Gebiet zu laufen. Knapp unterhalb des Grenzbereiches entstehen Lösungen, die nur wenige unzulässige Pairings enthalten. Deren Modifikation kann die Qualität der Lösung insgesamt kaum beeinflussen.

Eine ähnliche Wirkung des Admiss-Faktors konnte auch bei den F70-Daten beobachtet werden, wenn auch bei weitem nicht so deutlich, da sie von den hier üblichen Streuungen überlagert wurde.

Abbildung 3.8 zeigt, daß auch die Streuung der Kosten im Grenzbereich bzw. knapp darunter minimal sind. Obwohl die hohe Standardabweichung bei Admiss-Faktor 25 auf DH8-Daten als „Ausrutscher“ zu werten ist, zeigt sich, daß sich nicht nur die Lösungsqualität selbst, sondern auch ihre Stabilität bei einer „nicht zu strengen“ Bestrafung der Unzulässigkeit verbessern.

Doch wie verhält es sich mit der Struktur der Lösungen? Aus Abbildung 3.9 kann man die Zusammensetzung der Gesamtkosten am Beispiel der DH8-Daten ersehen. Im Bereich des Admiss-Faktors von 5 bis 15, in dem die Gesamtkosten relativ konstant bleiben und ihr Minimum annehmen, findet eine deutliche Verlagerung zwischen den Hauptkosten Proceedings und X-Work statt. Diese setzt sich bis Admiss-Faktor 20, also knapp über den Grenzbereich zwischen zulässigen und unzulässigen Annealing-Lösungen, fort.

Der Grund für diese Änderung der Kostenstruktur ist im Verhalten des Admissiblators zu suchen. Nur etwa 20% aller von ihm ausgeführten Moves sind Joins. Bei Splits, die also hauptsächlich ausgeführt werden, kommt es zu einer Verringerung der X-Work, während die Proceeding-Kosten durch die steigende Anzahl der Pairings zunehmen können. Die wenigen Joins können diesen Effekt nicht ausgleichen.

Für die Anwendung in der Praxis, wenn umfangreiche Parameterstudien nicht möglich sind, empfiehlt es sich, zunächst den Grenzbereich des Admiss-Faktors zwischen zulässigen und unzulässigen Lösungen zu bestimmen und mit einem etwas (10-20%) geringeren Faktor zu arbeiten. Bei noch kleineren Werten treten in verstärktem Maß die beschriebenen Veränderungen der

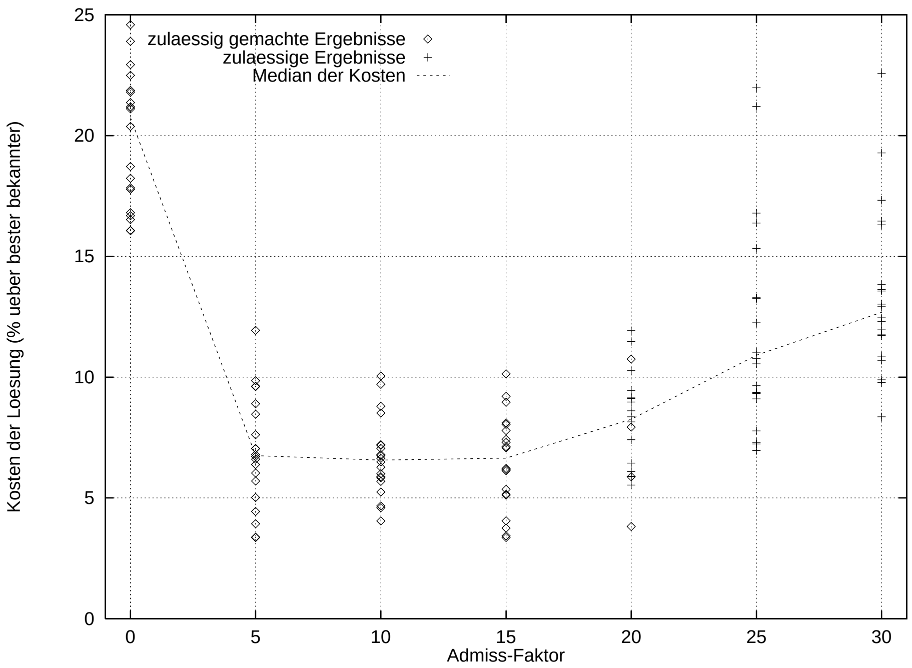


Abbildung 3.6: Admiss-Relaxierung
(DH8-Daten, Simulated Annealing mit Levellänge 60)

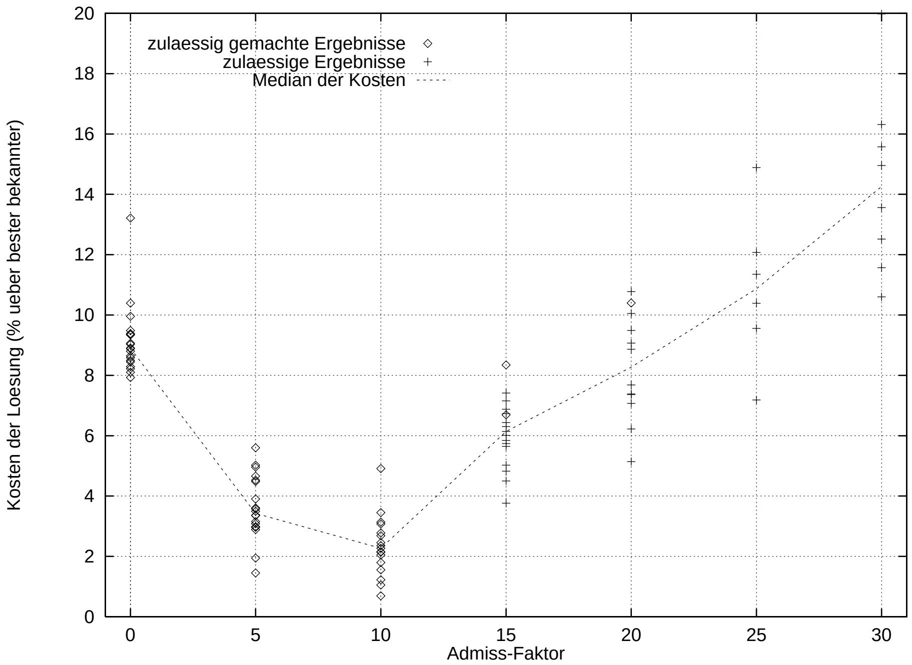
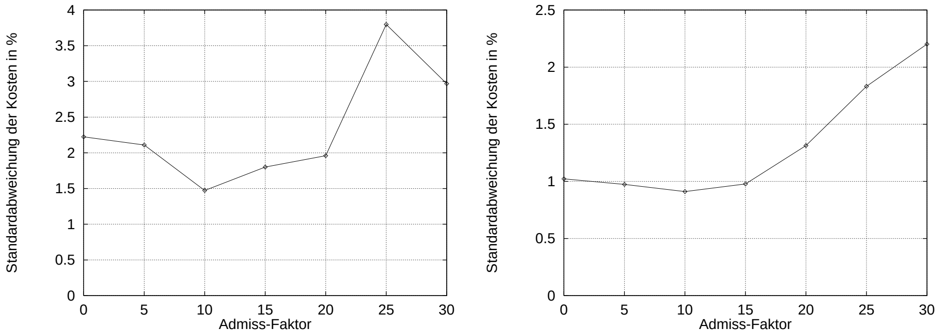


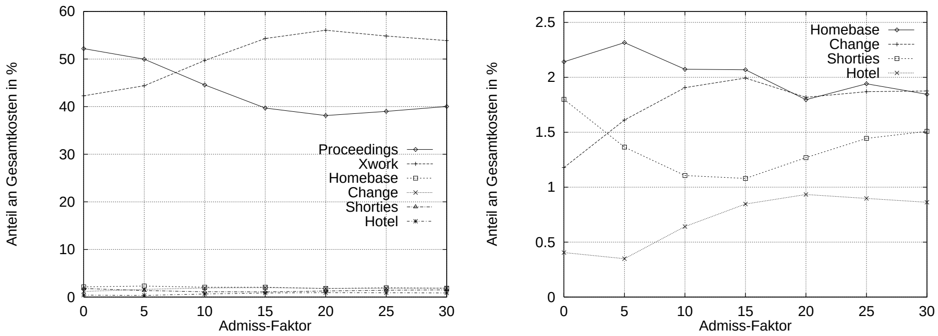
Abbildung 3.7: Admiss-Relaxierung
(X-Daten, Simulated Annealing mit Levellänge 60)



(a) DH8-Daten

(b) X-Daten

Abbildung 3.8: Die Standardabweichungen zu den Abbildungen 3.6 und 3.7 in % der Kosten.



(a) Gesamtdarstellung

(b) vergrößerter y-Bereich

Abbildung 3.9: Anteile der Kostenarten an den Gesamtkosten (DH8-Daten, Simulated Annealing mit Levellänge 60)

Kostenstruktur auf, die in der Regel nicht gewünscht sind. Optimiert man oberhalb des Grenzbereiches, gelangt man zu schlechteren Lösungen.

3.3 Initialtemperatur und Startlösung

Die Wahl der Initialtemperatur hat einen entscheidenden Einfluß auf die Laufzeit des Annealing-Algorithmus. Man ist daher bestrebt, sie so klein wie möglich zu wählen, ohne die Qualität der Lösungen zu verschlechtern.

Eigentlich haben Temperaturen beim Simulated Annealing als Absolutwert keinerlei Aussagekraft, da sie weder auf andere Probleminstanzen noch innerhalb einer Instanz bei veränderter Kostenfunktion übertragbar sind; hier bietet sich eher die Akzeptanzratio an (vgl. Kapitel 3.1). Dennoch sind sie Grundlage der folgenden Betrachtungen, da sie logarithmisch in die Rechenzeit eingehen (siehe Abbildung 3.10) und daher bei entsprechender Achsenbeschriftung einen guten Eindruck der Zeitentwicklung geben. Aus Abbildung 3.11 kann die entsprechende Akzeptanzratio ersehen werden.

Welchen Einfluß hat die Initialtemperatur auf die Optimierung? Bei sehr niedrigen Temperaturen werden kaum Verschlechterungen angenommen, so daß sich das Annealing wie ein Greedy verhält. Beginnt man mit einer zu niedrigen Temperatur, findet man nur ein nahes lokales Optimum. Eine zu hohe Starttemperatur sollte auf das Ergebnis keinen negativen Einfluß haben, jedoch auf die Rechenzeit. In [JAMS89] berichten Johnson et al., daß es nicht notwendig ist, wie angenommen mit einer Akzeptanzratio nahe 100% beginnen zu müssen, da für $acc_0 = 40\%$ gleiche Ergebnisse bei geringerem Zeitaufwand gefunden werden.

Es hat sich herausgestellt, daß bei unserem Problem auch die Wahl der Startlösung die Lösungsqualität nachhaltig beeinflußt. Es wurden Testreihen mit zwei Varianten durchgeführt:

„Singleton“ ist die triviale Startlösung, in der jedes zu verplanende Leg in einem eigenen Pairing untergebracht ist. Dieser Zustand liegt direkt nach Initialisierung der Datenstrukturen vor. Ein Preprocessing ist also nicht nötig.

„Linear Chainer“ ist die bereits in Kapitel 2.4 vorgestellte einfache Aufbauheuristik. Wie dort schon ausgeführt wurde, ist die notwendige Rechenzeit vernachlässigbar.

In Abbildung 3.12 sieht man zunächst eine Testreihe auf DH8-Daten mit Linear-Chainer-Startlösung und 20 Läufen pro Initialtemperatur. Die x-Achse

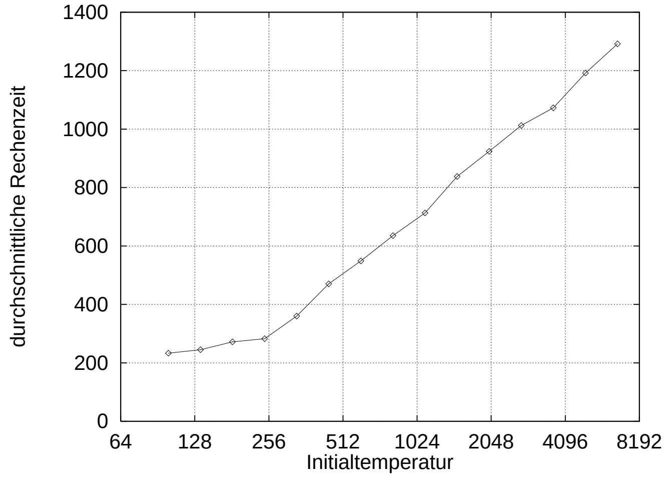


Abbildung 3.10: Die verbrauchte Rechenzeit bei verschiedenen Initialtemperaturen (zu Abbildung 3.12). Im Bereich ab Temperatur 246 zeigt sich die logarithmische Abhängigkeit der Rechenzeit recht deutlich. Lediglich im sehr kalten Bereich weicht die Kurve davon ab. Hier kommt es zu einer späteren Erfüllung des Abbruchkriteriums, da bei insgesamt sehr schlechten Kostenwerten des öfteren noch Verbesserungen in späten Leveln gefunden werden.

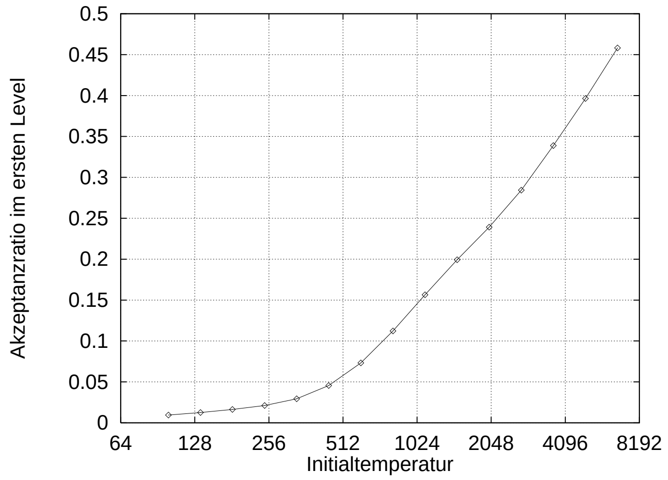


Abbildung 3.11: Die aus der Initialtemperatur resultierende Akzeptanzratio im ersten Level (zu Abbildung 3.12).

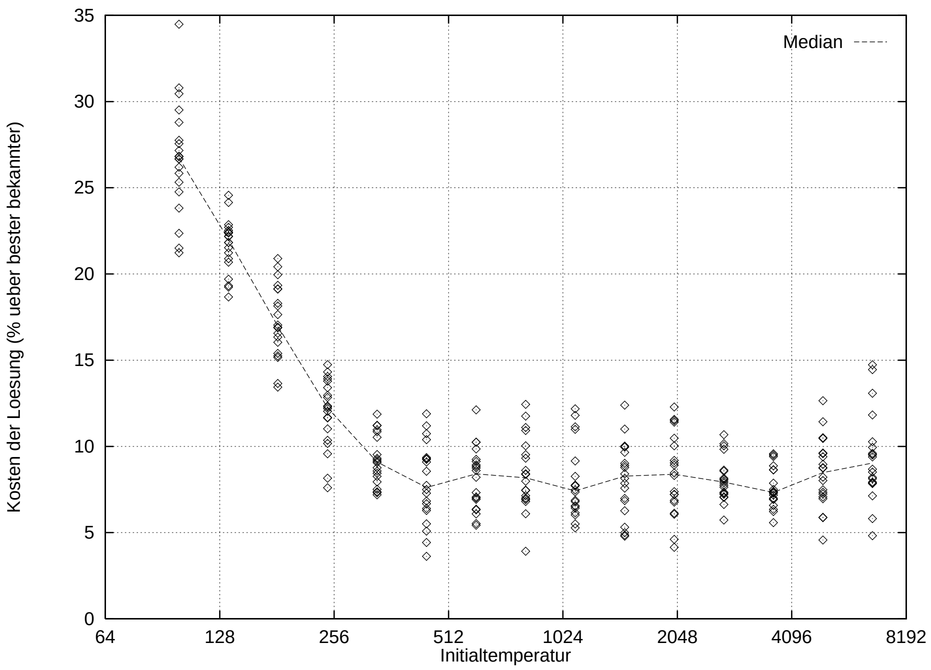
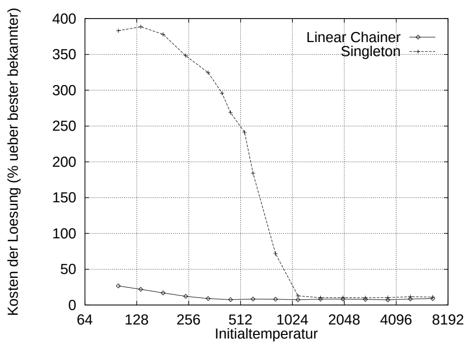
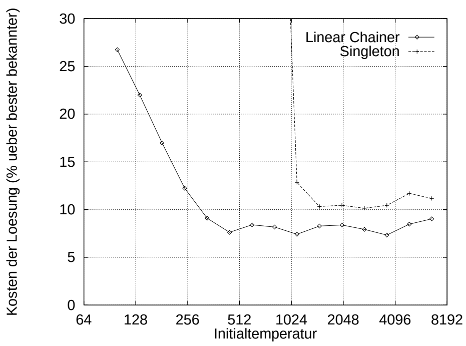


Abbildung 3.12: Initialtemperatur bei Linear-Chainer-Startlösung (DH8-Daten, Simulated Annealing mit Levellänge 60)



(a) Gesamtbereich



(b) vergrößerter y-Bereich

Abbildung 3.13: Initialtemperatur bei verschiedenen Startlösungen (DH8-Daten, Simulated Annealing mit Levellänge 60) ist logarithmisch in der Temperatur und damit linear in der benötigten Rechenzeit geteilt. Während im Temperaturbereich bis 448 die Lösungsqualität ansteigt, bleibt sie im oberen Bereich relativ konstant. Der günstigste Wert für die Initialtemperatur liegt an dieser Grenze, die einer Akzeptanzratio von 4,5% entspricht und damit entschieden niedriger als das von Johnson et al. gefundene Optimum.

Die Testreihe mit Singleton-Startlösung erzeugt zwar einen qualitativ ähnlichen Werteverlauf, allerdings auf einem quantitativ völlig anderen Niveau. In Abbildung 3.13 sieht man die Mediane der beiden Testreihen. Im unteren Temperaturbereich schneidet die Singleton-Variante deutlich schlechter ab und auch im oberen reicht sie nicht an Linear Chainer heran. Darüber hinaus liegt die optimale Temperatur, an der die Lösungskosten relativ konstant werden, bei 1489. Dies entspricht einer initialen Akzeptanzratio von etwa 20% und liegt damit weit hinter der Linear-Chainer-Variante zurück. Dort kommt der optimale Lauf mit 56% der hier notwendigen Rechenzeit aus. Daß der gefundene Wert für acc_0 immer noch unter dem aus [JAMS89] liegt, deutet darauf hin, daß für unser Problem generell niedrigere Starttemperaturen notwendig sind, aber der positive Einfluß der Linear-Chainer-Startlösung bleibt unverkennbar.

3.4 Die Abbruchbedingung

Wie bereits in Kapitel 3.1 dargelegt, wird der Abbruch des geometrischen Kühlschemas im wesentlichen durch die zwei Parameter Abbruchsakzeptanzratio acc_{end} und Kettenlänge ℓ bestimmt. Die Minimaltemperatur T_{min} bleibt in den folgenden Betrachtungen unberücksichtigt.

Um zunächst verschiedene Werte für acc_{end} zu untersuchen, wurde in den Abbildungen 3.14 und 3.15 die Kettenlänge auf 1 gesetzt. Der für DH8-Daten günstigste Wert liegt um etwa 1%. Bei größeren Akzeptanzratios nimmt die Lösungsqualität deutlich ab, während sie unterhalb 1% relativ konstant bleibt. Ein zu langes Annealing bei niedrigen Temperaturen bringt also keine weitere Qualitätsverbesserung, kostet aber Rechenzeit. Das gleiche Verhalten zeigt sich bei den X-Daten, sogar in einer noch deutlicheren Ausprägung. Es kommt zu einer dramatischen Verschlechterung bei Werten über 3%, das Minimum wird unter 0,25% konstant angenommen. Auch die Streuung der Ergebnisse nimmt bei beiden Datensätzen mit acc_{end} sichtlich ab.

In [JAMS89] lag der günstigste Wert für die Abbruchsakzeptanzratio bei 2%. Im Zusammenhang mit den Ergebnissen aus Kapitel 3.3 zeichnet sich ab, daß der für die Optimierung entscheidende Bereich der Akzeptanzratio bei unserem Problem insgesamt niedriger ist als beim Graph-Partitioning.

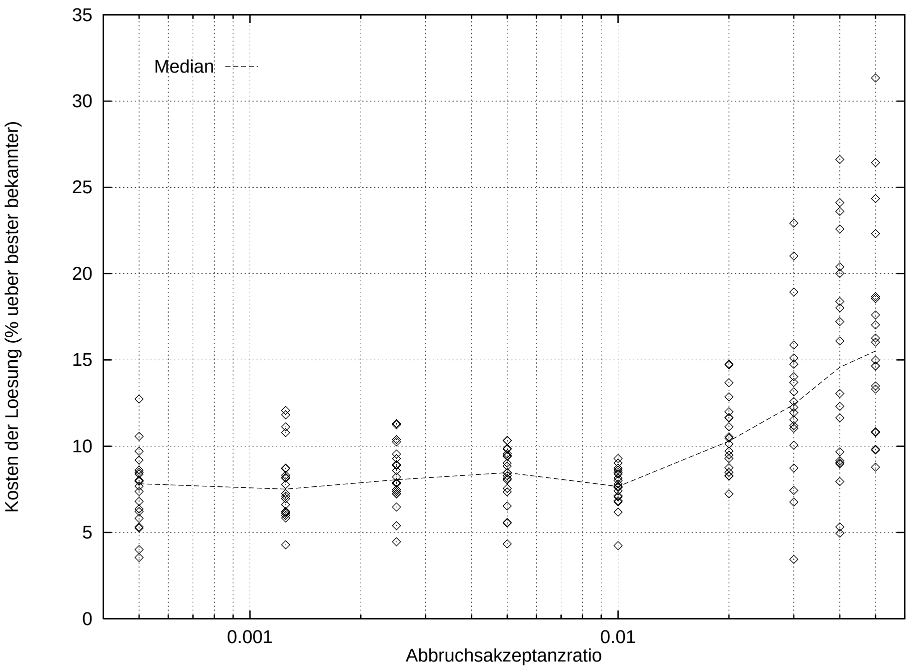


Abbildung 3.14: Abbruchsakzeptanzratio bei Kettenlänge 1
(DH8-Daten, Simulated Annealing mit Levellänge 60)

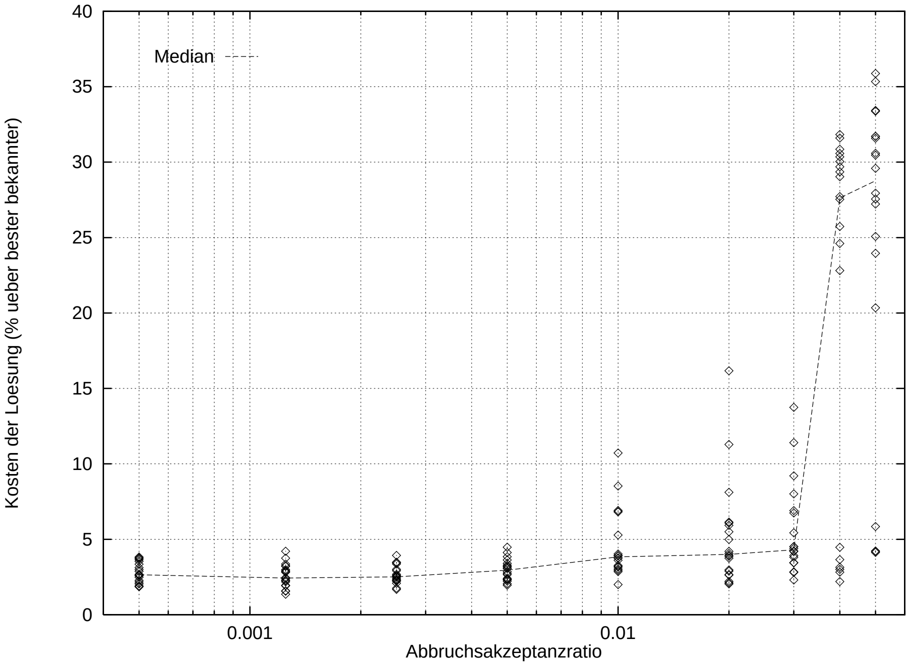


Abbildung 3.15: Abbruchsakzeptanzratio bei Kettenlänge 1
(X-Daten, Simulated Annealing mit Levellänge 60)

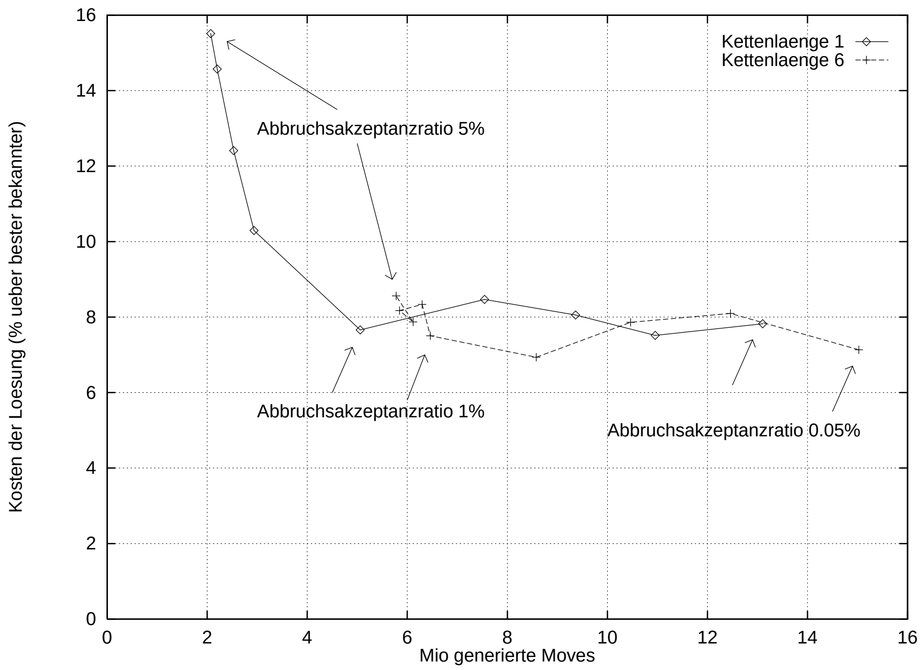


Abbildung 3.16: Die Rechenzeit bei verschiedenen Abbruchsakzeptanzratios und Kettenlängen 1 und 6 auf DH8-Daten. $acc_{end} \in \{5\%; 4\%; 3\%; 2\%; 1\%; 0{,}5\%; 0{,}25\%; 0{,}125\%; 0{,}05\%\}$

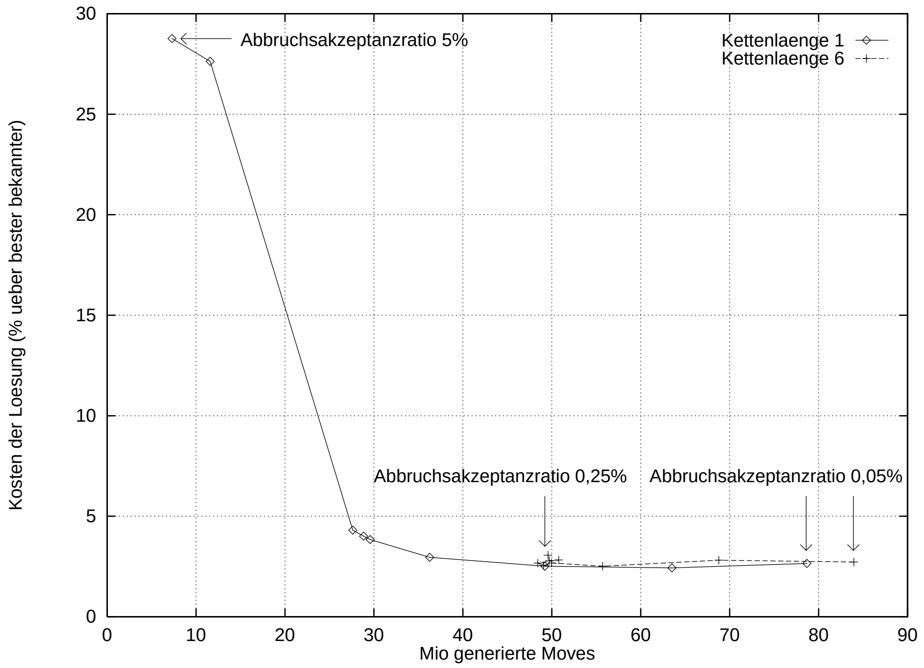


Abbildung 3.17: Analog Abbildung 3.16 für X-Daten

Die Abbildungen 3.16 und 3.17 zeigen den Einfluß der Kettenlänge auf das Abbruchkriterium. Dort ist zunächst die Lösungsqualität über der Gesamtzahl generierter Moves dargestellt. Als Laufzeitvergleich wurde hier nicht die verbrauchte Prozessorzeit genutzt, da deren Messung stets fehlerbehaftet ist[1]. Die exakt bestimmbare Zahl generierter Moves steigt streng monoton mit der (theoretisch) notwendigen Rechenzeit, da lediglich das Abbruchkriterium variiert wird.

Neben den Medianen der bereits vorgestellten Testreihen mit Kettenlänge 1 enthalten die Abbildungen noch je eine Kurve mit Kettenlänge 6 und sonst identischen Parametern. Es fällt sofort auf, daß bei $\ell = 6$ die Meßpunkte für zu hoch gewählte Abbruchsakzeptanzratios, bei DH8 $acc_{end} \geq 1\%$ und bei X $acc_{end} \geq 0,5\%$, die entstehenden Meßpunkte eine dichte Wolke bilden. Qualitativ befinden sich diese Wolken nur unwesentlich über den als optimal benannten Meßpunkten für $\ell = 1$. Ihre Rechenzeit liegt bei DH8 zwischen denen von acc_{end} = 0,5% und 1% und stimmt bei X mit der von acc_{end} = 0,25% überein. Für zu kleine Abbruchsakzeptanzratios zeigt sich das schon bekannte Bild: Bei erhöhter Rechenzeit bleiben die Kosten in etwa konstant, wobei für gleiche acc_{end} bei höherer Kettenlänge jeweils etwas mehr Zeit notwendig ist.

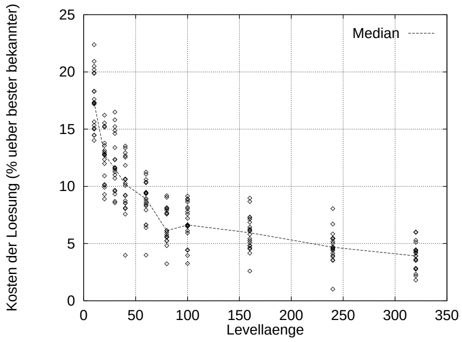
Generell kann zwar mit der optimalen Abbruchsakzeptanzratio auf große Kettenlängen verzichtet werden, aber wenn keine Möglichkeit besteht, diesen Wert genau zu bestimmen, kann mit Hilfe der Kettenlänge und einer etwas erhöhten Rechenzeit die Gefahr eines zu frühen Abbruchs stark reduziert werden. Die Möglichkeit der weiteren Qualitätsverbesserung durch verlängerte Läufe besteht allerdings nicht.

3.5 Ausnutzung der Rechenzeit

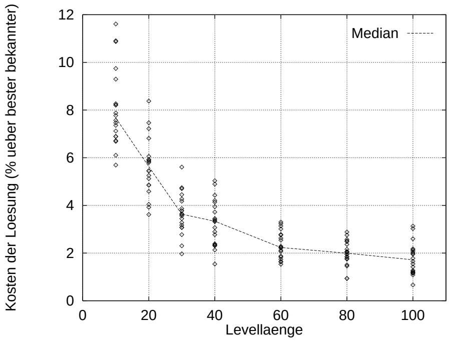
Die zur Verfügung stehende Rechenzeit hat einen großen Einfluß auf die Güte der produzierten Lösungen. Doch es stellt sich die Frage, wie diese Zeit am günstigsten zu nutzen ist bzw. über welche Parameter man sie am sinnvollsten steuern sollte. Initiale und minimale Akzeptanzratio sind dazu nicht geeignet, da sie einen optimalen Wert haben, ober- bzw. unterhalb dessen keine Verbesserungen mehr auftreten (vgl. Kapitel 3.3 und 3.4). Es bliebe die Steuerung der Abkühlungsgeschwindigkeit über die Faktoren Levellänge L und Temperaturreduktion α.

Aus Abbildung 3.18 kann man den Kostenverlauf bei verschiedenen Levellängen ersehen. Es ist hier kein günstigster Wert, sondern eher eine Koppelung

[1] Beim Start absolut identischer und nicht randomisierter Programme ergaben sich Meßfehler von bis zu 10%. Der Grund liegt vermutlich darin, daß einzelne Prozesse unterschiedlich oft geswappt werden und dies auf die Laufzeit angerechnet wird.

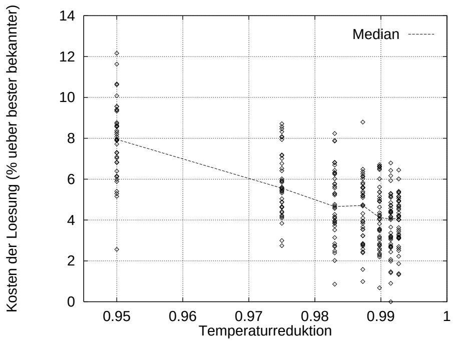


(a) DH8-Daten

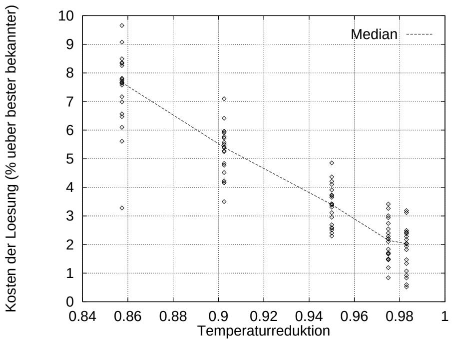


(b) X-Daten

Abbildung 3.18: verschiedene Levellängen bei konstanter Temperaturreduktion 0,95

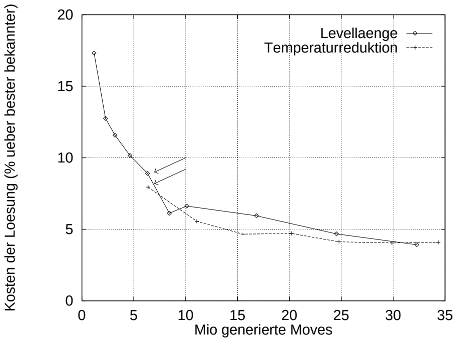


(a) DH8-Daten Levellänge 60

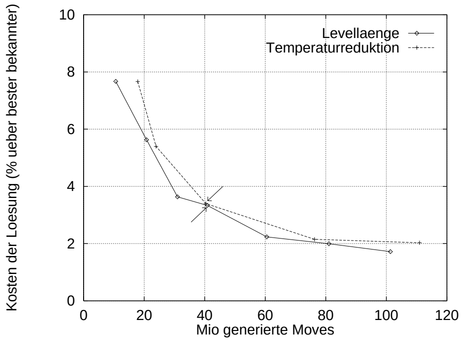


(b) X-Daten Levellänge 40

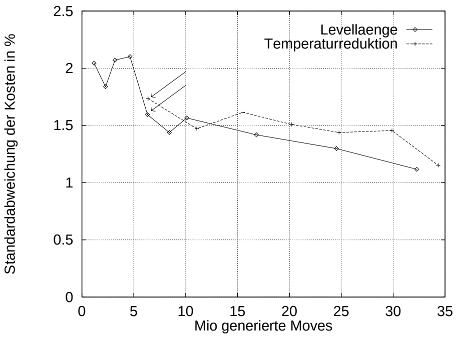
Abbildung 3.19: verschiedene Temperaturreduktionen bei konstanter Levellänge



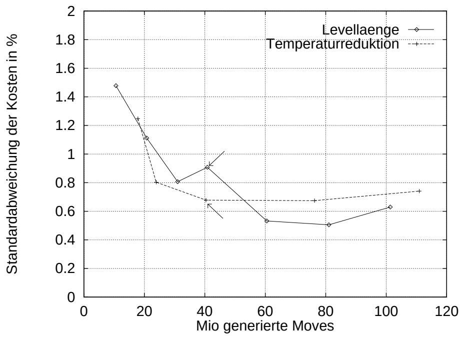
(a) DH8-Daten



(b) X-Daten



(c) σ zu (a)



(d) σ zu (b)

Abbildung 3.20: Vergleich von Laufzeiten und Streuung bei Erhöhung von Levellänge bzw. Temperaturreduktion. In dieser Abbildung wird deutlich, wie fehlerbehaftet die Messungen selbst nach 20 Runden noch sind. Die jeweils mit Pfeilen gekennzeichneten Punkte stellen Läufe mit identischen Parametern dar. Die Übereinstimmung in Bild (b) scheint auch eher zufällig zu sein.

zwischen Zeit und Qualität erkennbar; mit größerem Aufwand lassen sich bessere Lösungen produzieren. Ein ähnliches Bild bietet sich bei der Variation der Temperaturreduktionen in Abbildung 3.19. Auch hier scheinen die Lösungen mit wachsender Zeit immer besser zu werden.

Alle vier Testreihen sind als Laufzeitvergleich in Abbildung 3.20 zu finden. Bei DH8-Daten liegt die Kurve für die Temperaturreduktion zu großen Teilen unterhalb der für die Levellänge. Allerdings fallen sie im oberen Bereich wieder zusammen und die Differenz zwischen ihnen ist in Anbetracht der Streuungen nicht sehr deutlich. Bei X-Daten liegt die Kurve der Levellängenvariation generell unter ihrem Konkurrenten – sowohl ober- als auch unterhalb des gemeinsamen Punktes. Die einzige Erklärung für dieses Verhalten liegt in den nach 20 Runden immer noch stark schwankenden Ergebnissen. Für beide Datensätze sollten die Kurven daher als praktisch gleich gelten.

Auch bei den in Abbildung 3.20 (c) und (d) dargestellten Standardabweichungen schneidet keine der beiden Varianten deutlich besser ab. Erkennbar ist aber die tendenzielle Abnahme der Streuung mit wachsender Rechenzeit.

Entgegen Johnson et al., die einen leichten Vorteil in der Steigerung von α ausmachen konnten, würde ich keinen der beiden Parameter als günstiger für eine Qualitätssteigerung einstufen wollen.

3.6 Varianten des Annealing

Im folgenden werden zwei Versuche vorgestellt, mit einem vom klassischen Annealing abweichenden Algorithmus bessere Ergebnisse zu erzielen.

3.6.1 Permute

Wenn das Annealing bei sehr niedrigen Temperaturen angelangt ist, also auch die Akzeptanzratio sehr kleine Werte (in der Größenordnung von $\frac{1}{|\mathcal{S}_i|}$) annimmt, werden zwischen zwei akzeptierten Nachbarn sehr große Teile der Nachbarschaft der aktuellen Lösung abgesucht. Der Algorithmus betrachtet durch die zufällige Auswahl der Nachbarn einige mehrfach, andere gar nicht und findet eine Verbesserung eventuell nicht.

Statt einer gedächtnislosen zufälligen Auswahl haben wir daher die Möglichkeit vorgesehen, eine zufällige Permutation aller Nachbarn abzusuchen. Sobald ein Nachbar akzeptiert ist, wird eine neue Permutation begonnen, da ja kein Verbot irgendwelcher Nachbarn, sondern lediglich ihre gleichmäßigere Betrachtung erreicht werden soll.

Bei der Implementation wurde darauf geachtet, daß für die Berechnung der Permutation kein nennenswerter zusätzlicher Rechenaufwand entsteht. Exemplarisch wird in Algorithmus 3 die Erzeugung einer Permutation der Zahlen 1 bis n skizziert. Auf Einzelheiten, wie eine erneute Initialisierung nach n Aufrufen von „neuer_Wert", wird hier nicht eingegangen.

Algorithmus 3 Permutation der Zahlen 1 bis n

Prozedur: Initialisierung
  $z := 0$
  [*lege ein Feld* a *der Größe* n *an*]
  for $i := 0$ to n **do**
    $a[i] := i$
  end for

Prozedur: neuer_Wert
  $z := z + 1$
  [*wähle* r *zufällig aus* $\{z, ..., n\}$]
  if $(r \neq z)$ **then**
    [*vertausche die Inhalte von* $a[z]$ *und* $a[r]$]
  end if
  return$(a[z])$

Initial wird ein Feld a der Größe n angelegt, in dem jeder Eintrag mit seinem Index belegt ist. Ein Zähler z speichert die Anzahl der bereits abgefragten Permutationswerte. Vor und nach der Prozedur „neuer_Wert" gilt, daß für $1 \leq i \leq z$ in $a[i]$ der i-te Wert der Permutation zu finden ist und $a[z+1, ..., n]$ die noch nicht vergebenen Werte enthält. Zudem benötigt „neuer_Wert" nur konstante Zeit. Auch die Initialisierung kann in konstanter Zeit ablaufen, wenn das Feld a bereits angelegt ist und für die Eintragung der Zahlen 1 bis n in a das bereits erwähnte Zeitstempelverfahren verwendet wird. Ist ein $a[i]$ laut Zeitstempel ungültig, wird einfach i als sein Inhalt angenommen.

Daß die Größe der Nachbarschaft während des Annealing variiert, stellt kein Problem dar, da bei Programmstart die maximale Größe einer Nachbarschaft (Kantenzahl im Conngraphen) bekannt ist und a entsprechend eingerichtet werden kann. Bei der Initialisierung einer neuen Permutation wird lediglich die aktuelle Größe der Nachbarschaft abgelegt, der Zeitstempel erhöht und z auf 0 gesetzt. Aus dem jeweilig ausgelesenen Permutationswert kann mit Hilfe der Nachbarschaftsliste in konstanter Zeit der zu betrachtende Nachbar bestimmt werden.

In den Tabellen 3.1(a-c) sieht man neben Angaben zu Rechenzeit und Kostenfunktion auch einige implementationsunabhängige Größen zum Laufzeitvergleich: Die Gesamtzahl der Level, die Zahl der insgesamt generierten und die der akzeptierten Moves. Außerdem ist unter dem Titel „Rekordlevel" ver-

Tabelle 3.1: Vergleich zwischen Testreihen mit und ohne Permute

	Permute	ohne Permute	Differenz
Rechenzeit	109,2	82,8	31,88 %
Anzahl der Level	45,7	44,3	3,16 %
Rekordlevel	22,7	23,2	
generierte Moves	1511162,5	1477460,2	2,2811 %
akzeptierte Moves	39589,4	37400,8	5,85175 %
Kosten der Lösung	73362,5	73621,5	-0,351799 %
Standardabweichung	2,4461 %	2,4439 %	

(a) F70-Daten (120 Runden)

	Permute	ohne Permute	Differenz
Rechenzeit	633,07	503,23	25,79 %
Anzahl der Level	34,0	32,0	6,25 %
Rekordlevel	26,9	25,5	
generierte Moves	6704602,8	6321519,2	6,05999 %
akzeptierte Moves	119433,3	112581,9	6,0857 %
Kosten der Lösung	307158,0	302902,0	1,40507 %
Standardabweichung	1,8523 %	1,9827 %	

(b) DH8-Daten (20 Runden)

	Permute	ohne Permute	Differenz
Rechenzeit	3575,5	3264,1	9,54 %
Anzahl der Level	45,9	46,2	-0,649 %
Rekordlevel	34,2	34,5	
generierte Moves	40271731,5	40489657,7	-0,538227 %
akzeptierte Moves	378979,8	376855,7	0,563637 %
Kosten der Lösung	812510,5	809960,5	0,31483 %
Standardabweichung	0,6737 %	0,8364 %	

(c) X-Daten (20 Runden)

zeichnet, in welchem Level die beste Lösung durchschnittlich gefunden wurde. Die Standardabweichung der Kostenfunktion ist in Prozent der Kosten angegeben.

Zunächst zeigt sich, daß trotz gestiegenem Zeitaufwand bei den F70-Daten eine nur unwesentliche Verbesserung und bei DH8- und X-Daten sogar eine Verschlechterung der Kosten erreicht wurde. Doch selbst die Verbesserung bei den F70-Daten um 0,35% ist wertlos, da ein Lauf mit Levellänge 70 (statt 60) bei nur 100,47 Sekunden Rechenzeit eine Kostenminderung von 1,05% einbringt. Die Streuung der Ergebnisse ist bei F70- und DH8-Daten annähernd unverändert, bei X nimmt sie im Permute-Fall geringfügig ab. Eine signifikante Änderung liegt jedoch nicht vor.

Der Anstieg der Rechenzeit hat verschiedene Ursachen. Einerseits kommt es bei den F70- und DH8-Daten zu dem gewünschten Effekt, daß in den späteren Leveln häufiger gute Nachbarn gefunden werden. Dadurch steigt die Akzeptanzratio, und es kommt spät noch zu Rekorden, weshalb das Annealing später abbricht. Andererseits profitiert der Algorithmus nicht mehr von der Möglichkeit, die Kosten bereits betrachteter Nachbarn nicht neu berechnen zu müssen. Letzteres könnte den im Verhältnis zu den anderen Vergleichsgrößen enormen Anstieg der Rechenzeit erklären.

Um nachzuweisen, daß allein die Speicherung der Kosten schon bekannter Nachbarn Geschwindigkeitsvorteile in der beobachteten Größenordnung verursachen kann, wurde folgende Testreihe auf DH8-Daten durchgeführt: Zum einen lief 20mal der Algorithmus ohne Permute und mit den Parametern, die schon für Tabelle 3.1(b) benutzt wurden. Zu jedem einzelnen kam ein zweiter hinzu, der mit identischen Zufallszahlen versorgt wurde, der also während der gesamten Laufzeit die selben Entscheidungen traf und die gleichen Lösungen betrachtet. Der einzige Unterschied zum Originallauf bestand darin, daß die Frage, ob die Kosten eines Nachbarn bereits bekannt sind, immer mit „nein“ beantwortet wurde, was natürlich zu deren Neuberechnung führte.

Die Laufzeitzunahmen der einzelnen Paare lag zwischen 12,4% und 19,3%, im Durchschnitt bei 15,47%. Die unterschiedlichen Zeitdifferenzen sind, neben der statistisch schwankenden Anzahl mehrfach betrachteter Nachbarn, hauptsächlich auf verschieden frühe Abbrüche des Annealing zurückzuführen, da speziell in den letzten Leveln, also bei sehr niedriger Akzeptanzratio, vermehrt Nachbarn doppelt betrachtet werden. Die Paare mit eher kleiner Laufzeitzunahme (unter 14%) haben tatsächlich nach spätestens 31 Leveln abgebrochen, während die mit größerer Zunahme (über 17%) mindestens 38 Level durchliefen.

Da sich aus den in Tabelle 3.1(b) dargestellten 25,79% Zeitverlust durch Permute nach Verrechnung des ca. 6,1%igen Anstieges der anderen Vergleichsgrößen ein zu erklärender Verlust von 18,55% ergibt, scheint der nachgewiesene Laufzeiteffekt durch Kostenspeicherung mit durchschnittlich 15,47% etwas zu niedrig. Die Diskrepanz dürfte auf die bereits erwähnte Ungenauigkeit der Zeitmessungen zurückzuführen sein. Es läßt sich aber auf jeden Fall festhalten, daß die erwartete Größenordnung erreicht wird.

Insgesamt hat sich für die von uns verwendeten Datensätze die Permutation der Nachbarschaft als nicht vorteilhaft erwiesen.

3.6.2 Join/Split-Verhältnis

Während eines Testlaufes werden einige Millionen Moves vorgeschlagen, von denen einige zehn- bis hunderttausende angenommen werden. Bei jedem Split erhöht sich die Anzahl der Pairings um eins, während sie bei einem Join um eins vermindert wird. Die Zahl der Pairings einer Lösung liegt immer zwischen 1 und der Anzahl vorhandener Legs. Letztere ist stets sehr klein gegenüber der Zahl generierter oder auch akzeptierter Moves (unser größter Datensatz besteht aus 4612 Legs), weshalb man davon ausgehen kann, daß unter den akzeptierten Moves etwa gleich viele Joins und Splits sind.

In der Nachbarschaft einer Lösung befinden sich aber in der Regel mehr Joins als Splits. Aufgrund der gleichverteilten Auswahl eines Nachbarn werden Joins im Durchschnitt doppelt so oft wie Splits generiert, aber entsprechend seltener angenommen. Dies führte zu der Überlegung, von der Gleichverteilung abzusehen und vermehrt Splits vorzuschlagen, damit die Zahl generierter Moves und damit die Rechenzeit sinkt. Die einfachste Variante der Umsetzung bestand darin, direkt nach dem Generieren eines Moves im Falle eines Joins erneut zu würfeln. Wenn die Wahrscheinlichkeit eines Joins vorher p_J war, so sinkt sie damit auf p_J^2. Durch eine weitere Wiederholung wird p_J^3 erreicht. Sollte sich diese Änderung als nützlich erweisen, kann man die Umstrukturierung der Moveliste und damit eine professionellere Variante der Einflußnahme auf das Join-Split-Verhältnis angehen.

Tabelle 3.2: Auswirkungen eines veränderten Join-Split-Verhältnisses

	normales Annealing	erhöhte Splitzahl	stark erhöhte Splitzahl
Joinanteil	68,8%	56,9%	46,8%
Kostenfunktion	302458	310158	326760
Streuung	1,8074 %	2,6096 %	3,0222 %
Pairinglänge	3,758	3,686	3,648
generierte Moves	6113675	6851529	7143222
akzeptierte Moves	111318	124866	116407

Tabelle 3.2 zeigt die Ergebnisse einer Testreihe auf DH8-Daten. Im Gegensatz zu den vorher angestellten Überlegungen sinkt die Wahrscheinlichkeit eines Joins nicht auf $p_J^2 = 47{,}3\%$ und $p_J^3 = 32{,}6\%$. Dies ist darauf zurückzuführen, daß durch die neue Verteilung bei der Auswahl der Nachbarn ganz andere Bereiche des Suchraums bevorzugt werden und daher ein verändertes durchschnittliches Verhältnis zwischen Joins und Splits in der Moveliste vorliegt. Dafür spricht auch die leicht abnehmende Pairinglänge der Endergebnisse.

Bei einer steigenden Zahl generierter Moves bleibt die Zahl der akzeptierten tendenziell gleich, was zu einer leicht fallenden Akzeptanzratio führt. Eigentlich war der gegenteilige Effekt beabsichtigt; die Zahl der generierten Moves sollte sinken, da die „richtigen“ Moves vorgeschlagen werden.

Sowohl die Kostenfunktion als auch deren Streuung verschlechtern sich; spätestens an dieser Stelle muß der vorgenommene Versuch für gescheitert erklärt werden.

Es hat sich also gezeigt, daß es notwendig ist, mehr Joins als Splits vorzuschlagen, um zu den üblichen, guten Ergebnissen zu gelangen. Da sich die beiden Operatoren innerhalb der akzeptierten Moves die Waage halten, muß die Akzeptanzwahrscheinlichkeit eines Splits durchschnittlich etwas höher als die eines Joins liegen. Um den gewünschten Effekt zu erzielen müßte, auch dieses Mißverhältnis ausgeräumt werden, was aber ein Eingriff in die Kostenfunktion wäre.

3.7 Ein Ansatz für Parallelität: MIR

Aufgrund der recht starken Streuung unserer Ergebnisse bietet sich als Parallelisierung deren simpelste Form an: Man startet auf den p zur Verfügung stehenden Prozessoren p voneinander unabhängige Läufe und wählt am Ende das beste Ergebnis aus.

Fu-Hsieng Allisen Lee behandelt in seiner Doktorarbeit [Lee95] Simulated Annealing Ansätze auf parallelen Prozessoren, die via Message-Passing kommunizieren. Speziell geht es um optimales Design von Datenbanknetzwerken, ein Optimierungsproblem mit recht komplexer Kostenfunktion und einer Reihe von Nebenbedingungen. Dabei stellte er fest, daß der oben beschriebene MIR-Ansatz (Multiple Independent Runs) bessere Resultate hervorbrachte als eine tatsächliche Parallelisierung eines Simulated Annealing Laufes. Er geht sogar noch weiter: Die Qualität der Lösungen nimmt mit der Anzahl der unabhängigen Läufe zu, selbst wenn sie die Zahl der Prozessoren übersteigt. Es stellt sich also die Frage, ob auch auf einem Prozessor die Qualität ansteigt, wenn statt eines langen Laufes das Minimum mehrerer kurzer gebildet wird.

Um dies zu motivieren, nehmen wir für den Moment an, daß die in einer Testreihe mit konstantem Zeitaufwand t pro Lauf erreichten Kostenwerte normalverteilt sind. Median und Durchschnitt sind bei einer Normalverteilung identisch und hier mit μ_t bezeichnet; die Standardabweichung sei σ_t. Die Verteilungsfunktion $F_t(c)$ ist wie folgt definiert:

$$\begin{aligned}
F_t(c) &:= \mathbb{P}(\xi_t < c) && (3.7)\\
&= \int\limits_{-\infty}^{c} \frac{1}{\sqrt{2\pi}\sigma_t} e^{-\frac{1}{2}\left(\frac{x-\mu_t}{\sigma_t}\right)^2} dx && (3.8)\\
&= \int\limits_{-\infty}^{\frac{c-\mu_t}{\sigma_t}} \frac{1}{\sqrt{2\pi}} e^{-\frac{y^2}{2}} dy && (3.9)\\
&= \phi\left(\frac{c-\mu_t}{\sigma_t}\right) && (3.10)
\end{aligned}$$

ϕ ist die Standard-Normalverteilungsfunktion.

Betrachtet man je k Läufe mit Zeitaufwand t als einen Versuch, dessen Ergebnis das Minimum der erreichten Kosten ist, so ergibt sich eine Verteilungsfunktion $F_{k;t}(c)$:

$$\begin{aligned}
F_{k;t}(c) &:= \mathbb{P}(\min_{i=1}^{k}(\xi_t^i) < c) && (3.11)\\
&= \mathbb{P}(\bigvee_{i=1}^{k} \xi_t^i < c) &&\\
&= 1-(1-F_t(c))^k && (3.12)
\end{aligned}$$

Konsistenterweise gilt $F_t(c) = F_{1;t}(c)$, weshalb im folgenden nur noch letztere Schreibweise verwendet wird.

Der Median μ_F einer beliebigen (nicht notwendigerweise Normal-) Verteilungsfunktion F ist der Wert, den sie auf 0,5 abbildet (auch: 0,5-Quantil). Im Falle einer MIR-Testreihe mit je k Läufen ergibt sich für den Median $\mu_{F_{k;t}}$ (im folgenden kurz $\mu_{k;t}$):

$$\begin{aligned}
F_{k;t}(\mu_{k;t}) &= 0{,}5 && (3.13)\\
1-(1-F_{1;t}(\mu_{k;t}))^k &= 0{,}5 && (3.14)\\
F_{1;t}(\mu_{k;t}) &= 1-\sqrt[k]{0{,}5} && (3.15)\\
\phi\left(\frac{\mu_{k;t}-\mu_t}{\sigma_t}\right) &= 1-\sqrt[k]{0{,}5} && (3.16)\\
\mu_{k;t} &= \phi^{-1}\left(1-\sqrt[k]{0{,}5}\right)\cdot\sigma_t+\mu_t && (3.17)
\end{aligned}$$

Die Umformung zu (3.17) ist möglich, da ϕ streng monoton wächst und ϕ^{-1} folglich definiert ist. Für $k \geq 1$ ist $1 - \sqrt[k]{0{,}5} \leq 0{,}5$ und daher

$$p_k := -\phi^{-1}\left(1 - \sqrt[k]{0{,}5}\right) \geq 0. \tag{3.18}$$

Eine intuitivere Schreibweise für (3.17) ist

$$\mu_{k;t} = \mu_t - p_k \cdot \sigma_t. \tag{3.19}$$

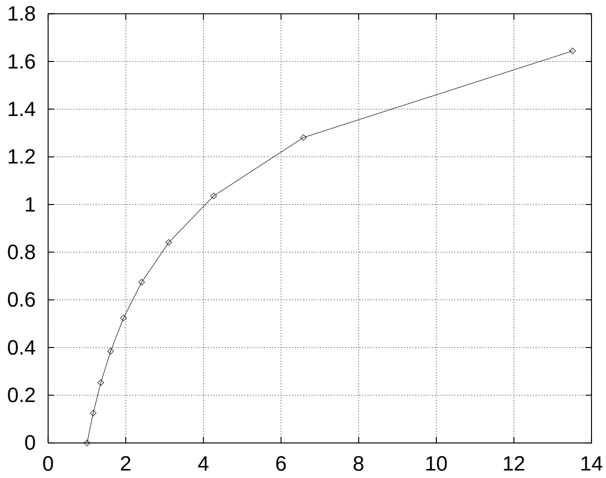


Abbildung 3.21: Die Funktion $-\phi^{-1}(1 - \sqrt[k]{0{,}5})$. Sie wurde mit Hilfe einer aus [Stö93] stammenden Tabelle für ϕ an einigen reellen Werten bestimmt und linear interpoliert. Die Werte p_k für natürliche k sollten sich anhand der Kurve abschätzen lassen.

$\mu_{k;t}$ läßt sich also aus Median und Standardabweichung einer Testreihe mit je einem Lauf bestimmen. Ein MIR-Ansatz mit k Läufen der Länge t lohnt sich gegenüber einem einzelnen Lauf mit Zeit kt, wenn $\mu_t - \mu_{1;kt}$[2], also die Verbesserung bei k-facher Zeit, kleiner als $p_k \cdot \sigma_t$, die Verbesserung durch Minimumbildung, ist.

Da für unser Problem keine Formel für $\mu_{1,t}$ oder σ_t vorliegt und die Annahme einer Normalverteilung schon wegen der Beschränktheit der Kostenfunktion nicht korrekt sein kann, läßt sich nur anhand eines Testlaufes eine Aussage machen. Die Herleitung von (3.19) sollte jedoch motivieren, daß der MIR-Ansatz auch für andere Verteilungen sinvoll sein könnte.

Abbildung 3.22 zeigt das Ergebnis eines MIR-Tests auf DH8-Daten und Abbildung 3.23 die zugehörigen Standardabweichungen. Für die Variation

[2] $\mu_{1;kt}$ könnte eigentlich als μ_{kt} geschrieben werden. Um aber Verwechslungen mit $\mu_{k;t}$ zu vermeiden, sehe ich davon ab.

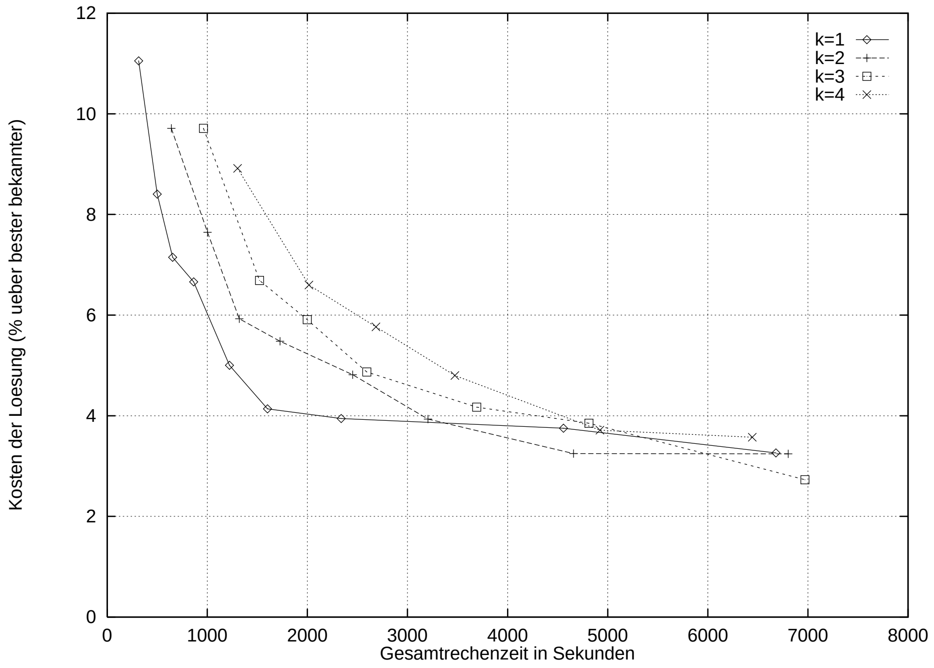


Abbildung 3.22: Multiple Independent Runs (DH8-Daten, 20 Runden). $\alpha \in \{0{,}95^{\frac{1}{i}} | i \in \{0{,}5; 1; 1{,}5; 2; 3; 4; 6; 9; 12; 18\}\}$

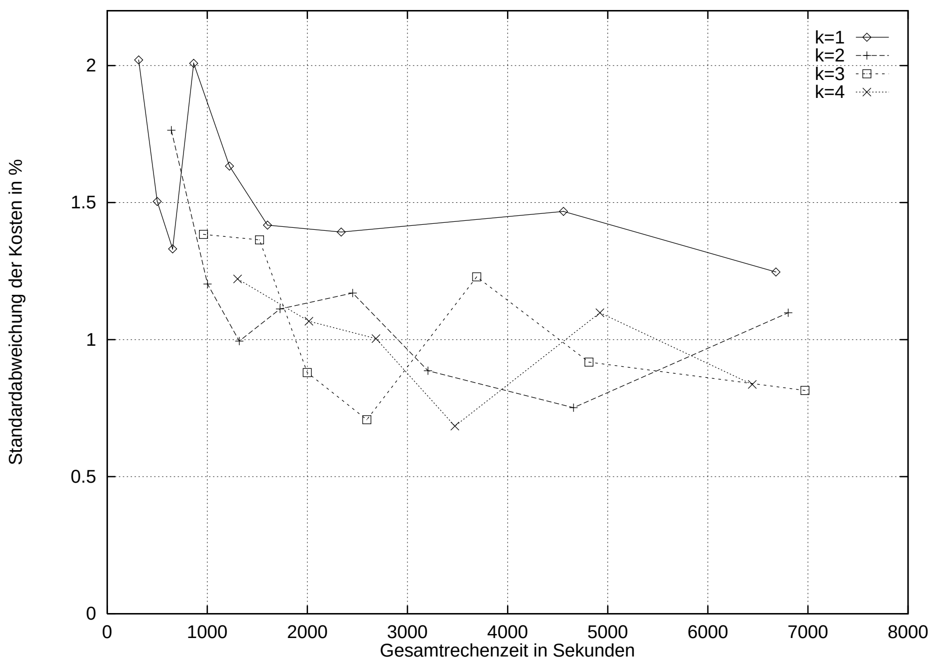


Abbildung 3.23: Die Standardabweichung zu Abbildung 3.22

der Rechenzeit wurde hier die Temperaturreduktion genutzt. Obwohl diese Testreihe mit einer Gesamtlaufzeit von 254,7 Stunden, also fast 11 Tagen, schon sehr aufwendig war, beginnen die interessanten Effekte erst im oberen Bereich. Bis zu einer Laufzeit von etwa 3300 Sekunden liegen die Kurven noch gestaffelt mit $k = 1$ als Favorit. Zwischen 3300 und 6000 liefert $k = 2$ die besten Werte und ab 6000 führt $k = 3$. Für einen Vorsprung von $k = 4$ genügt der abgedeckte Zeitbereich vermutlich noch nicht. Da die gemessene Standardabweichung bei $k = 1$ zwischen 1600 und 6700 Sekunden kaum abgenommen hat, müßte $k = 4$ bei noch höheren Laufzeiten wesentlich bessere Lösungen liefern. Abbildung 3.23 zeigt einen weiteren positiven Effekt des MIR-Ansatzes: die Abnahme der Streuung. Zumindest liegen die Standardabweichungen für $k = 2; 3; 4$ im Schnitt deutlich unter denen von $k = 1$, auch wenn sie sich untereinander kaum unterscheiden.

Eine MIR-Testreihe für F70 brachte leider keine verwertbaren Ergebnisse, da infolge der starken Streuung die erzeugten Kurven chaotisch verliefen. Für X-Daten wurde von einem Test abgesehen, da der notwendige Aufwand mit ca. 69 Tagen Prozessorzeit die vorhandenen Kapazitäten überlastet hätte.

Für die Verwendung auf nur einem Prozessor kann zusammenfassend festgestellt werden, daß der MIR-Ansatz von Vorteil ist, wenn sehr hohe Rechenzeiten zur Verfügung stehen, die optimal ausgenutzt werden sollen. Im niederen Zeitbereich dagegen bleibt ein einzelner Lauf die beste Wahl.

Für eine tatsächliche Parallelisierung scheint MIR ein sehr starkes Verfahren zu sein, da der Kommunikationsaufwand minimal ist und das sequentielle Programm unverändert übernommen werden kann.

Üblicherweise wird die Güte einer Parallelisierung an dem erreichten Geschwindigkeitsvorteil gemessen. Der sogenannte „Speedup“ ist ein Maß dafür und definiert als Quotient von sequentieller und paralleler Laufzeit. Bei der komplexitätstheoretischen Beurteilung eines parallelen Algorithmus wird als sequentielle Laufzeit die des besten bekannten Algorithmus verwendet (siehe [Akl89]), weshalb ein linearer Speedup, also eine k-fache Geschwindigkeit bei k Prozessoren, der beste erreichbare ist.

Davon abweichend möchte ich hier den MIR-Ansatz im Vergleich zum einfachen Annealing-Lauf bewerten. Grundlage hierfür ist die bereits vorhandene MIR-Testreihe. Die für ein $k > 1$ jeweils zusammengefaßten k Einzelläufe werden betrachtet, als wären sie parallel auf $p = k$ Prozessoren entstanden. Die notwendige Rechenzeit ergibt sich aus dem Maximum über den k Läufen.

Aussagen über verschiedene Laufzeiten sind natürlich nur sinnvoll, wenn die Ergebnisse gleich oder zumindest ähnlich sind. Da man beim Annealing die zu erreichende Lösungsqualität nicht direkt einstellen kann, wurde aus den vorhandenen Daten der MIR-Testreihe für jedes $k(= p)$ die Parametereinstellung α ausgewählt, bei der die Lösungsqualität im Median mindestens ein festgesetztes q erreichte. Abbildung 3.24 zeigt für verschiedene q den Geschwindigkeitsvorteil über der Anzahl der Prozessoren p. Die jeweils ausgewählten Testdaten können die Qualitätsvorgabe q verschieden stark unterbieten, weshalb die entstandenen Kurven nur als grobe Orientierung verstanden werden dürfen.

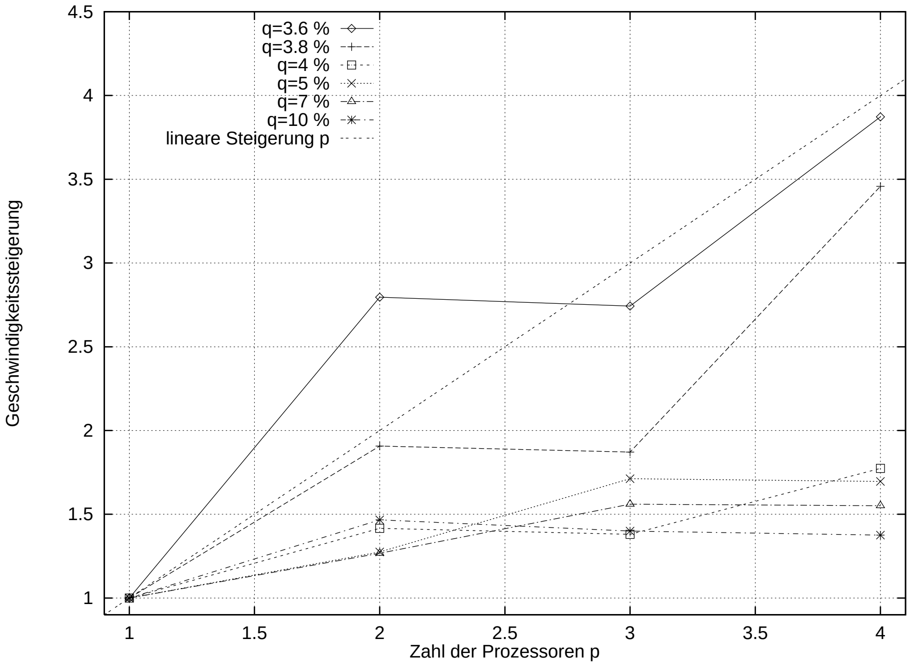


Abbildung 3.24: Die erreichten Geschwindigkeitssteigerungen der MIR-Parallelisierung bei verschiedenen Qualitätsvorgaben q.

Es wird deutlich, daß sich die Parallelisierung für große q (also kleine Rechenzeiten) kaum lohnt. Die Kurven $q \geq 4$ erreichen für zwei Prozessoren noch Werte zwischen 1,25 und 1,5, um dann aber bis $p = 4$ nur noch wenig zu steigen. Mit kleineren Werten für q werden weitaus bessere Steigerungsraten erreicht. Daß für $q = 3{,}6$ auf zwei Prozessoren sogar ein Wert über p liegt, sollte nach den Ergebnissen aus Abbildung 3.22 nicht mehr verwundern.

Eine Parallelisierung durch MIR ist also von Nutzen, wenn mit einem hohen Einsatz an Rechenzeit sehr gute Lösungen produziert werden sollen. Geht es jedoch darum, den Zeitaufwand für die Konstruktion von Lösungen minderer Güte zu senken, ist dieses simple Verfahren eher ungeeignet. Sowohl für den sequentiellen als auch den parallelen Fall gilt, daß MIR sinnvoll eingesetzt werden kann, wenn die Ergebnisse stark streuen und weitere Erhöhungen der Rechenzeit die produzierten Lösungen kaum noch verbessern.

Kapitel 4

Zusammenfassung

Da weder die Kosten einer optimalen Lösung noch eine gute untere Schranke dafür bekannt sind, ist keine absolute Beurteilung der durch Simulated Annealing produzierten Lösungen möglich. Wie in [Emd98] jedoch ausgeführt wird, ist die Lösungsqualität der anderen untersuchten lokalen Suchverfahren entweder mit denen des Annealing vergleichbar oder schlechter. Die Kosten der für den F70-Datensatz vorhandenen Handlösung konnten unterboten werden und die beiden getesteten Greedyverfahren produzieren vergleichsweise schlechte Lösungen.

Auf dieser Grundlage läßt sich feststellen, daß Simulated Annealing sehr gut für die Lösung des Problems der Pairing-Generierung geeignet ist. In vertretbarer Zeit werden bereits recht gute und bei langen Laufzeiten sehr gute Lösungen gefunden. Zudem ermöglicht das geometrische Kühlschema mit seinen Parametern Levellänge und Temperaturreduktion, die für einen Lauf ungefähr notwendige Rechenzeit im voraus festzulegen, wobei sich die Lösungsqualität erwartungsgemäß mit Zunahme der Laufzeit verbessert. Damit ist sowohl die langfristige Einsatzplanung, bei der die Kostenminimalität im Vordergrund steht, als auch die schnelle Planung bei kurzfristigen Änderungen mit dem gleichen Algorithmus möglich.

Die Relaxierung der Zulässigkeit einer Lösung hat im Zusammenhang mit einer entsprechend klein gewählten Bestrafung und dem „Admissiblator“ eine wesentliche Verbesserung der Lösungskosten erbracht. Auch mit der Wahl einer günstigen Startlösung wurde eine leichte Kostenabnahme, aber vor allem eine Reduzierung der notwendigen Rechenzeit erreicht. Beide Konzepte sollten bei einer praktischen Anwendung berücksichtigt werden. Im Gegensatz dazu konnten die beiden Annealing-Varianten aus Kapitel 3.6 nicht zu einer Verbesserung des Algorithmus beitragen.

Bei der industriellen Anwendung des Simulated Annealing könnte sich eventuell die Wahl der verschiedenen Parameter als problematisch erweisen, wenn keine Zeit für umfangreiche Testreihen zur Verfügung steht. Die in den Kapiteln 3.2, 3.3 und 3.4 beschriebenen Erkenntnisse sollten jedoch eine vernünftige Wahl der entsprechenden Parameter anhand weniger Testläufe ermöglichen.

Man sollte nicht aus den Augen verlieren, daß Simulated Annealing ein randomisiertes Verfahren ist. Alle hier gemachten Angaben zur Lösungsqualität sind statistische Werte, von denen das Ergebnis eines einzelen Laufes stark abweichen kann. Eine brauchbare obere Schranke der erzielten Kosten kann aus ihnen nicht gewonnen werden. Für die Berechnung hochwertiger Lösungen bietet sich die Benutzung des MIR-Ansatzes aus zweierlei Gründen an: Einerseits werden im hohen Zeitbereich bessere Lösungen produziert, andererseits sinkt schon im mittleren Zeitbereich die Streuung und damit die Gefahr, eine besonders schlechte Lösung zu generieren. Zudem ist MIR ein sehr simples Verfahren. Eine Parallelisierung auf mehrere Rechner erfordert kaum Aufwand und ist notfalls sogar „von Hand", also durch mehrfachen Start des gleichen Programms, möglich.

Insgesamt hoffe ich, daß die am Prototyp gesammelten und in dieser Arbeit dargestellten Ergebnisse und Erfahrungen eine gute Grundlage für die Entwicklung einer kommerziellen Planungssoftware bieten.

Literaturverzeichnis

[AK89] Aarts, Emile H.L. und Jan Korst: *Simulated Annealing and Boltzmann Machines.* John Wiley & Sons, 1989.

[Akl89] Akl, Selim G.: *The Design And Analysis Of Parallel Algorithms.* Prentice Hall, 1989.

[AS89] Aragon, Cecilia R. und Raimund G. Seidel: *Randomized Search Trees.* Proc. 30th IEEE FoCS, Seiten 540–545, 1989.

[AvL85] Aarts, Emile H.L. und P.J.M. van Laarhoven: *A new polynomial time cooling schedule.* Proc. IEEE Int. Conf. on Computer-Aided-Design, Seiten 206–208, 1985.

[Čer85] Černy, V.: *Thermodynamical Approach To The Traveling Salesman Problem; An Efficient Simulation Algorithm.* Journal of Optimization Theorie and Applications, 45:41–51, 1985.

[Emd98] Emden-Weinert, Thomas: *Kombinatorische Optimierungsverfahren für das Crew Scheduling Problem.* Doktorarbeit, Humboldt-Universität zu Berlin, 1998. in Vorbereitung.

[Fel50] Feller, William: *An Introduction to Probability Theory and Its Applications*, Band 1. John Wiley & Sons, 1950.

[Haj88] Hajek, Bruce: *Cooling Schedules For Optimal Annealing.* Mathematics of Operations Research, 13(2):311–329, May 1988.

[JAA] JAA Operations Committee: *Explanatory Note Covering The Joint Aviation Authorities Flight And Duty Time Limitations And Rest Requirements Proposal.*

[JAMS89] Johnson, David S., Cecilia R. Aragon, Lyle A. McGeoch und Catherine Schevon: *Optimization by Simulated Annealing: An experimental evaluation; Part I, Graph Partitioning.* Operations Research, 37(6):865–892, 1989.

[JAMS91] JOHNSON, DAVID S., CECILIA R. ARAGON, LYLE A. MCGEOCH und CATHERINE SCHEVON: *Optimization by Simulated Annealing: An experimental evaluation; Part II, Graph Coloring and Number Partitioning.* Operations Research, 39(3):378–406, 1991.

[KGV83] KIRKPATRICK, S., C.D. GELATT JR. und M.P. VECCHI: *Optimization By Simulated Annealing.* Science, 220(4598):671–680, 1983.

[Lee95] LEE, FU-HSIENG ALLISEN: *Parallel Simulated Annealing On A Message-Passing Multi-Computer.* Doktorarbeit, Utah State University, 1995.

[LW85] LEHN, JÜRGEN und HELMUT WEGMANN: *Einführung in die Statistik.* B.G. Teubner Stuttgard, 1985.

[MRR+53] METROPOLIS, N., A. ROSENBLUTH, M. ROSENBLUTH, A. TELLER und E. TELLER: *Equation of state calculations by fast computing machines.* Journal of Chemical Physics, 21:1087–1092, 1953.

[MRS86] MITRA, D., F. ROMEO und A.L. SANGOIVANNI-VINCENTELLI: *Convergence and finite-time behavior of simulated annealing.* Advances in Applied Probability, 18:747–771, 1986.

[PS82] PAPADIMITRIOU, CHRISTOS H. und KENNETH STEIGLITZ: *Combinatorial Optimization.* Prentice-Hall, 1982.

[Stö93] STÖCKER, HORST (Herausgeber): *Taschenbuch mathematischer Formeln und moderner Verfahren.* Verlag Harri Deutsch, 1993.

Thesen

1. Die Personalkosten einer Fluggesellschaft stellen nach denen für Treibstoff die zweithöchsten Betriebskosten dar. Ihre Optimierung trägt maßgeblich zur Wirtschaftlichkeit und Konkurrenzfähigkeit des Unternehmens bei.

2. Die Dienstplanung des Flugpersonals, das sogenannte Crew-Scheduling, wird üblicherweise in die zwei Phasen Pairinggenerierung und Rostering unterteilt. Innerhalb des PARALOR-Projektes hat unsere Arbeitsgruppe das Problem der Pairinggenerierung bearbeitet. Für dessen Lösung wurden verschiedene Verfahren der lokalen Suche auf ihre Eignung hin untersucht.

3. Lokale Suchverfahren abstrahieren von der eigentlichen Problemstellung, indem sie lediglich eine indirekt gegebene Lösungsmenge betrachten, auf der eine Kosten- und eine Nachbarschaftsfunktion definiert sind. Sie bieten damit den Vorteil einer leichten Übertragbarkeit auf verschiedenste kombinatorische Optimierungsprobleme.

4. Die von uns für das Pairingproblem gewählte Kostenfunktion setzt sich aus sieben Einzelkosten zusammen. Einerseits spiegelt sie die durch eine Lösung innerbetrieblich verursachten Kosten wider, andererseits dient sie dazu, das verwendete lokale Suchverfahren zu lenken und damit bestimmte Eigenschaften der Lösung zu fördern. So wurde beispielsweise die tarifvertragliche Zulässigkeit einer Pairingzerlegung als Teil der Kostenfunktion modelliert.

5. Bei Simulated Annealing handelt es sich um einen randomisierten lokalen Suchalgorithmus, der in Anlehnung an Theorien aus der statistischen Physik über das Kristallwachstum in Schmelzen entstanden ist. Die Güte der gefundenen Lösungen bzw. die Wahrscheinlichkeit, mit der gute Lösungen gefunden werden, sowie die benötigte Rechenzeit hängen von einer Reihe Parameter ab, die unter dem Begriff „Kühlschema“ zusammengefaßt werden.

6. Um mit Simulated Annealing optimale Lösungen mit Wahrscheinlichkeit 1 zu produzieren, ist ein sehr langsames Kühlschema notwendig, das für interessante Probleme in der Regel zu exponentiellen Laufzeiten führt. Aus diesem Grunde werden in der Praxis Kühlschemata benutzt, die in polynomieller Zeit möglichst gute, aber zumeist nicht optimale Lösungen erzeugen.

7. Im Rahmen des PARALOR-Projektes entstand ein Prototyp, mit dem zahlreiche Testreihen durchgeführt wurden, um praxisnahe statistische Aussagen über die verwendeten Verfahren und den Einfluß verschiedener Parameter zu gewinnen. Die benutzten Testdaten wurden vom Industriepartner zur Verfügung gestellt.

8. Die Lösungsqualität zweier getesteter Greedy-Verfahren ist unbefriedigend, obwohl von einem der beiden sehr hohe Laufzeiten benötigt wurden.

9. Für das Simulated Annealing wurden zwei Kühlheuristiken untersucht. Das geometrische Schema nach Johnson et al. hat sich dabei als günstiger erwiesen als das adaptive nach Aarts und van Laarhoven, da es etwas bessere Lösungen produziert und vor allem leichter zu steuern ist.

10. Die Relaxierung der Zulässigkeit einer Lösung, also die Bestrafung unzulässiger Lösungen duch die Kostenfunktion statt ihres generellen Verbotes hat sich für das Simulated Annealing als sehr erfolgreich erwiesen. Zulässige Lösungen, die mit einer geringeren Bestrafung erzeugt wurden, sind durchschnittlich besser als solche mit hoher Bestrafung. Die besten Lösungen entstehen sogar, wenn so gering bestraft wird, daß in fast jeder Lösung einige Unzulässigkeiten enthalten sind, die anschließend durch ein sehr simples Verfahren beseitigt werden. Das Annealing profitiert also von der Möglichkeit, während seiner Suche auch unzulässige Lösungen zu generieren.

11. Die Initiallösung, von der ausgehend der Lösungsraum durchsucht wird, hat großen Einfluß auf Simulated Annealing. Durch vorherigen Start einer recht einfachen Aufbauheuristik konnte sowohl die Lösungsqualität verbessert als auch die Laufzeit erheblich reduziert werden.

12. Die Abbruchbedingung des geometrischen Kühlschemas wird im wesentlichen durch die zwei Parameter Abbruchsakzeptanzratio und Kettenlänge bestimmt. Ohne die Verwendung der Kettenlänge gibt es für die Abbruchsakzeptanzratio einen optimalen Wert, oberhalb dessen durch einen vorzeitigen Abbruch schlechte Lösungen entstehen und unterhalb dessen unnötig Rechenzeit verbraucht wird. Mit einer vernünf-

tig gewählten Kettenlänge kann zumindest die Gefahr des zu frühen Abbruches abgewendet werden.

13. Das geometrische Kühlschema verfügt über zwei Parameter, mit denen die Laufzeit sehr gut gesteuert werden kann: Temperaturreduktion und Levellänge. Für Variationen jedes der beiden Parameter gilt, daß mit zunehmender Laufzeit bessere Lösungen erzeugt werden. Keiner der beiden hat sich jedoch als günstiger für die Laufzeitsteuerung erwiesen.

14. Zwei getestete Modifikationen des Annealing-Algorithmus haben leider nicht den erhofften Erfolg gebracht. Zum einen sollte statt einer gleichverteilt zufälligen Auswahl der betrachteten Nachbarlösungen eine zufällige Permutation über ihnen verwendet werden. Der dadurch erhoffte Laufzeitvorteil blieb nicht nur aus, es kam sogar zu einer Erhöhung der Rechenzeit. Auch die zweite Variante betrifft die Wahl der Nachbarn. Da die beiden Nachbarschaftsoperatoren im Durchschnitt gleichhäufig ausgeführt werden, sollten sie auch mit gleicher Wahrscheinlichkeit vorgeschlagen werden. Doch es kam zu einer Zunahme der Rechenzeit und einer Verschlechterung der Kosten.

15. Der in [Lee95] vorgeschlagene triviale Parallelisierungsansatz „MIR“ eignet sich für randomisierte Optimierungsverfahren, deren Ergebnisse stark streuen und bei denen eine Erhöhung der Rechenzeit kaum noch Verbesserungen der Lösungsqualität bewirkt. Unsere Testläufe belegen, daß sogar auf nur einem Prozessor mit MIR im Mittel bessere Lösungen gefunden werden, wenn sehr hohe Laufzeiten zur Verfügung stehen. Entsprechend hoch ist in diesem Zeitbereich das Potential bei einer Parallelisierung.