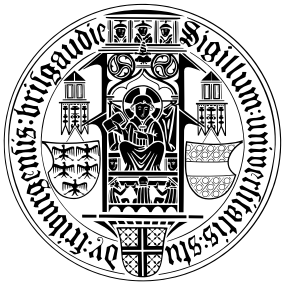

Prof. Dr. Bernhard Nebel
Lehrstuhl für
Grundlagen der Künstlichen Intelligenz
Albert-Ludwigs-Universität
Freiburg im Breisgau

Diplomarbeit

Komplexität und Berechnung der h^+-Heuristik

Christoph Betz

Betreuer: Malte Helmert

Februar 2009

Danksagungen

Bei Herrn Prof. Dr. Bernhard Nebel bedanke ich mich für die Möglichkeit, diese Arbeit an seinem Lehrstuhl anzufertigen, insbesondere für seine Bereitschaft, für diese Arbeit als Gutachter zur Verfügung zu stehen.

Ich möchte mich ganz herzlich bei meinem Betreuer Malte Helmert bedanken. Ungeachtet der Tatsache, dass er eigentlich nie Zeit hatte und neben einer Vorlesung noch Vorträge für mehrere Konferenzen vorzubereiten hatte, hat er sich immer Zeit für mich genommen. Wenn ein persönliches Treffen nicht möglich war, hat er meine Fragen stets geduldig per Mail beantwortet. Er hat mir als Programmieranfänger bei meinen programmiertechnischen Fragen geholfen und für die theoretischen Aspekte der Arbeit Ideen geliefert. Vielen, vielen Dank für diese hervorragende Betreuung, was sicherlich alles andere als selbstverständlich ist!

Weiterhin möchte ich mich bei meinen Kommilitonen Marina Klingele und Michael Meier bedanken, die meine Arbeit Korrektur gelesen haben und zahlreiche Anregungen zur Verbesserung hatten. Marina war außerdem nach der Korrektur so geduldig, mir meine Fragen zu ihren Änderungen zu beantworten.

Mein Dank gilt ebenfalls meinem Kommilitonen Matthias Heizmann. Er hat mir als absolutem Linuxbeginner, der sich wohl nie ganz mit diesem Betriebssystem anfreunden kann, die wichtigsten Tricks gezeigt, um damit klarzukommen. Ohne seine Hilfestellung hätte ich weder Archive erstellen können noch Ruhe im Zimmer gehabt, da ohne seinen Tipp das *screen*-Kommando zu nutzen mein Rechner Tag und Nacht gelaufen wäre.

Erklärung

Hiermit erkläre ich, dass ich diese Arbeit selbstständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Arbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Freiburg im Breisgau, Februar 2009, _____

Unterschrift

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die Wahl der Heuristik kann beim Planen mit Suchalgorithmen wie A^* darüber entscheiden, ob man nach kurzer Zeit eine optimale Lösung findet oder dies einige Zeit in Anspruch nimmt.
Eine Heuristik wird als gut angesehen, wenn sie möglichst dicht an den tatsächlichen Kosten liegt. Der alle zwei Jahre stattfindende IPC-Planungswettbewerb [McD00, LKS$^+$00, Bac01, LF03, HE05, HET$^+$06] stellt mehrere Probleme zur Verfügung, an denen neue Planer gemessen werden. Malte Helmert und Robert Mattmüller haben gezeigt [HM08], dass die h^+-Heuristik sich bei sieben bekannten Domänen der IPC-Planungswettbewerbe von den tatsächlichen Kosten nur um einen konstanten Faktor unterscheidet. Die konstanten Faktoren sind in Abbildung 1.1 dargestellt. Da die h^+-Heuristik zulässig und monoton ist, ist sie geeignet, optimale Lösungen mithilfe der A^*-Suche zu finden. Doch da die h^+-Heuristik im Allgemeinen NP-schwer zu berechnen ist [Byl94], wurde sie bisher selten in der Praxis untersucht.
Die konstanten Faktoren in Abbildung 1.1 sind nicht allgemeingültig zu verstehen. Wenn wir sagen, h^+ unterscheidet sich auf einer Domäne um den konstanten Faktor $c \in \mathbb{R}$ von den tatsächlichen Kosten h^*, so bedeutet dies formal, dass für den Anfangszustand s_0 jeder Instanz dieser Domäne $h^+(s_0) \geq ch^*(s_0) + o(h^*(s_0))$ gilt und dass es unendlich viele Instanzen gibt, so dass für den Anfangswert s_0 einer solchen Instanz $h^+(s_0) \leq ch^*(s_0) + o(h^*(s_0))$ gilt. Der konstante Faktor liefert neben einer unteren Abschätzung nur eine Grenzwertaussage.
In dieser Diplomarbeit werde ich die sieben erwähnten Domänen daraufhin untersuchen, ob sich die h^+-Heuristik polynomiell auf ihnen berechnen lässt und ob sich die Grenzwertaussage des konstanten Faktors in der Praxis bestätigt. Dafür habe ich die h^+-Heuristik für einige dieser Domänen implementiert. Für diese Domänen werde ich weiterhin den praktischen Nutzen der Heuristik untersuchen, indem ich sie mit der Merge-And-Shrink-Heuristik [HHH07], die den aktuellen Entwicklungsstand bei domänenun-

Domäne	konstanter Faktor
Miconic-Strips	6/7
Miconic-Simple-ADL	3/4
Schedule	1/4
Gripper	2/3
Blocksworld	1/4
Satellite	1/2
Logistics	3/4

Abbildung 1.1: Die konstanten Faktoren, um die sich die h^+-Heuristik von den tatsächlichen Kosten unterscheidet

abhängigem Planen darstellt, vergleiche.
Im folgenden Kapitel 2 werden die Grundlagen erklärt, die für das Verständnis dieser Arbeit nötig sind. In den darauffolgenden sechs Kapiteln, die in sich abgeschlossen sind und nach der Lektüre von Kapitel 2 unabhängig voneinander gelesen werden können, werden die einzelnen Domänen behandelt. Die Kapitel sind identisch aufgebaut, nach der Definition des Problems wird beschrieben, wie die h^+-Heuristik in der jeweiligen Domäne berechnet werden kann, wenn sie implementiert wurde, folgt ein praktischer Vergleich der h^+- und der Merge-And-Shrink-Heuristik. Letzteres ist bei Miconic-Strips, Gripper, Blocksworld, Satellite und Logistics der Fall, bei Miconic-Simple-ADL und Schedule wurde darauf verzichtet. Wegen der Ähnlichkeit von Miconic-Strips und Miconic-Simple-ADL werden beide gemeinsam in Kapitel 3 behandelt, die übrigen fünf Domänen erhalten ein eigenes Kapitel. In Kapitel 9 schließlich werden die Ergebnisse der Arbeit ausgewertet und die konstanten Faktoren aus Abbildung 1.1 mit der Praxis verglichen.

Kapitel 2

Grundlagen

In diesem Kapitel werden die Grundlagen erklärt, die für das Verständnis der Arbeit nötig sind. Neben allgemeinen Begriffen des Planens wird kurz auf das zugrundeliegende Suchverfahren A^* eingegangen.

2.1 Notation

Zunächst ein paar Symbole und Schreibweisen, die in der Arbeit auftreten werden:

$\mathbb{N}$	Die natürlichen Zahlen $\{1, 2, 3, \ldots\}$ beginnend bei 1
$\mathbb{R}$	Die reellen Zahlen
2^S	Die Potenzmenge der Menge S, d.h. $2^S = \{M \mid M \subseteq S\}$ ist die Menge aller Teilmengen von S
$f\|_{K'}$	Für eine Abbildung $f : K \to M$ bezeichnet $f\|_{K'} : K' \to M$ mit $f\|_{K'}(k') = f(k')$ für $k' \in K'$, die Einschränkung von f auf K'
$\|S\|$	Die Mächtigkeit der Menge S, d.h. die Anzahl der Elemente von S, wenn S endlich und ∞ sonst
ZHK	Abkürzung für Zusammenhangskomponente. Eine Zusammenhangskomponente in einem ungerichteten Graphen ist eine maximale wegweise verbundene Knotenmenge
1_p	Die bedingte 1-Funktion, die genau dann 1 annimmt, wenn p wahr ist und 0 sonst

2.2 SAS^+-Planen

Wenn von Planen gesprochen wird, ist meist der Begriff des STRIPS-Planens gemeint. Der Planer, in den ich meine Heuristik eingebaut habe, arbeitet mit einem etwas anderen Ansatz, SAS^+ genannt. Die Domänenbeschreibung bei

den Planungswettbewerben ist in STRIPS oder dessen Erweiterung ADL gehalten, Beschreibungen in diesen Beschreibungssprachen können nach SAS^+ transformiert werden. Die Beschreibung der Domänen in den nachfolgenden Kapiteln wird direkt in SAS^+ erfolgen, dies ist meist kürzer als die STRIPS-Kodierung. Die folgende Definition ist eine Abwandlung der Definition aus der Arbeit von Malte Helmert [Hel04].

Definition 2.1 (Zustandsraum)
Ein SAS^+-Zustandsraum oder kurz Zustandsraum ist ein 4-Tupel $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, S_* \rangle$, bestehend aus den folgenden Komponenten:

- *$\mathcal{V} = \{v_1, \ldots, v_n\}$ ist eine endliche Menge von Zustandsvariablen, jede Variable $v \in \mathcal{V}$ besitzt eine zugehörige endliche Domäne $\mathcal{D}_v$.*
  Eine partielle Variablenzuweisung über $\mathcal{V}$ ist eine Funktion s auf einer Teilmenge von $\mathcal{V}$, so dass $s(v) \in \mathcal{D}_v$, wenn $s(v)$ definiert ist. Wenn $s(v)$ für alle $v \in \mathcal{V}$ definiert ist, nennen wir s einen Zustand.
- *$\mathcal{O}$ ist eine Menge von Operatoren, wobei ein Operator ein Paar $\langle pre, eff \rangle$ von partiellen Variablenzuweisungen ist. Wir nennen pre Vorbedingungen und eff Effekte. Operatoren bezeichnen wir auch als Aktionen. Wenn die Vorbedingungen von $o \in \mathcal{O}$ in einem Zustand s erfüllt sind, sagen wir, o ist für s definiert.*
- *s_0 ist ein ausgezeichneter Zustand, Anfangszustand genannt.*
- *S_* ist eine ausgezeichnete Zustandsmenge, die Menge der Zielzustände genannt.*

In unserem Fall wird eine Variable meist als ein Tupel geschrieben, das aus Mengen und Funktionen besteht. In den späteren Kapiteln werden wir eine andere Beschreibung der Domänen als über den Zustandsraum geben, diese Beschreibung bezeichnen wir als Problemstellung. Als Domäne bezeichnen wir die Abbildung der Problemstellung auf den Zustandsraum.

Definition 2.2 (Relaxierung)
Die Relaxierung eines Zustandsraums $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, S_* \rangle$, der aus einer STRIPS- oder ADL-Beschreibung entstand, ist ein Zustandsraum $\Pi' = \langle \mathcal{V}, \mathcal{O}', s_0, S_* \rangle$. Die Menge $\mathcal{O}'$ der neuen Operatoren ist dabei die Menge der relaxierten Operatoren der zugrundeliegenden Beschreibung, d.h. die negativen Effekte wurden aus ihnen entfernt.

Bei uns werden die Variablen im Zustandsraum und seiner Relaxierung nicht identisch sein, die erwähnten Tupel enthalten in der Relaxierung häufig Potenzmengen der ursprünglichen Mengen und die Funktionen bilden auf Teilmengen ab. Dies entspricht strenggenommen nicht der Definition, wird

jedoch durch Identifizieren der einelementigen Mengen mit ihrem einzigen Element, d.h. $d \simeq \{d\}$, legitimiert. Als relaxierte Domäne werden wir das Abbilden eines Zustands s auf die Relaxierung bezeichnen. Der Zustand s wird der neue Anfangszustand sein, was strenggenommen nicht der Definition der Relaxierung entspricht.

2.3 Pläne und Heuristiken

Definition 2.3 (Plan)
Ein Plan $p = o_1, \ldots, o_m$ für einen Zustandsraum $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, S_* \rangle$ ist eine endliche Folge von Operatoren $o_i \in \mathcal{O}$, mit $i \in \{1, \ldots, m\}$, die den Anfangszustand in einen Zielzustand überführen. D.h. es existiert eine Menge $s_1, \ldots, s_m$ von Zuständen, so dass gilt,

- *o_i ist für Zustand s_{i-1} definiert und $o_i(s_{i-1}) = s_i$ für alle $i \in \{1, \ldots, m\}$.*
- *$s_m \in S_*$.*

Für einen Plan $p = o_1, \ldots, o_m$ nennen wir m seine Länge und schreiben dafür $|p|$.
Wenn für einen Plan p_* gilt, dass $|p_*| \leq |p|$ für alle Pläne p, so bezeichnen wir p_* als optimalen Plan.

Wir bezeichnen das Finden eines Plans in der Relaxierung auch als das Finden eines h^+-Plans oder relaxierten Plans.

Definition 2.4 (Domänen-Instanz, Domänen-Problemarten)
Eine $\mathcal{D}$-Instanz ist ein konkreter durch eine Domäne $\mathcal{D}$ entstandener Zustandsraum.
Eine relaxierte $\mathcal{D}$-Instanz ist ein konkreter durch eine relaxierte Domäne $\mathcal{D}$ entstandener Zustandsraum.
Das $\mathcal{D}$-Längen-Problem ist die Frage, ob es einen Plan mit Länge höchstens k für eine $\mathcal{D}$-Instanz gibt.
Das relaxierte $\mathcal{D}$-Längen-Problem ist die Frage, ob es einen Plan mit Länge höchstens k für eine relaxierte $\mathcal{D}$-Instanz gibt.
Das $\mathcal{D}$-Problem ist das Finden eines optimalen Plans für eine $\mathcal{D}$-Instanz.
Das relaxierte $\mathcal{D}$-Problem ist das Finden eines optimalen Plans für eine relaxierte $\mathcal{D}$-Instanz.

Wir meinen später gelegentlich, vor allem bei den Beispielen, nur den Anfangszustand, wenn wir von einer Instanz sprechen. Es wird aus dem Kontext hervorgehen, ob der gesamte Zustandsraum oder nur den Anfangszustand gemeint ist, so dass keine Verwechslung entsteht.

Definition 2.5 (Heuristik)
Eine Heuristik $h : s \to \mathbb{N} \cup \{0, \infty\}$ schätzt für einen Zustand s des Zustandsraumes die Länge eines optimalen Plans mit s als Anfangszustand.
Sei $p_*(s)$ ein optimaler Plan mit s als Anfangszustand, dann nennen wir $h^*(s) = |p_*(s)|$ die perfekte Heuristik. Falls kein Plan mit s als Anfangszustand existiert, ist $h^*(s) = \infty$.
Sei $p'_*(s)$ ein optimaler Plan in der Relaxierung mit s als Anfangszustand, dann nennen wir $h^+(s) = |p'_*(s)|$ die h^+-Heuristik. Falls für die Relaxierung kein Plan mit s als Anfangszustand existiert, ist $h^+(s) = \infty$.
Wir nennen eine Heuristik h genau dann zulässig, wenn $h(s) \leq h^*(s)$ für alle Zustände s.
Wir nennen eine Heuristik h genau dann monoton, wenn $h(s) \leq h(o(s)) + 1$ für alle Zustände s und alle Operatoren $o \in \mathcal{O}$.

Die Merge-And-Shrink-Heuristik, die ich zum Vergleich mit der h^+-Heuristik verwendet habe, lässt sich nicht so einfach beschreiben wie letztere. Es handelt sich um eine sogenannte Abstraktionsheuristik. Der Zustandsraum wird zu einem neuen, kleineren Zustandsraum abstrahiert, indem nacheinander Zustände miteinander verschmolzen werden und der Zustandsraum verkleinert wird, solange bis er eine festgelegte Größe erreicht. In diesem neuen Zustandsraum wird ein optimaler Plan gesucht. Für eine genauere Beschreibung der Merge-And-Shrink-Heuristik lese man die Arbeit, in der sie vorgestellt wurde [HHH07].
Da jeder Zustand entweder nie zu einem Zielzustand führt oder es einen optimalen Pfad von ihm zu einem Zielzustand gibt, ist jede monotone Heuristik zulässig, wenn wir zusätzlich für jeden Zielzustand $s_* \in S_*$ fordern, dass $h(s_*) = 0$ gilt. Im ersten Fall gilt dies trivialerweise, da dann $h^*(s) = \infty$, im zweiten Fall folgt dies induktiv aus der Monotonie angewandt auf die Zustände des optimalen Pfades, beginnend beim Zielzustand.

Bemerkung 2.1
Die h^+-Heuristik ist zulässig und monoton und für Zielzustände $s_* \in S_*$ gilt $h^+(s_*) = 0$.

2.4 A^*

A^* ist ein Suchalgorithmus für das Planen, der die Planlänge des entstehenden Planes abschätzt über die Summe seiner bisherigen Länge und einer Heuristik. Da der Planer, mit dem ich gearbeitet habe, auf der A^*-Suche beruht, gebe ich kurz eine Beschreibung von A^* und seine Eigenschaften an. Dem Suchalgorithmus A^* ist eine Heuristik h zugeordnet. Als Expandieren eines Zustandes s bezeichnet man das Überprüfen, ob $s \in S_*$, das Erstellen der Menge der Nachfolgezustände $\text{succ}(s) = \{o(s) \mid o \in \mathcal{O} \text{ ist in } s \text{ definiert}\}$,

sowie das Berechnen von $h(s')$ für alle $s' \in \text{succ}(s)$. Der Algorithmus speichert stets eine Menge von evaluierten Zuständen, d.h. Zuständen s, für die der Heuristikwert $h(s)$ ausgerechnet wurde, und von expandierten Zuständen. Es wird solange ein evaluierter Zustand mit kleinstem f-Wert expandiert, bis ein Zielzustand gefunden wurde. Dabei ist $f(s) = g(s) + h(s)$ und $g(s)$ bezeichnet die Anzahl der Operatoren, die auf s_0 angewendet wurden, um zu s zu gelangen, d.h. $f(s)$ ist eine Schätzung der Länge eines optimalen Plans, der über s führt.
Das Anwenden von A^* als Baumsuche bedeutet, dass jeder Zustand nur auf einem Weg von s_0 erreicht werden kann, d.h. es gibt einen eindeutigen Plan mit s als Zielzustand. Das Anwenden von A^* als Graphensuche bedeutet, dass diese Einschränkung nicht gelten muss. Bei uns wird meist zweiteres der Fall sein. Da die h^+-Heuristik monoton ist und $h^+(s_*) = 0$ für alle Zielzustände $s_* \in S_*$ gilt , findet der Planer dennoch stets eine optimale Lösung nach dem folgenden Satz, dessen Beweis man im bekannten Buch von Stuart Russell und Peter Norvig [RN03] findet.

Satz 2.1

- *Die A^*-Suche angewandt als Baumsuche findet stets eine optimale Lösung, wenn die zugeordnete Heuristik h zulässig ist.*
- *Die A^* Suche angewandt als Graphensuche findet stets eine optimale Lösung, wenn die zugeordnete Heuristik h monoton ist und $h(s_*) = 0$ für alle Zielzustände $s_* \in S_*$ gilt.*

Bei A^* bezeichnet man die Zustände auch als Knoten. In den folgenden Kapiteln werden wir im experimentellen Abschnitt vom Finden von Lösungen mit einer Heuristik h sprechen. Gemeint ist dabei das Finden eines optimalen Plans mit dem A^*-Suchalgorithmus mit h als zugeordneter Heuristik.

2.5 Komplexitätsanalyse des Domänen-Problems

Für Entscheidungsprobleme gibt es den bekannten Begriff der polynomiellen Reduzierbarkeit, der ausdrückt, dass eine Sprache nicht „deutlich schwieriger“ zu entscheiden ist als eine andere.

Definition 2.6 (polynomielle Reduktion)
Es seien L_1 und L_2 Sprachen über Σ_1 und Σ_2. Dann heißt L_1 polynomiell auf L_2 reduzierbar, Notation $L_1 \leq_p L_2$, wenn es eine polynomielle Transformation von L_1 nach L_2 gibt, d.h. wenn es eine in polynomieller Zeit berechenbare Funktion $f : \Sigma_1^* \to \Sigma_2^*$ gibt, so dass für alle $\omega \in \Sigma_1^*$ gilt:

$$\omega \in L_1 \Leftrightarrow f(\omega) \in L_2.$$

Eine Sprache L wird mit dieser aus dem Buch von Ingo Wegener [Weg99] stammenden Definition NP-schwer genannt, wenn für alle Sprachen L' aus NP gilt, dass $L' \leq_p L$. Sie heißt NP-vollständig, wenn sie zusätzlich in NP liegt.
Die Probleme, die uns in dieser Arbeit beschäftigen, sind allerdings Optimierungsprobleme, wir interessieren uns für die Länge eines optimalen h^+-Plans und das Domänen-Problem, d.h. das Finden eines solchen. Auf den ersten Blick scheint die obige Definition dafür nicht hilfreich zu sein. Der in diesem Fall übliche Ansatz, den wir hier auch wählen, ist es, vom Optimierungsproblem zum zugehörigen Entscheidungsproblem überzugehen. Anstatt nach der Länge eines optimalen h^+-Planes zu suchen, fragen wir, ob es einen h^+-Plan mit Länge höchstens k gibt. Für dieses Entscheidungsproblem können wir untersuchen, ob es in NP liegt und ob es NP-schwer ist. Aus den Ergebnissen dieser Untersuchung können Rückschlüsse auf das Optimierungsproblem gezogen werden. Falls die Planlänge eines optimalen Planes polynomiell in der Größe der Instanz beschränkt ist, folgt aus der NP-Vollständigkeit des Entscheidungsproblems (Domänen-Längen-Problem) die NP-Äquivalenz des Optimierungsproblems (Domänen-Problem). Wir müssen die Mitgliedschaft in NP für das (relaxierte) Domänen-Längen-Problem im Falle der polynomiellen Beschränktheit der optimalen Planlänge nicht explizit zeigen, dies gilt automatisch. Beide Resultate wurden von Malte Helmert im Rahmen seiner Diplomarbeit [Hel01] gezeigt. Da bei den relaxierten Instanzen ein einmal erreichter Effekt erhalten bleibt, muss jede Aktion höchstens einmal ausgeführt werden, somit existiert stets ein Plan der Länge höchstens $|O|$ oder es gibt keinen Plan. Daher ist die Vorraussetzung der polynomiellen Beschränktheit gegeben, alle relaxierten Domänen-Längen-Probleme liegen in NP.
Falls es einen polynomiellen Algorithmus zur Bestimmung der optimalen Planlänge oder sogar eine geschlossene Formel dafür gibt, werden wir den Algorithmus bzw. die Formel angeben. Falls dies nicht möglich ist, zeigen wir, dass das Domänen-Längen-Problem NP-schwer ist, woraus mit den obigen Bemerkungen die NP-Äquivalenz des Domänen-Problems folgt, was wir zukünftig nicht mehr explizit erwähnen werden.

Kapitel 3

Miconic

Dieses Kapitel behandelt zwei unterschiedliche Domänen, namentlich die Miconic-Strips- und die Miconic-Simple-ADL-Domäne. Beide werden über die gleiche Problemstellung definiert. Nach der Definition der Probleme untersuchen wir, ob und wie die h^+-Heuristik berechnet werden kann. Für Miconic-Strips werden wir eine geschlossene Formel angeben, für Miconic--Simple-ADL dagegen ist das Finden eines optimalen h^+-Plans NP-äquivalent. Die experimentellen Ergebnisse für Miconic-Strips schließen dieses Kapitel ab.

3.1 Definition des Problems

Wir behandeln zwei Varianten der Miconic-Domäne. Bei beiden Varianten geht es darum, Passagiere mit einem Aufzug von ihrem Startstockwerk zu ihrem Zielstockwerk zu transportieren.
Bei der Miconic-Strips-Domäne kann zu einem Zeitpunkt genau ein Passagier den Aufzug betreten oder verlassen. Bei der Miconic-Simple-ADL-Domäne können alle Passagiere eines Stockwerks gleichzeitig den Aufzug betreten, und alle Passagiere im Aufzug können ihn gleichzeitig auf dem aktuellen Stockwerk verlassen.
Wir beginnen mit einer formalen Beschreibung der Miconic-Problemstellung.

Definition 3.1 (Miconic-Problemstellung)
Eine Miconic-Problemstellung ist gegeben durch ein 5-Tupel $\langle P, F, l_0, f_0, f_* \rangle$, *bestehend aus folgenden Komponenten:*

- P *ist eine endliche Menge von Passagieren.*
- F *ist eine endliche Menge von Stockwerken.*

- $l_0 \in F$ *ist das Stockwerk, in dem sich der Aufzug zu Beginn befindet.*
- $f_0 : P \to F$ *bildet jeden Passagier auf das Stockwerk ab, in dem er sich anfangs befindet, seinen Anfangsort.*
- $f_* : P \to F$ *bildet jeden Passagier auf das Stockwerk ab, in welches er gelangen will, seinen Zielort.*

Es folgt eine Definition der nicht-relaxierten und der relaxierten Miconic-Strips-Domäne, welche Miconic-Problemstellungen auf unterschiedliche Zustandsräume abbildet.

Definition 3.2 (Miconic-Strips-Domäne)
Die Miconic-Strips-Domäne bildet Miconic-Problemstellungen mit Passagieren P und Stockwerken F wie folgt auf Zustandsräume ab:

Zustände: *Dies sind 2-Tupel $\langle f, l\rangle$, wobei $f : P \to F \cup \{\omega\}$, mit $\omega \notin F$, die Passagierfunktion ist, die jedem Passagier seine Position zuweist und $l \in F$ die Position des Fahrstuhls ist. Wenn $f(p) = \omega$ für einen Passagier $p \in P$ gilt, sagen wir, er befindet sich im Aufzug, falls $f(p) = f \in F$, sagen wir, er befindet sich in Stockwerk f.*

Anfangszustand: *Der Anfangszustand ist das 2-Tupel $\langle f_0, l_0\rangle$.*

Zielzustände: *Genau die Zustände $\langle f_*, l\rangle$ mit $l \in F$ sind Zielzustände.*

Operatoren: *Ein Passagier $p \in P$ kann in den Aufzug einsteigen, wenn der Passagier und der Aufzug sich im selben Stockwerk befinden und der Passagier noch nicht an seinem Zielort ist. Im entstehenden Zustand befindet er sich im Aufzug.*
Ein Passagier $p \in P$ kann aus dem Aufzug in Stockwerk $f \in F$ aussteigen, wenn der Passagier sich im Aufzug und der Aufzug in Stockwerk f befindet und f der Zielort von Passagier p ist, d.h. $f_*(p) = f$. Im entstehenden Zustand befindet sich der Passagier in Stockwerk f.
Der Aufzug kann zu Stockwerk $f \in F$ fahren. Im entstehenden Zustand befindet sich der Aufzug in Stockwerk f.

Definition 3.3 (relaxierte Miconic-Strips-Domäne)
Die relaxierte Miconic-Strips-Domäne bildet einen Zustand $\left\langle \hat{f}, \hat{l} \right\rangle$ einer (nicht-relaxierten) Miconic-Strips-Instanz auf Zustandsräume ab. Die neuen Zustände sind 2-Tupel $\langle f, l\rangle$. In der relaxierten Domäne bildet die Passagierfunktion nicht auf die Menge der Stockwerke ab, sondern auf deren Teilmengen. Die Position des Aufzugs ist eine Teilmenge der Stockwerke, d.h. er kann sich in mehreren Stockwerken gleichzeitig befinden.

Operatoren: *Ein Passagier $p \in P$ kann in den Aufzug einsteigen, wenn der Passagier und der Aufzug ein gemeinsames Stockwerk haben, d.h. wenn $f(p) \cap l \neq \emptyset$, und der Passagier noch nicht an seinem Zielort ist. Im entstehenden Zustand befindet sich der Passagier im Aufzug* **und** *an seinen bisherigen Positionen.*
Ein Passagier $p \in P$ kann aus dem Aufzug in Stockwerk $f \in F$ aussteigen, wenn der Passagier sich im Aufzug und der Aufzug in Stockwerk f befindet und f der Zielort von Passagier p ist. Im entstehenden Zustand befindet sich der Passagier in Stockwerk f **und** *an seinen bisherigen Positionen.*
Der Aufzug kann zu Stockwerk $f \in F$ fahren. Im entstehenden Zustand befindet sich der Aufzug in Stockwerk f **und** *an seinen bisherigen Positionen.*

Es folgt eine formale Beschreibung der Miconic-Simple-ADL-Domäne. Die Zustände sind die gleichen wie bei der Miconic-Strips-Domäne. Allerdings unterscheiden sich die Operatoren, da mehrere Passagiere ein- und aussteigen können.

Definition 3.4 (Miconic-Simple-ADL-Domäne)
Die Miconic-Simple-ADL-Domäne bildet Miconic-Problemstellungen mit Passagieren P und Stockwerken F wie folgt auf Zustandsräume ab:

Zustände: *Dies sind 2-Tupel $\langle f, l \rangle$, wobei $f : P \to F \cup \{\omega\}$, mit $\omega \notin F$, die Passagierfunktion ist, die jedem Passagier seine Position zuweist und $l \in F$ die Position des Fahrstuhls ist. Wenn $f(p) = \omega$ für einen Passagier $p \in P$ gilt, sagen wir, er befindet sich im Aufzug, falls $f(p) = f \in F$, sagen wir, er befindet sich in Stockwerk f.*

Anfangszustand: *Der Anfangszustand ist das 2-Tupel $\langle f_0, l_0 \rangle$.*

Zielzustände: *Genau die Zustände $\langle f_*, l \rangle$ mit $l \in F$ sind Zielzustände.*

Operatoren: *Der Aufzug kann in Stockwerk l anhalten. Im entstehenden Zustand steigen alle Passagiere, die sich in Stockwerk l befinden und noch nicht an ihrem Zielort sind, in den Aufzug. Alle Passagiere, die sich im Aufzug befinden und deren Zielort l ist, steigen aus. D.h. die neue Passagierfunktion f' ist*

$$f'(p) = \begin{cases} \omega, & \text{falls } f(p) = l \text{ und } f_*(p) \neq l \\ l, & \text{falls } f(p) = \omega \text{ und } f_*(p) = l \\ f(p), & \text{sonst} \end{cases},$$

wobei f die alte Passagierfunktion ist.
Der Aufzug kann zu Stockwerk $f \in F$ fahren. Im entstehenden Zustand befindet sich der Aufzug in Stockwerk f.

Definition 3.5 (relaxierte Miconic-Simple-ADL-Domäne)
Die relaxierte Miconic-Simple-ADL-Domäne bildet einen Zustand $\left\langle \hat{f}, \hat{l} \right\rangle$ *einer (nicht-relaxierten) Miconic-Simple-ADL-Instanz auf Zustandsräume ab. Die neuen Zustände sind 2-Tupel* $\langle f, l \rangle$*. In der relaxierten Domäne bildet die Passagierfunktion nicht auf die Menge der Stockwerke ab, sondern auf deren Teilmengen. Die Position des Aufzugs ist eine Teilmenge der Stockwerke, d.h. er kann sich in mehreren Stockwerken gleichzeitig befinden.*

Operatoren: *Der Aufzug kann in Stockwerk* l *anhalten. Im entstehenden Zustand steigen alle Passagiere, die sich in Stockwerk* l *befinden und nicht an ihrem Zielort sind, in den Aufzug. Alle Passagiere, die sich im Aufzug befinden und deren Zielort* l *ist, steigen aus. Die Passagiere befinden sich danach an ihrer neuen* **und** *an den alten Positionen. Die neue Passagierfunktion* f' *ist*

$$f'(p) = \begin{cases} f(p) \cup \{\omega\}, & \text{falls } l \cap f(p) \neq \emptyset \text{ und } f_*(p) \notin f(p) \\ f(p) \cup \{a\}, & \text{falls } \omega \in f(p) \text{ und } f_*(p) = a \text{ mit } a \in l \\ f(p), & \text{sonst} \end{cases},$$

wobei f *die alte Passagierfunktion ist.*
Der Aufzug kann zu Stockwerk $f \in F$ *fahren. Im entstehenden Zustand befindet sich der Aufzug in Stockwerk* f **und** *an seinen alten Positionen.*

3.2 Berechnung der h^+-Heuristik

Obwohl die beiden Domänen ähnlich sind und sich nur in der Art unterscheiden, wie Passagiere eingeladen werden, kann das relaxierte Miconic-Strips-Problem in linearer Zeit gelöst werden, wogegen das relaxierte Miconic-Simple-ADL-Längen-Problem NP-vollständig ist.

Satz 3.1
Das Finden eines optimalen h^+*-Plans für eine Miconic-Strips-Instanz ist in Zeit* $O(|P| + |F|)$ *möglich.*
Seien

- $\mathcal{P}_1 = \{p \in P \mid f(p) \neq f_*(p), f(p) \neq \omega\}$ *die Passagiere, die sich weder an ihrem Zielort noch im Aufzug befinden.*
- $\mathcal{P}_2 = \{p \in P \mid f(p) = \omega\}$ *die Passagiere im Fahrstuhl.*
- $\mathcal{F} = \{f \in F \setminus \{l_0\} \mid \exists p \in P : f(p) = f\}$ *die Stockwerke, auf denen sich Passagiere befinden, aber nicht der Aufzug.*

Dann gilt für jeden optimalen Plan h^+:

$$|h^+| = \mathcal{L}(\textit{Miconic-Strips}) := 2|\mathcal{P}_1| + |\mathcal{P}_2| + |\mathcal{F}|$$

Bemerkung 3.1
Die Menge $\mathcal{P}_2$ ist nur bei Zuständen nicht leer, in denen bereits Aktionen ausgeführt wurden. In der Definition der Miconics-Strips-Domäne ist der Aufzug anfangs leer.

Beweis:
„$|h^+| \leq \mathcal{L}$(Miconic-Strips)“:
Betrachte folgenden Plan für das Problem.
Fahre alle Anfangs- und Zielorte an ($|\mathcal{F}|$ Aktionen). Lade alle Passagiere, die sich auf einem anderen Stockwerk als ihrem Zielort befinden, in den Aufzug ein ($|\mathcal{P}_1|$ Aktionen). Lade alle Passagiere an ihrem Zielort ab ($|\mathcal{P}_2| + |\mathcal{P}_1|$ Aktionen).
„$|h^+| \geq \mathcal{L}$(Miconic-Strips)“
In jedem gültigen Plan müssen alle Anfangs- und Zielorte angefahren, die anfangs im Aufzug befindlichen Passagiere ausgeladen und die übrigen Passagiere ein- und ausgeladen werden. □

Für jede Miconic-Strips Instanz können wir somit einen optimalen relaxierten Plan in polynomieller Zeit finden, im nicht-relaxierten Fall ist das Miconic-Strips-Längen-Problem NP-vollständig [Hel08]. Bei der Miconic-Simple-ADL-Domäne ist die Frage nach der Existenz eines beschränkten Planes sowohl im nicht-relaxierten [Hel08] als auch im relaxierten Fall NP-vollständig, da die Schwierigkeit, eine Besuchsreihenfolge der Stockwerke zu finden, die möglichst wenige Halteaktionen erfordert, bei beiden Fällen vorhanden ist.

Satz 3.2
Das relaxierte Miconic-Simple-ADL-Längen-Problem ist NP-schwer.

Beweis:
Siehe dazu den Artikel von Malte Helmert [Hel03].
Der Beweis von Satz 18 daraus lässt sich wie in Bemerkung 20 und Satz 36 vorgeschlagen auf die Miconic-Simple-ADL-Domäne übertragen.
Beim Beweis ändert sich lediglich, dass im relaxierten Fall jeder Ort (Stock) nur einmal angefahren werden muss. Mindestens an den Orten eines Minimum-Feedback-Vertex-Sets muss immer noch zweimal gestoppt werden. Ein Feedback-Vertex-Set der Größe K existiert genau dann, wenn die Miconic-Simple-ADL-Instanz eine Lösung der Größe $2|V| + K$ besitzt. □

3.3 Experimentelle Ergebnisse

Die h^+-Heuristik wurde für die Miconic-Strips-Domäne implementiert. Bei der Miconic-Simple-ADL-Domäne habe ich darauf verzichtet. Der zugrunde liegende Planer hat Probleme, wenn die Spezifikationen über die von STRIPS hinausgehen, dies ist bei Miconic-Simple-ADL der Fall.
Es gab 30 Miconic-Strips-Instanzen in jeweils fünf Versionen beim Planungswettbewerb. Die h^+-Heuristik wurde auf allen 150 Instanzen ausgeführt und mit der Merge-And-Shrink-Heuristik verglichen. Für Letztere zeigte sich, dass sie mit wachsendem Parameter N bessere Ergebnisse lieferte, was die Anzahl der expandierten Knoten und die Menge der gelösten Instanzen betrifft. Wir listen hier die Ergebnisse mit zwei verschiedenen Parametern, $N = 5000$ und $N = 1000000$, auf. Die Experimente wurden auf einem Computer mit acht Intel Xeon Prozessoren mit einer Taktfrequenz von 2.66 GHZ durchgeführt. Falls nach 30 Minuten kein optimaler Plan gefunden wurde oder der Speicherbedarf 2 Gigabyte überschritten hat, wurde die Suche abgebrochen. Wenn eine der Heuristiken eine der Instanzen nicht lösen konnte, wird dies durch Striche dargestellt. Aus Platzgründen wird jeweils nur die erste Variante der 30 Instanzen gelistet. Ein Vergleich der Anfangswertschätzung, der Lösungsdauer und der Anzahl der expandierten Knoten zwischen der h^+- und Merge-And-Shrink-Heuristik findet sich in den Abbildungen 3.1, 3.2 und 3.3. Eine Übersicht der h^+-Heuristik ist in Abbildung 3.4 zu sehen. Es werden wieder nur die ersten Versionen der Instanzen gelistet.
Mit der h^+-Heuristik konnten 142 der 150 Instanzen gelöst werden. Nicht gelöst wurden die Instanzen 19-2, 23-1, 23-3, 25-2, 26-1, 27-0, 28-1 und 30-3. Mit der Merge-And-Shrink-Heuristik konnten mit dem Parameter $N = 5000$ die Instanzen bis einschließlich 11-4 gelöst werden, mit Parameter $N = 1000000$ zusätzlich die Instanzen 12-0, 12-1, 12-2 und 12-3. Wenn eine Instanz nicht gelöst werden konnte, lag dies bei der h^+- und der Merge-And-Shrink-Heuristik mit Parameter $N = 5000$ stets am begrenzten Speicher. Bei der Merge-And-Shrink-Heuristik mit Parameter $N = 1000000$ waren bei Instanz 15-4 und den Instanzen ab 16-1 die 30 Minuten um, ohne dass eine Lösung gefunden wurde, bei den übrigen Instanzen ging der Speicher aus.
Die größte Anomalie bei der h^+-Heuristik ist in Abbildung 3.4 gelistet, dies ist Instanz 20-0. Hier wurde 86 Sekunden gerechnet und 496091 Knoten wurden expandiert, bevor eine Lösung gefunden wurde. Unter den nicht gelisteten Instanzen gibt es drei Instanzen mit hervorzuhebenden Ergebnissen, dies sind die Instanzen 11-3, 13-1 und 17-4. Für 17-4 benötigte die h^+-Heuristik 12 Sekunden und 86730 Knoten mussten expandiert werden, um eine Lösung zu finden, die anderen beiden Instanzen benötigten unter 2 Sekunden und weniger als 20000 Knoten wurden expandiert. Sämtliche

anderen gelösten Instanzen waren in deutlich unter 1 Sekunde gelöst und weniger als 1000 Knoten mussten expandiert werden, um eine Lösung zu finden.

Problemname	optimale Planlänge	h^+-Heuristik	Merge-And-Shrink-Heuristik, N=5000	Merge-And-Shrink-Heuristik, N=1000000
Problem 1-0	4	3	4	4
Problem 2-0	7	7	7	7
Problem 3-0	10	10	10	10
Problem 4-0	14	14	14	14
Problem 5-0	17	17	17	17
Problem 6-0	19	18	19	19
Problem 7-0	23	23	22	23
Problem 8-0	27	27	23	27
Problem 9-0	31	30	26	**31**
Problem 10-0	33	33	25	33
Problem 11-0	37	36	28	**37**
Problem 12-0	40	**40**	-	38
Problem 13-0	44	**43**	-	-
Problem 14-0	45	**45**	-	-
Problem 15-0	46	**46**	-	-
Problem 16-0	53	**53**	-	-
Problem 17-0	56	**55**	-	-
Problem 18-0	59	**59**	-	-
Problem 19-0	62	**62**	-	-
Problem 20-0	64	**63**	-	-
Problem 21-0	70	**69**	-	-
Problem 22-0	73	**72**	-	-
Problem 23-0	76	**75**	-	-
Problem 24-0	79	**79**	-	-
Problem 25-0	81	**81**	-	-
Problem 26-0	84	**83**	-	-
Problem 27-0	-	-	-	-
Problem 28-0	95	**94**	-	-
Problem 29-0	94	**94**	-	-
Problem 30-0	95	**95**	-	-

Abbildung 3.1: Vergleich der Bewertung des Anfangszustands der h^+- und der Merge-And-Shrink-Heuristik

Problemname	optimale Planlänge	h^+-Heuristik	Merge-And-Shrink-Heuristik, N=5000	Merge-And-Shrink-Heuristik, N=1000000
Problem 1-0	4	0.00	0.00	0.00
Problem 2-0	7	0.00	0.00	0.00
Problem 3-0	10	0.00	0.00	0.00
Problem 4-0	14	**0.00**	0.01	0.01
Problem 5-0	17	**0.00**	0.06	0.06
Problem 6-0	19	**0.00**	0.12	0.53
Problem 7-0	23	**0.00**	0.32	4.01
Problem 8-0	27	**0.00**	1.68	27.66
Problem 9-0	31	**0.00**	10.28	141.17
Problem 10-0	33	**0.00**	63.39	300.23
Problem 11-0	37	**0.01**	314.89	869.70
Problem 12-0	40	**0.01**	-	911.17
Problem 13-0	44	**0.01**	-	-
Problem 14-0	45	**0.01**	-	-
Problem 15-0	46	**0.01**	-	-
Problem 16-0	53	**0.01**	-	-
Problem 17-0	56	**0.01**	-	-
Problem 18-0	59	**0.01**	-	-
Problem 19-0	62	**0.01**	-	-
Problem 20-0	64	**86.47**	-	-
Problem 21-0	70	**0.02**	-	-
Problem 22-0	73	**0.03**	-	-
Problem 23-0	76	**0.03**	-	-
Problem 24-0	79	**0.03**	-	-
Problem 25-0	81	**0.04**	-	-
Problem 26-0	84	**0.04**	-	-
Problem 27-0	-	-	-	-
Problem 28-0	95	**0.05**	-	-
Problem 29-0	94	**0.06**	-	-
Problem 30-0	95	**0.06**	-	-

Abbildung 3.2: Vergleich der Lösungsdauer der h^+- und der Merge-And-Shrink-Heuristik

Problemname	optimale Planlänge	h^+-Heuristik	Merge-And-Shrink-Heuristik, N=5000	Merge-And-Shrink-Heuristik, N=1000000
Problem 1-0	4	5	5	5
Problem 2-0	7	9	8	8
Problem 3-0	10	14	11	11
Problem 4-0	14	16	15	15
Problem 5-0	17	22	18	18
Problem 6-0	19	32	20	20
Problem 7-0	23	27	8241	**24**
Problem 8-0	27	31	92677	**28**
Problem 9-0	31	39	622602	**32**
Problem 10-0	33	37	3525685	**34**
Problem 11-0	37	**57**	15707297	780338
Problem 12-0	40	**46**	-	3993228
Problem 13-0	44	**49**	-	-
Problem 14-0	45	**60**	-	-
Problem 15-0	46	**63**	-	-
Problem 16-0	53	**60**	-	-
Problem 17-0	56	**79**	-	-
Problem 18-0	59	**77**	-	-
Problem 19-0	62	**96**	-	-
Problem 20-0	64	**496091**	-	-
Problem 21-0	70	**101**	-	-
Problem 22-0	73	**100**	-	-
Problem 23-0	76	**120**	-	-
Problem 24-0	79	**95**	-	-
Problem 25-0	81	**139**	-	-
Problem 26-0	84	**107**	-	-
Problem 27-0	-	-	-	-
Problem 28-0	95	**112**	-	-
Problem 29-0	94	**126**	-	-
Problem 30-0	95	**138**	-	-

Abbildung 3.3: Vergleich der expandierten Knoten der h^+- und der Merge-And-Shrink-Heuristik

Problemname	optimale Planlänge	Anfangswert	expandierte Knoten	Zeit
Problem 1-0	4	3	5	0.00
Problem 2-0	7	7	9	0.00
Problem 3-0	10	10	14	0.00
Problem 4-0	14	14	16	0.00
Problem 5-0	17	17	22	0.00
Problem 6-0	19	18	32	0.00
Problem 7-0	23	23	27	0.00
Problem 8-0	27	27	31	0.00
Problem 9-0	31	30	39	0.00
Problem 10-0	33	33	37	0.00
Problem 11-0	37	36	57	0.01
Problem 12-0	40	40	46	0.01
Problem 13-0	44	43	49	0.01
Problem 14-0	45	45	60	0.01
Problem 15-0	46	46	63	0.01
Problem 16-0	53	53	60	0.01
Problem 17-0	56	55	79	0.01
Problem 18-0	59	59	77	0.01
Problem 19-0	62	62	96	0.01
Problem 20-0	64	63	496091	86.47
Problem 21-0	70	69	101	0.02
Problem 22-0	73	72	100	0.03
Problem 23-0	76	75	120	0.03
Problem 24-0	79	79	95	0.03
Problem 25-0	81	81	139	0.04
Problem 26-0	84	83	107	0.04
Problem 27-0	-	-	-	-
Problem 28-0	95	94	112	0.05
Problem 29-0	94	94	126	0.06
Problem 30-0	95	95	138	0.06

Abbildung 3.4: Übersicht über die gelösten Probleme der h^+-Heuristik

Kapitel 4

Schedule

In diesem Kapitel beschäftigen wir uns mit der Schedule-Domäne. Die Definitionen des Problems sind hauptsächlich Übersetzungen der Doktorarbeit von Malte Helmert [Hel08].
Für das relaxierte Problem werden wir einen polynomiellen Algorithmus zum Finden eines optimalen Planes angeben. Tatsächlich existiert auch für das nicht-relaxierte Problem ein polynomieller Algorithmus. Wie bei der Miconic-Simple-ADL-Domäne im letzten Kapitel wurde wegen des zugrundeliegenden Planers auf eine Implementierung der h^+-Heuristik verzichtet.

4.1 Definition des Problems

Es sollen die Oberflächenstruktur, die Temperatur, die Form, die Farbe und die Anzahl der Löcher von Gegenständen verändert werden. Dafür stehen verschieden Maschinen zur Verfügung, die teilweise Anforderungen an die Temperatur des Gegenstandes stellen, bevor die Gegenstände bearbeitet werden können.

Definition 4.1 (Schedule-Problemstellung)
Die Menge der Temperaturen O_T, Oberflächenstrukturen (oder kurz Oberflächen) O_{SC}, Formen O_S, Farben O_C und Löcher O_H von Schedule und die Menge der Schedule-Gegenstandszustände O sind wie folgt definiert:

$$
\begin{aligned}
O_T &= \{\textit{kalt, heiß}\} \\
O_{SC} &= \{\textit{rau, glatt, poliert, keine Struktur}\} \\
O_S &= \{\textit{zylindrisch, ringförmig, rechteckig}\} \\
O_C &= \{\textit{blau, gelb, schwarz, keine Farbe}\} \\
O_H &= \{\textit{vorne1, vorne2, vorne3, hinten1, hinten2, hinten3}\} \\
O &= O_T \times O_{SC} \times O_S \times O_C \times 2^{O_H}
\end{aligned}
$$

Eine Schedule-Problemstellung ist eine endliche Sequenz von Paaren in $O \times 2^O$. *Für Schedule-Problemstellungen* $(\langle o_0^i, O_*^i \rangle)_{i=1}^n$ *sagen wir, dass* o_0^i *der Anfangszustand des* $i-$*ten Gegenstandes und* O_*^i *die Zielbeschreibung für den* $i-$*ten Gegenstand ist.*

Definition 4.2 (Schedule-Maschinen)
Die Menge der Schedule-Maschinen ist definiert als

$$M = \{\textit{Bohrmaschine, Schleifmaschine, Tauchlackierer, Drechselbank, Poliergerät, Ausstanzer, Abzugsrolle, Spritzlackierer}\}.$$

Jede Maschine $m \in M$ *hat eine dazugehörige Menge von Umformungen, die als partielle Funktionen auf den Gegenstandszuständen definiert sind. Wir beschreiben die Umformungen über Vorbedingungen und Effekte auf Gegenstandszuständen:*

- *Bohrmaschine, Poliergerät, Ausstanzer und Spritzlackierer können jeden Gegenstand umformen, dessen Temperatur kalt ist. Die anderen Maschinen können jeden Gegenstand umformen.*
- *Bohrmaschine und Ausstanzer verfügen über eine Umformung für jedes Loch* $h \in O_H$. *Diese Umformung fügt das Loch* h *zur Menge der Löcher des umgeformten Gegenstandes hinzu. Zusätzlich wird die Oberfläche bei Verwendung des Ausstanzers rau.*
- *Tauchlackierer und Spritzlackierer verfügen über eine Umformung für jede Farbe* $c \in O_C \setminus \{\textit{keine Farbe}\}$. *Diese Umformung verändert die Farbe des Gegenstandes zu c. Zusätzlich wird die Oberfläche bei Verwendung des Spritzlackierers keine Struktur.*
- *Der Polierer verfügt über eine einzige Umformung, welche die Oberfläche des umgeformten Gegenstandes zu poliert ändert.*
- *Die Abzugsrolle verfügt über eine einzige Umformung, welche die Oberfläche des umgeformten Gegenstandes zu keine Struktur und die Farbe zu keine Farbe ändert, alle Löcher entfernt (d.h. die Menge der Löcher wird die leere Menge), die Temperatur auf heiß und die Form auf zylindrisch ändert.*
- *Die Drechselbank verfügt über eine einzige Umformung, welche die Farbe des umgeformten Gegenstandes auf keine Farbe, die Oberfläche auf rau und die Form auf zylindrisch ändert.*
- *Die Schleifmaschine verfügt über eine einzige Umformung, welche die Farbe des umgeformten Gegenstandes auf keine Farbe und die Oberfläche auf glatt ändert.*

Definition 4.3 (Schedule-Domäne)
Die Schedule-Domäne bildet Schedule-Problemstellungen $(\langle o_0^i, O_*^i \rangle)_{i=1}^n$ *wie folgt auf Zustandsräume ab:*

Zustände: *Dies sind 3-Tupel* $\langle (o^i)_{i=1}^n, M_B, O_B \rangle$, *wobei* $(o^i)_{i=1}^n \in O^n$ *ein* $n-$*Tupel von Gegenstandszuständen ist, dessen* $i-$*te Komponente wir den Zustand des* $i-$*ten Gegenstandes nennen,* $M_B \subseteq M$ *nennen wir die Menge der beschäftigten Maschinen und* $O_B \subseteq \{1, \ldots, n\}$ *nennen wir die Menge der in Bearbeitung befindlichen Gegenstände. Wir sagen* $m \in M$ *ist genau dann beschäftigt, wenn* $m \in M_B$ *und der* $i-$*te Gegenstand ist genau dann in Bearbeitung, wenn* $i \in O_B$.

Anfangszustand: $\langle (o_0^i)_{i=1}^n, \emptyset, \emptyset \rangle$.

Zielzustände: *Genau die Zustände in denen für alle* $i \in \{1, \ldots, n\}$ *gilt, dass der* $i-$*te Gegenstand in* O_*^i *ist, sind Zielzustände.*

Operatoren: *Für jede Umformung* t *einer Maschine* $m \in M$ *und jedes* $i \in \{1, \ldots, n\}$ *kann Maschine* m *den* $i-$*ten Gegenstand genau dann bearbeiten, wenn* t *definiert ist für den Zustand des* $i-$*ten Gegenstands und weder* m *beschäftigt noch der* $i-$*te Gegenstand in Bearbeitung ist. Im entstehenden Zustand ist der Zustand des* $i-$*ten Gegenstandes* $t(o_i)$, *Maschine* m *ist beschäftigt und der* $i-$*te Gegenstand in Bearbeitung.*
Der Warteoperator ist genau dann anwendbar, wenn mindestens eine Maschine beschäftigt ist. Im entstehenden Zustand ist sowohl die Menge der beschäftigten Maschinen als auch die Menge der bearbeiteten Gegenstände leer.

Für die Relaxierung der Domäne können die Gegenstände mehrere Temperaturen, Oberflächenstrukturen, Formen, Farben und (wie im nicht-relaxierten Fall) Löcher gleichzeitig haben, d.h. der Zustand eines Gegenstandes ist im relaxierten Fall aus $O' = 2^{O_T} \times 2^{O_{SC}} \times 2^{O_S} \times 2^{O_C} \times 2^{O_H}$. Die relaxierten Maschinen M' haben die gleichen Vorbedingungen wie die Maschinen M, ihre Effekte ergänzen jedoch die alten Eigenschaften des Gegenstandes. So wird beispielsweise bei Verwendung des Spritzlackierers zur Umformung für die Farbe c die Oberfläche des Gegenstandes um keine Struktur ergänzt und der Gegenstand erhält zusätzlich zu seinen bisherigen Farben die Farbe c.

Definition 4.4 (relaxierte Schedule-Domäne)
Die relaxierte Schedule-Domäne bildet einen Zustand $(\langle \hat{o_0^i}, \hat{O_*^i} \rangle)_{i=1}^n$ *einer (nicht-relaxierten) Schedule-Instanz auf Zustandsräume ab. Die Zustände im entstehenden Zustandsraum sind 3-Tupel* $\langle (o^i)_{i=1}^n, M_B, O_B \rangle$, *dabei ist* $(o^i)_{i=1}^n \in (O')^n$ *ein* n*-Tupel, bei dem die* n *Gegenstände im Gegensatz*

zur nicht-relaxierten Instanz mehrere Temperaturen, Oberflächenstrukturen, Formen, Farben und Löcher gleichzeitig haben können. Bei $M_B \in M'$ handelt es sich um die beschäftigten relaxierten Maschinen, $O_B \subseteq \{1, \ldots, n\}$ sind wie im nicht-relaxierten Fall die in Bearbeitung befindlichen Gegenstände.

Operatoren: *Für jede Umformung t einer Maschine $m' \in M'$ und jedes $i \in \{1, \ldots, n\}$ kann Maschine m' den $i-$ten Gegenstand genau dann bearbeiten, wenn t definiert ist für den Zustand des $i-$ten Gegenstands und weder m' beschäftigt noch der $i-$te Gegenstand in Bearbeitung ist. Im entstehenden Zustand ist der Zustand des $i-$ten Gegenstandes $t(o_i)$, wobei die Umformung t der relaxierten Maschine die alten Eigenschaften des Gegenstands erhält. Maschine m' ist* ***nicht*** *beschäftigt und der $i-$te Gegenstand ist* ***nicht*** *in Bearbeitung.*
Der Warteoperator ist genau dann anwendbar, wenn mindestens eine Maschine beschäftigt ist. Im entstehenden Zustand ist sowohl die Menge der beschäftigten Maschinen als auch die Menge der bearbeiteten Gegenstände leer.

Bemerkung 4.1
In den Planungswettbewerben wurde die Zieleigenschaft der Löcher O_H nie verwendet, auch die Oberflächenstruktur rau taucht nie als Zielbeschreibung auf. Sie können, da wir uns auf domänenabhängige Lösungsalgorithmen beschränken, als nicht Teil der Problemdefinition angesehen werden. Daher kann auf die Maschinen Bohrmaschine und Ausstanzer ebenfalls verzichtet werden.

4.2 Berechnung der h^+-Heuristik

In der relaxierten Version bleiben alle einmal erfüllten Eigenschaften der Gegenstände erhalten. Da weiterhin die Erfüllung der Zieleigenschaften der einzelnen Gegenstände unabhängig von denen der anderen Gegenstände sind, genügt es für einen optimalen Plan alle Gegenstände durchzugehen und für jede Zieleigenschaft zu prüfen, ob eine passende Maschine existiert, die nicht beschäftigt ist. Falls alle Maschinen, die die nötige Umwandlung durchführen können, beschäftigt sind oder der Gegenstand bereits bearbeitet wird, führe eine Warteoperation aus. In Algorithmus 1 ist dieses Vorgehen nochmal detailliert beschrieben unter der Voraussetzung, dass es keine Zielbeschreibungen mit Löchern oder rauen Oberflächen gibt. Außerdem wurde auf eine geschickte Reihenfolge der Eigenschaften geachtet, da manche der Maschinen mehrere Eigenschaften erfüllen, was zu einem kürzeren Plan führt.

Satz 4.1
Das Finden eines optimalen h^+-Plans für eine Schedule-Instanz ist in Zeit $O(n)$ möglich, Algorithmus 1 liefert einen optimalen Plan.

Beweis:
Algorithmus liefert gültigen Plan:
Für jeden Gegenstand wird jede Eigenschaft seiner Zielbeschreibung, die nicht erfüllt ist, abgeändert. Nach den Umwandlungen erfüllt jeder Gegenstand seine Zielbeschreibung.
Kein Plan ist kürzer als der vom Algorihmus gelieferte:
Für jede Eigenschaft eines Gegenstandes, die nicht mit seiner Zielbeschreibung übereinstimmt, ist eine Aktion nötig. Falls ein Gegenstand anfangs bearbeitet wird oder eine benötigte Maschine beschäftigt ist, muss eine Warteoperation eingefügt werden. Somit benötigt jeder gültige Plan mindestens soviele Aktionen wie der vom Algorithmus gelieferte. □

Für jede relaxierte Instanz kann also ein optimaler Plan in linearer Zeit – linear in der Anzahl der Gegenstände – gefunden werden. Der Grund dafür ist, dass die Menge der Maschinen fix ist und eine Aktion einer Maschine keine bisher erreichten Zielbeschreibungen zerstört.
Im nicht-relaxierten Fall kann ein optimaler Plan ebenfalls in polynomieller Zeit gefunden werden, wie Malte Helmert gezeigt hat [Hel08]. Der von ihm angegebene Algorithmus besitzt jedoch einen sehr hohen Exponenten (> 600000). Die Maschinen zerstören bereits erfüllte Zielbeschreibungen, weswegen darauf geachtet werden muss, in welcher Reihenfolge sie benutzt werden.
Im nicht-relaxierten Fall kommt den Warteoperationen eine größere Bedeutung zu, jede Maschine kann nur einmal ohne eine solche benutzt werden. Hier kann die Planlänge auch anhand der Anzahl der Warteoperationen bewertet werden, ein nach diesem Kriterium optimaler Plan kann ebenfalls in polynomieller Zeit gefunden werden [Hel08].

Algorithm 1 Greedyschedule

Input: Gegenstände parts, Anfangszustand und Zielbeschreibung für jedes x:parts, Liste der beschäftigten Maschinen busyMaschines⊆M, Liste der in Bearbeitung befindlichen Gegenstände busyParts⊆parts

Output: Optimaler h^+-Plan, der jeden Gegenstand in seinen Zielzustand umwandelt

```
1: def warteoperation() do
2:   busyMachines = busyParts = ∅
3: end def
```

```
1: def process(obj, machines) do
2:   for all machine:machines do
3:     if not machine:busyMachines then
4:       use machine on obj
5:       return
6:     end if
7:     warteoperation()
8:     use any machine:machines on obj
9:   end for
10: end def
```

```
1: def solve() do
2:   for all ?x:part do
3:     if x:busyParts then
4:       warteoperation()
5:     end if
6:     if x.ziel_form=zylindrisch and x.form≠zylindrisch then
7:       process(x, {Drechselbank, Abzugsrolle})
8:     end if
9:     if x.ziel_oberfläche=glatt and x.oberfläche≠glatt then
10:      process(x, Schleifmaschine)
11:    else if x.ziel_oberfläche=poliert and x.oberfläche≠poliert then
12:      process(x, Poliergerät)
13:    end if
14:    if x.ziel_farbe=c && x.farbe≠c then
15:      process(x, {Tauchlackierer, Spritzlackierer})
16:    end if
17:  end for
18: end def
```

Kapitel 5

Gripper

Dieses Kapitel beschäftigt sich mit der Gripper-Domäne und der theoretischen und praktischen Berechnung der h^+-Heuristik auf dieser.
Im Anschluss an die Definition des Problems sind wir in der Lage, eine geschlossene Formel zur Berechnung der h^+-Heuristik anzugeben. Danach wird die Implementierung dieser Formel im Abschnitt über die Experimente mit der Merge-And-Shrink-Heuristik, der Goal-Count-Heuristik und der blinden Suche verglichen.

5.1 Definition des Problems

Es geht darum, Bälle zwischen zwei Räumen zu transportieren. Der Roboter, der diese Aufgabe erfüllen soll, verfügt über zwei Greifarme, in denen er jeweils einen Ball halten kann. Er kann zwischen den Räumen beliebig hin- und herfahren, um bei vollen Greifarmen einen weiteren Ball aufzunehmen, muss er einen gehaltenen Ball fallenlassen.

Definition 5.1 (Gripper-Problemstellung)
Eine Gripper-Problemstellung ist gegeben durch ein 5-Tupel $\langle B, R, g_0, g_*, l_0 \rangle$, *bestehend aus folgenden Komponenten:*

- B *ist eine endliche Menge von Bällen.*
- R *ist eine endliche Menge von Räumen.*
- $g_0 : B \to R$ *bildet jeden Ball auf seinen Anfangsraum ab.*
- $g_* : B \to R$ *bildet jeden Ball auf seinen Zielraum ab.*
- $l_0 \in R$ *gibt den Anfangsort des Roboters an.*

Bemerkung 5.1
In den Benchmarks der Planungswettbewerbe werden stärkere Restriktionen verwendet, die das Problem vereinfachen. Es gibt nur zwei Räume, alle Bälle und der Roboter sind anfangs in Raum 1 und alle Bälle müssen zu Raum 2 transportiert werden.
Da wir nur an einem domänenabhängigen Löser interessiert sind, gehen wir ab jetzt – auch bei der Definition der Domäne – von diesen Einschränkungen als Teil der Problembeschreibung aus.

Die Menge der Räume und die Menge der Greifarme müssen disjunkt sein. Aus Gründen der kürzeren Notation verwenden wir bei der Definition der Domäne trotzdem beides Mal die Menge $\{1, 2\}$. Da aus dem Kontext stets hervorgeht, um welche Menge es sich handelt, sollte keine Verwechslung entstehen.

Definition 5.2 (Gripper-Domäne)
Die Gripper-Domäne bildet Gripper-Problemstellungen mit Bällen B wie folgt auf Zustandsräume ab:

Zustände: *Dies sind 6-Tupel $\langle B^1, B^2, l, H_1, H_2, F\rangle$, wobei $B^1 \subseteq B$ die Menge der Bälle im ersten Raum, $B^2 \subseteq B$ die Menge der Bälle im zweiten Raum, $l \in \{1, 2\}$ der Ort des Roboters, $H_1, H_2 \in B \cup \{\omega\}$, mit $\omega \notin B$, der vom Roboter mit dem ersten bzw. zweiten Greifarm gehaltene Ball und $F \subseteq \{1, 2\}$ die Menge der freien Greifarme des Roboters ist. Es gilt $B = (B^1 \cup B^2 \cup H_1 \cup H_2) \setminus \{\omega\}$. Wir sagen der Roboter befindet sich in Raum l. Wenn $b \in H_i$, mit $i = 1$ oder $i = 2$, für einen Ball $b \in B$ gilt, sagen wir, der Roboter hält Ball b in Greifarm i, wenn $H_i = \omega$ mit $i = 1$ oder $i = 2$, sagen wir, Greifarm i ist leer. Wenn $g \in F$ für einen Greifarm g gilt, sagen wir, Greifarm g ist frei.*

Anfangszustand: *Der Anfangszustand ist das 5-Tupel $\langle B, \emptyset, 1, \omega, \omega, F_0\rangle$, wobei B die Menge der Bälle der Problemstellung sind und $F_0 = \{1, 2\}$ ist.*

Zielzustände: *Genau die Zustände $\langle \emptyset, B, l, H_1, H_2, F\rangle$ sind Zielzustände.*

Operatoren: *Der Roboter kann den Raum wechseln. Im entstehenden Zustand befindet er sich im anderen Raum $R \setminus \{l\}$.*
Der Roboter kann genau dann einen Ball $b \in B$ in Raum r mit Greifarm g aufnehmen, wenn $l = r$ und Greifarm g frei ist. Im entstehenden Zustand wird Ball b aus Raum r entfernt, der Roboter hält Ball b in Greifarm g und g wird aus der Menge der freien Greifarme entfernt.
Der Roboter kann genau dann einen Ball b in Raum r aus Greifarm g

fallenlassen, wenn $H_g = b$ und $l = r$. Im entstehenden Zustand befindet sich Ball b in Raum r, Greifarm g ist leer und wird zur Menge der freien Greifarme hinzugefügt.

Definition 5.3 (relaxierte Gripper-Domäne)
Die relaxierte Gripper-Domäne bildet einen Zustand $\left\langle \hat{B}^1, \hat{B}^2, \hat{l}, \hat{H}_1, \hat{H}_2, \hat{F} \right\rangle$ einer (nicht-relaxierten) Gripper-Instanz auf Zustandsräume ab. Die Zustände sind mit denen in der nicht-relaxierten Instanz identisch, außer dass l, H_1 und H_2 Mengen sind.

Operatoren: *Der Roboter kann den Raum wechseln. Im entstehenden Zustand befindet er sich **in beiden** Räumen.*
*Der Roboter kann genau dann einen Ball $b \in B$ in Raum r mit Greifarm g aufnehmen, wenn $r \in l$ und Greifarm g frei ist. Im entstehenden Zustand hält der Roboter Ball b in Greifarm g. Ball b wird **nicht** aus Raum r entfernt und g wird **nicht** aus der Menge der freien Greifarme entfernt.*
*Der Roboter kann genau dann einen Ball b in Raum r aus Greifarm g fallenlassen, wenn $b \in H_g$ und $r \in l$. Im entstehenden Zustand befindet sich Ball b in Raum r **und** in Greifarm g. Greifarm g wird zur Menge der freien Greifarme hinzugefügt.*

5.2 Berechnung der h^+-Heuristik

Mit den aus den Benchmarks resultierenden Beschränkungen kann ein relaxierter Plan relativ leicht gefunden werden. Im wesentlichen muss der Roboter einmal den Raum wechseln und jeden Ball aus dem ersten Raum aufheben und im anderen Raum fallenlassen.

Satz 5.1
Das Finden eines optimalen h^+-Plans für eine Gripper-Instanz ist in Zeit $O(|B| + |R|) = O(|B|)$ möglich. Für jeden optimalen Plan h^+ gilt

$$
\begin{aligned}
|h^+| &= \mathcal{L}(\mathit{Gripper}) \\
&:= \begin{cases}
0, & \text{falls } B^1 \text{ leer und } H_1 = H_2 = \omega \\
1_{H_1 \neq \omega} + 1_{H_2 \neq \omega}, & \text{falls } B^1 \text{ leer und der Roboter} \\
 & \text{sich in Raum 2 befindet} \\
2|B^1| + 1_{H_1 \neq \omega} + 1_{H_2 \neq \omega} + 1, & \text{sonst}
\end{cases}
\end{aligned}
$$

Beweis:
„$|h^+| \leq \mathcal{L}(\text{Gripper})$:“
Betrachte folgenden Plan für das Problem:

Falls beide Greifarme und der erste Raum leer sind, ist nichts zu tun.
Falls der erste Raum leer ist und der Roboter sich im zweiten befindet, lässt er die in seinen Greifarmen befindlichen Bällen fallen, daraufhin sind alle Bälle im zweiten Raum.
Andernfalls wechselt er den Raum, lässt alle Bälle, die er in den Greifarmen hält, im zweiten Raum fallen, nimmt alle Bälle aus dem ersten Raum auf und lässt sie im zweiten Raum fallen, daraufhin sind alle Bälle im zweiten Raum.
„$|h^+| \geq \mathcal{L}$(Gripper)“:
Der erste Fall ist klar.
Im zweiten Fall muss der Roboter in jedem gültigen Plan die gehaltenen Bälle fallenlassen ($1_{H_1 \neq \omega} + 1_{H_2 \neq \omega}$ Aktionen) damit sich alle Bälle im zweiten Raum befinden.
Im dritten Fall muss der Roboter in jedem gültigen Plan die gehaltenen Bälle fallenlassen ($1_{H_1 \neq \omega} + 1_{H_2 \neq \omega}$ Aktionen) und alle Bälle aus dem ersten Raum aufnehmen und wieder fallenlassen ($2|B^1|$ Aktionen). Da nach Voraussetzung der erste Raum Bälle enthält oder der Roboter sich im ersten Raum befindet, muss er einmal den Raum wechseln (1 Aktion), um die Aufnehm- und Fallenlassaktionen im richtigen Raum ausführen zu können. □

5.3 Experimentelle Ergebnisse

Die h^+-Heuristik wurde implementiert. Die Experimente wurden auf einem Computer mit acht Intel Xeon Prozessoren mit einer Taktfrequenz von 2.66 GHZ durchgeführt. Falls nach 30 Minuten kein optimaler Plan gefunden wurde oder der Speicherbedarf 2 Gigabyte überschritten hat, wurde die Suche abgebrochen. Neben der Merge-And-Shrink-Heuristik wurde die h^+-Heuristik mit der Goal-Count-Heuristik und mit der blinden Suche verglichen. In der Gripper-Domäne ist die Goal-Count-Heuristik, welche die Anzahl der nicht erfüllten Zielbedingungen zurückliefert, der h^+-Heuristik ähnlich. Die Anzahl der nicht erfüllten Zielbedingungen entspricht der Anzahl der Bälle, die sich **nicht** im zweiten Raum befinden, somit liefert die Goal-Count-Heuristik etwa den halben Wert der h^+-Heuristik zurück. Die blinde Suche ist eine Breitensuche auf dem Zustandsraum. Für die Merge-And-Shrink-Heuristik wurde durch Experimente der optimale Parameter $N = 5000$ für die Größe der Abstraktionen gefunden. Die Vergleiche der Anfangswertschätzung, Lösungsdauer und expandierten Knoten findet sich in den Abbildungen 5.1, 5.2 und 5.3. Es werden nur Probleme gelistet, die von mindestens einer der Heuristiken gelöst wurden.
Der Anfangswert der Goal-Count-Heuristik entspricht der Anzahl der Bälle der jeweiligen Instanz. Wie in den Tabellen zu sehen ist, liegen die Ergebnisse aller vier Heuristiken nahe beieinander. Die Goal-Count-Heuristik und

Problemname	optimale Planlänge	h^+-Heuristik	Merge-And-Shrink-Heuristik, N=5000	Goal-Count Heuristik	blinde Suche
Problem 01	11	9	**11**	4	1
Problem 02	17	13	**15**	6	1
Problem 03	23	**17**	11	8	1
Problem 04	29	**21**	13	10	1
Problem 05	35	**25**	14	12	1
Problem 06	41	**29**	16	14	1
Problem 07	47	**33**	18	16	1

Abbildung 5.1: Vergleich der Bewertung des Anfangszustands der h^+-, der Merge-And-Shrink-, der Goal-Count-Heuristik und der blinden Suche

Problemname	optimale Planlänge	h^+-Heuristik	Merge-And-Shrink-Heuristik, N=5000	Goal-Count Heuristik	blinde Suche
Problem 01	11	0.00	0.00	0.00	0.00
Problem 02	17	0.00	0.10	0.00	0.00
Problem 03	23	0.06	0.34	0.05	0.05
Problem 04	29	0.44	1.04	**0.33**	0.35
Problem 05	35	2.86	3.59	2.15	**2.09**
Problem 06	41	17.79	16.19	13.10	**12.70**
Problem 07	47	97.60	79.83	70.92	**69.11**

Abbildung 5.2: Vergleich der Laufzeit der h^+-, der Merge-And-Shrink-, der Goal-Count-Heuristik und der blinden Suche

die blinde Suche schlagen die h^+- und die Merge-And-Shrink-Heuristik bei der Laufzeit sogar, da sie deutlich einfacher zu berechnen sind. Alle Instanzen mit mehr als 16 Bällen konnten mit keiner Heuristik gelöst werden, was jeweils am Speicherlimit lag. Bei Problem 7 mussten zum Lösen bereits mehr als 10 Millionen Zustände expandiert werden. Dieses schlechte Resultat verwundert nicht, Malte Helmert und Gabrielle Röger haben gezeigt [HR08], dass selbst eine fast-perfekte Heuristik, die sich von der tatsächlichen Planlänge nur um 1 unterscheidet, mindestens die Hälfte aller Zustände des gesamten Zustandsraums expandieren muss. Das Suchen nach einem optimalen Plan mithilfe einer Heuristik ist somit unpraktikabel, dagegen kann ein optimaler Plan für die Instanz in linearer Zeit mit einem trivialen Algorithmus gefunden werden [Hel08].
In Abbildung 5.4 ist eine Übersicht der h^+-Heuristik auf den gelösten Instanzen zu sehen.

Problemname	optimale Planlänge	h^+-Heuristik	Merge-And Shrink-Heuristik, N=5000	Goal-Count Heuristik	blinde Suche
Problem 01	11	82	**12**	229	236
Problem 02	17	1249	**975**	1803	1826
Problem 03	23	**10304**	11506	11689	11736
Problem 04	29	**65687**	68380	68479	68558
Problem 05	35	**371726**	376510	376653	376772
Problem 06	41	**1974285**	1982018	1982227	1982394
Problem 07	47	**10080252**	10091970	10092241	10092464

Abbildung 5.3: Vergleich der expandierten Knoten der h^+-, der Merge-And-Shrink-, der Goal-Count-Heuristik und der blinden Suche

Problemname	optimale Planlänge	Anfangswert	expandierte Knoten	Zeit
Problem 01	11	9	82	0.00
Problem 02	17	13	1249	0.00
Problem 03	23	17	10304	0.06
Problem 04	29	21	65687	0.44
Problem 05	35	25	371726	2.86
Problem 06	41	29	1974285	17.79
Problem 07	47	33	10080252	97.60

Abbildung 5.4: Übersicht über die gelösten Probleme der h^+-Heuristik

Kapitel 6

Blocksworld

Dieses Kapitel beschäftigt sich mit der Blocksworld-Domäne.
Zunächst wird das Problem definiert, bevor wir uns mit dem Verfahren zur Berechnung der h^+-Heuristik befassen. Da das Verfahren implementiert wurde, folgt danach der Abschnitt über die Experimente, in welchem die h^+-Heuristik mit der Merge-And-Shrink-Heuristik verglichen wird.

6.1 Definition des Problems

Es gilt, Blöcke auf einem Tisch gemäß Zielvorgaben aufeinanderzustapeln. Dabei kann ein Block sich entweder auf genau einem anderen Block oder auf dem Tisch befinden. Zum Stapeln der Blöcke steht ein Roboter mit einem Greifarm zur Verfügung.

Definition 6.1 (Blocksworld-Problemstellung)
Eine Blocksworld-Problemstellung ist gegeben durch ein 5-Tupel $\langle B, \text{on}_0, \text{on}_*, \text{clear}_0, \text{clear}_* \rangle$*, bestehend aus folgenden Komponenten:*

- *B ist eine endliche Menge von Blöcken, wobei table $\notin B$.*
- $\text{on}_0 : B \to B \cup \{table\}$ *ist die Anfangsstapelfunktion. Wenn* $\text{on}_0(b) = b'$*, schreiben wir auch* $\text{on}_0(b, b')$ *und sagen, Block b liegt anfangs auf Block b', bzw. Block b liegt anfangs auf dem Tisch, falls $b' = table$. Deswegen nennen wir die Anfangsstapelfunktion auch die Anfangsstapelrelation.*
- $\text{on}_* : B' \to B \cup \{table\}$ *mit $B' \subseteq B$ ist die Zielstapelfunktion. Wenn* $\text{on}_*(b) = b'$*, so schreiben wir auch* $\text{on}_*(b, b')$ *und sagen, Block b soll im Ziel auf Block b', bzw. Block b soll im Ziel auf dem Tisch, falls $b' = table$, liegen. Deswegen nennen wir die Zielstapelfunktion auch die Zielstapelrelation.*

- $\text{clear}_0 \subseteq B$ *ist die Menge der Blöcke, auf denen sich anfangs kein anderer Block befindet. Für einen Block* $b \in B$ *gilt genau dann* $b \in \text{clear}_0$*, wenn es keinen Block* $b' \in B$ *gibt, so dass* $\text{on}_0(b', b)$*. Wenn* $b \in \text{clear}_0$*, so sagen wir,* b *liegt anfangs oben auf einem Stapel und schreiben dafür auch* $\text{clear}_0(b)$*.*

- $\text{clear}_* \subseteq B$ *ist die Menge der Blöcke, auf denen sich im Ziel kein Block befinden soll. Wenn für einen Block* $b \in B$ *gilt* $b \in \text{clear}_*$*, so gibt es keinen Block* $b' \in B$ *mit* $\text{on}_*(b', b)$*. Wenn* $b \in \text{clear}_*$*, so sagen wir,* b *liegt im Ziel oben auf einem Stapel und schreiben dafür auch* $\text{clear}_*(b)$*.*

Definition 6.2 (Blocksworld-Domäne)
Die Blocksworld-Domäne bildet Blocksworld-Problemstellungen mit Blöcken B *wie folgt auf Zustandsräume ab:*

Zustände: *Dies sind 4-Tupel* $\langle \text{on}, \text{clear}, H, F \rangle$*. Dabei ist* $\text{on} : B \to B \cup \{table\}$ *die Stapelfunktion,* $\text{clear} \in B$ *die Menge der freien Blöcke,* $H \in B \cup \{\omega\}$*, mit* $\omega \notin B$*, der vom Roboter gehaltene Block und* $F \in \{true, false\}$ *gibt an, ob der Greifarm frei ist. Für* $\text{on}(b) = b'$ *schreiben wir auch* $\text{on}(b, b')$ *und sagen, Block* b *befindet sich auf Block* b' *bzw. auf dem Tisch, falls* $b' = table$*. Wir sagen, Block* $b \in B$ *ist frei, wenn* $b \in \text{clear}$ *und schreiben dafür auch* $\text{clear}(b)$*. Wir sagen, der Roboter hält Block* $b \in B$*, falls* $H = b$*, bzw. der Greifarm des Roboters ist leer, falls* $H = \omega$*. Wir sagen, der Greifarm des Roboters ist frei, wenn* $F = true$ *und der Greifarm ist besetzt andernfalls.*

Anfangszustand: *Der Anfangszustand ist das 4-Tupel* $\langle \text{on}_0, \text{clear}_0, \omega, true \rangle$*, wobei* on_0 *die Anfangsstapelfunktion und* clear_0 *die Menge der Blöcke, die anfangs auf Stapeln oben sind, ist.*

Zielzustände: *Genau die Zustände* $\langle \text{on}, \text{clear}, \omega, true \rangle$ *mit* $\text{on}(b) = b'$ *für alle Zielstapelrelationen* $\text{on}_*(b, b')$ ***und*** $\text{clear}(c)$ *für alle Blöcke* $c \in B$ *mit* $\text{clear}_*(c)$ *sind Zielzustände.*

Operatoren: *Der Roboter kann genau dann einen Block* $b \in B$ *aufnehmen, wenn* b *und der Greifarm frei sind. Im entstehenden Zustand hält der Roboter Block* b*, der Block befindet sich weder auf dem Tisch noch auf einem anderen Block und der Block und der Greifarm sind nicht frei. Falls sich* b *vorher auf einem Block* b' *befand, so ist* b' *nun frei.*
Der Roboter kann genau dann einen Block b *auf dem Tisch ablegen, wenn er Block* b *hält. Im entstehenden Zustand ist der Greifarm leer und frei und* b *befindet sich auf dem Tisch.*
Der Roboter kann genau dann einen Block b *auf einem anderen Block* b' *ablegen, wenn er Block* b *hält und* b' *frei ist. Im entstehenden Zustand ist der Greifarm leer und frei,* b *befindet sich auf* b' *und* b' *ist nicht frei.*

Definition 6.3 (relaxierte Blocksworld-Domäne)
Die relaxierte Blocksworld-Domäne bildet einen Zustand $\left\langle \hat{\text{on}}, \hat{\text{clear}}, \hat{H}, \hat{F} \right\rangle$ *einer (nicht-relaxierten) Blocksworld-Instanz auf Zustandsräume ab. In den neuen Zustandsräumen bildet* $\text{on} : B \to 2^{B \cup \{table\}}$ *auf Teilmengen ab,* $H \subseteq B \cup \{\omega\}$ *und* $F \subseteq \{true, false\}$ *sind Teilmengen. Identisch mit der nicht-relaxierten Instanz bleibt* $\text{clear} \subseteq B$*. Die Operatoren unterscheiden sich in der relaxierten Domäne:*

Operatoren: *Der Roboter kann genau dann einen Block* $b \in B$ *aufnehmen, wenn* b *und der Greifarm frei sind. Im entstehenden Zustand hält der Roboter Block* b ***und*** *seine bisher gehaltenen Blöcke, der Block* ***befindet*** *sich auf dem Tisch bzw. dem Block, auf dem er sich befand und der Block und der Greifarm sind* ***frei****.*
Der Roboter kann genau dann einen Block b *auf dem Tisch ablegen, wenn er Block* b *hält. Im entstehenden Zustand* ***hält er Block*** b*, der Greifarm ist frei und* b *befindet sich auf dem Tisch.*
Der Roboter kann genau dann einen Block b *auf einem anderen Block* b' *ablegen, wenn er Block* b *hält und* b' *frei ist. Im entstehenden Zustand* ***hält er Block*** b*, der Greifarm ist frei,* b *befindet sich auf* b' *und* b' *ist* ***frei****.*

6.2 Berechnung der h^+-Heuristik

Im Falle, dass für jeden Block im Ziel seine Position angegeben ist, d.h. wenn es nur einen Zielzustand gibt, kann man die h^+-Heuristik berechnen, in dem man etwas ähnliches wie Partitionen auf den Blöcken bildet. Im allgemeinen Fall führt eine Abwandlung dieses Vorgehens zum Ziel.

Satz 6.1
Für Blocksworld-Instanzen, in denen für jeden Block $b \in B$ *die Zielstapelfunktion* on_* *definiert und der Greifarm im Startzustand frei ist, ist das Finden eines optimalen* $h^+ - Plans$ *in Zeit* $O(|B|^2)$ *möglich.*
Bilde Tupel $\mathcal{B}_1, \ldots, \mathcal{B}_m$ *derart, dass gilt,*

- *werden die Tupel als Mengen aufgefasst, ist* $\bigcup_{i=1}^{m} \mathcal{B}_i = \mathcal{B} \cup \{table\}$.
- *werden die Tupel als Mengen aufgefasst, ist* $\mathcal{B}_i \cap \mathcal{B}_j = \emptyset$ *oder* $\mathcal{B}_i \cap \mathcal{B}_j = \{table\}$ *für* $i \neq j$.
- *für alle* i *ist* $\mathcal{B}_i \neq \langle table \rangle$.

- *$\mathcal{B}_1, \ldots, \mathcal{B}_m$ sind gerade die maximalen Mengen zusammengehörender Blöcke, die im aktuellen und im Zielzustand in der gleichen Reihenfolge vorkommen, aufgezählt von unten nach oben. D.h. für $\mathcal{B}_i = \langle b_1, \ldots, b_{k_i} \rangle$ gilt,*
  - *$\text{on}(b_{j+1}, b_j)$ und $\text{on}_*(b_{j+1}, b_j)$ für $j \in \{1, \ldots, k_i - 1\}$.*
  - *$\text{on}(b_1) \neq$ table oder $\text{on}_*(b_1) \neq$ table. D.h. der erste Block des Tupels liegt im aktuellen Zustand oder im Ziel **nicht** auf dem Tisch. Falls der erste Eintrag des Tupels kein Block, sondern der Tisch ist, ist dies trivialerweise erfüllt.*
  - *kein Block aus einem anderen Tupel $\mathcal{B}_j$ liegt im aktuellen Zustand **und** im Zielzustand auf dem letzten Eintrag von $\mathcal{B}_i$, d.h. für alle $\mathcal{B}_j = \left\langle b'_1, \ldots, b'_{k_j} \right\rangle$ mit $j \neq i$, gilt $\text{on}(b'_p) \neq b_{k_i}$ oder $\text{on}_*(b'_p) \neq b_{k_i}$ für alle $p \in \{1, \ldots, k_j\}$.*

Œ *seien $\mathcal{B}_1, \ldots, \mathcal{B}_k$ die einzigen Tupel, deren erster Eintrag table ist. Dann gilt für jeden optimalen Plan h^+*

$$|h^+| = \mathcal{L}(\textit{Blocksworld}) := \underbrace{\sum_{i=k+1}^{m} |\mathcal{B}_i|}_{\textit{müssen entstapelt werden}} + \underbrace{(m-k)}_{\textit{müssen gestapelt werden}}$$

Beweis:
„$|h^+| \geq \mathcal{L}$(Blocksworld)“:
In jedem gültigen Plan muss jeder Block, der auf einem falschen Block liegt, auf den richtigen gesetzt werden. Um ihn dorthin zu setzen, muss er in die Hand genommen werden. Ist er nicht der oberste Block, so ist er nicht frei und es müssen alle Blöcke über ihm ebenfalls in die Hand genommen werden. Da dies insbesondere für die ersten Einträge aus $\mathcal{B}_{k+1}, \ldots, \mathcal{B}_m$ gilt – diese sind nach Voraussetzung Blöcke – da diese per Definition nicht auf dem richtigen Stein liegen, müssen auf jeden Fall alle Blöcke aus den Tupeln $\mathcal{B}_{k+1}, \ldots, \mathcal{B}_m$ in die Hand genommen werden ($\sum_{i=k+1}^{n} |\mathcal{B}_i|$ Operationen). Um die nicht erfüllten on_*-Relationen zu erfüllen, müssen Blöcke abgelegt werden. Zumindest die on_*-Relationen für die ersten Einträge aus den Tupeln $\mathcal{B}_{k+1}, \ldots, \mathcal{B}_m$ sind nicht erfüllt, dafür sind mindestens $m - k$ Operationen notwendig. Damit folgt für alle gültigen h^+-Pläne $|h^+| \geq \mathcal{L}$(Blocksworld).
„$|h^+| \leq \mathcal{L}$(Blocksworld)“:
Betrachte folgenden Plan für das Problem.
Nehme nach und nach alle Blöcke bis auf die Blöcke der Tupel $\mathcal{B}_1, \ldots, \mathcal{B}_k$ auf. Danach hält der Roboter alle Blöcke, bis auf die der Mengen $\mathcal{B}_1, \ldots, \mathcal{B}_k$ in der Hand, alle Blöcke in seiner Hand und jeweils der oberste von $\mathcal{B}_1, \ldots, \mathcal{B}_k$ sind frei. Lege die untersten Blöcke von $\mathcal{B}_{k+1}, \ldots, \mathcal{B}_m$ auf den richtigen Block

Abbildung 6.1: Beispiel für ein Blocksworld-Problem

ab, dies muss jeweils der oberste Block von $\mathcal{B}_1, \ldots, \mathcal{B}_m$ oder der Tisch selber sein. Damit sind alle on_*-Relationen erfüllt. Gemäß Definition der Domäne sind somit automatisch die clear_*-Relationen erfüllt. Die Länge dieses Plans ist $\mathcal{L}$(Blocksworld). □

Beispiel 6.1
Betrachte Abbildung 6.1.
Anfangs sind, wie in Start zu sehen, die Blöcke 1 bis 5 übereinandergestapelt, daneben ist ein Stapel mit den Blöcken 6 bis 8. Im Ziel sollen die Blöcke 1 bis 3 übereinander liegen, und auf einem zweiten Haufen die Blöcke 4 bis 8. Die in Satz 6.1 erwähnten Tupel sind
$\mathcal{B}_1 = \langle table, 1, 2, 3\rangle$
$\mathcal{B}_2 = \langle 4, 5\rangle$
$\mathcal{B}_3 = \langle 6, 7, 8\rangle$
Um die relaxierte Instanz zu lösen, nimmt der Roboter die Steine 4 *und* 5 *vom ersten Stapel, sowie die Steine* 6, 7 *und* 8*, also den kompletten zweiten Stapel, auf. Danach legt er Stein* 4 *auf dem Tisch und Stein* 6 *auf Stein* 5 *ab. Dies sind insgesamt 7 Operationen. Die Formel aus Satz 6.1 liefert ebenfalls* $\sum_{i=2}^{3} |\mathcal{B}_i| + (3-1) = 7$.

Die im Satz angegebene quadratische Laufzeit resultiert aus den Tupeln $\mathcal{B}_1, \ldots, \mathcal{B}_m$. Um sie zu finden, müssen für jeden Block $b \in B$ alle on- und on_*-Relationen durchgegangen werden.
In den meisten Instanzen sind die Voraussetzungen aus Satz 6.1 nicht gegeben. Der Roboter hält häufig einen Block, die on_*-Relation ist nicht für jeden Block definiert und es gibt Blöcke mit der clear_*-Relation, die nicht schon implizit durch die on_*-Relationen gegeben ist. Durch Anpassungen können wir das bisherige Resultat auf diese Fälle übertragen.

Korollar 6.1
Falls $\text{on}_*(b)$ *für Blöcke* $b \in B$ *nicht definiert ist, setze zur Erstellung der Tupel* $\mathcal{B}_1, \ldots, \mathcal{B}_m$ *die Zielstapelfunktion* $\text{on}_*(b) = \text{on}(b)$*. Gehe danach für* $j \in \{1, \ldots, k\}$ *wie folgt vor, um die Stapel* $\mathcal{B}_1, \ldots, \mathcal{B}_k$ *eventuell aufzuspalten.*

Sei $i \in \{1, \ldots, k_j - 1\}$ mit $\mathcal{B}_j = \langle b_1, \ldots, b_{k_j} \rangle$ der niedrigste Index, falls es einen solchen gibt, für den eine der beiden folgenden Eigenschaften erfüllt ist.

- *Die Zielstapelfunktion $\mathrm{on}_*(b_i)$ ist nicht definiert, d.h. es wurde $\mathrm{on}(b_i)$ zur Erstellung von $\mathcal{B}_j$ verwendet, und es existiert ein Block b mit $\mathrm{on}_*(b) = b_i$ **und** $\mathrm{on}(b) \neq b_i$.*
- *Der Block b_i soll im Ziel frei sein, d.h. $b_i \in \mathrm{clear}_*$.*

Falls ein solcher Index existiert, setze $\mathcal{B}_j = \langle b_1, \ldots, b_{i-1} \rangle$ und $\mathcal{B}_{m+1} = \langle b_i, \ldots, b_{k_j} \rangle$. Setze $m = m + 1$ und fahre mit dem nächsten j fort.
Berechne die Planlänge wie in Satz 6.1. Falls der Roboter einen Block $b \in B$ in seinem Greifarm hält, erhöht sich die Planlänge um 1. Weiter gilt,

- *falls $\mathrm{on}_*(b) = table$ oder $\mathrm{on}_*(b) = b'$, mit b' einem der Blöcke, der im aktuellen Zustand auf einem Stapel ganz oben liegt, erhöht sich die Länge des Plans nicht weiter.*
- *falls $\mathrm{on}_*(b) = b'$ für einen Block $b' \in \mathcal{B}_j$, $j > k$, der auf $\mathcal{B}_j$ nicht oben liegt, erhöht sich die Länge des Plans nochmals um 1.*
- *falls $\mathrm{on}_*(b) = b'$, mit b' der n-te Block von oben von $\mathcal{B}_j$, mit $j \leq k$, erhöht sich die Länge des Plans nochmals um $n - 1$.*

Beweis:
Führe den Plan aus Satz 6.1 durch. Es ändert sich, dass Mengen auf dem Tisch unter Umständen aufgespalten werden müssen.

- Blöcke b, für die $\mathrm{on}_*(b)$ nicht definiert ist, beeinflussen den Plan nur, wenn im Ziel ein Block auf ihnen liegen muss. Da alle Blöcke der Mengen $\mathcal{B}_{k+1}, \ldots, \mathcal{B}_m$ sowieso hochgehoben werden, müssen zusätzlich nur eventuelle Blöcke aus den Mengen $\mathcal{B}_1, \ldots, \mathcal{B}_k$ hochgehoben werden.
- Selbiges gilt für Blöcke mit der clear_*-Eigenschaft, auch sie können dafür sorgen, dass Blöcke aus den Tupeln $\mathcal{B}_1, \ldots, \mathcal{B}_k$ hochgehoben werden müssen.
- Wird ein Block in der Hand gehalten, muss dieser anfangs abgelegt werden, damit die Hand leer wird. Falls der Block aus der Hand auf den Tisch muss oder auf einen der Stapel des aktuellen Zustands, ist dies mit dem ersten Ablegen bereits möglich. Andernfalls muss er später erneut abgelegt werden. Außerdem müssen die Blöcke über dem Zielblock hochgehoben werden, falls der Zielblock aus einem der Tupel $\mathcal{B}_1, \ldots, \mathcal{B}_k$ stammt, da er in diesem Fall nicht frei ist.

Die ersten beiden Punkte entsprechen den durchgeführten Aufpaltungen, der letzte Punkt entspricht der Behandlung des in der Hand befindlichen Blocks im Korollar. □

Bei dieser Abänderung des Vorgehens müssen die Tupel $\mathcal{B}_1, \ldots, \mathcal{B}_k$ erneut durchgegangen und mit den Blöcken aus den anderen Tupeln verglichen werden. Da die Tupel bereits erstellt wurden, ist dies in $O(|B|^2)$ möglich, die asymptotische Laufzeit ändert sich nicht. Insbesondere ist das relaxierte Blocksworld-Problem polynomiell lösbar. Das nicht-relaxierte Blocksworld-Längen-Problem dagegen ist NP-schwer [GN92] und somit wegen der polynomiellen Beschränkheit eines optimalen Planes NP-vollständig.

6.3 Experimentelle Ergebnisse

Die h^+-Heuristik wurde implementiert. Neben den Instanzen aus dem Planungswettbewerb wurde sie auf bewiesen schwierigen Instanzen getestet und mit der Merge-And-Shrink-Heuristik verglichen. Für die Merge-And-Shrink-Heuristik wurden durch Experimente zwei optimale Parameter $N = 25000$ und $N = 50000$ gefunden. Die Experimente wurden auf einem Computer mit acht Intel Xeon Prozessoren mit einer Taktfrequenz von 2.66 GHZ durchgeführt. Falls nach 30 Minuten kein optimaler Plan gefunden wurde oder der Speicherbedarf 2 Gigabyte überschritten hat, wurde die Suche abgebrochen. Wenn eine der Heuristiken eine der Instanzen nicht lösen konnte, wird dies durch Striche dargestellt. Instanzen, die von keiner der Heuristiken gelöst werden konnten, fehlen in den Tabellen, ausser sie befinden sich zwischen gelösten Instanzen. Im letzten Fall werden sie durch Striche dargestellt.

6.3.1 Instanzen des Planungswettbewerbs

Der erste Teil der Vergleiche fand mit den Instanzen aus dem Planungswettbewerb statt. Es gibt Instanzen 4-17, jeweils in 1-3 Versionen. Die Nummer der Instanz entspricht der Anzahl der vorkommenden Blöcke. In den Abbildungen 6.2, 6.3 und 6.4 sind die Vergleiche der Anfangswertschätzung, Lösungsdauer und Anzahl der expandierten Knoten zwischen der h^+- und der Merge-And-Shrink-Heuristik zu sehen. Bei den Instanzen, die nicht gelöst werden konnten, lag dies bei beiden Heuristiken am begrenzten Speicher. In Abbildung 6.5 ist eine Übersicht der mit der h^+-Heuristik gelösten Instanzen zu sehen.

Problemname	optimale Planlänge	h^+-Heuristik	Merge-And-Shrink-Heuristik, N=25000	Merge-And-Shrink-Heuristik, N=50000
Problem 4-0	6	6	6	6
Problem 4-1	10	6	10	10
Problem 4-2	6	6	6	6
Problem 5-0	12	8	12	12
Problem 5-1	10	7	10	10
Problem 5-2	16	9	16	16
Problem 6-0	12	11	12	12
Problem 6-1	10	10	10	10
Problem 6-2	20	11	17	**18**
Problem 7-0	20	13	16	**18**
Problem 7-1	22	12	15	**16**
Problem 7-2	20	12	14	**16**
Problem 8-0	18	13	14	**16**
Problem 8-1	20	13	14	**16**
Problem 8-2	16	14	14	**16**
Problem 9-0	30	16	14	16
Problem 9-1	28	16	14	16
Problem 9-2	26	**17**	15	16
Problem 10-0	34	18	-	18
Problem 10-1	32	**19**	16	16
Problem 10-2	34	**19**	-	-
Problem 11-0	32	19	19	**21**
Problem 11-1	30	**21**	-	-
Problem 11-2	34	**19**	-	-
Problem 12-0	34	**22**	-	-
Problem 12-1	34	**22**	-	-
Problem 13-0	-	-	-	-
Problem 13-1	44	**25**	-	-
Problem 14-0	38	**25**	-	-
Problem 14-1	36	**27**	-	-
Problem 15-0	40	**28**	-	-

Abbildung 6.2: Vergleich der Bewertung des Anfangszustands der h^+- und der Merge-And-Shrink-Heuristik

Problemname	optimale Planlänge	h^+-Heuristik	Merge-And-Shrink-Heuristik, N=25000	Merge-And-Shrink-Heuristik, N=50000
Problem 4-0	6	**0.00**	0.04	0.04
Problem 4-1	10	**0.00**	0.04	0.04
Problem 4-2	6	**0.00**	0.04	0.04
Problem 5-0	12	**0.00**	0.98	1.43
Problem 5-1	10	**0.00**	1.02	1.42
Problem 5-2	16	**0.00**	1.05	1.41
Problem 6-0	12	**0.00**	6.35	11.09
Problem 6-1	10	**0.00**	4.89	9.65
Problem 6-2	20	**0.01**	5.85	11.32
Problem 7-0	20	**0.01**	10.74	27.24
Problem 7-1	22	**0.04**	9.86	23.78
Problem 7-2	20	**0.01**	10.33	21.34
Problem 8-0	18	**0.01**	21.00	36.99
Problem 8-1	20	**0.04**	23.93	35.20
Problem 8-2	16	**0.01**	27.39	35.80
Problem 9-0	30	**0.65**	60.44	191.12
Problem 9-1	28	**0.02**	52.82	69.25
Problem 9-2	26	**0.04**	55.77	90.12
Problem 10-0	34	**15.42**	-	367.82
Problem 10-1	32	**1.82**	286.03	316.28
Problem 10-2	34	**5.68**	-	-
Problem 11-0	32	**4.35**	220.29	199.01
Problem 11-1	30	**4.64**	-	-
Problem 11-2	34	**3.39**	-	-
Problem 12-0	34	**4.54**	-	-
Problem 12-1	34	**0.48**	-	-
Problem 13-0	-	-	-	-
Problem 13-1	44	**1078.59**	-	-
Problem 14-0	38	**10.64**	-	-
Problem 14-1	36	**19.99**	-	-
Problem 15-0	40	**420.91**	-	-

Abbildung 6.3: Vergleich der Lösungsdauer der h^+- und der Merge-And-Shrink-Heuristik

Problemname	optimale Planlänge	h^+-Heuristik	Merge-And-Shrink-Heuristik, N=25000	Merge-And-Shrink-Heuristik, N=50000
Problem 4-0	6	9	7	7
Problem 4-1	10	12	11	11
Problem 4-2	6	7	7	7
Problem 5-0	12	20	13	13
Problem 5-1	10	19	11	11
Problem 5-2	16	44	17	17
Problem 6-0	12	17	13	13
Problem 6-1	10	17	11	11
Problem 6-2	20	**260**	437	318
Problem 7-0	20	**72**	126	177
Problem 7-1	22	**1047**	3877	3836
Problem 7-2	20	**188**	333	940
Problem 8-0	18	**155**	4917	709
Problem 8-1	20	**1029**	39077	11905
Problem 8-2	16	**53**	270	275
Problem 9-0	30	**13215**	1917666	971774
Problem 9-1	28	**360**	168515	60311
Problem 9-2	26	**594**	79512	54583
Problem 10-0	34	**241489**	-	19143580
Problem 10-1	32	**29144**	15567572	12886413
Problem 10-2	34	**83007**	-	-
Problem 11-0	32	**63891**	12474668	7291064
Problem 11-1	30	**59340**	-	-
Problem 11-2	34	**53642**	-	-
Problem 12-0	34	**58124**	-	-
Problem 12-1	34	**6284**	-	-
Problem 13-0	-	-	-	-
Problem 13-1	44	**9990123**	-	-
Problem 14-0	38	**100499**	-	-
Problem 14-1	36	**160352**	-	-
Problem 15-0	40	**3540691**	-	-

Abbildung 6.4: Vergleich der expandierten Knoten der h^+- und der Merge-And-Shrink-Heuristik

Problemname	optimale Planlänge	Anfangswert	expandierte Knoten	Zeit
Problem 4-0	6	6	9	0.00
Problem 4-1	10	6	12	0.00
Problem 4-2	6	6	7	0.00
Problem 5-0	12	8	20	0.00
Problem 5-1	10	7	19	0.00
Problem 5-2	16	9	44	0.00
Problem 6-0	12	11	17	0.00
Problem 6-1	10	10	17	0.00
Problem 6-2	20	11	260	0.01
Problem 7-0	20	13	72	0.01
Problem 7-1	22	12	1047	0.04
Problem 7-2	20	12	188	0.01
Problem 8-0	18	13	155	0.01
Problem 8-1	20	13	1029	0.04
Problem 8-2	16	14	53	0.01
Problem 9-0	30	16	13215	0.65
Problem 9-1	28	16	360	0.02
Problem 9-2	26	17	594	0.04
Problem 10-0	34	18	241489	15.42
Problem 10-1	32	19	29144	1.82
Problem 10-2	34	19	83007	5.68
Problem 11-0	32	19	63891	4.35
Problem 11-1	30	21	59340	4.64
Problem 11-2	34	19	53642	3.39
Problem 12-0	34	22	58124	4.54
Problem 12-1	34	22	6284	0.48
Problem 13-0	-	-	-	-
Problem 13-1	44	25	9990123	1078.59
Problem 14-0	38	25	100499	10.64
Problem 14-1	36	27	160352	19.99
Problem 15-0	40	28	3540691	420.91

Abbildung 6.5: Übersicht über die gelösten Probleme der h^+-Heuristik

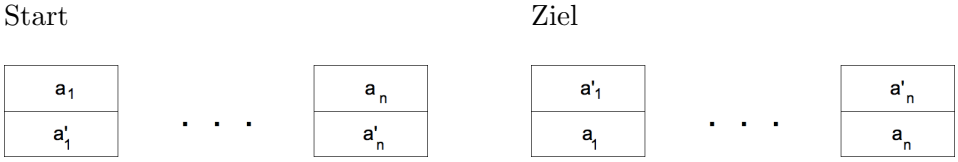

Abbildung 6.6: Schwierige Blocksworld-Instanz mit n Stapeln aus je 2 Blöcken

Abbildung 6.7: Schwierige Blocksworld-Instanz mit einem Stapel aus n Blöcken

6.3.2 Schwierige Instanzen

Malte Helmert und Gabrielle Röger haben für zwei Klassen von Blocksworld-Instanzen gezeigt, dass selbst eine fast-perfekte Heuristik, die die tatsächliche Planlänge nur um 1 unterschätzt, auf diesen Instanzen exponentiell viele Zustände exploriert [HR08]. Bei der ersten Klasse gibt es n Stapel bestehend aus jeweils 2 Blöcken, jeder dieser Stapel soll umgekehrt werden, siehe dazu Abbildung 6.6. Die Schwierigkeit resultiert hierbei daraus, dass die n Stapel unabhängig voneinander gelöst werden können, dies führt zu einer großen Anzahl an optimalen Plänen und damit Zuständen, die expandiert werden müssen. Bei der zweiten Klasse gibt es einen großen Stapel mit n Blöcken, der oberste Block soll nach unten, der restliche Stapel auf diesen, siehe dazu Abbildung 6.7. Die Schwierigkeit resultiert hierbei daraus, dass zu Beginn alle bis auf zwei on_*-Relationen erfüllt sind und auf dem Weg zum Ziel die bereits erfüllten on_*-Relationen zerstört werden müssen.
In den Abbildungen 6.8, 6.9 und 6.10 ist der Vergleich der h^+- und der Merge-And-Shrink-Heuristik auf der ersten Klasse von Instanzen mit n kleinen Stapeln nachzuvollziehen und in Abbildung 6.11 ist eine Übersicht der h^+-Heuristik auf dieser Klasse zu sehen. In den Abbildungen 6.12, 6.13 und 6.14 befindet sich der Vergleich der h^+- und der Merge-And-Shrink-Heuristik auf der zweiten Klasse von Instanzen mit einem großen Stapel, in Abbildung 6.15 ist eine Übersicht der h^+-Heuristik auf dieser Klasse zu sehen.
Der Vergleich der expandierten Knoten und die Übersicht zeigen sehr schön das exponentielle Zunehmen der expandierten Knoten bei wachsender Block-

Problemname	optimale Planlänge	h^+-Heuristik	Merge-And-Shrink-Heuristik, N=25000	Merge-And-Shrink-Heuristik, N=50000
1 kleine Stapel	4	3	4	4
2 kleine Stapel	8	6	8	8
3 kleine Stapel	12	9	12	12
4 kleine Stapel	16	12	16	16
5 kleine Stapel	20	15	15	**16**
6 kleine Stapel	24	**18**	-	-
7 kleine Stapel	28	**21**	-	-

Abbildung 6.8: Vergleich der Bewertung des Anfangszustands der h^+- und der Merge-And-Shrink-Heuristik für n kleine Stapel

Problemname	optimale Planlänge	h^+-Heuristik	Merge-And-Shrink-Heuristik, N=25000	Merge-And-Shrink-Heuristik, N=50000
1 kleine Stapel	4	0.00	0.00	0.00
2 kleine Stapel	8	**0.00**	0.03	0.04
3 kleine Stapel	12	**0.00**	2.10	3.47
4 kleine Stapel	16	**0.05**	15.63	32.00
5 kleine Stapel	20	**0.72**	128.87	188.95
6 kleine Stapel	24	**14.50**	-	-
7 kleine Stapel	28	**302.41**	-	-

Abbildung 6.9: Vergleich der Lösungsdauer der h^+- und der Merge-And-Shrink-Heuristik für n kleine Stapel

Problemname	optimale Planlänge	h^+-Heuristik	Merge-And-Shrink-Heuristik, N=25000	Merge-And-Shrink-Heuristik, N=50000
1 kleine Stapel	4	5	5	5
2 kleine Stapel	8	13	9	9
3 kleine Stapel	12	83	13	13
4 kleine Stapel	16	809	17	17
5 kleine Stapel	20	**9605**	129947	61951
6 kleine Stapel	24	**129771**	-	-
7 kleine Stapel	28	**1960005**	-	-

Abbildung 6.10: Vergleich der expandierten Knoten der h^+- und der Merge-And-Shrink-Heuristik für n kleine Stapel

Problemname	optimale Planlänge	Anfangswert	expandierte Knoten	Zeit
1 kleine Stapel	4	3	5	0.00
2 kleine Stapel	8	6	13	0.00
3 kleine Stapel	12	9	83	0.00
4 kleine Stapel	16	12	809	0.05
5 kleine Stapel	20	15	9605	0.72
6 kleine Stapel	24	18	129771	14.50
7 kleine Stapel	28	21	1960005	302.41

Abbildung 6.11: Übersicht über die gelösten Probleme der h^+-Heuristik für n kleine Stapel

Problemname	optimale Planlänge	h^+-Heuristik	Merge-And-Shrink-Heuristik, N=25000	Merge-And-Shrink-Heuristik, N=50000
2 Blöcke	4	3	4	4
3 Blöcke	8	4	8	8
4 Blöcke	12	5	12	12
5 Blöcke	16	6	16	16
6 Blöcke	20	7	15	**16**
7 Blöcke	24	8	14	**15**
8 Blöcke	28	9	14	14
9 Blöcke	32	10	12	**14**
10 Blöcke	36	**11**	-	-
11 Blöcke	40	**12**	-	-
12 Blöcke	44	**13**	-	-

Abbildung 6.12: Vergleich der Bewertung des Anfangszustands der h^+- und der Merge-And-Shrink-Heuristik für einen großen Stapel mit n Blöcken

Problemname	optimale Planlänge	h^+-Heuristik	Merge-And-Shrink-Heuristik, N=25000	Merge-And-Shrink-Heuristik, N=50000
2 Blöcke	4	0.00	0.00	0.00
3 Blöcke	8	0.00	0.00	0.00
4 Blöcke	12	**0.00**	0.04	0.04
5 Blöcke	16	**0.00**	1.06	1.37
6 Blöcke	20	**0.01**	4.11	8.47
7 Blöcke	24	**0.04**	9.16	19.00
8 Blöcke	28	**0.23**	17.75	39.71
9 Blöcke	32	**1.67**	44.64	88.47
10 Blöcke	36	**13.58**	-	-
11 Blöcke	40	**114.16**	-	-
12 Blöcke	44	**961.10**	-	-

Abbildung 6.13: Vergleich der Lösungsdauer der h^+- und der Merge-And-Shrink-Heuristik für einen großen Stapel mit n Blöcken

Problemname	optimale Planlänge	h^+-Heuristik	Merge-And-Shrink-Heuristik, N=25000	Merge-And-Shrink-Heuristik, N=50000
2 Blöcke	4	5	5	5
3 Blöcke	8	10	9	9
4 Blöcke	12	19	13	13
5 Blöcke	16	53	17	17
6 Blöcke	20	210	179	**134**
7 Blöcke	24	**1038**	2656	1864
8 Blöcke	28	**5850**	52358	52306
9 Blöcke	32	**35766**	933977	897989
10 Blöcke	36	**234347**	-	-
11 Blöcke	40	**1665091**	-	-
12 Blöcke	44	**12213093**	-	-

Abbildung 6.14: Vergleich der expandierten Knoten der h^+- und der Merge-And-Shrink-Heuristik für einen großen Stapel mit n Blöcken

Problemname	optimale Planlänge	Anfangswert	expandierte Knoten	Zeit
2 Blöcke	4	3	5	0.00
3 Blöcke	8	4	10	0.00
4 Blöcke	12	5	19	0.00
5 Blöcke	16	6	53	0.00
6 Blöcke	20	7	210	0.01
7 Blöcke	24	8	1038	0.04
8 Blöcke	28	9	5850	0.23
9 Blöcke	32	10	35766	1.67
10 Blöcke	36	11	234347	13.58
11 Blöcke	40	12	1665091	114.16
12 Blöcke	44	13	12213093	961.10

Abbildung 6.15: Übersicht über die gelösten Probleme der h^+-Heuristik für einen großen Stapel mit n Blöcken

anzahl in den beiden schwierigen Instanzen. Die h^+-Heuristik schneidet bei den schwierigen Instanzen wie auch auf den Instanzen des Planungswettbewerbs deutlich besser ab als die Merge-And-Shrink-Heuristik.

Kapitel 7

Satellite

Für die in diesem Kapitel vorgestellte Satellite-Domäne kann die h^+-Heuristik nicht über eine einfache Formel angegeben werden, wie das bei den bisherigen Domänen der Fall war, daher ist dieses Kapitel umfangreicher als die letzten.
Zunächst wird eine Definition der Problemstellung gegeben, dies sind hauptsächlich Übersetzungen aus der Doktorarbeit von Malte Helmert [Hel08]. Danach wird die Berechnung der h^+-Heuristik auf dieser Domäne erklärt, mittels eines exponentiellen Ansatzes, der im Wesentlichen alle Möglichkeiten durchgeht, gefolgt von Hinweisen zur Implementierung sowie möglichen Verbesserungen. Den eigentlichen Abschluss dieses Kapitels bildet schließlich der Abschnitt über den praktischen Vergleich der h^+-Heuristik mit der Merge-And-Shrink-Heuristik. Danach folgt ein kleiner Exkurs über eine mögliche Veränderung der Domäne mit interessantem Resultat.

7.1 Definition des Problems

Es gilt, von verschiedenen Himmelskörpern Aufnahmen in verschiedenen Modi, wie z.B. Infrarot oder Wärmebild, zu schießen. Um diese Aufgabe zu erledigen stehen Satelliten zur Verfügung, die verschiedene Instrumente an Bord haben, von denen jedes eine Auswahl der Modi unterstützt.

Definition 7.1 (Satellite-Problemstellung)
Eine Satellite-Problemstellung ist gegeben durch ein 10-Tupel $\langle S, I, M, D, l, C, cal, p_0, p_*, O_* \rangle$*, bestehend aus folgenden Komponenten:*

- S *ist eine endliche Menge von Satelliten.*
- I *ist eine endliche Menge von Instrumenten.*

- *M ist eine endliche Menge von Bildmodi oder kurz Modi.*
- *D ist eine endliche Menge von Ausrichtungen oder Winkeln.*
- *$l : I \to S$ bildet jedes Instrument auf den Satelliten ab, auf dem es montiert ist.*
- *$C \subseteq I \times M$ ist die binäre Relation der Instrumentenfähigkeiten. Wenn $\langle i, m \rangle \in C$ für ein Instrument i und einen Bildmodus m, sagen wir, i unterstützt m.*
- *$cal : I \to D$ ist die Kalibrierungszielfunktion.*
- *$p_0 : S \to D$ ist die Anfangsausrichtungsfunktion.*
- *$p_* : S' \to D$ mit $S' \subseteq S$ ist die Zielausrichtungsfunktion. Ein Paar $\langle s, d \rangle$ mit $p_*(s) = d$ wird Ausrichtungsziel genannt.*
- *$O_* \subseteq D \times M$ ist die Menge der Bildziele.*

Definition 7.2 (Satellite-Domäne)
Die Satellite-Domäne bildet Satellite-Problemstellungen mit Satelliten S, Instrumenten I, Bildmodi M und Ausrichtungen D wie folgt auf Zustandsräume ab:

Zustände: *Dies sind 5-Tupel $\langle p, S^P, I^P, I^C, O \rangle$, wobei $p : S \to D$ die Ausrichtungsfunktion, $S^P \subseteq S$ die Menge der Satelliten mit verfügbarem Strom, $I^P \subseteq I$ die Menge der eingeschalteten Instrumente, $I^C \subseteq I$ die Menge der kalibrierten Instrumente und $O \subseteq D \times M$ die Menge der offenen Bildziele ist. Wenn $p(s) = d$ sagen wir, dass Satellit $s \in S$ in Richtung $d \in D$ zeigt.*

Anfangszustand: *Der Anfangszustand ist das 5-Tupel $\langle p_0, S, \emptyset, \emptyset, O_* \rangle$, wobei p_0 die Anfangsausrichtungsfunktion und O_* die Menge der Bildziele der Problemstellung ist.*

Zielzustände: *Genau die Zustände $\langle p, S^P, I^P, I^C, \emptyset \rangle$ mit $p(s) = d$ für alle Ausrichtungsziele $\langle s, d \rangle$ sind Zielzustände.*

Operatoren: *Ein Satellit $s \in S$ kann sich in Richtung $d \in D$ drehen. Im entstehenden Zustand zeigt er nach d.*
Ein Instrument $i \in I$ kann genau dann eingeschaltet werden, wenn der Satellit, auf dem es montiert ist, über Strom verfügt. Im entstehenden Zustand wird i zur Menge der eingeschalteten Instrumente hinzugefügt und der Satellit aus der Menge der Satelliten mit verfügbarem Strom entfernt.
Ein Instrument $i \in I$ kann genau dann ausgeschaltet werden, wenn

es eingeschaltet ist. Im entstehenden Zustand wird i aus der Menge der eingeschalteten Instrumente entfernt, ebenso aus der Menge der kalibrierten Instrumente, falls es sich darin befand. Der Satellit, auf dem es montiert ist, wird zur Menge der Satelliten mit verfügbarem Strom hinzugefügt.
Ein Instrument $i \in I$ kann genau dann kalibriert werden, wenn es eingeschaltet ist und der Satellit, auf dem es montiert ist, in Richtung des Kalibrierungszieles $cal(I)$ zeigt. Im entstehenden Zustand wird i zur Menge der kalibrierten Instrumente hinzugefügt.
Ein Instrument $i \in I$ kann genau dann ein Bild einer Ausrichtung $d \in D$ im Modus $m \in M$ aufnehmen, wenn es eingeschaltet und kalibriert ist, den Modus m unterstützt und der Satellit, auf dem es montiert ist, in Richtung d zeigt. Im entstehenden Zustand wird $\langle d, m \rangle$ aus der Menge der offenen Bildziele entfernt, falls es ein solches war.

Definition 7.3 (relaxierte Satellite-Domäne)
Die relaxierte Satellite-Domäne bildet einen Zustand $\left\langle \hat{p}, \hat{S^P}, \hat{I^P}, \hat{I^C}, \hat{O} \right\rangle$ einer (nicht-relaxierten) Satellite-Instanz auf Zustandsräume ab. Die Zustände sind dabei bis auf die Ausrichtungsfunktion p mit denen der nicht-relaxierten Instanz identisch. In der relaxierten Domäne bildet die Ausrichtungsfunktion nicht auf die Menge der Ausrichtungen ab, sondern auf deren Teilmengen, d.h. $p : S \to 2^D$.

Operatoren: *Ein Satellit $s \in S$ kann sich in Richtung $d \in D$ drehen. Im entstehenden Zustand zeigt er nach d, d.h. die neue Ausrichtungsfunktion ist*

$$p'(s') = \begin{cases} p(s), & \text{falls } s' \neq s \\ p(s) \cup \{d\}, & \text{falls } s' = s \end{cases},$$

wobei p die alte Ausrichtungsfunktion bezeichnet. Die alte Ausrichtung des Satelliten ***bleibt*** *erhalten.*
Ein Instrument $i \in I$ kann genau dann eingeschaltet werden, wenn der Satellit, auf dem es montiert ist, in der Menge der Satelliten mit verfügbarem Strom enthalten ist. Im entstehenden Zustand wird i zur Menge der eingeschalteten Instrumente hinzugefügt. Der Satellit wird ***nicht*** *aus der Menge der Satelliten mit verfügbarem Strom entfernt.*
Ein Instrument $i \in I$ kann genau dann ausgeschaltet werden, wenn es eingeschaltet ist. Im entstehenden Zustand wird der Satellit, auf dem es montiert ist, zur Menge der Satelliten mit verfügbarem Strom hinzugefügt. Das Instrument wird ***nicht*** *aus der Menge der eingeschalteten Instrumente entfernt.*
Ein Instrument $i \in I$ kann genau dann kalibriert werden, wenn es eingeschaltet ist und der Satellit, auf dem es montiert ist, in Richtung des Kalibrierungszieles $cal(I)$ zeigt. Im entstehenden Zustand wird i

zur Menge der kalibrierten Instrumente hinzugefügt.
Ein Instrument $i \in I$ kann genau dann ein Bild einer Ausrichtung $d \in D$ im Modus $m \in M$ aufnehmen, wenn es eingeschaltet und kalibriert ist, den Modus m unterstützt und der Satellit, auf dem es montiert ist, in Richtung d zeigt. Im entstehenden Zustand wird $\langle d, m\rangle$ aus der Menge der offenen Bildziele entfernt, falls es ein solches war.

7.2 Berechnung der h^+-Heuristik

In der relaxierten Domäne muss jedes Instrument höchstens einmal eingeschaltet werden und je Satellit muss höchstens ein Instrument ausgeschaltet werden, um das Problem zu lösen. Es bleibt dennoch schwierig.

Satz 7.1
Das relaxierte Satellite-Längen-Problem ist NP-schwer.

Beweis:
Wir zeigen, dass man jede Mengenüberdeckungsinstanz als Satellite-Instanz darstellen kann. Die Gesamtmenge wird dabei auf die Menge der Bildmodi abgebildet. Jede Teilmenge wird kodiert als unterstützte Bildmodi eines der Instrumente. Wir werden sehen, dass es genau dann eine Mengenüberdeckung der Größe k gibt, wenn es in der entstehenden Satellite-Instanz einen Plan der Länge k' gibt.
Mengenüberdeckung:
Gegeben: Menge Ω, Teilmengen $\Omega_1, \ldots, \Omega_n \subseteq \Omega$.
Gefragt: Existieren k Teilmengen, so dass $\bigcup_{j=1,\ldots,k} \Omega_{i_j} = \Omega$?
Kodierung als Satellite-Instanz:

- Erstelle für jedes $\omega \in \Omega$ einen Bildmodus.
- Erstelle n Satelliten $s_1, \ldots, s_n$, alle in die gleiche Richtung d_0 blickend.
- Weise jedem Satelliten s_j ein Instrument i_j zu, das nach d_0 kalibriert wird und genau die Modi $\omega \in \Omega_j$ unterstützt.
- Setze $p_* = \emptyset$.
- Setze $O_* = \{\langle d_0, \omega\rangle \mid \omega \in \Omega\}$.
- Setze $D = \{d_0\}$.

Für einen minimalen Plan h^+ gilt genau dann $|h^+| \leq |\Omega| + 2k$, wenn es eine k-Überdeckung von Ω gibt.
„$\Leftarrow$“:

Es gebe eine k-Überdeckung. Œ sei diese $\Omega_1, \ldots, \Omega_k$. Schalte nacheinander $i_1, \ldots, i_k$ ein, kalibriere sie und schieße die nötigen Bilder. Bezeichnet man diesen Plan mit h, gilt

$$|h| = \underbrace{k}_{\text{einschalten}} + \underbrace{k}_{\text{kalibrieren}} + \underbrace{|\Omega|}_{\text{Bilder}}.$$

„⇒“:
In jedem gültigen Plan werden mindestens $|\Omega|$ Bilder geschossen. Jeder Satellit, der zum Bilderschiessen verwendet wird, muss eingeschaltet und kalibriert werden. Somit gilt für jeden gültigen Plan h', dass $|h'| \geq |\Omega| + 2l$, wobei l die Anzahl der verwendeten Satelliten ist. Wenn also $|h^+| \leq |\Omega| + 2k$ für einen gültigen Plan h^+ gilt, werden in diesem Plan weniger als k Satelliten zum Bilderschießen verwendet.
Somit kann die Menge mit k Teilmengen überdeckt werden. □

Wir wissen jetzt, dass wir zur Berechnung des optimalen h^+-Plans nicht auf einen polynomiellen Algorithmus hoffen sollten. Dennoch wollen wir einen (möglichst effizienten) Algorithmus finden, der das Problem löst. Eventuell findet sich ein bekanntes NP-vollständiges Problem, auf das wir das Satellite-Problem reduzieren können.
In Satz 7.1 haben wir gesehen, dass man das Mengenüberdeckungsproblem auf das relaxierte Satellite-Problem reduzieren kann. Unter bestimmten Voraussetzungen gilt auch die Rückrichtung. Wenn alle Instrumente eingeschaltet und kalibriert sind, liefert eine Mengenüberdeckung den optimalen Plan für das Satellite-Problem.

Lemma 7.1
Sei eine relaxierte Satellite-Instanz gegeben, in der alle Instrumente eingeschaltet und kalibriert sind und es keine Ausrichtungsziele gibt. Seien für alle $d \in D$

- $O_d = \{m \mid \langle d, m\rangle \in O_* \wedge \forall i \in I : (\langle i, m\rangle \notin C \vee p(l(i)) \neq d)\}$ *die Modi, in denen Bilder im Winkel d geschossen werden müssen, dafür aber eine Satellitendrehung notwendig ist.*
- $I^d = \{i \in I | \exists m \in O_d : \langle i, m\rangle \in C\}$ *die Instrumente, die mindestens einen Bildmodus aus O_d unterstützen.*
- $O_s^d = \{m | \exists i \in I^d : (l(i) = s \wedge (i, m) \in C)\}$ *für $s \in S$ die Bildmodi aus O^d, die von mindestens einem Instrument des Satelliten s unterstützt werden.*

Mit $S = s_1, \ldots, s_n$ gilt für den optimalen h^+-Plan für diese Satellite-Instanz

$$|h^+| = \mathcal{L}(Satellite, 1) := |O_*| + \sum_{d \in D} SetCover(O_d; O_{s_1}^d, \ldots, O_{s_n}^d).$$

Beweis:
„$|h| \leq \mathcal{L}(\text{Satellite}, 1)$“:
Betrachte folgenden Plan für das Problem.
Für jedes d drehe die Satelliten, die an der Mengenüberdeckung von O_d beteiligt sind, in Richtung d. Danach gibt es für jedes Bildziel $\langle d, m\rangle \in O_*$ mindestens einen Satelliten, der in Richtung d ausgerichtet ist und auf dem ein Instrument montiert ist, das m unterstützt. Falls $\langle d, m\rangle \in O_d$, gilt dies wegen der ausgeführten Drehbewegungen, ansonsten galt dies bereits davor. Schieße mit passenden Instrumenten Bilder.
„$|h| \geq \mathcal{L}(\text{Satellite}, 1)$“:
In jedem gültigen h^+-Plan sind mindestens $|O_*|$ Aktionen zum Schießen der Bilder nötig. Jedes $\langle d, m\rangle \in O_*$, so dass ein Satellit existiert, der bereits in Richtung d zeigt und auf dem ein Instrument montiert ist, welches den Bildmodus m unterstützt, kann aufgenommen werden. Für alle anderen $\langle d, m\rangle \in O_*$ müssen Satelliten gedreht werden. Das Drehen eines Satelliten in Richtung $d_2 \neq d_1$ beeinflusst niemals den Plan für $\langle d_1, m\rangle \in O_*$, da alle Instrumente bereits kalibriert sind. Somit kann das Gesamtproblem in die Probleme O_d, mit $d \in D$, aufgeteilt werden. Für eine feste Ausrichtung $d \in D$ entspricht eine minimale Anzahl an Satellitendrehungen einer minimalen Überdeckung von O_d durch die Mengen $O^d_{s_1}, \ldots, O^d_{s_n}$, somit liefert die Formel den kürzesten Plan. □

Bemerkung 7.1
Falls es Ausrichtungsziele gibt, können diese Ausrichtungen vor dem Suchen der Mengenüberdeckung durchgeführt werden. Da sie irgendwann ausgeführt werden müssen, aber in der relaxierten Domäne keinen negativen Effekt haben, ist der optimale Ausführungszeitpunkt zu Beginn. Dadurch können weitere Ziele aus O_* gestrichen werden, falls durch die Drehbewegung ein passender Satellit in Richtung eines Bildziels ausgerichtet wurde.

Für den Fall, dass nicht alle Instrumente eingeschaltet und kalibriert sind, geht es nicht so „einfach“. Das Einschalten eines Instruments benötigt eine Aktion, das Kalibrieren benötigt eine oder zwei Aktionen, aber eventuell ist der Satellit danach in Richtung eines Zieles ausgerichtet. Falls auf dem Satelliten ein Instrument eingeschaltet ist, ist für das Einschalten eines weiteren Instruments eine Aktion für das Ausschalten nötig. All diese Aktionen, die miteinander verwoben sind, erschweren das Problem im Vergleich zum einfacheren Fall aus Lemma 7.1. In diesem allgemeinen Fall muss die optimale Lösung nicht aus einer Mengenüberdeckung bestehen, wie folgendes Beispiel zeigt.

Beispiel 7.1
Sei eine Satellite-Instanz mit vier Instrumenten i_1, i_2, i_3 und i_4, zwei Satelliten s_1 und s_2, drei Modi m_1, m_2 und m_3 und zwei Ausrichtungen d_1

Instrument	Satellit	Modi			Kalibrierungs-winkel des Instrumentes	eingeschaltet/ kalibriert
		m_1	m_2	m_3		
i_1	s_1	x	x	-	d_2	nein/nein
i_2	s_1	-	-	x	d_1	ja/nein
i_3	s_2	x	-	-	d_1	ja/ja
i_4	s_2	-	x	-	d_1	nein/nein

Abbildung 7.1: Übersicht über die Eigenschaften der vier Instrumente

und d_2 gegeben. Anfangs zeigen beide Satelliten in Richtung d_1. In Abbildung 7.1 ist dargestellt, auf welchem Satelliten die Instrumente montiert sind, welche Modi sie unterstützen, über welchen Winkel sie kalibriert werden und ob sie eingeschaltet und kalibriert sind. Die Zielaufnahmen seien $O_* = \{\langle d_1, m_1\rangle, \langle d_1, m_2\rangle\}$.
Ein optimaler Plan schaltet auf s_2 das Instrument i_3 aus, das Instrument i_4 ein und kalbriert Instrument i_4, danach schießt er die beiden Bilder. Dieser Plan hat eine Gesamtlänge von 5. Die minimale Mengenüberdeckung wäre i_1, der zugehörige Plan hat eine Länge von 6.

Um eine relaxierte Satellite-Instanz zu lösen, dreht man alle Satelliten in ihre Zielausrichtung und schießt anschließend die Bildziele, für die ein eingeschaltetes und kalibriertes Instrument auf einem entsprechend ausgerichteten Satelliten existiert. Durch testweise Inbetriebnahme der Instrumente sämtlicher Satelliten mit anschließendem Lösen der Mengenüberdeckung mithilfe dieser Instrumente, werden für die restlichen Fotoziele die minimalen Kosten bestimmt. Der in Algorithmus 2 abgebildete SatelliteSolve-Algorithmus führt diese Schritte durch.

Satz 7.2
SatelliteSolve aus Algorithmus 2 liefert die optimale h^+-Planlänge für eine relaxierte Satellite-Instanz.

Beweis:
Es gibt einen Plan der angegebenen Länge, dieser befolgt die Schritte des Algorithmus und schießt danach alle Bildziele.
Noch zu zeigen ist, dass es keinen kürzeren Plan geben kann. In jedem gültigen h^+-Plan, der das Problem löst, müssen die $|O_*|$ Ziele fotografiert werden und irgendwann die Satelliten in die Zielrichtung ausgerichtet werden. Das Drehen der Satelliten in Zielrichtung bringt keinen Nachteil, je früher es stattfindet. Aber es bringt den Vorteil, dass die Satelliten sich in einer zusätzlichen Ausrichtung befinden, somit ist die hier durchgeführte frühestmögliche Drehung optimal. In jedem optimalen Plan werden bestimmte Instrumente eingeschaltet und kalibriert (nämlich diejenigen, die für die Aufnah-

Algorithm 2 SatelliteSolve

Input: Menge S der Satelliten, Menge I der Instrumente, Menge O_* der Zielaufnahmen, Menge p_* der Satellitenzielausrichtungen

Output: Minimale Kosten für das relaxierte Problem

1: minKosten = ∞
2: Drehe alle Satelliten in ihre Zielausrichtung.
3: Entferne alle Ziele $\langle d, m \rangle$ aus O_*, für die es ein eingeschaltetes und kalibriertes Instrument gibt, das m unterstützt und auf einem Satelliten montiert ist, der nach d zeigt.
4: **for all** Teilmengen $S' \subseteq S$ **do**
5: **if** Instrumente von S' überdecken nicht O_* **then**
6: continue
7: **end if**
8: **for all** Teilmengen I' der Instrumente von S', die O_* überdecken **do**
9: Führe für jedes Instrument i aus I' alle nötigen der folgenden Schritte aus

- Falls der Satellit keinen Strom hat, schalte ein eingeschaltetes Instrument aus.
- Schalte i ein.
- Bringe den zugehörigen Satelliten in die Kalibrierungsrichtung.
- Kalibriere i.

14: Lösche Ziele $\langle d, m \rangle$, für die ein kalibriertes Instrument existiert, das Modus m unterstützt und das auf einem Satelliten montiert ist, der nach d zeigt.
15: Finde für jeden Winkel aus O_* eine minimale Überdeckung mit den eingeschalteten und kalibrierten Instrumenten.
16: **if** aktuelle Kosten < minKosten **then**
17: minKosten = aktuelle Kosten
18: **end if**
19: **end for**
20: **end for**
21: **return** $|O_*| + minKosten$

men benutzt werden). Wenn nur mit eingeschalteten und kalibrierten Instrumenten gearbeitet wird, liefert eine Mengenüberdeckung gemäß Lemma 7.1 die optimale Planlänge. Da der Algorithmus alle Kombinationen von Instrumenten durchgeht, wird schließlich die optimale Planlänge gefunden und zurückgegeben. □

Es werden alle Teilmengen von Satelliten durchgegangen und für jeden Satelliten einer solchen Teilmenge alle nichtleeren Teilmengen der Instrumente desselben. Dies ist äquivalent dazu, sämtliche Teilmengen von Instrumenten durchzugehen. Für jede solche Teilmenge wird eine Mengenüberdeckungsinstanz gelöst, somit hat der angegebene Algorithmus eine Laufzeit von $O(2^{|I|} \cdot SC(|S|, |O_*|))$. Dabei ist $SC(n, m)$ die asymptotische Laufzeit, die man braucht, um ein Mengenüberdeckungsproblem mit n Teilmengen einer m-elementigen Gesamtmenge zu lösen.

7.3 Umsetzung

In diesem Abschnitt soll erklärt werden, wie Algorithmus 2 in C++ umgesetzt wurde. Anschließend werden einige Verbesserungen vorgestellt, die eingebaut wurden, um die Laufzeit zu beschleunigen.

7.3.1 Implementierung

Bei Algorithmus 2 werden in den Zeilen 4 und 8 Teilmengen ausgewählt, es ist noch zu klären, wie dies implentiert werden kann. In Zeile 4 sollen sämtliche Teilmengen der Satelliten ausgewählt werden, in Zeile 8 sämtliche nichtleeren Teilmengen von Instrumenten, da eine leere Teilmenge dem Verzicht auf den zugehörigen Satelliten entspricht. Somit muss nicht auf eine vorteilhafte Reihenfolge bei der Auswahl geachtet werden. Dennoch ist nicht klar, wie eine Teilmengenauswahl zu programmieren ist, bei der schließlich sämtliche Teilmengen vorkommen. Ich habe die Auswahl der Satelliten über Bitfelder implementiert, siehe dazu Algorithmus 3.

Beispiel 7.2
Ein Aufruf von initialize$(5, 3)$ *liefert das Bitfeld 00111. Aufeinanderfolgende next*() *Aufrufe liefern 01011, 01101, 01110, 10011, 10101, 10110, 11001, 11010, 11100.*

Ein Zustand eines Bitfeldes entspricht einer k-elementigen Teilmenge einer Obermenge der Größe size. Es wird, wenn auf der Obermenge eine Ordnung definiert ist, genau dann das i-te Element in die Teilmenge aufgenommen, wenn im Bitfeld das i-te Bit gesetzt ist. Das folgende Lemma zeigt, dass alle

Algorithm 3 Bitfeld-Klasse

```
1: class Bitfeld do
2:   function void initialize(int size, int k)
3:   function bool next()
4:   vector<bool> a
5: end class
```

```
1: def initialize(int size, int k) do
2:   erstelle vector<bool> a der Größe size
3:   setze die letzten k Bits von a
4: end def
```

```
1: def next() do
2:   if die ersten k Bits gesetzt sind then
3:     return false
4:   else
5:     suche von hinten die erste Kette von 1ern, diese habe die Länge n
6:     verschiebe die erste 1 der Kette um ein Bit nach vorne, lösche den
       Rest der Kette
7:     setze die letzten n − 1 Bits
8:     return true
9:   end if
10: end def
```

k-elementigen Teilmengen abgedeckt werden, wenn das Bitfeld nach dem Initialisieren solange mit *next*() hochgezählt wird, bis false zurückgeliefert wird.

Lemma 7.2
Für ein mit initialize(n, k) erstelltes Bitfeld gilt:

1. *Das Bitfeld hat zu jeder Zeit genau k gesetzte Bits.*

2. *Wird der Vektor a als Binärzahl aufgefasst, und sei a der Vektor vor und* succ(a) *der Vektor nach einem Aufruf von next(), der true zurückliefert. Dann gilt* succ$(a) > a$.

3. *Durch Aufruf von initialize(n, k) und anschließendes Aufrufen von next(), bis das erste Mal false zurückgeliefert wird, werden $\binom{n}{k}$ Binärzahlen durchlaufen.*

Beweis:

zu 1) Die Behauptung ist unmittelbar nach der Initialisierung wahr. Durch einen Aufruf von *next*(), der das Bitfeld verändert, werden nur 1en verschoben, aber nicht gelöscht. Somit ist die Behauptung auch nach dem Aufruf von *next*() wahr.

zu 2) Da durch den Aufruf von *next*() eine 1, die auf eine 0 folgt, nach vorne geschoben wird, ist die erste Stelle von vorne, an der sich a und succ(a) unterscheiden, bei succ(a) eine 1 und bei a eine 0. Somit gilt succ$(a) > a$ wie behauptet.

zu 3) Die erzeugten Binärzahlen haben die Gestalt $a = a_{n-1}, \ldots, a_0$ wobei nach dem Initialisieren $a = \underbrace{0 \ldots 0}_{n-k}\underbrace{1 \ldots 1}_{k}$ gilt. Wir zeigen durch Induktion über k, dass $\binom{n}{k}$ Binärzahlen durchlaufen werden.

$k = 0$: Nach dem Initialisieren liefert bereits der erste Aufruf von *next*() false zurück und $\binom{n}{0}$ wertet sich ebenfalls zu 1 aus.

$k \to k + 1$: Die vorderste 1 durchläuft die Stellen $m = k, \ldots, n-1$, die nachfolgenden 1en durchlaufen die gleiche Sequenz wie in einem mit *initialize(m, k)* initialisierten Bitfeld. Somit gilt nach Indukti-

onsvoraussetzung für die Anzahl der durchlaufenen Binärzahlen:

$$\begin{aligned}\sum_{m=k}^{n-1}\binom{m}{k} &= \sum_{m=0}^{n-1-k}\binom{k+m}{k} \\ &\overset{\text{bin. Summenregel}}{=} \binom{k+n-1-k+1}{k+1} \\ &= \binom{n}{k+1}\end{aligned}$$

Wir haben nicht gezeigt, dass die Binärzahlen notwendig verschieden sind. Dies folgt zusammen mit Eigenschaft 2).

□

Aus den Eigenschaften 1)-3) von Lemma 7.2 folgt, dass die Menge der durchlaufenen Binärzahlen eine $\binom{n}{k}$-elementige Teilmenge der Menge

$$\mathcal{M} = \{(a_{n-1}, \ldots, a_0) | a_i \in \{0,1\} \text{ für } i = 0, \ldots, n-1; \sum_{i=0}^{n} a_i = k\},$$

der n-stelligen Binärzahlen mit k gesetzten Bits, ist. Da die Menge $\mathcal{M}$ nur $\binom{n}{k}$ Elemente enthält, entspricht die Menge der durchlaufenen Binärzahlen $\mathcal{M}$.
Falls also mit Hilfe von Bitfeldern k-elementige Teilmengen ausgesucht werden, ist garantiert, dass alle k-elementigen Teilmengen durchlaufen werden. Für die Auswahl der Satelliten wird k von 0 bis $|S|$ hochgezählt und die k-elementigen Teilmengen bestimmt.
In Zeile 8 könnte man die Auswahl der Instrumente genauso handhaben, allerdings müsste für jeden Satelliten ein eigenes Bitfeld erstellt werden und es ist nicht klar, wie man dafür sorgt, dass jede Kombination von eingeschalteten Instrumenten durchlaufen wird. Deswegen wurde eine eigene Instrumentcounter-Klasse erstellt, die mehrere Bitfelder enthält und das Hochzählen verwaltet. Sie bietet außerdem den Vorteil, dass ihr eine Mindest- und eine Maximalzahl an gleichzeitig eingeschalteten Instrumenten übergeben werden kann, was für die Optimierung des Algorithmus eine Rolle spielen wird. In Algorithmus 4 wird die Instrumentcounter-Klasse vorgestellt. Beim Initialisieren werden ihr ein Zahlenvektor und zwei Zahlen für die Grenzen – low und high – mitgeteilt. Im Vektor steht für jeden Satelliten, wieviele Instrumente er besitzt, und die Grenzen geben an, wieviele Instrumente insgesamt mindestens bzw. höchstens verwendet werden sollen. Bei der *next*()-Methode muss beim Erhöhen der Bits eines Bitfeldes aufgepasst werden, dass die obere Grenze eingehalten wird, beim Erhöhen der Position mit Initialisieren des vorigen Bitfeldes muss aufgepasst werden, dass

Algorithm 4 Counter-Klasse für die Instrumente

```
1: class Counter do
2:   function void initialize(vector<int> v, int l, int h)
3:   function bool next()
4:   vector<Bitfeld> c
5:   int low
6:   int high
7: end class
```

```
1: def initialize(vector<int> v, int l, int h) do
2:   low = max(l, |v|), high = min(h, sum_{i=0}^{|v|-1} v_i)
3:   Erstelle |v| Bitfelder der Größen, die in v angegeben sind.
4:   Fülle die ersten Bitfelder komplett mit 1en, die letzten mit genau
     einer 1, und das Bitfeld dazwischen mit genausovielen 1en, dass in
     allen Bitfeldern zusammen genau low Bits gesetzt sind.
5:   Weise c den Vektor bestehend aus diesen Bitfeldern zu.
6: end def
```

```
1: def next() do
2:   position = first, endwhile = false
3:   while (not endwhile) or position = last do
4:     if an position next() = true then
5:       endwhile = false
6:     else if an position Erhöhung der gesetzten Bits möglich then
7:       Initialisiere das Bitfeld an position mit einem Bit mehr.
8:       endwhile = false
9:     else
10:      Erhöhe position.
11:      Initialisiere Bitfeld an (position-1) mit minimal möglicher Anzahl
         an Bits.
12:      endwhile = true
13:    end if
14:  end while
15:  if endwhile then
16:    return false
17:  else
18:    return true
19:  end if
20: end def
```

die untere Grenze eingehalten wird. Aus Gründen der einfachen Darstellung spiegeln sich diese Tatsachen im Pseudocode nicht wieder.
Diese Counterklasse wird anfangs mit der Anzahl der Instrumente pro Satellit und im naiven Algorithmus mit den Werten $low = 0$ und $high = \infty$ initialisiert. Danach können durch wiederholten Aufruf von $next()$ alle Teilmengen an Instrumenten erhalten werden. Im nächsten Unterabschnitt über mögliche Verbesserungen werden wir sehen, wie bessere untere und obere Grenzen gefunden werden können.
Es ist noch unklar, wie in Algorithmus 2 in Zeile 15 die minimale Überdeckung gefunden werden kann. Eine einfache Lösungsvariante für das SetCover-Problem ist in Algorithmus 5 zu sehen. Preprocess aus Zeile 5

Algorithm 5 SetCover

Input: Menge Ω, Teilmengen $\Omega_1, \ldots, \Omega_n$

Output: Eine minimale Zahl k, so dass $\bigcup_{j=1}^{k} \Omega_{i_j} = \Omega$ oder ∞, falls nicht möglich

```
1: Schneide alle Ω_i mit Ω, falls nötig.
2: if ⋃_{i=1}^{n} Ω_i ≠ Ω then
3:    return ∞
4: end if
5: preprocess(Ω, Ω_1, ..., Ω_n)
6: min = lower_bound(Ω, Ω_1, ..., Ω_n)
7: for k=min,...,n do
8:    if SetCover_k(Ω, Ω_1, ..., Ω_n) then
9:       return k
10:   end if
11: end for
```

ignoriert in der implementierten Version Ω und eliminiert lediglich die Ω_i, die vollständig in einem Ω_j, mit $i \neq j$, enthalten sind. Außerdem werden die Ω_i nach der Anzahl ihrer Elemente sortiert, was beim Aufruf von $SetCover_k$ einem Greedy-Ansatz entspricht. Lower_bound schätzt die minimale Anzahl an Teilmengen, die für eine Überdeckung nötig sind. Es liefert das kleinste k, so dass es eine Teilfolge der Größe k gibt, für die $\sum_{j=1}^{k} |\Omega_{i_j}| \geq |\Omega|$ gilt. Schließlich überprüft $SetCover_k$, ob es eine k-Überdeckung von Ω durch $\Omega_1, \ldots, \Omega_n$ gibt. Die Teilmengen werden über Bitfelder generiert. Da die Teilmengen von preprocess der Größe nach angeordnet wurden, werden zunächst die größten Teilmengen getestet. Dieser Greedyansatz kann jedoch bestenfalls im letzten Durchlauf, also bei $k = SetCover(\Omega, \Omega_1, \ldots, \Omega_n)$ einen Vorteil bringen, bei den Aufrufen mit kleinerem k werden alle Teilmengen durchlaufen.

7.3.2 Optimierung des Algorithmus

Die in Algorithmus 5 implementierte Lösung des Mengenüberdeckungsproblems arbeitet relativ naiv. Die Laufzeit des Algorithmus ist im Worst Case exponentiell in der Anzahl der Teilmengen. Dies ist bei der Satellite-Domäne allerdings kein größeres Problem, da die Anzahl der Teilmengen höchstens der Anzahl an Satelliten entspricht, diese ist bei den Benchmarks vergleichsweise klein.
Das Problem an Algorithmus 2 ist, dass er wegen Zeile 4 und 8 garantiert exponentiell in der Anzahl der Instrumente ist. Eine einfache Verbesserung des Algorithmus ist es, nur die Instrumente in Erwägung zu ziehen, die den Modus mindestens eines Bildzieles unterstützen. Entsprechend werden nur Satelliten betrachtet, die mindestens ein solches Instrument an Bord haben. Effektiver wäre es, wenn zusätzlich obere und untere Schranken für die Anzahl der zugeschalteten Satelliten bzw. Instrumente in Zeile 4 und 8 gefunden werden könnten. Dies wird in diesem Abschnitt versucht.

Lemma 7.3
Es seien alle Ziele $\langle d, m\rangle \in O_*$ entfernt, bei denen es ein kalibriertes Instrument gibt, welches m unterstützt und auf einem Satelliten montiert ist, der nach d zeigt. Weiter gäbe es keine Ausrichtungsziele.
Œ *sei $D = \{1, \ldots, k\}$ und $I = \{1, \ldots, n\}$. Seien*

- $\Omega_i = \{m | m \in M, \langle i, m\rangle \in C\}$ *die von Instrument i abgedeckten Modi.*
- $\Omega = \{m \in M \mid \exists j \in D : \langle j, m\rangle \in O_*\}$ *die Modi, in denen Bilder aufgenommen werden müssen.*

Dann gibt es einen optimalen h^+-Plan, in dem für die Anzahl $|inst|$ der insgesamt verwendeten Instrumente gilt,

1. $|inst| \leq |O_*|$.
2. $|inst| \geq SetCover(\Omega, \Omega_1, \ldots, \Omega_n)$.
3. $|inst| < 0,5 \cdot |O_*| + 2,5 \cdot SetCover(\Omega, \Omega_1, \ldots, \Omega_n)$.

Beweis:
Zur Abkürzung definiere $SC(I) := SetCover(\Omega, \Omega_1, \ldots, \Omega_n)$.

zu 1) Da jedes Instrument, welches in einem optimalen Plan verwendet wird, mindestens ein Bild aufnimmt – andernfalls könnte der Plan durch Weglassen des Instruments verkürzt werden – werden höchstens soviele Instrumente, wie es offene Bildziele gibt, verwendet.

zu 2) Um alle nötigen Modi von Bildzielen abzudecken, werden mindestens $SC(I)$ Instrumente benötigt. Somit werden auch in einem optimalen Plan mindestens soviele Instrumente verwendet, da andernfalls eines der Bildziele nicht geschossen werden könnte.

zu 3) Es gibt einen gültigen Plan, der $SC(I)$ Instrumente verwendet. Sei dieser Plan mit h' bezeichnet. Falls man auf diese Weise bereits einen optimalen Plan findet, gilt die Behauptung trivialerweise.
Andernfalls gibt es einen optimalen Plan h^+ mit $|h'| > |h^+|$ und $|h^+(I)| > SC(I)$, wobei $h^+(I)$ die in h^+ verwendeten Instrumente bezeichnet. Wir werden jetzt eine obere Schranke für $|h'|$ und eine untere Schranke für $|h^+|$ suchen, jeweils in Abhängigkeit von der Anzahl der Bildziele und der Anzahl der verwendeten Instrumente. Da wir wissen, dass h^+ kürzer ist, können wir anschließend die beiden Grenzen in Relation setzen und erhalten daraus die gewünschte Abschätzung für die in einem optimalen Plan verwendeten Instrumente.
In h' sind höchstens folgende Aktionen notwendig:

- Für alle Bildziele $\langle d, m\rangle \in O_*$ wird ein Satellit in Richtung d gebracht und das Bild im Modus m mit einem passenden Instrument aufgenommen.
- Für jedes Instrument, das an $SC(I)$ beteiligt ist, wird ein eventuell eingeschaltetes Instrument auf dem Satelliten ausgeschaltet, das Instrument wird eingeschaltet, der Satellit, auf dem es montiert ist, wird in die Kalibrierungsrichtung gebracht, das Instrument wird kalibriert.

Somit gilt:

$$\begin{aligned} |h'| &\leq 2|O_*| + 4SC(I) \\ &= 6SC(I) + 2(|O_*| - SC(I)) \end{aligned}$$

In h^+ sind mindestens folgende Aktionen notwendig:

- Für alle Bildziele $\langle d, m\rangle \in O_*$ wird ein Bild im Modus m aufgenommen.
- Für jedes Instrument aus $h^+(I)$ ist entweder eine Kalibrierungsaktion notwendig, oder der Satellit, auf dem es montiert ist, muss in Richtung eines Bildzieles geschwenkt werden, andernfalls wäre das Ziel nach Voraussetzung aus O_* entfernt worden.

Somit gilt:

$$\begin{aligned} |h^+| &\geq |O_*| + |h^+(I)| \\ &= 2|h^+(I)| + (|O_*| - |h^+(I)|) \\ &\overset{|h^+(I)|>SC(I)}{>} 2|h^+(I)| + (|O_*| - SC(I)) \end{aligned}$$

Wegen $|h^+| < |h'|$ folgt somit $|h^+(I)| < 3SC(I) + \frac{1}{2}(|O_*| - SC(I))$ wie behauptet. □

Bemerkung 7.2
Die untere Schranke für die Anzahl an Instrumenten kann entweder außerhalb der for-Schleife in Zeile 4 berechnet werden, oder innerhalb. Wird sie innerhalb berechnet, d.h. werden zur Überdeckung der Bildziele nur Instrumente auf den gerade verwendeten Satelliten herangezogen, liefert sie eine schärfere Schranke. Dadurch müssen weniger Teilmengen in Zeile 8 geprüft werden, allerdings kann sich die Laufzeit dennoch erhöhen, weil für jede Teilmenge von Satelliten ein Mengenüberdeckungsproblem gelöst werden muss. Ob sich der zusätzliche Aufwand lohnt, ist durch Experimente zu prüfen.

Lemma 7.4
Gelten die gleichen Voraussetzungen und seien Ω und $inst$ wie in Lemma 7.3. Für einen Satelliten $j \in S = \{1, \ldots, n\}$ sei

$$\Omega'_j = \{m \in M \mid \exists i \in I : \langle i, m\rangle \in C \wedge l(i) = j\}$$

die Menge der Modi, die von seinen Instrumenten abgedeckt werden.
Dann gibt es einen optimalen h^+-Plan, in dem für die Anzahl $|sat|$ an verwendeten Satelliten gilt:

1. $|sat| \leq |inst|$
2. $|sat| \geq SetCover(\Omega, \Omega'_1, \ldots, \Omega'_n)$

Beweis:

zu 1) Dies ist klar, da jeder Satellit, der benutzt wird, mindestens ein Instrument besitzt, müssen höchstens so viele Satelliten wie Instrumente verwendet werden.

zu 2) In jedem gültigen Plan müssen $|O_*|$ Fotos geschossen werden. Dafür müssen mindestens $SetCover(\Omega, \Omega'_1, \ldots, \Omega'_n)$ Satelliten verwendet werden. Da dies insbesondere für jeden optimalen Plan gilt, folgt die Behauptung.

□

Es sind somit Abschätzungen sowohl für die Anzahl der Satelliten, als auch für die Anzahl der Instrumente gefunden. Allerdings müssen im Worst Case dennoch mehr als die Hälfte aller Teilmengen von Instrumenten durchgegangen werden. Für $|O_*| \geq \max(2, |I|)$ ist die bei $SetCover(\Omega, \Omega_1, \ldots, \Omega_m) =$

$|I|/2.5$ in Lemma 7.3 gefundene untere Schranke für die Anzahl an Instrumenten $|I|/2.5$, die obere Schranke ist $|I|$. Wegen $\binom{n}{k} = \binom{n}{(n-k)}$ gilt für die Anzahl der überprüften Instrumentteilmengen:

$$\begin{aligned}\sum_{k=\frac{1}{2.5}|I|}^{|I|} \binom{|I|}{k} &> \sum_{k=\frac{1}{2}|I|}^{|I|} \binom{|I|}{k} \\ &= \frac{1}{2}\sum_{k=1}^{|I|} \binom{|I|}{k} \\ &= 2^{|I|-1}\end{aligned}$$

Bisher wurden A-Priori-Abschätzungen gefunden, sie werden gemacht, bevor man einen gültigen Plan gefunden hat. Nachdem in Algorithmus 2 die Schleife einmal durchlaufen wurde, hat man von einem gültigen Plan die Länge bestimmt. Dieses Wissen kann genutzt werden, um die Anzahl an Instrumenten einzuschränken. Wenn ein Plan kürzer ist, können nicht beliebig viele Instrumente eingeschaltet sein.

Lemma 7.5
Wir bezeichnen die Kosten, die in Abhängigkeit von den gewählten Satelliten und Instrumenten auftreten, als variable Kosten, d.h. die Kosten, die im Algorithmus zwischen Zeile 4 und Zeile 19 berechnet werden. Sei die Schleife aus Zeile 8 bereits mindestens einmal durchlaufen worden und seien die bisherigen minimalen variablen Kosten $v_{\min}$.
Beachte, dass in einem vom Startzustand erreichbaren Zustand einer nichtrelaxierten Instanz pro Satellit höchstens ein Instrument eingeschaltet sein kann. Definiere für einen Satelliten $s \in S$

$$P(s) = \begin{cases} 0, & \textit{ein Instrument ist eingeschaltet und kalibriert} \\ 1, & \textit{ein Instrument ist eingeschaltet, aber nicht kalibriert} \\ 2, & \textit{kein Instrument ist eingeschaltet} \end{cases}$$

Seien in Zeile 4 die Satelliten $s_1, \ldots, s_n$ ausgewählt. Von den Ausrichtungen, in denen die Bildziele aufgenommen werden sollen, werden m nicht von den Satelliten abgedeckt. Wenn ein Plan geringere variable Kosten als $v_{\min}$ hat, gilt für die Anzahl $|inst|$ ausgewählter Instrumente in Zeile 8

$$|inst| < n + (v_{\min} - \sum_{i=1}^{n} P(s_i) - m)/2.$$

Beweis:
Bei der Verwendung von insgesamt $|inst|$ Instrumenten auf den Satelliten $s_1, \ldots, s_n$, wobei pro Satellit mindestens ein Instrument verwendet wird,

können die Kosten wie folgt abgeschätzt werden. Pro Satellit s_i wird mindestens ein Instrument eingeschaltet und kalibriert, dafür sind Kosten von mindestens $P(s_i)$ nötig (es werden die Kosten ignoriert, die eventuell für das Drehen in Kalibrierungsausrichtung nötig sind). Die restlichen $|inst| - n$ Instrumente müssen eingeschaltet und kalibriert werden, was Kosten von mindestens 2 verursacht. Außerdem sind noch mindestens m Ausrichtungen nötig, um die Zielbilder aus dem richtigen Winkel aufnehmen zu können. Somit gilt für die variablen Kosten

$$v \geq \sum_{i=1}^{n} P(s_i) + 2(|inst| - n) + m,$$

was der Behauptung entspricht. □

Da für die A-Priori-Abschätzungen Mengenüberdeckungsinstanzen für die Instrumente gelöst werden müssen, kann sich eine Verbesserung des SetCover-Algorithmus auszahlen. Dafür bietet sich eine Binärsuche an, wie sie in Algorithmus 6 dargestellt wird.

Satz 7.3
Algorithmus 6 liefert eine minimale Zahl k, so dass eine Überdeckung der Größe k von Ω durch die Mengen $\Omega_1, \ldots, \Omega_n$ existiert. Falls keine solche Überdeckung existiert, liefert der Algorithmus ∞.

Beweis:
Dass der Algorithmus ∞ zurückliefert, falls keine Überdeckung existiert, ist klar. Existiere also eine Mengenüberdeckung für die Instanz.
Zeige folgende Invariante für die while-Schleife in Zeile 8: Es existiert eine Mengenüberdeckung der Größe max und es existiert keine Mengenüberdeckung der Größe $min - 1$ (d.h. eine minimale Überdeckung hat mindestens die Größe min und höchstens die Größe max).
Die Invariante gilt vor Eintritt in die Schleife. Es existiert eine Überdeckung, diese verwendet höchstens alle n Teilmengen. Da laut *preprocess*-Algorithmus in Zeile 5 jede Teilmenge mindestens ein Element enthält, sind höchstens $|\Omega|$ Teilmengen für eine Überdeckung nötig. Der lower_bound-Algorithmus in Zeile 6 liefert die minimale Zahl min, so dass min Teilmengen zusammen mindestens so viele (nicht notwendig verschiedene) Elemente enthalten wie Ω. Da $min - 1$ Teilmengen somit echt weniger Elemente enthalten als Ω, kann es keine Mengenüberdeckung der Größe $min - 1$ geben.
Die Invariante gilt nach einem Durchlauf der while-Schleife. Falls eine Mengenüberdeckung der Größe $middle$ existiert, wird max auf den Wert $middle$ gesetzt. Somit gilt der erste Teil der Invariante für das neue max, da min nicht verändert wird, gilt auch der zweite Teil der Invariante. Falls keine Mengenüberdeckung der Größe $middle$ existiert, wird min auf $middle + 1$

Algorithm 6 $\text{SetCover}_{\log}$

Input: Menge Ω, Teilmengen $\Omega_1, \ldots, \Omega_n$

Output: Eine minimale Zahl k, so dass $\bigcup_{j=1}^{k} \Omega_{i_j} = \Omega$ oder ∞, falls nicht möglich

1: Schneide alle Ω_i mit Ω, falls nötig.
2: **if** $\bigcup_{i=1}^{n} \Omega_i \neq \Omega$ **then**
3: **return** ∞
4: **end if**
5: preprocess$(\Omega, \Omega_1, \ldots, \Omega_n)$
6: min = lower_bound$(\Omega, \Omega_1, \ldots, \Omega_n)$
7: max = min$(|\Omega|, n)$
8: **while** max-min>1 **do**
9: middle = $\lfloor \frac{max+min}{2} \rfloor$
10: **if** $SetCover_{middle}(\Omega, \Omega_1, \ldots, \Omega_n)$ **then**
11: max = middle
12: **else**
13: min = middle + 1
14: **end if**
15: **end while**
16: **if** $(max \neq min)$ **and not** $SetCover_{min}(\Omega, \Omega_1, \ldots, \Omega_n)$ **then**
17: k = max
18: **else**
19: k = min
20: **end if**
21: **return** k

gesetzt. Somit gilt der zweite Teil der Invariante für das neue min, da max nicht verändert wird, gilt auch der erste Teil.
Die while-Schleife bricht nach endlich vielen Schritten ab. Es gilt $min < \lfloor \frac{max+min}{2} \rfloor < max$ für alle $min, max \in \mathbb{N}, max > min + 1$. Also erhöht sich nach jedem Durchlauf der while-Schleife entweder min, oder es verringert sich max. Damit wird die Schleifenbedingung $max - min > 1$ nach endlich vielen Schritten verletzt.
Vor Eintritt in die while-Schleife gilt $min \leq max$, da min eine untere Schätzung für die Größe der Mengenüberdeckung ist. Nach Durchlauf der while-Schleife gilt $max = min$ oder $max = min + 1$ und die Schleifeninvariante. Falls $max = min$, hat wegen der Schleifeninvariante die minimale Mengenüberdeckung die Größe min. Falls $max = min + 1$, hat die minimale Mengenüberdeckung wegen der Schleifeninvariante entweder die Größe min oder die Größe max. Auch in diesem Fall liefert der Algorithmus die richtige Größe. □

Algorithmus 6 durchläuft sowohl im Best-Case als auch im Worst-Case ca. $\log_2 n$ verschiedene Werte für die Größe der Mengenüberdeckung. Somit ist er in Fällen, in denen es eine kleine Überdeckung gibt, schlechter als der naive Algorithmus 5. In Fällen, in denen viele Mengen für eine Mengenüberdeckung nötig sind, ist er schneller. Auch hier muss experimentell geprüft werden, ob sich der Einsatz von Algortihmus 6 gegenüber Algorithmus 5 auszahlt.

7.4 Experimentelle Ergebnisse

Die h^+-Heuristik wurde mit den im letzten Abschnitt vorgestellten Verbesserungen implementiert. Jede der Optimierungen kann unabhängig von den anderen ein- und ausgeschaltet werden, um die Effizenz zu überprüfen. Die Experimente wurden auf einem Computer mit acht Intel Xeon Prozessoren mit einer Taktfrequenz von 2.66 GHZ durchgeführt. Falls nach 30 Minuten kein optimaler Plan gefunden wurde oder der Speicherbedarf 2 Gigabyte überschritten hat, wurde die Suche abgebrochen.
Bei Verzicht auf die Optimierung aus Satz 7.5, die die Instrumentenanzahl anhand der bisher gefundenen Pläne einschränkt, konnten die Probleme 1-7 und 10 des IPC3 Wettbewerbs gelöst werden. Für die restlichen Probleme konnte innerhalb der 30 Minuten keine Lösung gefunden werden. Folgende sechs Optimierungen wurden in verschiedenen Kombinationen getestet:

a: Verwende nur Instrumente, die mindestens einen Zielmodus unterstützen.

b: Verwende Grenzen für die Instrumente ausserhalb der Schleife.

keine Optimierung	a	b	c	d	f	a, b
1400.49	918.06	838.80	1410.12	1402.36	530.76	714.41

b, d	b, c, d	a, b, c, d	a, f	a, e, f	a, b, e, f
839.28	851.30	729.54	470.40	471.29	**470.14**

Abbildung 7.2: Aufsummierte Lösungsdauer des A*-Algorithmus mittels der h^+-Heuristik auf den acht gelösten Problemen

c: Verwende Grenzen für die Instrumente innerhalb der Schleife.

d: Verwende obere und untere Grenzen für die Satelliten.

e: Verwende binäre Suche zum Lösen der Mengenüberdeckungsinstanzen.

f: Verwende anhand der bisher kürzesten Planlänge abgeschätzte Instrumentengrenzen.

Eine Übersicht über die Gesamtlösungsdauer dieser acht Probleme bei verschiedenen Kombinationen der Optimierungen findet sich in Abbildung 7.2. Da aus Zeitgründen nicht alle Kombinationen getestet werden konnten, wurde eine Auswahl getroffen. Optimierung e wurde nicht alleine getestet, da beim Standardalgorithmus dadurch keine nennenswerte Verbesserung zu erwarten ist, dafür sind die darin zu lösenden Mengenüberdeckungsprobleme zu klein. Neben dem Testen einzelner Optimierungen wurden danach Kombinationen der ersten vier Optimierungen getestet. Da durch Hinzunahme von c und d keine Laufzeitverbesserung erzielt werden konnte – siehe Unterschied von a, b zu a, b, c, d in Abbildung 7.2 – wurden die Optimierungen e und f nur mit a und b getestet.
Durch alleiniges Zuschalten von f kann die Laufzeit des ursprünglichen Algorithmus mehr als halbiert werden. Außerdem wird dadurch zusätzlich Problem 9 gelöst. Keine der getesteten Kombinationen der ersten vier Optimierungen war dazu imstande. Die gute Laufzeit der Optimierung f konnte durch Hinzunahme der Optimierungen a, b und e noch gesteigert werden. Deswegen wird für die weiteren Vergleiche mit dieser optimierten Version gearbeitet. Einen Vergleich mit der Merge-And-Shrink-Heuristik findet sich in den Abbildungen 7.3, 7.4 und 7.5. Die Merge-And-Shrink-Heuristk wurde dabei mit einem optimierten Parameter – $N = 10000$, 3 Abstraktionen – aufgerufen [HHH07]. Probleme die nicht gelistet sind, wurden von keiner der beiden Heuristiken gelöst. Falls die h^+-Heuristik ein Problem nicht lösen konnte, lag dies ausnahmslos an der Rechendauer. Bei der Merge-And-Shrink-Heuristik ging bei den Problemen 23-36 die Zeit aus, bei den Problemen 7-22 der Speicher. Die h^+-Heuristik schlägt die Merge-And-Shrink-Heuristik bei sämtlichen ausser den drei einfachsten Problemen in allen drei

Problemname	optimale Planlänge	h^+-Heuristik	Merge-And-Shrink-Heuristik, 3 Abstraktionen, N=10000
Problem 01	9	8	**9**
Problem 02	13	12	**13**
Problem 03	11	10	**11**
Problem 04	17	17	17
Problem 05	15	**14**	12
Problem 06	20	**18**	16
Problem 07	21	**20**	-
Problem 08	-	-	-
Problem 09	27	**26**	-
Problem 10	29	**29**	-

Abbildung 7.3: Vergleich der Bewertung des Anfangszustands der h^+- und der Merge-And-Shrink-Heuristik

Problemname	optimale Planlänge	h^+-Heuristik	Merge-And-Shrink-Heuristik, 3 Abstraktionen, N=10000
Problem 01	9	0.00	0.00
Problem 02	13	**0.01**	0.09
Problem 03	11	**0.05**	4.01
Problem 04	17	**0.11**	7.57
Problem 05	15	**3.40**	44.66
Problem 06	20	**11.07**	48.73
Problem 07	21	**174.58**	-
Problem 08	-	-	-
Problem 09	27	**949.02**	-
Problem 10	29	**280.92**	-

Abbildung 7.4: Vergleich der Laufzeit der h^+- und der Merge-And-Shrink-Heuristik

Problemname	optimale Planlänge	h^+-Heuristik	Merge-And-Shrink-Heuristik, 3 Abstraktionen, N=10000
Problem 01	9	10	10
Problem 02	13	14	14
Problem 03	11	21	**12**
Problem 04	17	**26**	237
Problem 05	15	**34**	38598
Problem 06	20	**526**	375938
Problem 07	21	**812**	-
Problem 08	-	-	-
Problem 09	27	**264**	-
Problem 10	29	**40**	-

Abbildung 7.5: Vergleich der expandierten Knoten der h^+- und der Merge-And-Shrink-Heuristik

Kriterien. Weiterhin löst sie 3 Probleme, die von der Merge-And-Shrink-Heuristik innerhalb der Zeit-/Speichergrenze nicht gelöst werden konnten. Schließlich ist in Abbildung 7.6 noch eine Übersicht der gelösten Probleme mit Anfangswertschätzung, Anzahl der expandierten Knoten und der Zeit zum Finden des optimalen Plans zu sehen. Wie zu erkennen ist, liegt das gute Abschneiden der h^+-Heuristik im Vergleich zur Merge-And-Shrink-Heuristik hauptsächlich an der guten Anfangswertschätzung. Die Berechnung der einzelnen Knoten ist, wie man besonders deutlich an Problem 09 sieht, auch in der optimierten Version zeitaufwendig.

7.5 Ein verändertes Satellite-Problem

In den bisherigen Abschnitten haben wir gesehen, dass das Finden eines optimalen h^+-Planes für eine Satellite-Instanz NP-äquivalent ist, wir haben einen Algorithmus gefunden, der das Problem löst, haben ihn verbessert und die Ergebnisse experimentell ausgewertet. Wir haben bisher nicht erwähnt, dass auch das Finden eines optimalen Planes für eine nicht-relaxierte Satellite-Instanz NP-äquivalent ist. Dies gilt jedoch, Malte Helmert [Hel08] hat sogar das stärkere Ergebnis bewiesen, dass es kein polynomielles Approximationsschema gibt, falls $P \neq NP$.
Im folgenden Abschnitt soll gezeigt werden, dass die Satellite-Domäne derart abgeändert werden kann, dass das Finden eines optimalen Planes darin trivial wird. Für die relaxierte Variante des Längen-Problems wird gelten, dass sie immer noch NP-vollständig ist. Somit haben wir ein Beispiel, das zeigt, dass entgegen der Anschauung das Relaxieren eine Domäne nicht immer vereinfacht.

Problemname	optimale Planlänge	Anfangswert	expandierte Knoten	Zeit
Problem 01	9	8	10	0.00
Problem 02	13	12	14	0.02
Problem 03	11	10	21	0.07
Problem 04	17	17	26	0.15
Problem 05	15	14	34	5.02
Problem 06	20	18	526	16.23
Problem 07	21	20	812	250.37
Problem 08	-	-	-	-
Problem 09	27	26	264	1350.72
Problem 10	29	29	40	401.41

Abbildung 7.6: Übersicht über die gelösten Probleme der h^+-Heuristik

Wir erhalten das veränderte Problem, indem wir die verschiedenen Ausrichtungen entfernen und nur noch mit unterschiedlichen Modi arbeiten. Außerdem lassen wir die Satelliten nach der Aufnahme eines Bildes automatisch ausschalten und verzichten auf das Kalibrieren. Wir können uns das Problem vorstellen als das Aufnehmen eines einzigen Himmelskörpers in unterschiedlichen Modi mithilfe von Satelliten, deren Instrumente sich nach einer Bildaufnahme ausschalten, um sich wieder aufzuladen.

Definition 7.4 (vereinfachte Satellite-Problemstellung)
Eine vereinfachte Satellite-Problemstellung ist gegeben durch ein 6-Tupel $\langle S, I, M, l, C, O_* \rangle$*, bestehend aus folgenden Komponenten:*

- S *ist eine endliche Menge von Satelliten.*
- I *ist eine endliche Menge von Instrumenten.*
- M *ist eine endliche Menge von Bildmodi oder kurz Modi.*
- $l : I \to S$ *bildet jedes Instrument auf den Satelliten ab, auf dem es montiert ist.*
- $C \subseteq I \times M$ *ist die binäre Relation der Instrumentenfähigkeiten. Wenn* $\langle i, m \rangle \in C$ *für ein Instrument* i *und einen Bildmodus* m*, sagen wir,* i *unterstützt* m*.*
- $O_* \subseteq M$ *ist die Menge der Bildziele.*

Wir definieren als Nächstes die nicht-relaxierte und die relaxierte Domäne, um das gewünschte Problem zu erhalten.

Definition 7.5 (vereinfachte Satellite-Domäne)
Die vereinfachte Satellite-Domäne bildet vereinfachte Satellite-Problemstellungen mit Satelliten S, Instrumenten I und Bildmodi M wie folgt auf Zustandsräume ab:

Zustände: *Dies sind 3-Tupel $\langle S^P, I^P, O \rangle$, wobei $S^P \subseteq S$ die Menge der Satelliten mit verfügbarem Strom, $I^P \subseteq I$ die Menge der eingeschalteten Instrumente und $O \subseteq M$ die Menge der offenen Bildziele ist.*

Anfangszustand: *Der Anfangszustand ist das 3-Tupel $\langle S, \emptyset, O_* \rangle$, wobei O_* die Menge der Bildziele der Problemstellung ist.*

Zielzustände: *Genau die Zustände $\langle S^P, I^P, \emptyset \rangle$ sind Zielzustände.*

Operatoren: *Ein Instrument $i \in I$ kann genau dann eingeschaltet werden, wenn der Satellit, auf dem es montiert ist, über Strom verfügt. Im entstehenden Zustand wird i zur Menge der eingeschalteten Instrumente hinzugefügt und der Satellit aus der Menge der Satelliten mit verfügbarem Strom entfernt.*
Ein Instrument $i \in I$ kann genau dann ausgeschaltet werden, wenn es eingeschaltet ist. Im entstehenden Zustand wird i aus der Menge der eingeschalteten Instrumente entfernt und der Satellit zur Menge der Satelliten mit verfügbarem Strom hinzugefügt.
Ein Instrument $i \in I$ kann genau dann ein Bild im Modus $m \in M$ aufnehmen, wenn es eingeschaltet ist und den Modus m unterstützt. Im entstandenen Zustand wird das Instrument aus der Menge der eingeschalteten Instrumente, sowie m aus der Menge der offenen Bildziele entfernt, falls es ein solches war. Der Satellit, auf dem das Instrument montiert ist, wird zur Menge der Satelliten mit verfügbarem Strom hinzugefügt.

Definition 7.6 (relaxierte vereinfachte Satellite-Domäne)
Die relaxierte vereinfachte Satellite-Domäne bildet einen Zustand $\left\langle \hat{S^P}, \hat{I^P}, \hat{O} \right\rangle$ einer (nicht-relaxierten) vereinfachten Satellite-Instanz auf Zustandsräume ab. Die relaxierten Operatoren sind die folgenden:

Operatoren: *Ein Instrument $i \in I$ kann genau dann eingeschaltet werden, wenn der Satellit auf dem es montiert ist, in der Menge der Satelliten mit verfügbarem Strom enthalten ist. Im entstehenden Zustand wird i zur Menge der eingeschalteten Instrumente hinzugefügt. Der Satellit auf dem es montiert ist, wird **nicht** aus der Menge der Satelliten mit verfügbarem Strom entfernt.*
Ein Instrument $i \in I$ kann genau dann ausgeschaltet werden, wenn es eingeschaltet ist. Im entstehenden Zustand wird der Satellit, auf dem es montiert ist, zur Menge der Satelliten mit verfügbarem Strom

hinzugefügt.
Ein Instrument $i \in I$ kann genau dann ein Bild im Modus $m \in M$ aufnehmen, wenn es eingeschaltet ist und den Modus m unterstützt. Im entstandenen Zustand wird der Satellit, auf dem es montiert ist, zur Menge der Satelliten mit verfügbarem Strom hinzugefügt, sowie m aus der Menge der offenen Bildziele entfernt, falls es ein solches war. Das Instrument wird ***nicht*** *aus der Menge der eingeschalteten Instrumente entfernt.*

Für eine vereinfachte Instanz ist das Finden eines optimalen Plans sehr einfach. Da jedes Instrument nach dem Schießen eines Bildes wieder ausgeschaltet wird und die Instrumente nicht ausgerichtet werden müssen, muss für jedes Bildziel genau ein Instrument eingeschaltet und das Bild geschossen werden. Dies führt zu einer trivialen Lösung, was der folgende Satz noch einmal zusammenfasst.

Satz 7.4
Das Finden eines optimalen Plans für eine vereinfachte Satellite-Instanz ist in Zeit $O(|M| \cdot |S| \cdot |I|)$ möglich.

Beweis:
Betrachte hierzu Algorithmus 7. Wenn das Problem lösbar ist, so hat der optimale Plan die Länge $2|O_*|$. Der Algorithmus findet in Zeit $O(|O_*| \cdot |S| \cdot |I|) = O(|M| \cdot |S| \cdot |I|)$ heraus, ob das Problem lösbar ist und liefert im Falle der Lösbarkeit einen optimalen Plan. □

Algorithm 7 SatelliteEasySolver

```
for all m:O_* do
  if exists s:S, i:I with l(i)=s, (i,m):C then
    switch_on(i, s)
    take_image(s, i, m)
  else
    print „Das Problem hat leider keine Lösung!“
    return error
  end if
end for
```

Die Vereinfachung wirkt sich jedoch nicht auf die Komplexität der Relaxierung aus, die nach wie vor schwierig ist.

Satz 7.5
Das relaxierte vereinfachte Satellite-Längen-Problem ist NP-schwer.

Beweis:
In einer relaxierten vereinfachten Satellite-Instanz bleiben die Instrumente nach dem Schießen eines Bildes eingeschaltet. Im Beweis zu Satz 7.1 haben wir nicht von den verschiedenen Ausrichtungen der Satelliten Gebrauch gemacht. Somit können wir ihn direkt auf das vereinfachte Problem übertragen, es müssen immer noch genau die Instrumente einer Mengenüberdeckung (einmal) eingeschaltet werden. □

Kapitel 8

Logistics

Die in diesem Kapitel vorgestellte Logistics-Domäne ist wie Miconic und Gripper ein Spezialfall des allgemeinen Transportproblems. Deswegen sind die Problemdefinitionen im ersten Abschnitt Spezialisierungen der Definitionen des allgemeinen Transportproblems aus der Doktorarbeit von Malte Helmert [Hel08]. Danach folgt der Abschnitt über die Berechnung der h^+-Heuristik für Logistics, da dies NP-äquivalent ist wird ein exponentieller Ansatz vorgestellt, inklusive Hinweisen zur Implementierung und möglichen Verbesserungen. Außerdem wird eine in polynomieller Zeit berechenbare Approximation für die h^+-Heuristik gegeben. Den Abschluss dieses letzten Domänen-Kapitels bildet der Abschnitt über den Vergleich der h^+-Heuristik mit der Merge-And-Shrink-Heuristik.

8.1 Definition des Problems

Es sollen Güter zwischen Orten in unterschiedlichen Städten transportiert werden. Zum Transport innerhalb der Städte stehen Fahrzeuge zur Verfügung, innerhalb einer Stadt ist jeder Ort mit jedem anderen verbunden. Zum Transport zwischen den Städten stehen Flugzeuge zur Verfügung, jede Stadt ist mit jeder anderen verbunden.

Definition 8.1 (Logistics-Problemstellung)
Eine Logistics-Problemstellung ist gegeben durch ein 8-Tupel $\langle C, (L_c)_{c \in C}, A, T, P, G, l_0, l_* \rangle$*, bestehend aus folgenden Komponenten:*

- C *ist eine endliche Menge von Städten.*
- L_c *ist für jede Stadt* $c \in C$ *die endliche Menge seiner Orte, die Orte der Städte überschneiden sich nicht, d.h. für* $c, c' \in C$*, mit* $c \neq c'$*, gilt* $L_c \cap L_{c'} = \emptyset$*. Für die Menge aller Orte schreiben wir auch* $L := \bigcup_{c \in C} L_c$*.*

- *A ist eine endliche Menge von Flughäfen. Jede Stadt hat genau einen Flughafen, dieser ist einer der Orte der Stadt, d.h. es gibt für jede Stadt $c \in C$ genau einen Ort $a_c \in L$, so dass $A \cap L_c = \{a_c\}$. Der Ort a_c wird Flughafen von Stadt c genannt.*
- *T ist eine endliche Menge von Fahrzeugen.*
- *P ist eine endliche Menge von Flugzeugen.*
- *G ist eine endliche Menge von Gütern.*
- *$l_0 : (T \cup P \cup G) \to L$ ist die Anfangsortfunktion. Für ein Flugzeug $p \in P$ ist $l_0(p) \in A$.*
- *$l_* : G' \to L$ mit $G' \subseteq G$ ist die Zielortfunktion. Ein Paar $\langle g, l \rangle$ mit $l_*(g) = l$ wird Güterziel genannt.*

Definition 8.2 (Logistics-Domäne)
Die Logistics-Domäne bildet Logistics-Problemstellungen mit Städten C, Orten $(L_C)_{c \in C}$, Fahrzeugen T, Flugzeugen P und Gütern G wie folgt auf Zustandsräume ab:

Zustände: *Dies sind 3-Tupel $\langle l_T, l_P, l_G \rangle$, wobei $l_T : T \to L$ die Fahrzeugsfunktion, $l_P : P \to A$ die Flugzeugsfunktion und $l_G : G \to L \cup T \cup P$ die Güterfunktion ist. Wenn $l_T(t) = l$ sagen wir, dass sich Fahrzeug $t \in T$ an Ort $l \in L$ befindet, wir nennen $l_T(t)$ den Standort von Fahrzeug t. Wenn $l_P(p) = a$ sagen wir, dass sich Flugzeug $p \in P$ an Flughafen $a \in A$ befindet, wir nennen $l_P(p)$ den Standort von Flugzeug p. Wenn $l_G(g) = l$ für ein Gut $g \in G$ und einen Ort $l \in L$ gilt, sagen wir, dass sich Gut g an l befindet, wir nennen $l_G(g)$ den Standort von Gut g. Für ein Fahrzeug $t \in T$ und $l_G(g) = t$ sagen wir, dass sich Gut g in Fahrzeug t befindet, für ein Flugzeug $p \in P$ und $l_G(g) = p$ sagen wir, dass sich Gut g in Flugzeug p befindet.*

Anfangszustand: *Der Anfangszustand ist das 3-Tupel $\langle l_0|_T, l_0|_P, l_0|_G \rangle$, wobei l_0 die Anfangsortfunktion der Problemstellung ist.*

Zielzustände: *Genau die Zustände $\langle l_T, l_P, l_G \rangle$ mit $l_G(g) = l$ für alle Güterziele $\langle g, l \rangle$ sind Zielzustände.*

Operatoren: *Ein Fahrzeug $t \in T$ kann genau dann zu einem Ort $l \in L_c$ fahren, wenn es sich an einem Ort der Stadt c befindet. Im entstehenden Zustand befindet es sich an l.*
Ein Flugzeug $p \in P$ kann zu einem Flughafen $a \in A$ fliegen. Im entstehenden Zustand befindet es sich an a.
Ein Fahrzeug $t \in T$ kann genau dann ein Gut $g \in G$ aufnehmen, wenn

sich das Fahrzeug und das Gut am gleichen Ort befinden. Im entstehenden Zustand befindet sich das Gut g im Fahrzeug t.
Ein Fahrzeug $t \in T$ kann genau dann ein Gut $g \in G$ an einem Ort $l \in L$ abladen, wenn sich das Fahrzeug t an Ort l befindet und das Gut g im Fahrzeug t. Im entstehenden Zustand befindet sich das Gut g an l.
Ein Flugzeug $p \in P$ kann genau dann ein Gut $g \in G$ aufnehmen, wenn sich das Flugzeug und das Gut am gleichen Ort befinden. Im entstehenden Zustand befindet sich das Gut g im Flugzeug p.
Ein Flugzeug $p \in P$ kann genau dann ein Gut $g \in G$ an einem Flughafen $a \in A$ abladen, wenn sich das Flugzeug p an Flughafen a befindet und das Gut g im Flugzeug p. Im entstehenden Zustand befindet sich das Gut g an a.

Definition 8.3 (relaxierte Logistics-Domäne)
Die relaxierte Logistics-Domäne bildet einen Zustand $\left\langle \hat{l_T}, \hat{l_P}, \hat{l_G} \right\rangle$ einer (nicht-relaxierten) Logistics-Instanz auf Zustandsräume ab. Die neuen Zustände sind 3-Tupel $\langle l_T, l_P, l_G \rangle$, dabei bilden $l_T : T \to 2^L, l_P : P \to 2^A$, und $l_G : G \to 2^{L \cup T \cup P}$ auf Teilmengen ab, die Fahrzeuge, Flugzeuge und Güter können sich an mehreren Orten gleichzeitig befinden.

Operatoren: *Ein Fahrzeug $t \in T$ kann genau dann zu einem Ort $l \in L_c$ fahren, wenn es sich an einem Ort der Stadt c befindet, d.h. $l_T(t) \subseteq L_c$. Im entstehenden Zustand befindet es sich an l* **und** *an seinen bisherigen Orten.*
Ein Flugzeug $p \in P$ kann zu einem Flughafen $a \in A$ fliegen. Im entstehenden Zustand befindet es sich an a **und** *an seinen bisherigen Flughäfen.*
Ein Fahrzeug $t \in T$ kann genau dann ein Gut $g \in G$ aufnehmen, wenn sich das Fahrzeug und das Gut am gleichen Ort befinden, d.h. $l_T(t) \cap l_G(g) \neq \emptyset$. Im entstehenden Zustand befindet sich das Gut g im Fahrzeug t **und** *an seinen bisherigen Positionen.*
Ein Fahrzeug $t \in T$ kann genau dann ein Gut $g \in G$ an einem Ort $l \in L$ abladen, wenn sich das Fahrzeug t an Ort l befindet und das Gut g im Fahrzeug t. Im entstehenden Zustand befindet sich das Gut g an l **und** *an seinen bisherigen Positionen.*
Ein Flugzeug $p \in P$ kann genau dann ein Gut $g \in G$ aufnehmen, wenn sich das Flugzeug und das Gut am gleichen Ort befinden, d.h. $l_P(p) \cap l_G(g) \neq \emptyset$. Im entstehenden Zustand befindet sich das Gut g im Flugzeug p **und** *an seinen bisherigen Positionen.*
Ein Flugzeug $p \in P$ kann genau dann ein Gut $g \in G$ an einem Flughafen $a \in A$ abladen, wenn sich das Flugzeug p an Flughafen a befindet und das Gut g im Flugzeug p. Im entstehenden Zustand befindet sich das Gut g an a **und** *an seinen bisherigen Positionen.*

8.2 Berechnung der h^+-Heuristik

Genau wie bei Satellite bleibt auch bei Logistics die relaxierte Version des NP-vollständigen [Hel08] Längen-Problems schwierig. Bis auf wenige Spezialfälle kann man auf keinen polynomiellen Algorithmus hoffen. Deswegen wird in diesem Abschnitt, neben einem exponentiellen Ansatz für den allgemeinen Fall, eine polynomiell berechenbare Approximation vorgestellt.

8.2.1 Bestimmung des Schwierigkeitsgrades

Korollar 8.1
Das Finden eines optimalen h^+-Planes für eine Logistics-Instanz mit keinem Flugzeug, einer Stadt c und einem Fahrzeug t, ist in Zeit $O(|L|+|G|)$ möglich. Für $g \in G$ und $l \in L$ sei $\text{important}(g) = \exists l \in L : l_*(g) = l \wedge (l_*(g) \neq l_G(g))$ *und* $\text{important}(g, l) = \text{important}(g) \wedge (l_*(g) = l \vee l_G(g) = l)$.

- *Seien $\mathcal{O} = \{l \in L \setminus l_T(t) \mid \exists g \in G : \text{important}(g, l))\}$ die Orte ohne dem Standort des Fahrzeugs, an denen sich zu transportierende Güter befinden oder zu denen Güter transportiert werden müssen.*
- *Seien $\mathcal{G}' = \{g \in G \mid \text{important}(g) \wedge l_G(g) \neq t\}$ die Güter, die sich weder an ihrem Zielort noch im Fahrzeug befinden.*
- *Seien $\mathcal{G}(t) = \{g \in G \mid \text{important}(g) \wedge l_G(g) = t\}$ die Güter, die sich im Fahrzeug befinden.*

Dann gilt für jeden optimalen Plan h^+

$$|h^+| = \mathcal{L}(\mathit{Logistics}, 1) := |\mathcal{O}| + 2|\mathcal{G}'| + |\mathcal{G}(t)|$$

Beweis:
Dies ist eine direkte Folgerung aus Satz 3.1. Die Logistics-Domäne mit einer Stadt und einem Fahrzeug ist eine Verallgemeinerung der Miconic-Strips-Domäne, bei der die Passagiere (hier Güter) an jedem Ort, nicht nur ihrem Zielort, abgeladen werden können. Für einen optimalen Plan spielt diese Verallgemeinerung keine Rolle, da im optimalen Plan jedes Gut höchstens einmal ein- und ausgeladen wird. Somit kann das Ergebnis der Miconic-Strips-Domäne übernommen werden. □

Offensichtlich kann man ebenso den Fall ein Flugzeug und n Städte mit jeweils dem Flughafen als einzigem Ort, behandeln. Die Erweiterung auf ein Flugzeug und n Städte mit jeweils einem Fahrzeug ist ebenfalls nicht schwierig, wie der folgende Satz zeigt.

Satz 8.1
Die Logistics-Domäne ist für den Fall ein Flugzeug p, n Städte C und höchstens ein Fahrzeug pro Stadt, in Zeit $O(|L| + |G|)$ möglich. Für $g \in G$ und $l \in L$ seien important(g) *und* important(g, l) *wie in Korollar 8.1 definiert. Für eine Stadt $c \in C$ sei t_c das sich in der Stadt befindliche Fahrzeug und*

- $\mathcal{A}_c = \begin{cases} \{a_c\}, & \text{falls } \exists g \in G : ((l_G(g) \in L_c \vee l_G(g) \in t_c) \wedge l_*(g) \notin L_c) \\ & \vee (l_G(g) \notin L_c \wedge l_G(g) \notin t_c \wedge l_*(g) \in L_c) \\ \emptyset, & \text{sonst} \end{cases}$
  der Flughafen der Stadt, falls Güter aus oder in die Stadt transportiert werden müssen und die leere Menge sonst.

- $\mathcal{O}_c = (\{l \in L_c \mid \exists g \in G : \text{important}(g, l))\} \cup \mathcal{A}_c) \setminus l_T(t_c)$ *die Orte der Stadt, an denen sich zu transportierende Güter befinden oder zu denen Güter transportiert werden müssen, zusammen mit dem Flughafen der Stadt, falls Güter aus oder in die Stadt transportiert werden sollen, jedoch ohne dem Standort des Fahrzeugs der Stadt.*

- $\mathcal{G}'_c = \{g \in G \mid \exists l_c \in L_c : \text{important}(g, l_c) \wedge l_G(g) \neq t_c\}$ *die Güter, die von oder zu einem der Orte der Stadt transportiert werden müssen und sich nicht im Fahrzeug der Stadt befinden.*

- $\mathcal{G}(t_c) = \{g \in G \mid \text{important}(g) \wedge l_G(g) = t_c\}$ *die Güter, die sich im Fahrzeug der Stadt befinden.*

Weiter seien

- $\mathcal{H} = \{c \in C \mid \mathcal{A}_c \neq \emptyset\}$ *die Städte, aus oder zu denen Güter transportiert werden müssen.*

- $\mathcal{W} = \{g \in G \mid \text{important}(g) \wedge \text{wrongcity}(g)\}$, *wobei* $\text{wrongcity}(g) = \exists c, c' \in C : c \neq c' \wedge (l_G(g) \in L_c \vee l_G(g) \in t_c) \wedge l_*(g) \in L_{c'}$, *die Güter, die sich nicht in der Stadt ihres Zielortes befinden.*

- $\mathcal{G}(p) = \{g \in G \mid \text{important}(g) \wedge l_G(g) \in p\}$ *die Güter, die sich im Flugzeug befinden.*

Dann gilt für jeden optimalen h^+-Plan h^+

$$|h^+| = \mathcal{L}(Logistics, 2) := |\mathcal{H}| + 2|\mathcal{W}| + |\mathcal{G}(p)| + \sum_{c \in C}(|\mathcal{O}_c| + 2|\mathcal{G}'_c| + |\mathcal{G}(t_c)|$$

Beweis:
$|h^+| \leq \mathcal{L}(\text{Logistics}, 2)$:
Betrachte folgenden Plan für das Problem.

Fahre in jeder Stadt mit dem jeweiligen Fahrzeug alle Start- und Zielorte an. Fliege mit dem Flugzeug alle Start- und Zielflughäfen an.
Lade alle Güter, die transportiert werden müssen, in das Fahrzeug der jeweiligen Stadt ein und an ihrem Zielort aus. Jetzt sind alle Güter, die innerhalb der Stadt transportiert werden müssen, an ihrem Zielort und alle Güter, die einen Zielort außerhalb ihrer Ursprungsstadt haben, sind am Flughafen ihrer Ursprungsstadt.
Lade alle nötigen Güter in das Flugzeug und in der Zielstadt wieder aus. Erledige den Transport der geflogenen Güter innerhalb ihrer Zielstadt.
$|h^+| \geq \mathcal{L}(\text{Logistics}, 2)$:
Jedes Gut, welches von einer Stadt in eine andere transportiert werden muss, muss sich irgendwann am Flughafen der Zielstadt befinden. Es handelt sich bei dieser Eigenschaft um einen Meilenstein, d.h. wenn dieser Effekt aus allen Aktionen entfernt würde, wäre das Problem unlösbar. Zur genauen Erklärung von Meilensteinen siehe den Artikel von Matthias Westphal [RHW08]. Ebenso muss sich das Gut irgendwann am Flughafen der Ursprungsstadt befinden, ein weiterer Meilenstein. Somit hat jeder gültige Plan mindestens die Länge der Summe der optimalen Planlängen für die einzelnen Städte ($\sum_{c \in C} |\mathcal{O}_c| + |\mathcal{G}'_c| + |\mathcal{G}(t_c)|$ Aktionen).
Damit die Güter vom Flughafen der Startstadt zum Flughafen der Zielstadt gelangen, sind mindestens $2|\mathcal{W}|$ Ein- bzw. Ausladeaktionen und $|\mathcal{H}|$ Bewegungsaktionen des Flugzeugs nötig, zum Ausladen der Güter, die sich im Flugzeug befanden, weitere $|\mathcal{G}(p)|$ Aktionen.
Somit hat jeder gültige Plan mindestens die angegebene Länge. □

Wenn es also ein Flugzeug und pro Stadt ein Fahrzeug gibt, ist das Problem einfach lösbar, unabhängig davon, ob die Fahrzeuge und das Flugzeug anfangs beladen sind. Wenn es mehr als ein Fahrzeug gibt, welches den Transport von Gütern übernehmen kann, wird es schwierig, da nicht klar ist, welches Fahrzeug man für den Transport auswählen soll. Wir werden jedoch sehen, dass diese Wahl nicht wichtig ist, falls anfangs alle Fahrzeuge und das Flugzeug unbeladen sind.
Wir brauchen für das weitere Vorgehen den Begriff des Gütertransportgraphen. Dies ist ein Graph bestehend aus den Orten einer Stadt, bei dem gerade die Orte verbunden sind, zwischen denen Transporte erfolgen.

Definition 8.4 (Gütertransportgraph)
Gegeben eine Logistics-Instanz mit keinem Flugzeug, einer Stadt c und nur unbeladenen Fahrzeugen, ist der zugehörige Gütertransportgraph $GTG = (L_c, E)$. Dabei gilt $(l, l') \in E$ genau dann, wenn es ein Gut gibt mit Startort l und Zielort l' oder Startort l' und Zielort l.

Satz 8.2
Das Finden eines optimalen Planes für eine relaxierte Logistics-Instanz mit

n Städten, bei der alle Fahrzeuge und Flugzeuge nicht beladen sind, ist in Zeit $O(|L| + |G| + |T|)$ möglich.

Beweis:
Die Idee zum Satz und Beweis stammt von Malte Helmert.
Wegen der Meilensteinidee aus Satz 8.1 genügt es, den Fall einer Stadt mit nur unbeladenen Fahrzeugen zu betrachten, da sich die Lösung für n Städte aus den Lösungen für die einzelnen Städte zusammensetzt. Das Problem der Flugroute für Flugzeuge ist zum Problem der Route der Fahrzeuge in einer Stadt identisch.
Die Idee für den Beweis ist, dass wir die Orte der Stadt anhand der Transportwege einteilen und dann für die zusammengehörigen Orte ein Fahrzeug auswählen. Das Verwenden mehrerer Fahrzeuge für den Transport innerhalb zusammengehörender Orte bringt keinen Vorteil, wie wir später zeigen werden.
Wir geben einen kürzesten Plan an, bei dem pro Zusammenhangskomponente (ZHK) des Gütertransportgraphen höchstens ein Fahrzeug verwendet wird. Erstelle dazu zunächst den Gütertransportgraphen, dies ist in Zeit $O(|G| + |L|)$ möglich, z.B. mit Tarjans Algorithmus [Tar72]. Weise danach jeder ZHK falls möglich ein Fahrzeug der ZHK zu, andernfalls ein beliebiges. Dies ist in Algorithmus 8 dargestellt. Der benötigte Zeitaufwand ist $O(|T| + |L|)$. Im relaxierten Plan fährt jedes Fahrzeug alle Orte seiner zugewiesenen ZHK an und lädt alle entsprechenden Güter ein und an ihrem Zielort aus.
Dass dies ein gültiger Plan ist, ist klar.
Dass es keinen kürzeren geben kann, sieht man wie folgt ein. Die Anzahl der Ein- und Ausladeaktionen ist offensichtlich optimal. Für die Fahraktionen der Fahrzeuge genügt es gemäß Lemma 8.1 alle anderen Pläne zu betrachten, bei denen pro Zusammenhangskomponente höchstens ein Fahrzeug verwendet wird. Bei allen solchen Plänen muss für jede ZHK, die kein eigenes Fahrzeug enthält, eines aus einer anderen ZHK den Transport übernehmen. Die Fahraktionen innerhalb einer ZHK, die ein Fahrzeug enthält, sind offensichtlich optimal, da jeder Ort der ZHK Güter enthält oder empfängt und somit angefahren werden muss. Der Ort, an welchem sich das Fahrzeug befindet, muss nicht angefahren werden. Welches dieser Ort ist, hängt vom gewählten Fahrzeug ab, beeinflusst jedoch nicht die Anzahl der Fahraktionen. Somit kann es keinen kürzeren Plan geben.
Das Logistics-Problem mit einer Stadt und unbeladenen Fahrzeugen ist somit in Gesamtzeit $O(|L| + |G| + |T|)$ lösbar. □

Wir müssen noch den Nachweis liefern, dass die Verwendung mehrerer Fahrzeuge keinen Vorteil für den Transport bringt.

Algorithm 8 Fahrzeug zu Zusammenhangskomponente

Input: Zusammenhangskomponenten $Z_1, \ldots, Z_n$, Fahrzeuge $t_1, \ldots, t_m$
Output: Zuweisung eines Fahrzeugs an jede Zusammenhangskomponente

```
for i = 1, ..., m do
  if t_i steht auf Ort einer noch nicht zugewiesenen Zusammenhangskomponente then
    weise t_i an seine Zusammenhangskomponente zu
  end if
end for
for i = 1, ..., n do
  if Z_i hat noch kein Fahrzeug zugewiesen then
    weise t_1 an Z_i zu
  end if
end for
```

Lemma 8.1
Zu einem Logistics-Problem mit anfangs leeren Fahrzeugen gibt es einen kürzesten h^+-Plan, bei dem für jede Zusammenhangskomponente des dazugehörigen Gütertransportgraphen höchstens ein Fahrzeug verwendet wird.

Beweis:
Sei ein kürzester Plan h für das Problem gegeben. Betrachte eine Zusammenhangskomponente Z des Gütertransportgraphen, h verwende m Fahrzeuge für Z. Wegen des Optimalitätsprinzips wird in einem optimalen Plan für das Gesamtproblem auch das Teilproblem von Z optimal gelöst.
Zeige, dass es einen Plan gibt, bei dem nur ein Fahrzeug für Z verwendet wird und dessen Kosten die von h nicht übersteigen. Induktion über m.

$m = 1$ *klar*

$m \to m + 1 \quad (m + 1 \geq 2)$

Es werden $m+1$ Fahrzeuge in h benutzt. Dann existiert ein Ort l und Fahrzeuge t und t' mit $t \neq t'$, so dass sich am Ende des relaxierten Planes t und t' an l befinden. Andernfalls wäre entweder der Plan nicht gültig (ein Gut wurde nicht zum Ziel transportiert), oder Z nicht zusammenhängend.
Zeige, dass der Plan nicht länger wird, wenn t die Arbeit von t' übernimmt. Sei t am Ende von h an den Orten $l_1, \ldots, l_n$ und t' an den Orten $l'_1, \ldots, l'_k$. Da sich jedes Fahrzeug anfangs an einem Ort befand, waren dafür mindestens $n + k - 2$ Bewegungsaktionen notwendig. Erstelle einen neuen Plan h', in dem t die Orte $l'_1, \ldots, l'_k$ übernimmt. Da darunter l ist, benötigt t höchstens $k - 1$ zusätzliche Fahraktionen. Also sind in h' für t höchstens $n - 1 + k - 1$ Fahraktionen nötig.
Da beide Fahrzeuge anfangs leer waren, sind alle von Fahrzeug t' trans-

portierten Güter unterwegs ein- und wieder ausgeladen worden. Dies kann t zu den gleichen Kosten durchführen.
Somit wird der Plan nicht teurer, wenn t die Arbeit von t' übernimmt.
$\Rightarrow$ Plan h' ist gültig, verwendet m Fahrzeuge und es gilt $|h'| \leq |h|$
$\overset{IV}{\Rightarrow}$ Plan h'' existiert, der ein Fahrzeug verwendet und für den gilt $|h''| \leq |h'| \leq |h|$

Somit gibt es einen optimalen Plan, der für Z genau ein Fahrzeug verwendet. Beim Übertragen auf das Gesamtproblem können sich mehrere Zusammenhangskomponenten das gleiche Fahrzeug teilen, deswegen wird im Gesamtplan höchstens ein Fahrzeug pro Zusammenhangskomponente verwendet.
□

Nach diesen positiven Resultaten folgt das negative. Im allgemeinen Fall, bei dem die Fahrzeuge anfangs beladen sind, ist das Längen-Problem NP-vollständig. Auch wenn dies in der Definition der Logistics-Domäne für den Anfangszustand ausgeschlossen ist, ist dieses Resultat nicht nur in der Theorie interessant. Die h^+-Heuristik wird zur Bewertung von Zuständen einer Logistics-Instanz herangezogen und in diesen Zuständen können die Fahrzeuge beladen sein.

Satz 8.3
Das relaxierte Logistics-Längen-Problem ist für allgemeine Logistics-Instanzen, bei denen die Fahrzeuge anfangs beladen sein dürfen, NP-schwer.

Beweis:
Wir zeigen, dass man jede Mengenüberdeckungsinstanz als Logistics-Instanz darstellen kann. Wir erstellen dabei für jedes Element der Gesamtmenge mehrere Orte, für die Elemente der Teilmengen erstellen wir Güter **in** Fahrzeugen. Zusätzlich wird es einen Ort mit Gütern geben. Die Auswahl einer minimalen Menge von Fahrzeugen, die wir zu diesem Ort schicken, wird einer minimalen Mengenüberdeckung in der Mengenüberdeckungsinstanz entsprechen. Es wird genau dann eine Mengenüberdeckung der Größe k geben, wenn es für die relaxierte Logistics-Instanz einen Plan der Länge k' gibt.
Mengenüberdeckung:
Gegeben: Menge Ω, Teilmengen $\Omega_1, \ldots, \Omega_n \subseteq \Omega$.
Gefragt: Existieren k Teilmengen, so dass $\bigcup_{j=1,\ldots,k} \Omega_{i_j} = \Omega$?
Kodierung als Logistics-Problem:
Erstelle für jedes $\omega \in \Omega$ jeweils l Orte. Dieses l wird später näher bestimmt. Erstelle n Fahrzeuge $t_1, \ldots, t_n$, wobei Fahrzeug t_i gerade Güter mit den $l \cdot |\Omega_i|$ Zielorten aus Ω_i enthält. Platziere die Fahrzeuge an neuen, verschiedenen Orten. Erstelle einen neuen Ort, den Ω-Ort, mit $l \cdot |\Omega|$ Gütern, welche an alle Orte aus Ω müssen. Die Stadt hat somit insgesamt $l \cdot |\Omega| + n + 1$ Orte.

Es gibt genau dann einen h^+-Plan der Länge $2l \sum_{i=1}^{n} |\Omega_i| + 2l|\Omega| + k$, wenn eine Mengenüberdeckung der Größe k existiert.

„⇐“:

Es gebe eine k-Überdeckung. Œ sei diese $\Omega_1, \ldots, \Omega_k$. Fahre mit $t_1, \ldots, t_k$ zum Ω-Ort, dies sind k Aktionen. Lade dort für jedes $\omega \in \Omega$ alle dazugehörigen l Güter in ein Fahrzeug t_i ein, so dass $\omega \in \Omega_i$ gilt, dies sind $l \cdot |\Omega|$ Aktionen. Fahre mit allen Fahrzeugen zu den Orten, für die sie Güter geladen haben und lade die entsprechenden Güter aus, dies sind $2l \sum_{i=1}^{n} |\Omega_i| + l|\Omega|$ Aktionen.

„⇒“:

Für die Hinrichtung wählen wir $l = k$.

In jedem gültigen Plan sind für die $k \sum_{i=1}^{n} |\Omega_i|$ Güter in den Fahrzeugen jeweils mindestens 2 Aktionen nötig. Denn jedes Gut muss ausgeladen werden und entweder fährt das Fahrzeug zum Ziel oder das Gut wird in ein anderes Fahrzeug geladen. Auch für die $k|\Omega|$ Güter am Ω-Ort sind jeweils mindestens 2 Aktionen nötig, da sie in ein Fahrzeug geladen und aus diesem wieder ausgeladen werden müssen. Somit benötigen wir für die Güter in den Fahrzeugen und die Güter am Ω-Ort $2k \sum_{i=1}^{n} |\Omega_i| + 2k|\Omega|$ Aktionen, es müssen aber noch Fahrzeuge zum Ω-Ort fahren, um den Transport der dortigen Güter durchzuführen. Für jedes $\omega \in \Omega$ sind k Güter vom Ω-Ort zu Zielorten zu fahren. Somit wird, da ein nicht überdecktes $\omega \in \Omega$ nicht mit der Planlänge vereinbar ist, Ω von k Teilmengen überdeckt. □

Beispiel 8.1 (zum Beweis)
Sei $\Omega = \{1, 2, 3, 4, 5\}$, $\Omega_1 = \{1, 3, 5\}$, $\Omega_2 = \{1, 2, 3\}$, $\Omega_3 = \{3, 4\}$ *und* $\Omega_4 = \{1, 4\}$. *Gibt es eine Mengenüberdeckung der Größe 3?*
Für die zugehörige Logistics-Instanz siehe Abbildung 8.1. Der Ω*-Ort wird als grüner Kreis dargestellt, die vier Fahrzeuge als Quadrate, ihr Standort als Kreis darunter. Die zu einem* $\omega \in \Omega$ *gehörenden drei Orte werden der Übersichtlichkeit wegen in einem großen Ort zusammengefasst. Die dicken Pfeile deuten an, dass die Güter jeweils zu allen drei Orten, die im großen Ort liegen, transportiert werden müssen. Damit man sie besser unterscheiden kann, sind die von den Orten der Fahrzeuge ausgehenden Pfeile in unterschiedlicher Farbe gezeichnet.*
Das Logistics-Problem hat eine Lösung der Größe $2 \cdot 6 \cdot (3 + 3 + 2 + 2) + 2 \cdot 6 \cdot 5 + 3 = 183$. *Darin fahren alle Fahrzeuge zu den Orten ihrer Güter und laden selbige aus. Anschließend fahren die Fahrzeuge* t_1, t_2 *und* t_3 *zum* Ω*-Ort,* t_1 *lädt* $1, 3$ *und* 5 *ein,* t_2 *lädt* 2 *ein und* t_3 *lädt* 4 *ein. Danach laden diese Fahrzeuge ihre Güter bei den Zielorten aus. Die Mengenüberdeckungsinstanz hat somit eine Lösung.*

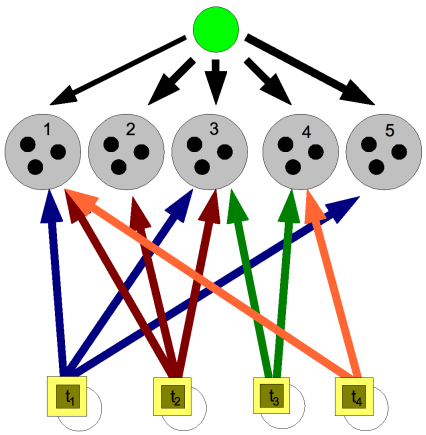

Abbildung 8.1: Beispiel, wie eine Mengenüberdeckungsinstanz als Logistics-Instanz kodiert wird

Dass für jedes $\omega \in \Omega$ jeweils k Orte erstellt werden, ist notwendig, da sonst die Hinrichtung des Beweises nicht mehr gelten würde. Im Fall $l < k$ werden die Fahrzeuge nicht genug bestraft, wenn sie zu Orten fahren, die anfangs nichtgeladenen Gütern entsprechen. Je nach Mengenüberdeckungsinstanz könnten Fahrzeuge die Aufgaben für andere Fahrzeuge übernehmen, so dass die Logistics-Instanz eine Lösung der angegebenen Größe hätte, obwohl die Mengenüberdeckungsinstanz nicht lösbar ist. Folgendes Beispiel liefert für jedes $n \in \mathbb{N}$ eine solche Mengenüberdeckungsinstanz.

Beispiel 8.2
Sei $\Omega = \{1, \ldots, n\}$, $\Omega_1 = \{2, \ldots, n\}$ *und* $\Omega_i = \{i\}$ *für* $i = 2, \ldots, n$. *Dann gibt es für kein* $k \in \{1, \ldots, n\}$ *eine* k*-Überdeckung von* Ω.
Wenn für jedes $\omega \in \Omega$ *in der Logistics-Instanz* $l < k$ *Orte erstellt werden, findet man einen* h^+*-Plan der Länge*

$$2l \sum_{i=1}^{n} |\Omega_i| + 2l|\Omega| + (l+1) \leq 2l \sum_{i=1}^{n} |\Omega_i| + 2l|\Omega| + k.$$

Es genügt, wenn Fahrzeug t_1 *zum* Ω*-Ort fährt. Da* t_1 *die gesamte Menge bis auf ein Element überdeckt, muss es nur noch die* l *Orte anfahren, welche die* 1 *repräsentieren.*

8.2.2 Approximation für Logistics

Da das relaxierte Logistics-Problem im Allgemeinen NP-äquivalent ist, werden wir in diesem Teilabschnitt eine Approximation für das Problem angeben, die in polynomieller Zeit berechnet werden kann. Die minimale Anzahl

der Ein- und Ausladeaktionen ist nicht schwer zu bestimmen, deswegen beschränken wir uns darauf, die Bewegungsaktionen abzuschätzen. Das Problem kann, wie wir gesehen haben, in seine Städte und sogar in seine Zusammenhangskomponenten aufgeteilt werden, weswegen wir die Approximation nur für eine Zusammenhangskomponente angeben. Doch zunächst müssen wir den Begriff des Gütertransportgraphen erweitern, bisher haben wir ihn nur für den Fall unbeladener Fahrzeuge definiert. Diese Definition wird auf natürliche Weise auf den Fall beladener Fahrzeuge erweitert, indem mit den Gütern in einem Fahrzeug genauso umgegangen wird, als befänden sie sich am Standort des Fahrzeugs.

Definition 8.5 (erweiterter Gütertransportgraph)
Gegeben eine Logistics-Instanz mit keinem Flugzeug, einer Stadt c und Fahrzeugen T ist der zugehörige erweiterte Gütertransportgraph $EGTG = (L_C, E)$. Dabei ist $E = \{(l, l') \mid l, l' \in L_c, (\text{connected}(l, l') \vee \text{connected}(l', l))\}$, wobei $\text{connected}(l, l') = (l \neq l') \wedge \exists g \in G : l_*(g) = l' \wedge (l_G(g) = l \vee (\exists t \in T : l_G(g) = t \wedge l_T(t) = l))$ bedeutet, dass Güter, die sich an l befinden oder in einem Fahrzeug, das sich an l befindet, geladen sind, zu l' transportiert werden müssen.

In Zukunft sind, wenn von erweiterten Zusammenhangskomponenten (oder erweiterten ZHK) gesprochen wird, Zusammenhangskomponenten im erweiterten Gütertransportgraphen gemeint, die mindestens ein beladenes Fahrzeug enthalten.
An jedem Ort einer erweiterten ZHK, an dem sich kein Fahrzeug befindet, liegt ein Gut oder es muss ein Gut dorthin transportiert werden. Somit muss jeder solche Ort in jedem gültigen Plan mindestens einmal angefahren werden. Wir nehmen an, er müsste genau einmal angefahren werden und erhalten unsere Approximation, indem wir die Ein- und Ausladeaktionen, sowie die zum Ausladen der beladenen Fahrzeuge notwendigen Fahraktionen dazunehmen. Den zur Approximation gehörenden Plan bezeichnen wir mit h^-, seine Länge mit $|h^-|$. Algorithmus 9 liefert eine alternative Definition der Approximation über ein neues Fahrzeug, dass sich an allen Standorten der beladenen Fahrzeuge befindet. Im Algorithmus werden alle beladenen Fahrzeuge ausgeladen. Für die an Orten stehenden Güter ist ein neues Fahrzeug zuständig, welches an allen Orten steht, an denen sich die beladenen Fahrzeuge nach dem Ausladen in einem h^+-Plan befinden würden. Zur Erleichterung des Verständnisses folgt ein Beispiel.

Beispiel 8.3 (zur Approximation)
Betrachte Abbildung 8.2. In a) ist eine erweiterte Zusammenhangskomponente zu sehen mit zwei Fahrzeugen, von denen eines zwei Güter auszuladen hat, das andere eines, dies wird durch von den Fahrzeugen ausgehende Pfeile gekennzeichnet. Es gibt einen Ort an dem drei zu transportierende Güter

Algorithm 9 Fahrzeug zu erweiterter Zusammenhangskomponente

Input: Zusammenhangskomponente Z, Menge der beladenen Fahrzeuge $T_b \subseteq T$ der Zusammenhangskomponente

Output: Zuweisung eines Fahrzeugs an die Zusammenhangskomponente

if Z ist erweiterte Zusammenhangskomponente **then**
Fahre alle Fahrzeuge aus T_b zu den Zielorten ihrer geladenen Güter, dies verändert ihre Standorte l_T.
Lade alle Güter in Fahrzeugen aus T_b an ihrem Zielort aus.
Erstelle neues Fahrzeug $t_* \notin T$ mit $l_T(t_*) = \bigcup_{t_b \in T_b} l_T(t_b)$.
Weise Fahrzeug t_* an Z zu.
else {Z ist gewöhnliche Zusammenhangskomponente}
Falls Z ein Fahrzeug enthält, weise ein solches Z zu, andernfalls weise beliebiges Fahrzeug $t \in T$ zu.
end if

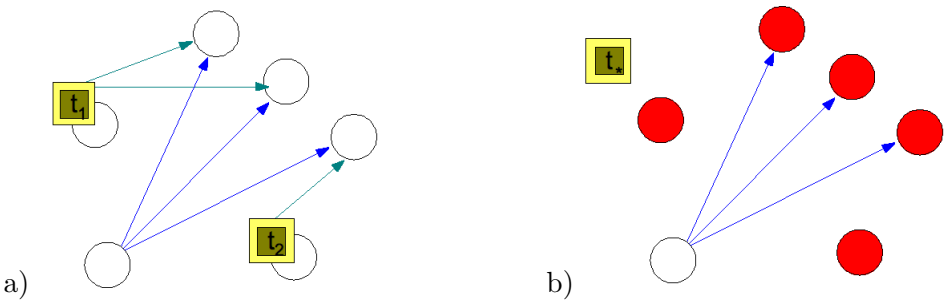

Abbildung 8.2: Beispiel zur Approximation. Links eine Instanz, rechts die Approximation nach Ausladen der beiden Fahrzeuge

liegen, die Zielorte sind durch Pfeile gekennzeichnet. In b) ist die zugehörige Approximation nach dem Ausladen der beiden Fahrzeuge zu sehen. Es gibt ein neues Fahrzeug t_*, dessen Standorte $l_T(t_*)$ rot markiert sind. Da die beiden ursprünglichen Fahrzeuge für die Approximation nach Erstellen von t_* nicht relevant sind, sind sie in b) nicht dargestellt.
Die Länge der Approximation ist 10, es müssen die beiden Fahrzeuge entladen werden, das neue Fahrzeug muss zum Standort der anderen Güter fahren und die drei Güter jeweils ein- und ausladen. Jeder kürzeste h^+-Plan für das Problem hat die Länge 11, entweder fahren beide Fahrzeuge zum Startort der Güter oder das linke zu dem ihm fehlenden Ort.

Als nächstes zeigen wir, dass die in Algorithmus 9 beschriebene Approximation mit der anfangs gegebenen Anschauung übereinstimmt. Damit können wir in Zukunft beide Charakterisierungen der Approximation verwenden, was für den Beweis seiner Eigenschaften von Vorteil sein wird.

Satz 8.4
Sei eine Logistics-Instanz gegeben. Œ gebe es nur eine Stadt, diese bestehe aus einer erweiterten ZHK Z. Seien

- $T_b = \{t \in T \mid \exists g \in G : l_G(g) = t\}$ *die beladenen Fahrzeuge von* Z.
- $G(t) = \{g \in G \mid l_G(g) = t\}$ *die Güter in Fahrzeug* $t \in T_b$.
- $D(t) = \{l \in L \setminus l_T(t) \mid \exists g \in G(t) : l_*(g) = l\}$ *die Orte, die Fahrzeug* $t \in T_b$ *anfahren muss, um seine Güter abzuladen.*
- $G' = \{g \in G \mid \exists l \in L : l_G(g) = l \wedge l_*(g) \neq l\}$ *die Güter, die sich an Orten, aber noch nicht an ihren Zielorten befinden.*

Dann gilt für die Länge $|h^-|$ *der Approximation aus Algorithmus 9:*

$$\begin{aligned} |h^-| &= \sum_{t \in T_b} (|G(t)| + |D(t)|) + 2|G'| + \\ &\quad |\{l \in L \mid \forall t \in T_b : l \notin D(t) \wedge l_T(t) \neq l \wedge \text{important}(l)\}| \end{aligned}$$

Dabei ist $\text{important}(l) = \exists g \in G' : l_G(g) = l \vee l_*(g) = l$.

Beweis:
Für das Ausladen der Fahrzeuge sind $\sum_{t \in T_b} (|G(t)| + |D(t)|)$ Aktionen nötig. Das Ein- und Ausladen des neuen Fahrzeugs mit den übrigen Gütern kostet $2|G'|$ Aktionen. Das neue Fahrzeug t_* der Approximation muss alle Orte anfahren, an denen Güter liegen oder zu denen welche transportiert werden müssen. Abzüglich der Orte $l_G(t_*)$, an denen es sich befindet, sind dies gerade die Orte der Menge $\{l \in L \mid \forall t \in T_b : l \notin D(t) \wedge l_T(t) \neq l \wedge \text{important}(l)\}$. □

Wir wollen nun einige Eigenschaften der Approximation nachweisen. Die meisten der Eigenschaften sind gut für die Praxis, die letzte leider nicht. Sie sagt uns, dass die Approximation nicht monoton ist, was bedeutet, dass sie in Verbindung mit der A^*-Suche nicht unbedingt eine optimale Lösung findet.

Satz 8.5
Die Approximation hat folgende Eigenschaften:

a) *Sie kann in polynomieller Zeit berechnet werden.*
b) *Wenn es im aktuellen Zustand keine beladenen Fahrzeuge gibt, stimmt sie mit der* h^+*-Heuristik überein.*
c) *Ihr Wert übersteigt nie den der* h^+*-Heuristik. Insbesondere ist die Approximation eine zulässige Heuristik.*

d) Sei $|h^+|$ die Länge eines optimalen h^+-Plans. Dann gilt $|h^-| \geq c|h^+|$ mit $c \geq \frac{1}{2}$.

e) Die Approximation ist nicht monoton.

Beweis:

zu a) Dies ist eigentlich klar, im Wesentlichen ist nur $l_T(t_*)$ zu berechnen. Das Bestimmen von $l_T(t_*)$ ist in Zeit $O(|G| \cdot |T|)$ möglich.

zu b) Klar.

zu c) Betrachte eine erweiterte Zusammenhangskomponente, für gewöhnliche gilt die Aussage trivialerweise.
In jedem gültigen h^+-Plan müssen alle Güter aus Fahrzeugen und von Orten an ihrem Ziel abgeladen werden. Für die Güter in den Fahrzeugen ist dies nicht billiger als über den direkten Transport möglich, ein Umladen lohnt sich nicht. Intuitiv ist dies klar, der formale Beweis wird in Lemma 8.2 nachgeliefert. Also könnte der h^+-Plan höchstens Fahrtkosten für die an Orten befindlichen Güter sparen. Doch auch in ihm müssen alle Orte, an denen sich Güter befinden oder zu denen Güter transportiert werden müssen, mindestens einmal besucht werden. Somit kann er nicht günstiger sein als der Plan der Approximation.

zu d) Sei eine Logistics-Instanz gegeben, deren Gütertransportgraph eine erweiterte Zusammenhangskomponente ist und seien die Bezeichnungen $G(t), D(t)$ und G' wie in Satz 8.4. Seien $S = \{l \in L | \exists g \in G' : l_G(g) = l\}$ die Orte mit Gütern. In der Approximation fährt Fahrzeug t_* alle Orte $s \in S$ an, die kein Standort von ihm sind. Bei einer gegebenen Besuchsreihenfolge der Orte aus S wird t_* für $s \in S$ zu r_s Orten fahren müssen, um die Güter von s ein- und auszuladen. Damit hat h^- eine Länge von $\sum_{t \in T_b} (|G(t)| + |D(t)|) + 2|G'| + \sum_{s \in S} r_s$ mit $r_s \in \mathbb{N} \cup \{0\}$.
In einem festen optimalen h^+-Plan h^+ werden ebenfalls Fahrzeuge zu $s \in S$ geschickt und nach s werden weitere Orte besucht. Da die Reihenfolge der Besuche der Orte aus S eine andere sein kann und weil die Standorte $l_T(t_*)$ hier auf mehrere Fahrzeuge aufgeteilt sind, sind dies r'_s Orte und es gilt nicht notwendigerweise $r'_s = r_s$. Da in h^+ mehr als ein Fahrzeug zu diesen Orten geschickt werden kann, ist die Planlänge $|h^+| = \sum_{t \in T_b} (|G(t)| + |D(t)|) + 2|G'| + \sum_{s \in S} (m_s + r'_s)$ mit $m_s, r'_s \in \mathbb{N} \cup \{0\}$ für $s \in S$, dabei ist m_s die Anzahl der zu s geschickten Fahrzeuge.

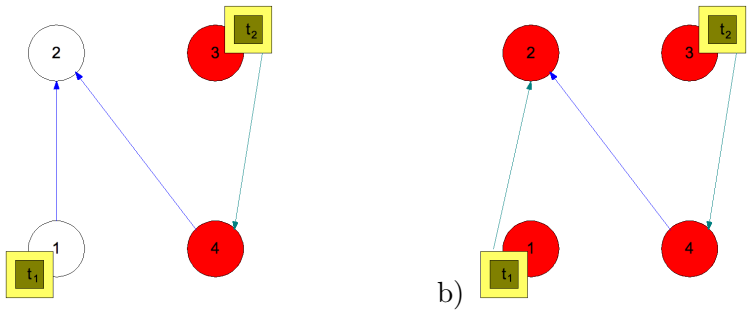

Abbildung 8.3: Logistics-Instanz, auf der die Approximation nicht monoton ist

Somit ergibt sich ein Verhältnis von

$$\frac{|h^-|}{|h^+|} = \frac{\sum_{t\in T_b}(|G(t)|+|D(t)|)+2|G'|+\sum_{s\in S} r_s}{\sum_{t\in T_b}(|G(t)|+|D(t)|)+2|G'|+\sum_{s\in S}(m_s+r'_s)}$$

Dabei kann $\sum_{s\in S}(m_s+r'_s)$ nicht beliebig groß werden, sondern es gilt $\sum_{s\in S}(m_s+r'_s) \leq \sum_{s\in S} r_s + \sum_{t\in T_b}(|D(t)|+1)$. Denn ein gültiger h^+-Plan wäre es, ein beliebiges Fahrzeug an alle relevanten Orte zu schicken, an denen es sich nicht befindet. Die Summe aller Zielorte ist höchstens $\sum_{s\in S} r_s + \sum_{t\in T_b}(|D(t)|+1)$. Somit gilt für das Verhältnis:

$$\begin{aligned}
\frac{|h^-|}{|h^+|} &\geq \frac{\sum_{t\in T_b}(|G(t)|+|D(t)|)+2|G'|+\sum_{s\in S} r_s}{\sum_{t\in T_b}(|G(t)|+|D(t)|)+2|G'|+\sum_{s\in S} r_s+\sum_{t\in T_b}(|D(t)|+1)} \\
&\overset{|G(t)|\geq 1}{\geq} \frac{\sum_{t\in T_b}(|G(t)|+|D(t_b)|)+2|G'|+\sum_{s\in S} r_s}{2\sum_{t\in T_b}(|G(t)|+|D(t)|)+2|G'|+\sum_{s\in S} r_s)} \\
&\geq \frac{\sum_{t\in T_b}(|G(t)|+|D(b)|)+2|G'|+\sum_{s\in S} r_s}{2\sum_{t\in T_b}(|G(t)|+|D(t)|)+4|G'|+2\sum_{s\in S} r_s)} \\
&= \frac{1}{2}
\end{aligned}$$

zu e) Betrachte Abbildung 8.3. In a) ist der Vorgängerzustand zu sehen. Es befinden sich zwei Fahrzeuge in der ZHK, ein unbeladenes, t_1, an Ort 1 und ein beladenes, t_2, an Ort 3. Fahrzeug t_2 hat ein Gut mit Zielort 4 geladen. An Ort 1 und an Ort 4 befindet sich jeweils ein Gut mit Zielort 2. Die Kosten der Approximation sind in diesem Fall

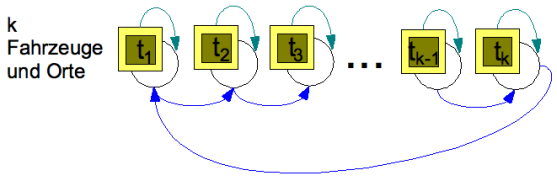

Abbildung 8.4: Logistics Instanz, bei der die Approximation die h^+-Plankosten um 25% unterschätzt

– unbeladene Fahrzeuge werden von der Approximation ignoriert – 4 für das Ein- und Ausladen der Güter an Orten, 1 für das Ausladen des beladenen Fahrzeugs und Fahrkosten von 3 (2 nicht abgedeckte Orte und die Fahrkosten von Fahrzeug t_2), somit insgesamt 8.
In b) ist der Nachfolgezustand zu sehen, bei dem Fahrzeug t_1 das Gut von Ort 1 eingeladen hat. Die Kosten der Approximation sind in diesem Fall 2 für das Ein- und Ausladen des Gutes an Ort 4, 2 für das Ausladen der beladenen Fahrzeuge und Fahrkosten von 2 (keine nicht abgedeckten Orte, aber Fahrkosten für Fahrzeug t_1 und t_2), somit insgesamt 6.
Damit fallen die Kosten nach Einladen des Gutes in t_1 um 2, also ist die Approximation nicht monoton.

□

Die Konstante c aus Satz 8.5 d) kann eventuell verbessert werden, doch es gilt $\frac{1}{2} \leq c \leq \frac{3}{4}$. Betrachte dazu Abbildung 8.4. Es gibt k Orte, an denen je ein Fahrzeug steht, welches ein Gut an diesem Ort ausladen muss. Die Orte sind kreisförmig zusammengeschlossen, jeder Ort muss ein Gut zum nächsten transportieren.
In der Approximation sind k Aktionen zum Ausladen der Fahrzeuge notwendig. Danach wird ein Fahrzeug erstellt, welches an allen k Orten steht. Es sind je k Ein- und Ausladeaktionen notwendig, was eine Gesamtplanlänge von $3k$ ergibt.
In jedem h^+-Plan sind k Aktionen zum Ausladen der Fahrzeuge notwendig, sowie $2k$ Aktionen für das Ein- und Ausladen der Güter. Es sind mindestens $k-1$ Fahraktionen nötig, es muss jeweils ein Fahrzeug von einem Ort zum nächsten fahren. Falls es mehr als einen Ort geben würde, von dem kein Fahrzeug ab- oder dorthin fahren würde, könnte im h^+-Plan kein Gut von einem dieser Orte transportiert werden.
Somit gibt es für jedes $k \in \mathbb{N}$ eine Instanz, bei der das Verhältnis $\frac{|h^-|}{|h^+|} = \frac{3k}{4k-1}$ beträgt. Somit muss $c \leq \frac{3}{4}$ gelten.

8.2.3 Genaue Berechnung für Logistics

Die Approximation ist, wie wir gesehen haben, nicht monoton und es gibt Instanzen, auf denen sie einen um 25% schlechteren Wert als die h^+-Heuristik liefert. In diesem Teilabschnitt wollen wir versuchen, einen Weg zur Berechnung der NP-schweren h^+-Heuristik zu finden. Eventuell findet sich ein bekanntes NP-vollständiges Problem, auf das wir das Logistics-Problem reduzieren können. Zunächst definieren wir ein neues NP-vollständiges Problem, welches uns zur Berechnung der h^+-Heuristik von Nutzen sein wird. Wir werden uns bei der Angabe von Lösungen weiterhin auf den Fall einer erweiterten ZHK in einer Stadt beschränken.

Definition 8.6 (billigste Teilüberdeckung)
Sei Ω eine Menge und seien $\Omega_1, \ldots, \Omega_n \subseteq \Omega$ Teilmengen.

- ***Entscheidungsvariante:*** *Gibt es für ein vorgegebenes $k \in \mathbb{N}$ Teilmengen $\Omega_{i_1}, \ldots, \Omega_{i_m}$, so dass $m + |\Omega \setminus (\bigcup_{j=1}^{m} \Omega_{i_j})| \leq k$? Falls ja, sagen wir, es gibt eine Teilüberdeckung von Ω mit Kosten höchstens k.*
- ***Optimierungsvariante:*** *Finde das minimale $k \in \mathbb{N}$, so dass es eine Teilüberdeckung von Ω mit Kosten höchstens k gibt. Wir nennen dieses k die Kosten der billigsten Teilüberdeckung von Ω und schreiben dafür $BT\ddot{U}(\Omega; \Omega_1, \ldots, \Omega_n)$.*

Wir bezeichnen die Entscheidungsvariante mit $BT\ddot{U}_k$ und die Optimierungsvariante mit $BT\ddot{U}$.

Beispiel 8.4
Sei $\Omega = \{1, 2, 3, 4, 5\}$ und $\Omega_1 = \{2, 4, 5\}$, $\Omega_2 = \{1, 4\}$, $\Omega_3 = \{1, 3, 5\}$, $\Omega_4 = \{2, 5\}$. Die Kosten der billigsten Teilüberdeckung von Ω sind 3, eine solche Teilüberdeckung ist beispielsweise die Auswahl von Ω_1 und Ω_2, da Element 3 nicht überdeckt wird, wird 1 addiert.

Korollar 8.2
Die Entscheidungsvariante des billigste-Teilüberdeckungs-Problems aus Definition 8.6 ist NP-vollständig.

Beweis:

- Die Zugehörigkeit zu NP ist klar.
- Es gilt SetCover Entscheidungsvariante $\leq_p$ $\text{BT}\ddot{\text{U}}_k$. Der Beweis funktioniert wie in Satz 8.3, für jedes Element aus Ω werden k Duplikate

erzeugt. Es gibt in der billigsten-Teilüberdeckungs-Instanz genau dann eine Teilüberdeckung mit Kosten höchstens $2k \sum_{i=1}^{m} |\Omega_i| + 2k|\Omega| + k$, wenn die Mengenüberdeckungsinstanz eine Lösung der Größe höchstens k besitzt.

Zusammen folgt die NP-Vollständigkeit für $\text{BT}\ddot{\text{U}}_k$. □

Die „Rückrichtung“ $\text{BT}\ddot{\text{U}}_k \leq_p$ SetCover Entscheidungsvariante kann ebenfalls mit wenig Aufwand bewiesen werden. Wir werden dies später zeigen, zunächst wird erklärt, weshalb das billigste-Teilüberdeckungs-Problem für die Logistics-Domäne eine wichtige Rolle spielt. Wenn es nur einen Ort gibt, an dem sich Güter befinden, setzen sich die Kosten einer optimalen Lösung aus den bekannten Kosten und den Kosten einer billigsten Teilüberdeckung zusammen.

Satz 8.6
Gegeben sei eine erweiterte Zusammenhangskomponente in einer Logistics-Instanz mit beladenen Fahrzeugen und genau einem Ort l_0, an dem sich zu transportierende Güter befinden. Seien $T_b, G(t), D(t)$ und G' wie in Satz 8.4. Seien

- $\overline{D(t)} = D(t) \cup \{l_T(t)\}$ *die Standorte von Fahrzeug $t \in T_b$ nach Ausladen seiner Güter.*
- $T_0 = \{t \in T_b \mid L_T(t) = l_0 \vee \exists g \in G : l_G(g) = t \wedge l_*(g) = l_0\}$ *die Fahrzeuge, die sich an l_0 befinden oder dorthin fahren müssen und*
- $\overline{D(T_0)} = \{l \in L \mid \exists t \in T_0 : (l_T(t) = l \vee \exists g \in G : l_G(g) = t \wedge l_*(g) = l)\}$ *die Orte, an denen sie sich nach Ausladen ihrer Güter befinden.*
- $\overline{L_0} = \{l \in L \mid \exists g \in G : l_G(g) \neq l_*(g) \wedge l_G(g) = l_0 \wedge l_*(g) = l\}$ *die Zielorte der Güter von l_0.*
- $L_0 = \overline{L_0} \setminus \overline{D(T_0)}$ *die Zielorte der Güter von l_0, zu denen noch kein Fahrzeug von l_0 unterwegs ist.*

Mit $T_b = \{t_1, \ldots, t_m\}$ gilt für jeden optimalen h^+-Plan

$$
\begin{aligned}
|h^+| &= \mathcal{L}(\mathit{Logistics}, 3) \\
&:= \sum_{i=1}^{m} (|G(t_i)| + |D(t_i)|) + 2|G'| + \\
&\quad \begin{cases} |\overline{L_0} \cup \{l_0\}|, & \text{falls } \overline{L_0} \cap \overline{D(t_i)} = \emptyset \\ & \forall i \in \{1, \ldots, m\} \\ & \text{und } T_0 = \emptyset \\ BT\ddot{U}(L_0; \overline{D(t_1)} \cap L_0, \ldots, \overline{D(t_m)} \cap L_0), & \text{sonst} \end{cases}
\end{aligned}
$$

Für den Beweis des Satzes benötigen wir das bereits erwähnte Lemma, welches es uns ermöglicht, uns auf Pläne zu beschränken, in denen die beladenen Fahrzeuge ihre geladenen Güter am Zielort ausladen.

Lemma 8.2
Es gibt einen optimalen h^+-Plan, in dem alle Fahrzeuge ihre Güter an den jeweiligen Zielorten ausladen.

Beweis:
Sei h^+ ein optimaler h^+-Plan, in dem es ein beladenes Fahrzeug t gibt, das ein Gut $g \in G(t)$ nicht am Zielort $l_*(g)$ auslädt. Dann muss g in h^+ in ein Fahrzeug t' geladen werden. Für das Umladen sind 2 Aktionen nötig, für das Ausladen von g an $l_*(g)$ mindestens 1 weitere.
Alternativ kann t das Gut g mit höchstens 2 Aktionen an $l_*(g)$ ausladen. Danach kann t' mit Kosten 1 nach $l_*(g)$ fahren, falls es sich nicht dort befindet. In diesem abgeänderten Plan sind alle Fahrzeuge mindestens an den Orten, an denen sie sich in h^+ befinden und Fahrzeug t liefert g an $l_*(g)$ ab.
Auf diese Weise kann induktiv ein gültiger h^+-Plan gefunden werden, in dem alle Fahrzeuge ihre Güter am Zielort ausladen und der die Kosten von h^+ nicht übersteigt. □

Beweis zu Satz 8.6:
$|h^+| \leq \mathcal{L}(\text{Logistics}, 3)$:
Fahre mit den beladenen Fahrzeugen an die Zielorte ihrer geladenen Güter. Falls $T_0 = \emptyset$ und $\overline{L_0} \cap \overline{D(t_i)} = \emptyset$ für alle $i \in \{1, \ldots, m\}$, fahre mit einem beliebigen Fahrzeug zu l_0, lade alle Güter $g \in G'$ ein, fahre die Zielorte der Güter an und lade sie aus. Sonst lade mit den Fahrzeugen aus T_0 die Güter von l_0 ein, an denen sich das jeweilige Fahrzeug befindet. Fahre dann mit den Fahrzeugen einer billigsten Teilüberdeckung zu l_0, lade alle Güter, an deren Zielort sich ein Fahrzeug der billigsten Teilüberdeckung befindet in dieses ein. Lade die übrigen Güter aus G' in ein beliebiges Fahrzeug und fahre an deren Zielorte. Lade die Güter in den Fahrzeugen an ihren Zielorten aus.
$|h^+| \geq \mathcal{L}(\text{Logistics}, 3)$:
Sei h^+ ein optimaler h^+-Plan für das Problem. Wegen Lemma 8.2 können wir annehmen, dass in h^+ alle beladenen Fahrzeuge ihre Güter an den Zielorten entladen. Dafür sind $\sum_{i=1}^{m}(|G(t_i)| + |D(t_i)|)$ Aktionen nötig. Für das Ein- und Ausladen der Güter von l_0 sind $2|G'|$ Aktionen nötig. Falls $T_0 = \emptyset$ und $\overline{L_0} \cap \overline{D(t_i)} = \emptyset$ für alle $i \in \{1, \ldots, m\}$, befindet sich weder an l_0 noch an einem Zielort seiner Güter ein Fahrzeug. Da alle diese Orte angefahren werden müssen, sind mindestens $|\overline{L_0} \cup \{l_0\}|$ weitere Fahraktionen nötig. Sonst sind für die Güter $g \in G'$ mit Zielorten $l \notin \overline{D(T_0)}$ per Definition mindestens $BT\ddot{U}(L_0; \overline{D(t_1)} \cap L_0, \ldots, \overline{D(t_m)} \cap L_0)$ Fahraktionen nötig. Somit hat in beiden Fällen jeder gültige Plan mindestens die angegebene Länge. □

Als nächstes zeigen wir, dass $\text{BT}\ddot{\text{U}}_k \leq_p$ SetCover Entscheidungsvariante. Somit ist nicht nur explizit gezeigt, dass $\text{BT}\ddot{\text{U}}_k$ in NP liegt, wegen der einfachen Kodierung als Mengenüberdeckungsproblem stellt sich heraus, dass BTÜ ein „getarntes“ Mengenüberdeckungsproblem ist. Dies ist für die Implementierung wichtig, so können wir auf unsere Algorithmen zum Lösen von SetCover zurückgreifen.

Satz 8.7
Es gilt $\text{BT}\ddot{\text{U}}_k \leq_p$ SetCover Entscheidungsvariante.

Beweis:
Zur Erinnerung, im billigste Teilüberdeckungs-Problem hat man eine Menge Ω und n Teilmengen $\Omega_1, \ldots, \Omega_n \subseteq \Omega$ gegeben und ist an der Minimierung von $m + |(\Omega \setminus (\bigcup_{j=1}^{m} \Omega_{i_j})|$ interessiert.
Bei der Optimierung des Wertes lohnt es sich nicht, auf eine Menge Ω_i zu verzichten, die ein neues Element liefert.

$$\begin{aligned} m + |\Omega \setminus (\bigcup_{j=1}^{m} \Omega_{i_j})| &= (m-1) + 1 + \underbrace{|(\Omega \setminus (\bigcup_{\substack{j=1 \\ i_j \neq i}}^{m} \Omega_{i_j})) \setminus \Omega_i|}_{\text{VSS} \leq |\Omega \setminus (\bigcup_{\substack{j=1 \\ i_j \neq i}}^{m} \Omega_{i_j})| - 1} \\ &\leq (m-1) + |\Omega \setminus (\bigcup_{j=1, i_j \neq i}^{m} \Omega_{i_j})| \end{aligned}$$

Somit liefert eine minimale Überdeckung von Ω, falls dies möglich ist, eine billigste Teilüberdeckung. Für die Entscheidungsvariante können wir also die folgenden Mengenüberdeckungsinstanzen erstellen.

Fall 1: $\bigcup_{i=1}^{m} \Omega_i = \Omega$.
Setze $k' = k$ und finde eine k'-Überdeckung von Ω.

Fall 2: $\bigcup_{i=1}^{m} \Omega_i \neq \Omega$
Setze $\Omega' = \bigcup_{i=1}^{m} \Omega_i$ und $k' = k - |\Omega \setminus \Omega'| = k + |\Omega'| - |\Omega|$ und finde eine k'-Überdeckung von Ω'.

Somit gilt die Aussage, denn eine Mengenüberdeckung der Größe k' liefert eine billigste Teilüberdeckung mit Kosten k. □

Da k im Beweis monoton in k' wächst, liefert eine minimale Überdeckung von Ω' eine billigste Teilüberdeckung für Ω. Dadurch folgt aus Satz 8.6 direkt das folgende Korollar, das uns sagt, dass wir das Logistics-Problem, bei dem es in jeder ZHK genau einen Ort mit Gütern gibt, auf das Mengenüberdeckungsproblem zurückführen können.

Korollar 8.3
Gelten die Voraussetzungen aus Satz 8.6. Neben den dortigen Bezeichnungen seien

- $\hat{D}(t_i) = \overline{D(t_i)} \cap L_0$ *die von Fahrzeug* $t_i \in T_b$ *angefahrenen Zielorte von Gütern von* l_0.
- $\hat{L}_0 = \bigcup_{i=1}^{m} \hat{D}(t_i)$ *die insgesamt von beladenen Fahrzeugen angefahrenen Zielorte von Gütern, die sich an* l_0 *befinden.*

Dann gilt für die Länge jedes optimalen h^+*-Plans:*

$$|h^+| = \sum_{i=1}^{m}(|G(t_i)| + |D(t_i)|) + 2|G'| + |L_0| - |\hat{L}_0| + \begin{cases} 1, & \text{falls } \overline{L_0} \cap \overline{D(t_i)} = \emptyset \\ & \forall i \in \{1, \ldots, m\} \\ & \text{und } T_0 = \emptyset \\ SetCover(\hat{L}_0; \hat{D}(t_1), \ldots, \hat{D}(t_m)), & \text{sonst} \end{cases}$$

□

Dieses Resultat lässt sich nicht einfach auf den Fall mit mehreren Orten übertragen. Bei n Orten können sich die Zielorte der Güter überlagern, so dass die Probleme für die Orte nicht unabhängig sind. Die optimalen Fahrtkosten lassen sich nicht als Summe der einzelnen billigsten Teilüberdeckungen ausdrücken.

Beispiel 8.5
Es sei eine Logistics-Instanz mit $m+n$ *Orten gegeben. Es gibt* m *Fahrzeuge, jedes steht an einem eigenen Ort und ist mit einem Gut beladen. Das Gut im Fahrzeug hat als Ziel den Ort, an dem sich das Fahrzeug befindet. An den restlichen* n *Orten befinden sich jeweils* m *Güter, jedes hat einen der* m *Orte, an denen sich die Fahrzeuge befinden, als Zielort. In Abbildung 8.5 ist diese Instanz zu sehen.*
Wenn man das Problem für jeden Ort separat nach Satz 8.6 löst, sind die Fahrtkosten für jeden Ort m*, es werden beispielsweise alle Fahrzeuge zu jedem Ort geschickt. In einem optimalen Plan fährt ein Fahrzeug die* $m-1$

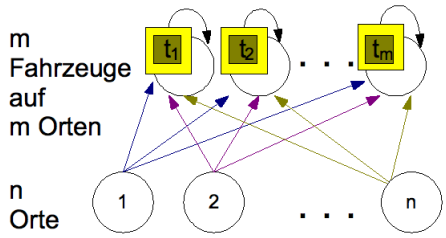

Abbildung 8.5: Logistics Instanz mit $n + m$ Orten, bei der die Fahrzeugzuweisungen der Orte nicht unabhängig voneinander sind

fehlenden Orte an und übernimmt den Transport aller $n \cdot m$ Güter. Dies liefert Fahrkosten von $n + m - 1$, diese sind um den Faktor $\min(n, m)/2$ kürzer als die Summe der Fahrtkosten der Orte nach Satz 8.6.

Für den allgemeinen Fall mit mehreren Orten, an denen sich Güter befinden, fiel mir lediglich der naive Algorithmus ein. In diesem werden alle Anfahrten von Fahrzeugen zu Orten und für jede solche Kombination für jeden Ort alle Einladungen von Gütern in die Fahrzeuge, die sich am Ort befinden, durchgegangen. Dies ist in Algorithmus 10 zu sehen.

Satz 8.8
Sei eine erweiterte Zusammenhangskomponente einer Logistics-Instanz mit m beladenen Fahrzeugen $t_1, \ldots, t_m$ und n Orten $l_1, \ldots, l_n$, an denen sich Güter befinden, die transportiert werden müssen, gegeben. Seien $G(t), L(t)$ und $\overline{D(t)}$ wie in Satz 8.6. Seien

- $G' = \{g \in G \mid \exists i \in \{1, \ldots, n\} : l_G(g) = l_i \wedge l_*(g) \neq l_i\}$ *die Güter die an anderen Orten als ihren Zielorten sind.*
- $L_i = \{l \in L \mid \exists g \in G : l_G(g) \neq l_*(g) \wedge l_G(g) = l_i \wedge l_*(g) = l\}$ *die Zielorte der Güter von l_i.*

Wenn mit $L_{Naiv}(L_1, l_1, \ldots, L_n, l_n, D_1, \ldots, D_m)$ der Rückgabewert von Algorithmus 10 bezeichnet wird, gilt für die Länge $|h^+|$ jedes optimalen h^+-Plans

$$\begin{aligned} |h^+| &= \sum_{i=1}^{m}(|G(t_i)| + |L(t_i)|) + 2|G'| + \\ &\quad L_{Naiv}(L_1, l_1, \ldots, L_n, l_n, \overline{D(t_1)}, \ldots, \overline{D(t_m)}) \end{aligned}$$

Beweis:
Die Existenz eines Planes der angegebenen Länge kann wie in dem Fall gezeigt werden, in welchem nur ein Ort existiert, an dem sich Güter befinden.

Algorithm 10 L-Naiv

Input: Menge L_i von Zielorten und Startorte $l_i \in L_i$ für $i = 1, \ldots, n$, Menge D_j für $j = 1, \ldots, m$ von Standorten der Fahrzeuge

Output: Fahrtkosten für die Güter an den Startorten

1: **if** $n = 0$ **then**
2: **return** 0
3: **end if**
4: min $= \infty$
5: **for** $i = 1, \ldots, n$ **do**
6: $L_i{}' = L_i \setminus (\bigcup_{\substack{j=1,\ldots m \\ l_i \in D_j}} D_j)$
7: **for all** Teilmengen $D_{i_1}, \ldots, D_{i_p}$ aus $D_1, \ldots, D_m$ **do**
8: $L^{Rest} = L_i{}' \setminus (\bigcup_{j=1}^{p} D_{i_j})$
9: Ortskosten $= |\{j \mid j \in \{1, \ldots, p\},\ l_i \notin D_{i_j}\}| + |L^{Rest}|$
10: **for all** Partitionen $(L_{i_1}^{Rest}, \ldots, L_{i_q}^{Rest})$ von L^{Rest} mit $q \leq p$ **do**
11: **for all** Teilmengen $D'_1, \ldots, D'_q$ von $D_{i_1}, \ldots, D_{i_p}$ **do**
12: $D'_{i_k} = D'_k \cup L_{i_k}^{Rest},\ k = 1, \ldots, q$
13: $D'_i = D_i$, für alle anderen i
14: Gesamtkosten = Ortskosten +
15: L-Naiv$(L_1, l_1, \ldots, L_{i-1}, l_{i-1},$
16: $L_{i+1}, l_{i+1}, \ldots, L_n, l_n, D'_1, \ldots, D'_m)$
17: **if** Gesamtkosten $<$ min **then**
18: min = Gesamtkosten
19: **end if**
20: **end for**
21: **end for**
22: **end for**
23: **end for**
24: **return** min

Bleibt zu zeigen, dass jeder gültige Plan mindestens die angegebene Länge hat. Wie in Satz 8.6 kann gezeigt werden, dass $\sum_{i=1}^{m}(|G(t_i)| + |L(t_i)|) + 2|G'|$ Aktionen zum Ausladen der Fahrzeuge und Ein- und Ausladen der Güter an Orten notwendig sind. Es muss noch gezeigt werden, dass mindestens $L_{\text{Naiv}}(L_1, l_1, \ldots, L_n, l_n, \overline{D(t_1)}, \ldots, \overline{D(t_m)})$ weitere Fahraktionen notwendig sind. L-Naiv geht für einen Ort sämtliche Fahrzeugzuweisungen durch, lädt dabei die Güter, deren Zielorte noch nicht abgedeckt werden, nacheinander in die Fahrzeuge, die zum Ort geschickt wurden. Es werden nur beladene Fahrzeuge getestet, denn Verwenden eines unbeladenen Fahrzeugs der gleichen ZHK führt zu keinem günstigeren Plan. Für jede solche Zuweisung berechnet er die optimale Planlänge für die restlichen Orte, wie induktiv gezeigt werden kann. Somit liefert er schließlich die minimale Anzahl an Fahraktionen zurück. □

Die Laufzeit dieses naiven Algorithmus ist größer als $O(n! \cdot n \cdot 2^m)$. Die genaue Laufzeit hängt von L_{Rest} in Zeile 8 ab, welches die Anzahl der Partitionen in Zeile 10 bestimmt.
Auf den Durchgang aller Städte in Zeile 7 kann verzichtet werden. Eine feste Stadtreihenfolge entspricht einer Reihenfolge, in der die Städte besucht werden. Da die Fahrzeuge in einem h^+-Plan an einem Ort bleiben, wenn sie ihn einmal besucht haben, hat die Reihenfolge, in der die Orte angefahren werden, keinen Einfluss auf die Planlänge. Für den Fall $n = 1$ kann $L_{\text{Naiv}}(L_1, l_1, \ldots, L_n, l_n, \overline{D(t_1)}, \ldots, \overline{D(t_m)})$ wie in Korollar 8.3 gesehen über die Lösung eines Mengenüberdeckungsproblems dargestellt werden. Somit kann Algorithmus 10 vereinfacht werden zu Algorithmus 11. Die Laufzeit ist allerdings nach wie vor mindestens exponentiell in der Anzahl der Fahrzeuge.

8.3 Umsetzung

In diesem Abschnitt soll erklärt werden, wie Algorithmus 11 in C++ umgesetzt wurde. Anschließend werden einige Verbesserungen vorgestellt, die eingebaut wurden, um die Laufzeit zu beschleunigen.

8.3.1 Implementierung

Die meisten Methoden in Algorithmus 11 wurden bereits im Kapitel über die Satellite-Domäne vorgestellt. Wie in Zeile 10 die Auswahl der Teilmengen, sowie in Zeile 5 die Implementierung der SetCover-Funktion aussieht, wurde dort erklärt. Es ist noch zu klären, wie die Auswahl der Partitionen in Zeile 13 erfolgen kann. Die Idee dazu stammt von Malte Helmert.

Algorithm 11 L-Semi-Naiv

Input: Menge L_i von Zielorten und Startorte $l_i \in L_i$ für $i = 1, \ldots, n$, Menge D_j für $j = 1, \ldots, m$ von Standorten der Fahrzeuge

Output: Fahrtkosten für die Güter an den Startorten

1: **if** $n = 0$ **then**
2: **return** 0
3: **else if** $n = 1$ **then**
4: $L'_1 = L_1 \setminus (\bigcup_{\substack{j=1,\ldots m \\ l_1 \in D_j}} D_j)$
5: $L^{Rest} = L'_1 \setminus (\bigcup_{j=1}^{m} D_j)$
6: **return** $|L^{Rest}| + SetCover(L'_1 \setminus L^{Rest}; D_1, \ldots, D_m)$
7: **end if**
8: min $= \infty$
9: $L_1{}' = L_1 \setminus (\bigcup_{\substack{j=1,\ldots m \\ l_1 \in D_j}} D_j)$
10: **for all** Teilmengen $D_{i_1}, \ldots, D_{i_p}$ aus $D_1, \ldots, D_m$ **do**
11: $L^{Rest} = L_1{}' \setminus (\bigcup_{j=1}^{p} D_{i_j})$
12: Ortskosten $= |\{j \mid j \in \{1, \ldots, p\},\ l_1 \notin D_{i_j}\}| + |L^{Rest}|$
13: **for all** Partitionen $(L^{Rest}_{i_1}, \ldots, L^{Rest}_{i_q})$ von L^{Rest} mit $q \leq p$ **do**
14: **for all** Teilmengen $D'_1, \ldots, D'_q$ von $D_{i_1}, \ldots, D_{i_p}$ **do**
15: $D'_{i_k} = D'_k \cup L^{Rest}_{i_k},\ k = 1, \ldots, q$
16: $D'_i = D_i$, für alle anderen i
17: Gesamtkosten = Ortskosten +
18: L-Naiv$(L_2, l_2, \ldots, L_n, l_n, D'_1, \ldots, D'_m)$
19: **if** Gesamtkosten $<$ min **then**
20: min = Gesamtkosten
21: **end if**
22: **end for**
23: **end for**
24: **end for**
25: **return** min

Definition 8.7 (Partitionsfunktion)
Eine Partitionsfunktion einer Menge Ω ist eine (nicht notwendig surjektive) Abbildung $f : \Omega \mapsto \{1, \ldots, n\}$.

Die Menge der Partitionsfunktionen einer Menge Ω mit Bildbereich $\{1, \ldots, n\}$ steht zu den Partitionstupeln der Größe höchstens n auf natürliche Weise in eineindeutiger Beziehung. Eine Partitionsfunktion kann interpretiert werden als Partition $(\Omega_1, \ldots, \Omega_n)$. Dabei ist $\Omega_i = \{\omega \in \Omega \mid f(\omega) = i\}$ für $i \in \{1, \ldots, n\}$.

Beispiel 8.6
Sei $\Omega = \{13, 36, 49, 30, 42\}$. Bei der Einteilung in höchstens vier Partitionen entspricht die Partitionsfunktion

$$f = \{13 \mapsto 1, 36 \mapsto 1, 49 \mapsto 1, 30 \mapsto 1, 42 \mapsto 1\}$$

der Partition $(\{13, 36, 49, 30, 42\}, \emptyset, \emptyset, \emptyset)$. Die Partitionsfunktion

$$g = \{13 \mapsto 2, 36 \mapsto 4, 49 \mapsto 2, 30 \mapsto 2, 42 \mapsto 4\}$$

entspricht der Partition $(\emptyset, \{13, 49, 30\}, \emptyset, \{36, 42\})$.

Die Partitionsfunktionen können effizient über einen $|\Omega|$-stelligen $n-$ären Zähler durchgezählt werden. Siehe dazu Algorithmus 12. Dass ein Binärzähler mit einer Partitionsfunktion identifiziert werden kann, ist klar, wenn man die Elemente von Ω durchnummeriert zu $\omega_1, \ldots, \omega_{|\Omega|}$. Element ω_i wird auf den Wert der i-ten Stelle des Binärzählers abgebildet.

8.3.2 Optimierung des Algorithmus

Von Malte Helmert stammt die Idee, die Laufzeit zu verbessern, indem dominierte Fahrzeuge entfernt werden. Ein Fahrzeug t wird dominiert, wenn es in seiner erweiterten ZHK ein Fahrzeug $t' \neq t$ gibt, so dass entweder t leer ist oder t' sich an allen Standorten von t befindet. Formal bedeutet dies mit den Bezeichnungen aus Satz 8.6, Fahrzeug t wird von Fahrzeug t' dominiert, wenn $t \notin T_b$ oder $\overline{D(t)} \subseteq \overline{D(t')}$.

Lemma 8.3
Sei eine erweiterte ZHK gegeben. Falls in ihr ein Fahrzeug t dominiert wird, gibt es einen optimalen h^+-Plan, in dem t außer für das Ausladen seiner Güter nicht benutzt wird.

Beweis:
Wenn t unbeladen ist, wissen wir bereits, dass es einen optimalen Plan gibt,

Algorithm 12 Partitionsfunktion-Klasse

```
1: class Partitionsfunktion do
2:   function void initialize(vector<anytype> Ω, int k)
3:   function bool next()
4:   function getPartition(int i)
5:   vector<int> a
6:   int number_of_partitions
7: end class
```

```
1: def initialize(vector<anytype> Ω, int k) do
2:   erstelle vector< int > a der Größe |Ω|
3:   setze alle Stellen auf 1
4:   number_of_partitions=k
5: end def
```

```
1: def next() do
2:   if a[i] = number_of_partitions für alle i then
3:     return false
4:   else
5:     for i = 0...|Ω| − 1 do
6:       if a[i] = number_of_partitions then
7:         a[i] = 1
8:       else
9:         a[i] + +
10:        break
11:      end if
12:    end for
13:    return true
14:  end if
15: end def
```

```
1: def getPartition(int i) do
2:   return {Ω[j] | a[j] = i}
3: end def
```

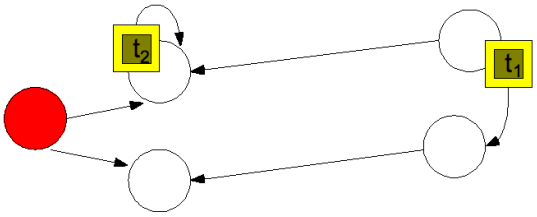

Abbildung 8.6: ZHK, bei der Fahrzeug t_1 in jedem optimalen Plan zu einem Ort geschickt wird, obwohl es sich weder an ihm noch einem der Zielorte seiner Güter befindet

der t nicht verwendet, da t alle Orte bis auf einen der ZHK anfahren muss und es nach Voraussetzung ein weiteres Fahrzeug in der ZHK gibt.
Sei also t beladen und t' ein Fahrzeug, welches t dominiert. Wir wissen aus Lemma 8.2, dass es einen optimalen h^+-Plan h^+ gibt, in dem jedes Fahrzeug seine eigenen Güter an ihren Zielorten auslädt. Danach befindet sich t' nach Voraussetzung an allen Orten, an denen sich t befindet. Somit muss t' höchstens soviele Fahraktionen (und genausoviele Ein- und Ausladeaktionen) wie t in h^+ ausführen, um die Aktionen von t zu übernehmen. Somit erhalten wir einen neuen optimalen h^+-Plan h', in dem t seine eigenen Güter auslädt und sonst keine Aktionen ausführt. □

Das Lemma hilft uns, die Anzahl der Teilmengen in Zeile 10 einzuschränken, indem wir am Anfang von Algorithmus 11 einen Dominanztest für die Standorte der Fahrzeuge durchführen.
Man könnte vermuten, dass in ähnlicher Weise in Zeile 10 auf Fahrzeuge verzichtet werden kann, die sich weder am aktuell betrachteten Startort noch einem seiner Zielorte befinden. Also in Zeile 10 Fahrzeuge t mit $D(t) \cap L_1 = \emptyset$ nicht zu betrachten. Diese Beschneidungstechnik haben wir bereits bei der Satellite-Domäne erfolgreich eingesetzt, als wir auf Instrumente verzichtet haben, die keinen offenen Zielmodus unterstützen.
Bei der Logistics-Domäne kann auf diese Fahrzeuge allerdings nicht verzichtet werden, da die Orte nicht unabhängig sein müssen, wie folgendes Beispiel zeigt.

Beispiel 8.7
Betrachte Abbildung 8.6. Es gibt insgesamt drei Orte, an denen sich Güter befinden, einer davon rot dargestellt. An einem der Zielorte der Güter des roten Ortes befindet sich Fahrzeug t_2. Weiter gibt es Fahrzeug t_1, welches sich weder am roten Ort noch einem der Zielorte seiner Güter befindet und auch keine Güter mit einem dieser Ziele geladen hat. Dennoch wird in jedem optimalen h^+-Plan Fahrzeug t_1 den Transport der Güter des roten Ortes übernehmen. In solch einem Plan werden 4 Fahraktionen durchgeführt. In

jedem h^+-Plan, der die minimale Anzahl an Ein- und Ausladeaktionen verwendet und Fahrzeug t_1 nicht zum roten Ort fährt, werden mindestens 5 Fahraktionen durchgeführt.

Wenn wir den Fall mit einem Fahrzeug in der aktuellen ZHK betrachten, stellen wir fest, dass in diesem Fall der Wert des halbnaiven Algorithmus mit dem der Approximation übereinstimmt. Die Anzahl der Fahrtkosten in der Approximation lässt sich dabei vereinfachen. In einer erweiterten Zusammenhangskomponente mit genau einem beladenen Fahrzeug t ist jeder Ort Start- oder Zielort eines Gutes, oder einer der Standorte des Fahrzeugs. Seien die Orte der ZHK $l_1, \ldots, l_n$ und seien $l_{i_1}, \ldots, l_{i_m}$ die Standorte von Fahrzeug t. Dann gilt für den Rückgabewert von Algorithmus 11, den wir mit $L_{\text{Semi-Naiv}}$ bezeichnen, mit Hilfe von Satz 8.4

$$\begin{aligned} L_{\text{Semi-Naiv}} &= |\{l \in \{l_1, \ldots, l_n\} \mid l \notin \overline{D(t)} \wedge \text{important}(l)\}| \\ &= |\{l_1, \ldots, l_n\} \setminus D(t)| \\ &= |\{l_1, \ldots, l_n\} \setminus \{l_{i_1}, \ldots, l_{i_m}\}| \\ &= n - m. \end{aligned}$$

Da sich dieser Wert über die Subtraktion der Größe der Mengen schneller berechnen lässt als durch Durchgehen der Güterstartorte im halbnaiven Algorithmus, sollte dieser Fall abgefangen werden.
Beim Übergang vom naiven zum halbnaiven Algorithmus wurde erwähnt, dass die Reihenfolge, in der die Städte besucht werden, keine Rolle spielt. Im halbnaiven Algorithmus wird einfach eine feste Reihenfolge angenommen. Eventuell lohnt es sich, eine Anordnung zu finden, die das Problem vereinfacht. Ich habe mir folgenden Ansatz zur Anordnung der Städte überlegt und ihn implementiert. Das Entfernen von Zielen in Zeile 9 kann auch für sämtliche Orte erfolgen. Dann haben die Startorte nur noch die Orte als Ziele, an denen sich die Fahrzeuge des Startortes nicht befinden. Nun werden die Orte aufsteigend nach der Zahl ihrer Zielorte angeordnet, d.h. der halbnaive Algorithmus greift immer den Ort mit den wenigsten Zielorten heraus. Dieser Ansatz hat eine Ähnlichkeit mit der Wahl der *most constrained variable* bei CSPs [RN03]. Dabei wird bei der Suche in einem *constrained satisfactory problem* jeweils die Variable als nächste mit Werten belegt, die am wenigsten Auswahlmöglichkeiten für die Belegung bietet. In unserem Fall stimmt dies nicht ganz. Da jedes zum Ort geschickte Fahrzeug eine unterschiedliche Anzahl an Zielorten abdeckt und die abgedeckten Orte sich überschneiden können, muss die Anzahl der möglichen Partitionen in Zeile 13 nicht minimal unter den Startorten sein. Da pro Zielort höchstens ein Fahrzeug verwendet werden muss, ist aber die Anzahl der Teilmengen in Zeile 10 garantiert minimal unter allen Startorten. Da sich durch den rekursiven Aufruf des Algorithmus die Standorte der Fahrzeuge ändern, muss die Anordnung bei jedem Aufruf neu berechnet werden. Dies bringt den Vorteil,

dass Orte mit „leeren“ – da von den am Ort befindlichen Fahrzeug abgedeckten – Zielorten entfernt werden können. Dadurch wird der Basisfall früher erreicht. Es sind weitere, komplexere Anordnungen denkbar. Dies hätte jedoch den Rahmen dieser Diplomarbeit gesprengt, daher habe ich mich auf diese einfache Anordnung beschränkt.
Schließlich kann ein Abbruchkriterium angegeben werden, welches das Absteigen im Algorithmus, also das Aufrufen von L-Semi-Naiv mit einem Startort weniger, unterbricht. Ähnliches haben wir bei der Satellite-Domäne gesehen, als wir die Anzahl der einzuschaltenden Instrumente anhand der bisherigen minimalen Planlänge beschränkt haben. In der Logistics-Domäne können die Kosten mithilfe der Approximation nach unten abgeschätzt werden. Wird bei einem Aufruf des halbnaiven Algorithmus die bisher gefundene minimale Planlänge übergeben, kann die Berechnung am Anfang des rekursiven Aufrufs abgebrochen werden, falls die Approximation diesen Wert übersteigt.

8.4 Experimentelle Ergebnisse

Von Logistics wurden in zwei Jahrgängen des Planungswettbewerbes Instanzen gestellt. Dies war in den Jahren 1998 und 2000. Die 35 Instanzen aus dem Jahr 1998 sind schwieriger zu lösen, es wurden keinerlei Einschränkungen an die Domäne gestellt. Im Jahr 2000 kamen 28 einfacher zu lösende Instanzen zum Einsatz. Jede Stadt hat nur zwei Orte, einer davon ein Flughafen. In jeder Stadt gibt es ein Fahrzeug. In den meisten der Instanzen gibt es nur ein Flugzeug. Auf diesen Instanzen stimmt, wie wir im letzten Abschnitt erwähnten, die Approximation mit der h^+-Heuristik überein.
Die h^+-Heuristik wurde mit den im letzten Abschnitt vorgestellten Verbesserungen implementiert. Jede der Optimierungen kann unabhängig von den anderen ein- und ausgeschalten werden, um die Effizenz zu überprüfen. Die Experimente wurden auf einem Computer mit acht Intel Xeon Prozessoren mit einer Taktfrequenz von 2.66 GHZ durchgeführt. Falls nach 30 Minuten kein optimaler Plan gefunden wurde oder der Speicherbedarf 2 Gigabyte überschritten hat, wurde die Suche abgebrochen. Folgende vier Optimierungen wurden in verschiedenen Kombinationen getestet:

a: Entferne dominierte Fahrzeuge.

b: Ordne die Orte anhand der Anzahl der Zielorte ihrer Güter.

c: Berechne Zusammenhangskomponenten mit einem Fahrzeug anhand der Approximation.

d: Nutze die Approximation, um den rekursiven Aufruf von L-Semi-Naiv abzubrechen.

keine Optimierung	a	b	c	d	a, b
1398.25	1462.38	1494.79	897.64	1407.48	1554.17

b, c	c, d	a, b, c	b, c, d	a, b, c, d
897.04	907.32	898.57	899.24	**895.99**

Abbildung 8.7: Aufsummierte Lösungsdauer der Optimierungen der h^+-Heuristik auf den gelösten Instanzen des Jahres 2000

keine Optimierung	a	b	c	d	a, b
31.80	32.36	32.48	**27.91**	31.87	33.07

b, c	c, d	a, b, c	b, c, d	a, b, c, d
28.00	28.10	28.07	28.19	28.05

Abbildung 8.8: Aufsummierte Lösungsdauer der Optimierungen der h^+-Heuristik auf den gelösten Instanzen des Jahres 1998

Es sei vorab bemerkt, dass weder bei den 98er noch den 2000er Instanzen eine der Verbesserungen mehr Instanzen lösen konnte als eine andere. Eine Übersicht über die Gesamtlösungsdauer bei den gelösten Instanzen bei verschiedenen Kombinationen der Optimierungen findet sich in den Abbildungen 8.7 und 8.8. Diese Ergebnisse zeigen, dass die Optimierungen a, b und d in der Praxis fast keinen Einfluss auf die Lösungsdauer haben, Optimierung c bringt dagegen einen signifikanten Geschwindigkeitsgewinn. Da das Zuschalten aller Optimierungen bei den 2000er Instanzen in der besten und in den 98er Instanzen der drittbesten Gesamtlösungsdauer resultierte, werden für die weiteren Vergleiche alle Optimierungen verwendet.
Die optimierte Version der h^+-Heuristik wurde auf den 98er und den 2000er Instanzen mit der Approximation und der Merge-And-Shrink-Heuristik verglichen. Die Approximation ist, wie wir in Satz 8.5 gezeigt haben, nicht monoton und findet somit nicht immer eine optimale Lösung. Glücklicherweise wurde der zugrundeliegende Planer so programmiert, dass er diese Fälle erkennt und die Suche abbricht, so dass wir sie trotzdem für die Vergleiche verwenden können. Für die Merge-And-Shrink-Heuristik wurde ein optimierter Parameter – $N = 200000$ – verwendet [HHH07]. Die Vergleiche bezüglich Anfangswertschätzung, Lösungsdauer und Anzahl der expandierten Knoten findet sich für die 2000er Instanzen in den Abbildungen 8.9, 8.10 und 8.11, für die 98er Instanzen in den Abbildungen 8.13, 8.14 und 8.15. Es werden nur Instanzen dargestellt, die von einer der Heuristiken gelöst wurden. Wenn eine Heuristik eine Instanz nicht lösen konnte, wird dies durch einen Strich dargestellt. Bei der Anfangswertschätzung fehlt die Approximation, da sie im Anfangszustand, wie in Satz 8.5 gezeigt, mit der h^+-Heuristik

übereinstimmt. Ebenfalls weggelassen wurden die Anzahl der expandierten Knoten der Approximation auf den 2000er Instanzen, sie stimmten auf diesen Instanzen stets mit denen der h^+-Heuristik überein. Eine Übersicht über die gelösten Probleme der h^+-Heuristik auf den 2000er Instanzen findet sich in Abbildung 8.12, über die der 98er Instanzen in Abbildung 8.16.
Bei den Instanzen 14-0, 14-1 und 15-1 des Planungswettbewerbs 2000 brach der Planer aufgrund einer Monotonieverletzung der Approximation ab, bei den restlichen von der Approximation ungelösten Instanzen dieses Jahrgangs war die Speichergrenze erreicht. Bei den Instanzen 2, 15, 16, 19, 23 und 34 des Planungswettbewerbs 1998 brach der Planer aufgrund einer Monotonieverletzung der Approximation ab, bei Instanz 11 war die Speichergrenze erreicht und bei den restlichen von der Approximation ungelösten Instanzen dieses Jahrgangs waren die 30 Minuten vorbei, ohne dass eine Lösung gefunden wurde. Bei allen von der Merge-And-Shrink ungelösten Instanzen des Planungswettbewerbs 2000 war die Speichergrenze erreicht. Bei den Instanzen 6, 9, 10, 12, 13, 14, 18, 19, 20, 21, 22, 23, 26, 27, 28, 29, 34 und 35 des Planungswettbewerbs 1998 war bei der Merge-And-Shrink-Heuristik die Speichergrenze erreicht, bei den restlichen ungelösten Instanzen dieses Jahrgangs waren die 30 Minuten vorbei, ohne dass eine Lösung gefunden wurde. Bei allen von der h^+-Heuristik ungelösten Instanzen des Planungswettbewerbs 2000 waren die Speichergrenze erreicht. Bei den Instanzen 11, 15 und 34 des Planungswettbewerbs 1998 war bei der h^+-Heuristik die Speichergrenze erreicht, bei den restlichen ungelösten Instanzen dieses Jahrgangs waren die 30 Minuten vorbei, ohne dass eine Lösung gefunden wurde.

Problemname	optimale Planlänge	h^+-Heuristik	Merge-And-Shrink-Heuristik, N=200000
Problem 4-0	20	19	**20**
Problem 4-1	19	17	**19**
Problem 4-2	15	13	**15**
Problem 5-0	27	25	**27**
Problem 5-1	17	15	**17**
Problem 5-2	8	8	8
Problem 6-0	25	23	**25**
Problem 6-1	14	13	**14**
Problem 6-2	25	23	**25**
Problem 6-9	24	21	**24**
Problem 7-0	36	33	**36**
Problem 7-1	44	39	**44**
Problem 8-0	31	29	**31**
Problem 8-1	44	41	**44**
Problem 9-0	36	33	**36**
Problem 9-1	30	29	**30**
Problem 10-0	45	41	**45**
Problem 10-1	42	39	**42**
Problem 11-0	48	45	**48**
Problem 11-1	60	55	**59**
Problem 12-0	42	39	**42**
Problem 12-1	68	-	**68**

Abbildung 8.9: Vergleich der Bewertung des Anfangszustands der h^+- und der Merge-And-Shrink-Heuristik auf den 2000er Instanzen

Problemname	optimale Planlänge	h^+-Heuristik	Approximation	Merge-And-Shrink-Heuristik, N=200000
Problem 4-0	20	**0.00**	0.01	0.07
Problem 4-1	19	**0.00**	0.01	0.06
Problem 4-2	15	0.00	0.00	0.06
Problem 5-0	27	0.05	**0.04**	0.72
Problem 5-1	17	**0.00**	0.01	0.75
Problem 5-2	8	0.00	0.00	0.74
Problem 6-0	25	0.06	**0.05**	3.24
Problem 6-1	14	0.00	0.00	3.33
Problem 6-2	25	0.03	0.03	3.31
Problem 6-9	24	0.03	0.03	3.27
Problem 7-0	36	**0.72**	0.73	19.06
Problem 7-1	44	11.33	**11.22**	19.08
Problem 8-0	31	0.32	0.32	26.75
Problem 8-1	44	3.87	3.87	28.17
Problem 9-0	36	1.62	**1.57**	37.58
Problem 9-1	30	**0.10**	0.11	37.49
Problem 10-0	45	**26.39**	26.46	79.12
Problem 10-1	42	**24.88**	24.96	86.22
Problem 11-0	48	28.81	**28.73**	87.99
Problem 11-1	60	775.53	775.10	**187.85**
Problem 12-0	42	**22.25**	22.29	117.98
Problem 12-1	68	-	-	**137.67**

Abbildung 8.10: Vergleich der Lösungsdauer der Approximation, der h^+- und der Merge-And-Shrink-Heuristik auf den 2000er Instanzen

Problemname	optimale Planlänge	h^+-Heuristik	Merge-And-Shrink Heuristik, N=200000
Problem 4-0	20	76	**21**
Problem 4-1	19	195	**20**
Problem 4-2	15	53	**16**
Problem 5-0	27	936	**28**
Problem 5-1	17	181	**18**
Problem 5-2	8	9	9
Problem 6-0	25	934	**26**
Problem 6-1	14	35	**15**
Problem 6-2	25	516	**26**
Problem 6-9	24	537	**25**
Problem 7-0	36	7751	**37**
Problem 7-1	44	155289	**45**
Problem 8-0	31	3269	**32**
Problem 8-1	44	43665	**45**
Problem 9-0	36	14090	**37**
Problem 9-1	30	707	**31**
Problem 10-0	45	**193846**	196342
Problem 10-1	42	**165006**	518215
Problem 11-0	48	156585	**12822**
Problem 11-1	60	5649882	**2608870**
Problem 12-0	42	**116555**	272878
Problem 12-1	68	-	**828540**

Abbildung 8.11: Vergleich der expandierten Knoten der h^+- und der Merge-And-Shrink-Heuristik auf den 2000er Instanzen

Problemname	optimale Planlänge	Anfangswert	expandierte Knoten	Zeit
Problem 4-0	20	19	76	0.00
Problem 4-1	19	17	195	0.00
Problem 4-2	15	13	53	0.00
Problem 5-0	27	25	936	0.05
Problem 5-1	17	15	181	0.00
Problem 5-2	8	8	9	0.00
Problem 6-0	25	23	934	0.06
Problem 6-1	14	13	35	0.00
Problem 6-2	25	23	516	0.03
Problem 6-9	24	21	537	0.03
Problem 7-0	36	33	7751	0.72
Problem 7-1	44	39	155289	11.33
Problem 8-0	31	29	3269	0.32
Problem 8-1	44	41	43665	3.87
Problem 9-0	36	33	14090	1.62
Problem 9-1	30	29	707	0.10
Problem 10-0	45	41	193846	26.39
Problem 10-1	42	39	165006	24.88
Problem 11-0	48	45	156585	28.81
Problem 11-1	60	55	5649882	775.53
Problem 12-0	42	39	116555	22.25

Abbildung 8.12: Übersicht über die gelösten Probleme der h^+-Heuristik auf den 2000er Instanzen

Problemname	optimale Planlänge	h^+-Heuristik	Merge-And-Shrink-Heuristik, N=200000
Problem 01	26	24	**25**
Problem 05	22	**22**	20
Problem 17	42	**42**	-
Problem 31	13	12	**13**
Problem 32	20	18	**20**
Problem 33	27	25	**27**
Problem 35	30	**29**	-

Abbildung 8.13: Vergleich der Bewertung des Anfangszustands der h^+- und der Merge-And-Shrink-Heuristik auf den 98er Instanzen

Problemname	optimale Planlänge	h^+-Heuristik	Approximation	Merge-And-Shrink-Heuristik, N=200000
Problem 01	26	**2.91**	3.16	67.64
Problem 05	22	0.03	**0.02**	99.94
Problem 17	42	**0.62**	44.28	-
Problem 31	13	0.02	**0.01**	7.22
Problem 32	20	0.02	0.02	1.97
Problem 33	27	21.80	**21.57**	81.01
Problem 35	30	**2.65**	2.66	-

Abbildung 8.14: Vergleich der Lösungsdauer der Approximation, der h^+- und der Merge-And-Shrink-Heuristik auf den 98er Instanzen

Problemname	optimale Planlänge	h^+-Heuristik	Approximation	Merge-And-Shrink-Heuristik, N=200000
Problem 01	26	**8623**	9459	375885
Problem 05	22	30	30	527498
Problem 17	42	**67**	8990	-
Problem 31	13	68	68	**14**
Problem 32	20	116	116	**21**
Problem 33	27	88629	89429	**28992**
Problem 35	30	1682	1682	-

Abbildung 8.15: Vergleich der expandierten Knoten der Approximation, der h^+- und der Merge-And-Shrink-Heuristik auf den 98er Instanzen

Problemname	optimale Planlänge	Anfangswert	expandierte Knoten	Zeit
Problem 01	26	24	8623	2.91
Problem 05	22	22	30	0.03
Problem 17	42	42	67	0.62
Problem 31	13	12	68	0.02
Problem 32	20	18	116	0.02
Problem 33	27	25	88629	21.80
Problem 35	30	29	1682	2.65

Abbildung 8.16: Übersicht über die gelösten Probleme der h^+-Heuristik auf den 98er Instanzen

Kapitel 9

Schlusswort

Wir haben gesehen, dass für vier der sieben Domänen die h^+-Heuristik in polynomieller Zeit berechnet werden kann, für die restlichen drei (Miconic-Simple-ADL, Satellite und Logistics) ist dies NP-äquivalent. Wir haben bei einer Abänderung der Satellite-Aufgabenstellung gesehen, dass das Finden eines relaxierten Planes nicht immer leichter sein muss als das Finden eines nicht-relaxierten Planes. Für fünf der sieben Domänen habe ich die Heuristik implementiert, bei zwei (Miconic-Simple-ADL und Schedule) habe ich aufgrund der über STRIPS hinausgehenden Anforderungen in den Problembeschreibungen darauf verzichtet.
In Abbildung 9.1 findet sich ein Überblick über den Schwierigkeitsgrad des Längen-Problems in der nicht-relaxierten und der relaxierten Domäne. Im Falle der NP-Vollständigkeit des Längen-Problems gilt die entsprechende Aussage der NP-Äquivalenz für das Finden eines optimalen Plans, wie im Grundlagenkapitel erläutert wurde. In Abbildung 9.2 findet sich ein Überblick über die Anzahl der gelösten Instanzen der h^+- und der Merge-And-Shrink-Heuristik. Bei den Logistics-Instanzen aus dem Jahr 2000 konnte die Merge-And-Shrink-Heuristik eine Instanz mehr lösen als die h^+-Heuristik. Bei allen anderen Domänen waren die von der h^+-Heuristik gelösten Instanzen Obermengen der von der Merge-And-Shrink-Heuristik gelösten.
Wir können jetzt die Fragestellung aus der Einleitung aufgreifen und kontrollieren, ob sich die Grenzwertaussagen über das Verhältnis von Planlänge in der nicht-relaxierten und der relaxierten Domäne in der Praxis bestätigen. Eine Auflistung nach den untersuchten Domänen findet sich in Abbildung 9.3. Es flossen dabei alle gelösten Instanzen der jeweiligen Domäne in den praktischen Grenzwert ein, d.h. bei Miconic-Strips wurden auch die nicht-gelisteten Instanzen verwendet und bei Blocksworld sind die schwierigen Instanzen, die nicht Teil des IPC-Wettbewerbs sind, nicht dabei. Das praktische Verhältnis ist bis auf die Gripper-Domäne deutlich besser als der theoretische Grenzwert. Bei der Interpretation dieses Wertes ist jedoch Vorsicht

Domäne	nicht-relaxierte Version	relaxierte Version
Miconic-Strips	NP-vollständig	$\in$P
Miconic-Simple-ADL	NP-vollständig	NP-vollständig
Schedule	$\in$P	$\in$P
Gripper	$\in$P	$\in$P
Blocksworld	NP-vollständig	$\in$P
Satellite	NP-vollständig	NP-vollständig
Logistics	NP-vollständig	NP-vollständig

Abbildung 9.1: Schwierigkeit des Längen-Problems

Domäne	Instanzen gesamt	h^+-Heuristik	Merge-And-Shrink-Heuristik
Miconic-Strips	150	142	56
Gripper	20	7	7
Blocksworld	35	30	21
Satellite	36	9	6
Logistics 2000	28	21	22
Logistics 1998	35	7	5

Abbildung 9.2: Vergleich der gelösten Instanzen der beiden Heuristiken

geboten. Dies muss nicht zwangsläufig daran liegen, dass der theoretische Wert zu pessimistisch für die Praxis ist. Vor allem bei der Satellite- und der Logistics-Domäne, in denen die h^+-Heuristik über einen exponentiellen Algorithmus implementiert wurde, kann dies eine andere Ursache haben. Aufgrund der langen Laufzeit können innerhalb der Zeitvorgabe nur wenige Knoten expandiert werden. Dies führt dazu, dass nur diejenigen Instanzen gelöst werden können, bei denen für die Lösung nicht deutlich mehr Aktionen nötig sind als in der Anfangsschätzung.
Die in dieser Arbeit erreichten Resultate zeigen auf, dass es sich lohnen würde, eine domänenunabhängige Version der h^+-Heuristik zu implementieren. Dabei sollte man sich aber hüten, allzu euphorisch zu sein, die Ergebnisse werden sich nicht direkt auf eine allgemeine Implementierung übertragen lassen. In das Finden des Heuristikwertes flossen viele Informationen aus der Domäne mit ein. Diese Informationen zu sehen ist einem Menschen möglich, ein Computer wird die wichtigen Informationen bei domänenunabhängigem Planen kaum erkennen. Dementsprechend könnte es bei einer domänenunabhängigen Version passieren, dass für die von uns polynomiell berechenbaren relaxierten Domänen nur in exponentieller Zeit der h^+-Wert berechnet werden kann. Die guten Laufzeitergebnisse werden sich also eventuell nicht wiederspiegeln. Die Anzahl der expandierten Knoten ist dagegen unabhängig von der Implementierung und wird somit auch bei einer domänenunabhängigen Version übereinstimmen. Da der Speicher für gewöhnlich ein größeres

Domäne	theoretischer Grenzwert $\lvert h^+\rvert/\lvert h^*\rvert$	experimenteller Wert $\lvert h^+\rvert/\lvert h^*\rvert$
Miconic-Strips	$2/3 \simeq 0.6667$	$6924/6997 \simeq 0.9896$
Miconic-Simple-ADL	$3/4 = 0.7500$	-
Schedule	$1/4 = 0.7500$	-
Gripper	$2/3 \simeq 0.6667$	$147/203 \simeq 0.7241$
Blocksworld	$1/4 = 0.2500$	$464/724 \simeq 0.6409$
Satellite	$1/2 = 0.5000$	$154/162 \simeq 0.9506$
Logistics 2000	$3/4 = 0.7500$	$600/652 \simeq 0.9202$
Logistics 1998	$3/4 = 0.7500$	$172/180 \simeq 0.9556$

Abbildung 9.3: Theoretisches und praktisches Verhältnis zwischen Anfangswertschätzung und optimaler Planlänge

Nadelöhr ist als die Rechenleistung/Laufzeit, ist eine allgemeine Implementierung dennoch lohnenswert.

Literaturverzeichnis

[Bac01] Fahiem Bacchus. The AIPS'00 Planning Competition. *AI Magazine*, 22(3):47–56, 2001.

[Byl94] Tom Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69:165–204, 1994.

[GN92] Naresh Gupta und Dana S. Nau. On the Complexity of Blocks-World Planning. *Artificial Intelligence*, 56(2–3):223–254, 1992.

[HE05] Jörg Hoffmann und Stefan Edelkamp. The Deterministic Part of IPC-4: An Overview. *Journal of Artificial Intelligence Research*, 24:519–579, 2005.

[Hel01] Malte Helmert. On the Complexity of Planning in Transportation and Manipulation Domains. Diplomarbeit, Albert-Ludwigs-Universität Freiburg, 2001.

[Hel03] Malte Helmert. Complexity results for standard benchmark domains in planning. *Artificial Intelligence*, 143(2):219–262, 2003.

[Hel04] Malte Helmert. A Planning Heuristic Based on Causal Graph Analysis. In Shlomo Zilberstein, Jana Koehler und Sven Koenig, Hrsg., *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004)*, Seiten 161–170. AAAI Press, 2004.

[Hel08] Malte Helmert. *Understanding Planning Tasks – Domain Complexity and Heuristic Decomposition*, Jgg. 4929 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2008.

[HET+06] Jörg Hoffmann, Stefan Edelkamp, Sylvie Thiébaux, Roman Englert, Frederico dos Santos Liporace und Sebastian Trüg. Engineering Benchmarks for Planning: the Domains Used in the Deterministic Part of IPC-4. *Journal of Artificial Intelligence Research*, 26:453–541, 2006.

[HHH07] Malte Helmert, Patrik Haslum und Jörg Hoffmann. Flexible Abstraction Heuristics for Optimal Sequential Planning. In Mark Boddy, Maria Fox und Sylvie Thiébaux, Hrsg., *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS 2007)*, Seiten 176–183. AAAI Press, 2007.

[HM08] Malte Helmert und Robert Mattmüller. Accuracy of Admissible Heuristic Functions in Selected Planning Domains, 2008.

[HR08] Malte Helmert und Gabriele Röger. How Good is Almost Perfect? In Dieter Fox und Carla P. Gomes, Hrsg., *AAAI*, Seiten 944–949. AAAI Press, 2008.

[LF03] Derek Long und Maria Fox. The 3rd International Planning Competition: Results and Analysis. *Journal of Artificial Intelligence Research*, 20:1–59, 2003.

[LKS+00] Derek Long, Henry Kautz, Bart Selman, Blai Bonet, Héctor Geffner, Jana Koehler, Michael Brenner, Jörg Hoffmann, Frank Rittinger, Corin R. Anderson, Daniel S. Weld, David E. Smith und Maria Fox. The AIPS-98 Planning Competition. *AI Magazine*, 21(2):13–33, 2000.

[McD00] Drew McDermott. The 1998 AI Planning Systems Competition. *AI Magazine*, 21(2):35–55, 2000.

[RHW08] Silvia Richter, Malte Helmert und Matthias Westphal. Landmarks Revisited, 2008.

[RN03] Stuart Russell und Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2. Auflage, 2003.

[Tar72] Robert Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[Weg99] Ingo Wegener. *Theoretische Informatik - eine algorithmenorientierte Einführung*. Teubner, 2. Auflage, 1999.