Albert-Ludwigs-Universität Freiburg
Technische Fakultät
Institut für Informatik

Diplomarbeit

Vereinfachung numerischer Planungsprobleme

Stefan Schleipen

29. April 2009

Betreut von
Prof. Dr. Bernhard Nebel
Dr. Malte Helmert

Inhaltsverzeichnis

1 Einleitung **6**

2 PDDL **7**
2.1 Domänenbeschreibung in PDDL 7
2.2 Problembeschreibung in PDDL 8
2.3 Numerische Erweiterungen 9

3 Theoretischer Teil **10**
3.1 Numerische Variablen und Konstanten 10
3.2 Formale Analyse 12
3.3 Methoden zur Reduktion 15

4 Implementation **20**
4.1 Optimal 20
4.2 Approximierend 22
4.3 Hilfsfunktionen 26

5 Experimenteller Teil **28**
5.1 Depot 29
5.2 Satellite 31
5.3 Zenotravel 33
5.4 Woodworking 35
5.5 Transport 38
5.6 Ergebnisse der Konvertierung 41

6 Fazit **47**

Danksagung

An dieser Stelle möchte ich mich bei Professor Nebel für die Möglichkeit bedanken diese Diplomarbeit am seinem Lehrstuhl, Grundlagen der Künstlichen Intelligenz, verwirklichen zu können.

Weiter bedanke ich mich bei Dr. Malte Helmert für seine sehr gute Betreuung und das er trotz etlicher Deadlines immer Zeit und etwas Kaffee für mich gefunden hat. Auch großen Dank für seine Geduld vor allem und gerade weil er es sicherlich nicht immer einfach mit mir hatte :-)

Dank geht auch an Dr. Kai O. Arras, der so freundlich war mich in einem seiner Büros niederzulassen und an Matthias Luber, der es so lange mit mir in selbigen ausgehalten hat.

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.

Stefan Schleipen
Freiburg im Breisgau, April 2009

Zusammenfassung

Obwohl bei internationalen Planungskonferenzen, z.B. der International Planning Competition (IPC), neben klassischen Problemen auch numerische Problemstellungen behandelt werden, ist die Zahl numerischer Planer in den letzten Jahren überschaubar geblieben. Im Gegensatz zu den klassischen Planern, deren Anzahl seither gestiegen ist. So ist auch die Qualität klassischer Planer im Vergleich zu den Numerischen wesentlich besser. Inhalt dieser Arbeit ist es numerische Probleme so zu verändern, dass sie mit klassischen Planern gelöst werden können. Dazu müssen die numerischen Teile des Problems extrahiert, konvertiert und wiederum in die Problemstellung zurückgeführt werden. In diesem Zusammenhang erfolgt eine Analyse der gestellten Probleme zur Erkennung numerischer Variablen und Konstanten. Es werden Methoden zur Reduktion der Wertebereiche (optimal bzw. approximierend) und zur Konvertierung von numerischen Problemen in STRIPS konforme Probleme vorgestellt.

1 Einleitung

Das Interesse an einer automatisierten Lösung eines Problems hat seit der Computerisierung zunehmend an Bedeutung gewonnen. Besonders im Bereich der Logistik werden mit Hilfe von Planern die immer komplexer werdenden Probleme gelöst [FL03]. Um diese Probleme verarbeiten zu können, müssen sie zunächst formalisiert werden. Probleme werden durch eine Domäne und eine konkret zu lösende Aufgabe gegeben. Eine Domäne beschreibt die Umwelt, welche aus Objekten und Aktionen besteht. Eine Aufgabe bestimmt welche Objekte existieren, wie diese zu Beginn zueinander stehen und wie der zu erreichende Endzustand aussehen soll. Aktionen werden dazu verwendet Zustände einer Domäne zu verändern. Ist das Problem formalisiert, kann mit einem Planer der Plan erzeugt werden, der die Reihenfolge von Aktionen bestimmt, die vom Ausgangs- zum Endzustand führen. Die Lösung einer Aufgabe muss nicht eindeutig sein. Ein Planer kann mehrere Pläne erzeugen. Von Interesse sind dabei die optimalen oder annähernd optimalen Pläne. Ob ein Plan und die damit verbundene Lösung optimal ist, wird durch die Definition von Zielvorgaben bestimmt. So kann optimal bedeuten möglichst wenige Aktionen zum Erreichen des Endzustands zu benötigen oder entsprechend viele oder wenige Veränderungen an der Umwelt zu verursachen. Im Laufe der Entwicklung von Planungsalgorithmen sind unterschiedliche Planungsansätze entstanden. In dieser Arbeit werden numerische und später klassische Domänen behandelt. Beim klassischen Planen besteht die Domäne nur aus wahren oder unwahren Aussagen. Bei numerischen Planern werden diese um Zahlenwerte erweitert.

Im Gegensatz zu klassischen Planern ist die Zahl numerischer Planer gering geblieben und keiner der vorhandenen Planer [HG01, DK01, GSS03, WC06] ist in der Lage optimale Lösungen für komplexere Probleme zu finden. Statt einen neuen, verbesserten numerischen Planer zu entwickeln, kann ein vorhandener Planer durch Vereinfachung des Problems zu besseren Lösungen kommen. Ein anderer Ansatz ist, das numerische Problem direkt in ein klassisches Problem zu konvertieren. Der Vorteil dieser Methode ist die Vorteile der klassischen Planer für numerische Probleme zu nutzen.

Vor einer Vereinfachung der numerischen Probleme müssen diese zunächst untersucht werden. Dabei zeigt sich, dass sich viele der numerischen Probleme auf klassische Probleme der Komplexitätstheorie adaptieren lassen. So bestehen die Probleme häufig aus Varianten des Rucksackproblems. Dort müssen Objekte mit unterschiedlichen Gewichten oder Größen in ein anderes Objekt mit begrenzter Kapazität gepackt werden. Unterschiede zum klassischen Rucksackproblem bestehen darin, dass die Objekte teilweise wieder entnommen werden können oder nicht alle Objekte gepackt werden müssen. Im Folgenden werden diese Probleme als Packingprobleme bezeichnet.

Eine weitere Variante, die in vielen der analysierten Domänen Verwendung findet, sind die so genannten Kontingentprobleme. Bei diesen Problemen wird eine anfangs vorhandene Menge an Kontingent verbraucht. Im Unterschied zu den Packingproblemen kann das verbrauchte Kontingent nicht durch eine umgekehrte Aktion wieder hergestellt werden. Außerdem können gleiche Kosten mehrfach verursacht werden, d.h. eine Aktion kann, mehrfach gewählt werden und ihre Kosten nochmals verursachen. Bei einigen Domänen besteht die Möglichkeit das Kontingent durch spezielle Aktionen wieder auf den Ausgangswert zurück zu setzen.

Ziel der Arbeit ist es numerische Probleme zu vereinfachen und diese in klassische zu überführen, um sie anschließend mit einem klassischen Planer zu Lösen. Die Arbeit ist in folgende Kapitel gegliedert. Zuerst erfolgt eine Einführung in PDDL. Im Anschluss folgt die formale Beschreibung numerischer Variablen. Im Kapitel Implementation wird der verwendete Code beschrieben. Abschließend werden die Ergebnisse der Arbeit dargestellt.

2 PDDL

In dieser Arbeit wird die Beschreibungssprache PDDL in der Version 2.1 von Maria Fox und Derek Long [FL03] als Grundlage verwendet. Dieses Kapitel stellt PDDL in seinen für diese Arbeit wichtigen Punkten vor und beschreibt diese genauer. PDDL (*Planning Domain Definition Language*) wurde entwickelt, um die erste *International Planning Competition* (IPC) zu realisieren. PDDL vereint ADL (Action description language) [Ped89], STRIPS (Stanford Research Institute Problem Solver) [FN71, Lif87], sowie einige weitere Sprachen. So wurde eine Sprache geschaffen, die alle Qualitäten der anderen Sprachen in einer gemeinsamen Form bietet. Dies ist die Grundvoraussetzung, um die Leistungsfähigkeit eines Planers mit dem eines anderen zu messen. Eine detailliertere Beschreibung von PDDL findet sich unter Fox und Long [FL03]. Die ursprüngliche Definition der Sprache, siehe [MGH+98], beschreibt die grundlegenden Teile der Sprache, wobei nur einige für diese Arbeit interessant sind. Die nicht verwendeten Teile beziehen sich auf Erweiterungen der Aktionen, Safety Constraints oder Expression Evaluation.

Im Folgenden werden die Merkmale von PDDL vorgestellt und in kurzer Form beschrieben. In einem eigenen Abschnitt werden später noch die numerischen Teile der Sprache besprochen, da numerische Probleme erst in der erweiterten Version 2.1 der Sprache hinzugefügt wurden. Diese Arbeit beschränkt sich dabei auf den *Level 2* der Sprache. *Level2* von PDDL ist die Erweiterung von PDDL um numerische Komponenten. Genaueres dazu findet sich im entsprechenden Abschnitt in diesem Kapitel. Die temporalen Erweiterungen der Sprache (*Level 3*), die ebenfalls mit [FL03] eingeführt wurden, werden in dieser Arbeit nicht beachtet.

2.1 Domänenbeschreibung in PDDL

In diesem Abschnitt werden die von PDDL für Domänenbeschreibungen benötigten Argumente vorgestellt, welche zwingend für klassisches Planen benötigt werden.

:requirements definiert, welche Voraussetzungen ein Planer erfüllen muss, um diese Domäne bearbeiten zu können. In diesem Eintrag wird beispielsweise angegeben, ob es sich um eine Domäne mit numerischen Komponenten handelt. Der entsprechende Eintrag lautet :fluents. Die angegebenen Einträge können wiederum andere :requirements implizieren. Eine vollständige Liste aller verfügbaren Optionen findet sich in [FL03]. Wird :requirements ausgelassen, wird :strips angenommen.

:types deklariert alle Typen, die in der Domäne verwendet werden. Ein Eintrag besteht aus einem Namen und dem Typen dem es angehört. Es ist möglich alle Einträge des gleichen Typs in einer Zeile zu deklarieren, siehe unten. Ebenso können Typen selbst einem anderen Typ angehören. Es gibt eine Reihe fest definierter Typen, Einträge ohne Typ werden als *object* bezeichnet.

$$\begin{aligned} \text{name0} \cdots \text{nameN} &- \text{type0} \\ \text{type0} &- \text{type1} \end{aligned} \tag{1}$$

:predicates besteht aus einer Liste von deklarierten Prädikaten. Eine Deklaration besteht aus einem Prädikatnamen und $n \geq 0$ Variablen. Neben dem Namen der Variable muss zu jeder

Variable der Typ angegeben werden. Der Name darf frei gewählt werden der Typ muss bereits in :types eingeführt worden sein.

$$\text{(predicatename ?var0 - type0 ?var1 - tpye1)} \tag{2}$$

:actions besteht immer aus **:parameters**, **:precondition** und **:effect**. Die Bedeutung der Werte wird im Folgenden ausgeführt.

1. **:parameters** besteht aus einer Liste von Variablen die bei dieser Aktion benötigt werden. Die Variablen werden mit Variablennamen und Typ angegeben. Zulässig sind nur bereits deklarierte Variablen und Typen.
2. **:precondition** enthält alle Bedingungen die zum Ausführen der Aktion erfüllt sein müssen. Die Bedingungen können aus einfachen Variableneinträgen oder Prädikaten bestehen
3. **:effect** gibt an, welche Veränderung nach Ausführen der Aktion mit den in den Effekten aufgeführten Variablen geschehen.

2.2 Problembeschreibung in PDDL

In diesem Abschnitt werden die für eine Problembeschreibung nötigen Einträge beschrieben. Der erste Eintrag einer Problembeschreibung besteht aus der Definition des Problems. Neben dem Namen des Problems wird die zugehörige Domäne festgelegt.

$$define\ (problem\ \text{NAME})\ (:domain\ \text{DOMÄNE}) \tag{3}$$

Anschließend folgen die Einträge der verwendeten Objekte des Initialzustands und dem Zielzustand.

:objects weist jedem Objekt des Problems einen Typ aus der Domänenbeschreibung zu. Es können auch mehrere Objekte eines Problems einem Typ zugewiesen werden. Die Syntax dafür sieht wie folgt aus.

$$\begin{aligned}&(:objects \\ &\ name0\ name1\ ...\ nameN\ -\ Type \\ &\vdots \\ &)\end{aligned} \tag{4}$$

Die Namen dürfen frei gewählt werden, müssen jedoch eindeutig sein. Der Typ muss in der Domänenbeschreibung deklariert sein.

:init besteht aus einer Liste von Propositionen, in denen der Initialzustand beschrieben wird. Diese müssen dabei in der Form angegeben werden, in der sie in den Vorbedingungen der Aktionen verwendet werden (siehe **predicates** unter Domänenbeschreibung).

$$\begin{aligned}&(:init \\ &\ (predicatename\ object0\ \cdots\ objectN) \\ &\vdots \\ &)\end{aligned} \tag{5}$$

:goal definiert den zu erreichenden Zielzustand. Die goal description wird dabei in funktionsfreier Logik erster Stufe angegeben. Diese kann aus einer Konjunktion und/oder Disjunktion von atomaren Formeln bestehen. Mögliche Operatoren sind: *exists*, *forall* oder *not*.

2.3 Numerische Erweiterungen

Mit Version 2.1 von Fox und Long werden in PDDL die Erweiterungen um numerische Komponenten eingeführt. Die Erweiterungen betreffen Domänenbeschreibung und Problembeschreibung.

Domänenbeschreibung

:functions dient dazu, numerische Werte an Funktionen zu binden. Es ist möglich einen Wert einer Funktion mit nur einer Variable zuzuweisen, einer Funktion mit mehreren Variablen oder einer Funktion ohne Variablen. Letztere Funktion wird globale Funktion genannt. Beispiele dafür sind:

$$\begin{aligned} &\text{(functionname0 ?var0 } - \text{ type0)} \\ &\text{(functionname1 ?var1 ?var2 } - \text{ type1)} \\ &\text{(globalfunction)} \end{aligned} \tag{6}$$

Problembeschreibung

:metric oder Plan Metric, ermöglicht durch einen arithmetischen Ausdruck eine zusätzliche goal description anzugeben. Zulässig sind die Operatoren $+, -, *, /$. Der Ausdruck soll von einem Planer minimiert werden.

$$\text{(:metric minimize arithmetic\_expression)} \tag{7}$$

init wird ebenfalls um numerische Werte erweitert. Dabei werden allen Funktionen die entsprechenden Werte zugewiesen. Die Zuweisung beginnt immer mit $=$.

$$\begin{aligned} &(= \text{ (function0) value0)} \\ &(= \text{ (function1 object0 } \cdots \text{ objektN) value1)} \end{aligned} \tag{8}$$

3 Theoretischer Teil

Bevor eine Vereinfachung numerischer Werte möglich ist, müssen diese zunächst durch eine Formel beschrieben werden. Die Ausarbeitung der Formalisierung findet in diesem Kapitel statt. Dazu werden die numerischen Bereiche eines Planungsproblems isoliert betrachtet. Einem Menschen fällt es recht leicht bei einem Problem zu erkennen, welches die entscheidenden Werte sind und ob diese wichtig für das Lösen eines Problems sind. Angenommen man möchte zwei Briefe mit seinem Auto zur Post bringen. Die Gewichte der Briefe und die maximale Zuladung des Autos sind in diesem Fall nicht weiter interessant. Hat man anstatt der Briefe zwei Pakete, die 500kg und 300kg wiegen, sind Gewicht und Zuladung entscheidend zum Lösen des Problems. Wenn das Auto nur eine maximale Zuladung von 600kg hat, müssen die Pakete einzeln gefahren werden. Welches Paket als erstes gefahren wird, ist dabei nicht wichtig. Außerdem besitzt das Auto nach dem Entladen eines Pakets wieder die volle Zuladung. Bei diesem Beispiel ist die wichtige Information das das Auto nur ein Paket zu selben Zeit transportieren kann. Man kann die eigentlichen Gewichte vernachlässigen und jedem Paket den Wert eins zuordnen. Ersetzt man die maximale Zuladung des Autos nun ebenfalls auf eins, sind alle Verhältnisse bezüglich der ursprünglichen Werte wieder hergestellt, die Art und Weise das Problem zu lösen jedoch unverändert. Intuitiv ist dieses Beispiel einem Menschen verständlich, für einen Computer ist diese Erkenntnis allerdings nicht trivial. Um dies zu leisten, werden die im Folgenden gezeigten Formalisierungen und die dazu vorgestellten Verfahren zur Reduktion benötigt.

3.1 Numerische Variablen und Konstanten

Domänbeschreibung

Die Domänen werden in PDDL formuliert. Die wichtigen Bereiche für eine Analyse sind dabei *functions* und *actions*. Eine Variable wird zunächst immer im Bereich für Funktionen deklariert. Aktionen sind die einzigen Bereiche, in denen einer Variable ein neuer Wert zugewiesen werden kann.

Funktionen sind im Rahmen der Analyse von Interesse, da hier alle numerischen Komponenten der Domäne eingeführt werden. Die hier eingeführten Werte lassen sich in Konstanten und Variablen unterteilen. Eine Konstante ist ein Wert, der sich im Laufe eines Plans nicht verändert, wie z.B. das *load_limit* eines *truck* in der Depot Domäne. Zwar ändern sich die aktuellen Zuladungen, diese werden jedoch durch *load_limit* und *current_load* ermittelt und durch *current_load* repräsentiert. Eine Variable zeichnet sich dadurch aus, dass sich ihr Wert im Laufe des Plans verändert. Wobei nicht zwischen monotonen oder fluktuierenden Änderungen unterschieden wird. Eine Unterscheidung zwischen Konstanten und Variablen kann allerdings erst nach Betrachten der Aktionen geschehen. Variablen oder Konstanten, die in diesem Bereich ohne Zuweisung auf eine Variable der Domänenrepresentation deklariert werden, sind in der Regel globale Werte und dienen als eine Art counter. Diese Werte sind für eine weitere Analyse nicht interessant.

Aktionen bestehen aus drei Teilen: Parameter, Vorbedingung und Effekt. Parameter legen fest, welche Variablen für eine Aktion benötigt werden. Enthalten ansonsten aber keine Informationen über numerische Werte. Die Vorbedingungen geben die Zusammenhänge zwischen den Variablen an. Beispielsweise wird an dieser Stelle getestet, ob eine Kiste noch auf einen LKW passt und nicht die Zuladung überschritten wird. Der Entsprechende Test sieht wie folgt aus:

$$(<= \ (+ \ (current\_load \ ?z) \ (weight \ ?y)) \ (load\_limit \ ?z))) \tag{9}$$

Zuerst wird *current_load* mit dem Gewicht der Kiste addiert und anschließend mit *load_limit* des LKW verglichen. Aus diesem Vergleich geht die Abhängigkeit zwischen momentaner und maximaler Zuladung hervor.Weiter lässt sich daraus schließen, dass die momentane Zuladung wahrscheinlich ein variabler Wert ist. Sicherheit darüber kann durch Betrachtung der Effekte erlangt werden. Operatoren, die auf das Verändern einer numerischen Variable hindeuten sind, *increase* und *decrease*. Diese Operatoren erhöhen bzw. verringern einen Wert um eine Konstante oder Variable. Im Fall der Depot Domäne sieht der Eintrag im Effekt entsprechend aus.

$$(increase\ (current\_load\ ?z)\ (weight\ ?y)) \tag{10}$$

current_load ist also eine Variable, die um das Gewicht einer Kiste erhöht werden kann. In diesem Zusammenhang muss getestet werden, ob es sich bei *weight* um eine Konstante handelt oder ebenfalls um eine Variable. Um dies zu testen reicht es aus, in den Effekten aller Aktionen zu betrachten, ob diese Variable in einem davon verändert wird. Ist dies nicht der Fall handelt es sich um eine Konstante. Weiter muss geprüft werden, ob es sich bei *current_load* um eine Variable handelt, die stetig wächst oder auch verringert werden kann. Dazu müssen die Effekte der anderen Aktionen betrachtet werden, ob diese die Variable in einem verringern. Im Falle der Depot Domäne existiert eine entsprechende Aktion, die Unload heißt. Hier wird in den Effekten der *current_load* wieder um das Gewicht einer Kiste verringert.

$$(decrease\ (current\_load\ ?z)\ (weight\ ?y)) \tag{11}$$

Weiter offen bleibt die Frage, in welchem Wertebereich sich dieser Wert bewegt. Bekannt ist nur, dass *current_load* niemals größer als *load_limit* werden kann, da die einzige Aktion in der erhöht wird diese Vorbedingung hat. Bei der Unload Aktion ist keine solche Vorbedingung gegeben. Es kann also sein, dass der Wert unendlich oft verringert werden könnte. Dies kann mit Invariantensuche geklärt werden.

Problembeschreibung

In der Problembeschreibung werden konkrete Werte an numerische Variablen gebunden. Diese sind für eine Analyse weniger brauchbar, da keine Aussage darüber getroffen werden kann, ob es sich um Variablen oder Konstanten handelt. Die numerischen Variablen, die jedoch nicht an ein Objekt gebunden sind, sondern global definiert werden, müssen für eine spätere Analyse behalten werden. Diese Werte sind diejenigen, die potentiell für Vergleiche verwendet werden. Gerade für die Suche nach Konstanten sind diese Werte interessant. Es muss dabei zwischen denen unterschieden werden, die an eine Objektvariable gebunden wurden und denen die frei deklariert werden. Letztere müssen nicht beachtet werden.

Zusammenführen von Domäne und Problem

Durch Instanzieren des Problems werden die Variablen aus den Aktionen durch konkrete Werte ersetzt. Aus jeder zulässigen Kombination von Variablen der Problembeschreibung wird eine Aktion generiert. Das Ergebnis ist eine Menge von möglichen Aktionen. Im Falle der Depot Domäne und des oben beschriebenen Problems der *current_load* Variable ist es jetzt möglich eine Aussage über das Verhalten zu treffen. Es werden alle Aktionen, in deren Effekt eine vorher gewählte Variable vorkommt, gesondert betrachtet. Aus dieser verringerten Menge an Aktionen können nun weitere Teilmengen gebildet werden. Eine Teilmenge bilden dabei Aktionen, in denen die Variable in Verbindung mit einer anderen Variable vorkommt, beispielsweise ein LKW in Verbindung mit einer der Kisten. Für eine Kiste bestünde diese Teilmenge aus genau zwei Aktionen (Load

und Unload). Aus den Vorbedingungen lässt sich feststellen, dass keine der Aktionen mehrmals hintereinander ausgeführt werden kann. Daraus folgt, dass nach Ausführen einer Load Aktion nur eine Unload Aktion zulässig ist und umgekehrt. Bei diesem Beispiel würde sich zeigen, dass beide Aktionen immer im Wechsel ausgeführt werden würden und die Variable immer zwischen ihrem Ausgangswert und dem um das Gewicht der Kiste erhöhten Wert wechselt.

Es gibt in den betrachteten Domänen noch eine zweite Variante von Variablen. Diese charakterisieren sich durch Aktionen, die den Wert einer Variable verringern und mehr als einmal nacheinander ausgeführt werden können. Dies geht aus den Vorbedingungen hervor, die keine Beschränkungen besitzen, ob eine Aktion in Verbindung mit einer vorher bestimmten Variable bereits ausgeführt wurde. Bei manchen Varianten gibt es Aktionen, die den Wert der Variable wieder erhöhen. Diese kommen allerdings nicht in Verbindung mit einer der verringernden Aktionen und deren Variablen vor. Wie bei dem oben erklärten Beispiel lassen sich für diese Variante Grenzwerte durch Vorbedingungen finden, was auf einen endlichen Wertebereich schließen lässt. Eine formale Beschreibung, beider Variablentypen erfolgt im Folgenden.

3.2 Formale Analyse

Um durch Invariantensuche eine numerische Variable finden und einordnen zu können, muss diese formal beschrieben werden. In den folgenden beiden Kapiteln werden die Charakteristiken von Variablen herausgearbeitet und anhand eines Beispiels bewiesen.

Am Beispiel der Variable *current_load* aus der Depot Domäne wird gezeigt, wie eine numerische Variable aussieht und welche Eigenschaften sie besitzt. *current_load* muss sich immer zwischen Null und dem *load_limit* eines LKW befinden. Für *current_load* gibt es genau zwei Aktionen deren Effekte den Wert verändern: Load, die einen *increase* als Effekt hat, und Unload die einen *decrease* bewirkt. Eine Aktion kann nur ausgeführt werden, wenn sie vorher noch nicht ausgeführt wurde oder die entsprechende Gegenaktion zu einem zurückliegenden Zeitpunkt ausgeführt wurde (Eine Kiste kann nicht zwei mal entladen werden ohne zwischendurch wieder aufgeladen worden zu sein). Dass der Wert von *current_load load_limit* nicht übersteigt, wird in den Aktionen durch die Bedingung $(<= \quad (+ \ (current\_load \ ?x) \ (weight \ ?y)) \ (max\_limit \ ?x))$ festgelegt. Werte unter Null sind aufgrund der Bedingung, dass ein Unload nur stattfinden kann, wenn zuvor das entsprechende Load ausgeführt wurde, nicht möglich. Es bleibt zu zeigen, dass sich bei einem Load bzw. Unload *current_load* nur um das Gewicht der Kiste ändert. Die Induktionsbeweise dafür finden sich im folgenden Beispiel.

Nach Induktionsvoraussetzung gilt vor einer Load Aktion:

$$current\_load(x,t) = \sum_{y} at(x,y,t) \cdot weight(y,t) \tag{12}$$

x ist ein beliebiger Lkw, y eine Kiste, $at(x,y,t) \rightarrow \{0,1\}$ gibt an, ob die Kiste bereits zum Zeitpunkt t mit einer Load Aktion auf den LKW geladen wurde. $weight(y,t) \rightarrow \mathbb{R}$ ist das Gewicht der Kiste.

Für die Load Aktion gilt es zu zeigen, dass nach der Aktion zum Zeitpunkt t' *current_load* um exakt das Gewicht der hinzukommenden Kiste erhöht wird:

$$current\_load(x,t') = \sum_{y} at(x,y,t') \cdot weight(y,t') \tag{13}$$

Für *current_load* gilt:

$$current\_load(x,t') = current\_load(x,t) + weight(\tilde{y},t) \tag{14}$$

wobei $\tilde{y}$ das Gewicht der zugeladenen Kiste ist. Die Gewichte sind zu jedem Zeitpunkt konstant. Es gilt daher: $weight(y, t') = weight(y, t)$. Für die rechte Seite gilt:

$$\begin{aligned} & \sum_{y} at(x, y, t') \cdot weight(y, t') \qquad (15) \\ = & \sum_{y \neq \tilde{y}} at(x, y, t) \cdot weight(y, t) + at(x, \tilde{y}, t') \cdot weight(\tilde{y}, t') \\ = & \sum_{y \neq \tilde{y}} at(x, y, t) \cdot weight(y, t) + 1 \cdot weight(\tilde{y}, t') \\ = & \sum_{y \neq \tilde{y}} at(x, y, t) \cdot weight(y, t) + weight(\tilde{y}, t) \\ = & \; current\_load(x, t) + weight(\tilde{y}, t) \\ = & \; linke\ Seite. \end{aligned}$$

Die Trennung von t und t' kann vollzogen werden, da sich von t zu t' an den übrigen $at(x, y, t)$ Zuständen nichts verändert. Das neue Gewicht kann somit aus der Summe gezogen werden. $at(x, \tilde{y}, t')$ bildet auf 1 ab, da das Gewicht von $\tilde{y}$ zur Summe der Gewichte hinzukommt. Die Summe kann nach Induktionsvoraussetzung durch $current\_load(x, t)$ substituiert werden.

Die Induktion der Unload Aktion und dem dadurch ausgelösten *decrease* erfolgt analog zur Load Aktion. Die Induktion sieht wie folgt aus.

$$current\_load(x, t') = \sum_{\alpha} at(x, y, t') \cdot weight(y, t') \qquad (16)$$

Für *current_load* gilt, dass sich der Wert in t' um das Gewicht der entladenen Kiste verringert:

$$current\_load(x, t') = current\_load(x, t) - weight(\tilde{y}, t) \qquad (17)$$

Für die rechte Seite gilt:

$$\begin{aligned} & \sum_{y} at(x, y, t') \cdot weight(y, t') \qquad (18) \\ = & \sum_{y \neq \tilde{y}} at(x, y, t) \cdot weight(y, t) + at(x, \tilde{y}, t') \cdot weight(\tilde{y}, t') \\ = & \sum_{y \neq \tilde{y}} at(x, y, t) \cdot weight(y, t) + 0 \cdot weight(\tilde{y}, t') \\ = & \sum_{y \neq \tilde{y}} at(x, y, t) \cdot weight(y, t) -^{*} weight(\tilde{y}, t) \\ = & \; current\_load(x, t) - weight(\tilde{y}, t) \\ = & \; linke\ Seite. \end{aligned}$$

* Durch entladen der Kiste wird $at(x, \tilde{y}, t')$ zu 0. Dies entspricht einer Subtraktion des Gewichtes der entladenen Kiste von *current_load* zum Zeitpunkt t.

Im Allgemeinen

Allgemein sind sechs Fälle zu betrachten: Zwei Fälle für Packing verwandte Variablen und vier Fälle für Kontingentvariablen. Bei den Packingvariablen besteht der Unterschied in der Begrenzung der Kapazität. Gibt es keine Begrenzung der Variable, so können die numerischen Werte vernachlässigt werden. Die Domäne kann direkt in STRIPS überführt werden. Da es keine Obergrenze gibt ist es möglich alle Objekte, die vorhanden sind, in jedem Zustand komplett aufzunehmen. Deshalb können die Werte entfernt werden. Der zweite Fall sind Packingvariablen mit Grenzwerten. Eine solche Variable hat folgende Form:

$$\varphi(x,n) = \sum_{t=1}^{n} \sum_{y} \alpha_t(x,y) \cdot \omega(y) \tag{19}$$

Wobei $\alpha_t(x,y) \in \{0,1\}$ den Wert 1 hat, wenn zum Zeitpunkt t eine Aktion ausgeführt wird, die y einlädt. $\omega(y)$ stellt die Kosten von Objekt y und $\varphi(x,n)$ der aktuelle Wert der Variable x zum Zeitpunkt n dar.

Kontingentvariablen unterscheiden sich durch die Option die Variable wieder auf ihren Ausgangswert zurück zu setzen. Solche Aktionen werden als *refill* Aktionen bezeichnet. Variablen werden nach den Kriterien kein, teilweiser oder kompletter *refill* unterschieden. Der wesentliche Unterschied zwischen dieser Art von Variable und der in (19) formalisierten, besteht darin, dass eine Aktion und die damit verbundenen Kosten mehrfach anfallen können. Aus diesem Grund müssen die Aktionen in jedem Zeitschritt separat berechnet werden. Die Formel für Variablen ohne die Option eines *refill* hat folgende Form.

$$\varphi(x,n) = \psi(x,0) - \sum_{t=1}^{n} \sum_{y} \alpha_t(x,y) \cdot \omega(y) \tag{20}$$

Wobei $\varphi(x,n) \in \mathbb{R}$ der aktuelle Wert der Variable x zum Zeitpunkt n ist, $\psi(x,0) \in \mathbb{R}$ der Initiale Wert von x zum Zeitpunkt 0, $\alpha_t(x,y) \in \{0,1\}$ eine Aktion zum Zeitpunkt t und $\omega(y) \in \mathbb{R}$ die Kosten von y. $\varphi(x,t)$ ist das zu einem Zeitpunkt t verbleibende Kontingent, das sich aus der Summe der Aktionen die bis zum Zeitpunkt $t = n$ ausgeführt wurden multipliziert mit deren Kosten $\omega(y)$ berechnet, die von $\psi(x,0)$ abgezogen werden. $\psi(x,0)$ ist der Initiale Wert von x zum Zeitpunkt $t = 0$.

Für Variablen die über einen kompletten *refill* wieder auf ihren Ausgangswert gesetzt werden, gilt diese Form ebenfalls. Die Formel stellt dann den jeweiligen Verbrauch zwischen zwei *refill* Intervallen dar. Um dieses Verhalten allgemeiner zu beschreiben wird eine solche Variable wie folgt definiert:

$$\varphi(x,n) = \psi(x,m) - \sum_{t=m}^{n} \sum_{y} \alpha_t(x,y) \cdot \omega(y) \tag{21}$$

Wobei $m-1$ der Zeitpunkt ist, an dem die letzte *refill* Aktion gewählt wurde. Der Wert von $\psi(x,m)$ beträgt das Ausgangsniveau.

Variablen, bei denen ein teilweiser *refill* möglich ist, haben ein leicht anderes Verhalten. Hierbei reicht keine Intervallbetrachtung, es müssen alle *refill* Aktionen betrachtet werden. Die Formel erweitert sich wie folgt:

$$\varphi(x,n) = \psi(x,0) + \sum_{t=0}^{n} \sum_{r} \alpha_t(x,r) \cdot \omega(r) - \sum_{t=0}^{n} \sum_{y} \alpha_t(x,y) \cdot \omega(y) \tag{22}$$

Die Erweiterung der Formel um $\sum_{t=0}^{n} \sum_{r} \alpha_t(x, r) \cdot \omega(r)$ beschreibt die Summe über alle *refill* Aktionen, die bis zum Zeitpunkt $t = n$ ausgeführt wurden. Wobei $\alpha_t(x, r) \in \{0, 1\}$ und $\omega(r) \in \mathbb{R}$. $\omega(r)$ der Wert ist, um den das Kontingent wieder erhöht wird.

Der vierte Fall sind Kontingentvariablen die keinen Grenzwert besitzen. Im Gegensatz zu Packingvariablen führt diese fehlende Einschränkung nicht zu einer Eliminierung der numerischen Variablen. Dieser Fall kann nur in Verbindung mit einem teilweisen *refill* vorkommen. Eine weitere Betrachtung dieser Fälle macht jedoch keinen Sinn, da eine Reduktion nicht möglich ist. Einzig wenn die *refill* Aktion aus der Domäne entfernt wird, wäre ein Entfernen aller Werte denkbar. Dies wäre aber keine äquivalente Konvertierung.

3.3 Methoden zur Reduktion

Nachdem eine numerische Variable durch Invariantensuche gefunden wurde, kann damit begonnen werden deren Wertebereich zu reduzieren. Das Ziel ist es, aus einem numerischen Problem ein äquivalentes Problem zu erzeugen, dass keine numerischen Werte enthält. Dieses kann dann von einem klassischen Planer bearbeitet werden. Je kleiner der Wertebereich der numerischen Variablen des zu überführenden Problems ist, desto kleinere Repräsentationen werden später in der konvertierten Variante des Problems benötigt.

Optimale Lösung

Eine effiziente Methode zur Reduktion bietet Lineare Programmierung, genauer Integer Linear Programing (ILP) [Ber00, BFG+00]. Ein ILP-Solver kann aus einer Menge von Variablen und Bedingungen die Werte der Variablen bestimmen, so dass die gegebenen Bedingungen erfüllt werden. Die Idee diesen Ansatz zur Reduktion der Wertebereiche der Variablen zu verwenden ist, aus den originalen Werten des Problems eine Menge von Bedingungen zu erzeugen und diese dem ILP-Solver zu übergeben. Verbunden mit dem Ziel, die kleinst möglichen Werte für alle Unbekannten zu ermitteln, bekommt man den reduzierten Wertebereich für alle numerischen Konstanten und Variablen. Im Falle von Packingvariablen ist aus dem vorherigen Abschnitt bekannt, dass die Summe der gepackten Werte den Wert der Packingvariable nicht überschreiten dürfen. Eine entsprechende, allgemeine Formulierung sieht wie folgt aus:

$$W \geq \sum_{t \in T} w_t \tag{23}$$

$W \in \mathbb{N}_0$ ist der Wert der Variable, $w_1, \cdots, w_n \in \mathbb{N}_0$ sind die Kosten der zu packenden Objekte. $T \subseteq w_1, \cdots, w_n$ sind alle Teilmengen aus den Kosten, deren Summe kleiner oder gleich dem Wert der Variable ist. Gesucht wird ein W' das die folgende Bedingung erfüllt.

$$W \geq \sum_{t \in T} w_t \leftrightarrow W' \geq \sum_{t \in T} w'_t \tag{24}$$

Neben allen Ungleichungen, die diese Form erfüllen, müssen auch alle Teilmengen die diese nicht erfüllen übergeben werden. Diese erfüllen dann die Bedingung:

$$W < \sum_{t \in T} w_t \leftrightarrow W' < \sum_{t \in T} w'_t \tag{25}$$

Zusammen mit der Optimierungsbedingung minimiere W', werden für alle w_i und W die kleinst möglichen Werte bestimmt. Die Menge der übergebenen Bedingungen entspricht der Potenzmenge der w_i. Bei n Elementen wären dies genau 2^n.

Bei Kontingentvariablen sind die Optimierungsbedingungen gleich formuliert. Gegeben sind: $K \in \mathbb{N}_0$ die Mengen an verfügbarem Kontingent, $k_t \in \mathbb{N}_0$ ein bestimmter Kostenwert und T die Menge aller Kosten. ς_t gibt an, wie oft eine Aktion deren Kosten kleiner oder gleich der Hälfte von K sind, nacheinander ausgeführt werden könnte ohne das Kontingent zu erschöpfen. Kosten können niemals öfter als ihr ς_t anfallen. ς_t berechnet sich aus $\frac{K}{k_t}$, abgerundet.

$$K \geq \sum_{t \in T} \varsigma_t \cdot k_t \leftrightarrow K' \geq \sum_{t \in T} \varsigma_t \cdot k'_t \tag{26}$$

Beim Generieren der Eingabe für den ILP-Solver wird jedes Element aus der Menge der Kosten entsprechend seinem ς_t oft aufgenommen und beim Erstellen der Bedingungen berücksichtigt. Der Grund dafür ist, dass Kosten mehrfach anfallen können und nicht wie bei Packingvariablen nur einmal.

Approximierte Lösung

Obwohl eine optimale Lösung angestrebt werden sollte, bietet dieser Ansatz den entscheidenden Nachteil, dass die Berechnungen für eine große Anzahl an Konstanten und Variablen, exponentielle Laufzeit benötigen. Der Grund dafür ist, dass ILP NP-vollständig ist (siehe Garey und Johnson [GJ79]). Da dieser Ansatz als Vorverarbeitungsschritt angesehen werden soll, sollten Laufzeit und Speicherverbrauch der Vereinfachung entsprechend gering sein.

Verfahren

Die grundlegende Idee eine Menge an Werten und einen dazu in Abhängigkeit stehenden Grenzwert zu verringern ist, einen der Werte um einen bestimmten Wert zu senken und dann zu testen, ob alle Belegungen die mit den alten Werten galten weiterhin gelten. Um eine möglichst gute Näherung zu erreichen ist es sinnvoll die Werte um kleine Beträge zu verringern. Diese Methode kann, mit leichten Unterschieden, auf beide Typen von Variablen angewendet werden. Die Methoden sind im Folgenden genauer beschrieben.

Packingvariablen Um zu Prüfen, ob eine Verringerung eines Werte um einen konstanten Betrag zulässig ist, müssen alle Kombinationen die zuvor nicht den Grenzwert überschritten haben auch danach noch gelten. Die formale Beschreibung der Bedingungen sieht dabei wie folgt aus:

$$\sum_{i=0}^{n} x_i \cdot c_i \leq W \tag{27}$$

Diese Ungleichung beschreibt die Ausgangsbedingung. $c \in \mathbb{N}$ entspricht den Kosten der einzelnen Objekte, $x_i, i \in 0, \cdots, n, x \in \{0, 1\}$ beschreibt ob ein Objekt gepackt ist oder nicht, $W \in \mathbb{N}$ ist der Grenzwert. Wenn einer der Werte um einen bestimmten Wert verringert wird ändert sich die Ungleichung wie folgt.

$$\sum_{0}^{n-1} x_i \cdot c_i + (x_n - C) \leq W \tag{28}$$

$C \in \mathbb{N}$ ist der Wert um den c_i verringert werden soll. Wenn eine Belegung für $x_0, \cdot, x_n$ in (27) gültig ist, muss diese auch in (28) gültig sein. Kann keine Belegung gefunden werden die ein Gegenbeispiel erzeugt, ist die Verringerung zulässig.

Diese Prüfung gilt nur für den linken Teil. Der Test für den Grenzwert sieht wie folgt aus:

$$\sum_{0}^{n} x_i \cdot c_i = W \tag{29}$$

Falls es eine Belegung gibt die genau auf den Grenzwert abbildet, darf dieser nicht verringert werden. Kann keine Belegung gefunden werden, darf W um den Wert 1 verringert werden. Eine Verringerung um einen größeren Wert ist mit dieser Form nicht bewiesen. Es ist allerdings möglich durch eine Änderung des Tests um größere Werte zu verringern. Dazu wird, für den Fall, dass keine erfüllende Belegung gefunden wird, die größte Summe aus allen Werten ermittelt, die nicht größer oder gleich dem Grenzwert ist. W kann dann auf den so größten erzeugten Wert gesetzt werden. Diese Variante der Verringerung wird in einem späteren Abschnitt beschrieben.

Kontingentvariablen Die Approximation für Kontingentvariablen ist ähnlich zur Packing Variante. Die Unterschiede bestehen darin, dass beim Test, ob ein Wert verringert werden kann, x_i nicht nach $\{0, 1\}$ abbildet, sondern nach $\mathbb{N}$. Der Grund dafür ist, wie bereits im Teil über optimale Lösungen erwähnt, dass die Kosten mehrfach anfallen können. Ansonsten ist die Ungleichung äquivalent zu (27) und (28).

$$\sum_{0}^{n} x_i \cdot c_i \leq W \tag{30}$$

$$\sum_{0}^{n-1} x_i \cdot c_i + (x_n - C) \cdot x_n \leq W$$

Die Bedingungen für ein Verringern des Grenzwertes sind bis auf die mehrfach möglichen gleichen Kosten äquivalent zu denen für Packingvariablen.

Verbesserte Methoden Eine einfache Reduzierung der Werte, wie sie für eine approximierende Methode beschrieben wird, führt nicht in allen Fällen zu einer zufriedenstellenden Näherung. Aus diesem Grund ist es notwendig die Methoden zu erweitern. Zum Einen wurde eine Instanz in den Algorithmus eingebaut, die in jedem Rekursionsschritt eine GGT Suche über Menge und Grenzwert durchführt. Die führt im Idealfall dazu, dass die Summe über Menge und Grenzwert wenigstens halbiert wird. Der kleinste von 1 verschiedene gemeinsame Teiler ist 2 und damit würde jeder Wert halbiert. Dieser Schritt ermöglicht es, lokale Minima zu überspringen. Es hat sich gezeigt, dass nur durch das Integrieren dieses Schrittes, die Summe aller Werte in einigen Fällen um ein Vielfaches kleiner ausfiel als mit dem ursprünglichen Verfahren ohne GGT Suche. Der Rechenaufwand für diesen Schritt ist im Vergleich zu dem bisherigen Verfahren gering. Für den Fall, das ein Teiler größer 1 gefunden werden kann, verringert sich die Laufzeit. Es fallen entsprechend weniger Rechenschritte an, bei denen die Werte schrittweise verringert und getestet werden. Ausgehend von dieser Idee wurde die Methode zum Verringern leicht verändert, damit beim Reduzieren gezielt auf gerade Werte verringert wird. Dadurch wird eine effizientere Reduzierung durch GGT erreicht. Erst dann wird, in einem zweiten Durchlauf, die ursprüngliche Methode auf dem bereits verkleinerten Wertebereich ausgeführt. Das diese Methode funktioniert, konnte anhand mehrerer Testläufe gezeigt werden. Beispiele dafür finden sich in Abbildung 1. Neben teilweise deutlich kleineren Wertebereichen zeigte sich eine Verkürzung der Laufzeit. Die in Abbildung 1 gezeigten Werte basieren alle auf Tests mit Approximationen, die GLPK [glp] als Grundlage verwendeten.

Abbildung 1: Unterschiede in Wertebereichsreduktion und Laufzeit zwischen ursprünglicher Anwendung der Methode und vorgestelltem verändertem Durchlauf.

Problem	einfach	Zeit	zweifach	Zeit
Depot p3	481	49855.04	241	3314.24
Depot p4	602	5426.07	295	73.82
Depot p7	376	22727.74	100	3653.76
Depot p13	283	35589.11	283	1178.46
Transport p3	223	3666.78	112	1837.95
Transport p6	513	366.49	424	207.65
ZenoTravel p4	669	35.51	245	35.31

Die zweite Änderung findet während der Evaluation der Daten statt. Es zeigte sich, dass die Reihenfolge der Werte eine erhebliche Auswirkung auf das Ergebnis der Approximation haben kann. Experimente mit permutierter Kostenliste haben gezeigt, dass die Ergebnisse der Approximation teilweise um ein vielfaches besser ausfallen, wenn eine Liste mit möglichen Permutationen der Ursprungsliste verwendet wird. Dabei wurde nacheinander jede der Permutationen mit der Reduktionsmethode bearbeitet und die Ergebnisse gespeichert. Die so gewonnenen Reduktionen unterscheiden sich teilweise stark von einander, trotz dass jede der Listen die selben Elemente jedoch in unterschiedlicher Reihenfolge enthielten. Einige Beispiele für verbesserte Ergebnisse finden sich in Abbildung 2. Neben der Summe der Werte des ursprünglichen Problems findet sich die Summe der Variante ohne permutierte und die mit permutierter Liste. In Abbildung 3 sind außerdem Beispiele für unterschiedliche Ergebnisse bezüglich bester und schlechtester Reduktion.

Abbildung 2: Unterschiede zwischen permutierter und nicht permutierter Menge. oP - ohne Permutation, P - mit Permutation

Problem	original	oP	P
Depot p13	594	581	283
Transport p5	619	608	124
Transport p4	563	524	208

Abbildung 3: Unterschiede zwischen bester und schlechtester Reduktion innerhalb einer Problemstellung, bezogen auf ein Teilproblem wie ein bestimmter LKW, etc.

Problem	bester Wert	schlechtester Wert
Depot p7	100	137
Depot p13	283	586
Transport p3	112	158
Transport p6	424	647

Die Verwendung permutierter Listen macht allerdings nur bei kleineren Mengen Sinn. Bei einer Liste von 10 Elementen wären bereits über 3.6 Millionen Permutationen zu testen. Neben dem Generieren der Permutaionsliste fällt zusätzlich für jede Permutation die Berechnung der Approximation an. Um dieses Problem in den Griff zu bekommen, wird bei Listen mit mehr als 6 Elementen nicht mehr die komplette Permutationsmenge gebildet, sondern eine bestimmte Menge von zufälligen Permutationen der Ursprungsmenge. Der Berechnungsaufwand ist dadurch

geringer als mit dem ursprünglichen Verfahren und der Variante ohne Permutationen vorzuziehen. Neben einer geringeren Berechnungszeit verringert sich allerdings auch die Qualität der Ergebnisse. So konnte bei *transport*, Problem 5 die Summe von 619 auf die Summe von 124 reduziert werden. Bei einer Reduktion mit 100 zufälligen Permutationen lag der kleinste Wert allerdings nur bei 258. Etwa um den Faktor zwei schlechter. Im schlechtesten Fall kann es sein, dass genau die 100 Listen erzeugt werden, die zu denn 100 schlechtesten Reduktionen führen. Die aktuell beste Reduktion etwa liegt bei einer Summe von 504. Dem gegenüber steht allerdings eine Laufzeit von etwa 12.5 Tagen bei vollständiger Permutationsliste und eine Laufzeit von ca. 3 - 8 Minuten für die mit 100 zufälligen Permutationen.

Dynamische Programmierung Wie bereits oben erwähnt kann der Test, ob es möglich ist den Grenzwert abzusenken, durch ein Verfahren ersetzt werden, das aus der Menge von Kosten alle möglichen erreichbaren Werte generiert, die kleiner oder gleich dem Grenzwert sind. Der Vorteil ist ein Test, der weniger Speicher intensiv als glpsolver ist. Ein weiterer Vorteil ist, dass neben dem Grenzwert auch sämtliche Zwischenergebnisse verfügbar sind. Selbst wenn sich der Grenzwert nicht allzu sehr verringern lässt ist es denkbar, dass die Teilmenge der erreichbaren Werte gering ausfällt. Man benötigt also eine geringere Menge an Variablen in der STRIPS Variante für die Konvertierung. Dieser Ansatz kann für beide Variablentypen verwendet werden um zu testen ob der Grenzwert reduzierbar ist. Bei Kontingentvariablen muss jedoch jedes Element mehrfach in die Menge von Kosten aufgenommen werden, wenn dieses kleiner gleich der Hälfte des Grenzwertes sind.

Es ist auch möglich bei Packungsvariablen so den linken Teil der Verringerung zu testen. Da mit Hilfe dynamischer Programmierung aus einer Liste und einem Grenzwert alle möglichen Kombinationen generiert werden, die die Bedingung zum Grenzwert nicht verletzen reicht es die Anzahl der Kombinationen zu zählen und diese mit der Anzahl zu vergleichen, die aus dem selben Grenzwert und einer Menge besteht, welche einen um 1 verringerten Wert enthält. Ist die Anzahl identisch, so ist die Verringerung zulässig. Eine detaillierte Darstellung der Ergebnisse beider Methoden findet sich in den jeweiligen Kapiteln über die Domänen. Dort werden Laufzeiten und prozentuale Reduktion direkt miteinander verglichen.

4 Implementation

In diesem Kapitel werden die für diese Arbeit benötigten Implementationen besprochen. Die Domänen- und Problembeschreibungen lagen zu dieser Arbeit in PDDL vor. Das Extrahieren der Daten erfolgte mit Hilfe des Parser der Teil der Implementation von Gabi Rögers temporalem Plannungsalgorithmus ist. Für das Lösen der Ungleichungen die für den optimalen Ansatz benötigt wurden, wurde das freie Programm *GNU Linear Programming Kit* [glp] kurz GLPK verwendet. Alle weiteren Schritte wurden mit eigens dafür entwickelten Algorithmen bewerkstelligt. Die Implementation dieser Algorithmen wurde in Python realisiert.

4.1 Optimal

Die Implementation für den optimalen Ansatz bedient sich überwiegend der Funktionalität des GLPK. Im Folgenden werden die Algorithmen für das Generieren der Eingaben des GLPK und die Aufrufe, sowie die erzeugten Ausgaben erklärt. Optimal bedeutet dabei, dass die erzeugten Werte das Minimum aller einzelnen Werte darstellen, den die Variablen annehmen können. Dies wird erreicht, indem aus den Ursprungswerten Bedingungen in Form von Ungleichungen gebildet werden. Anhand dieser Bedingungen können dann die minimalen Werte die eben diese erfüllen bestimmt werden.

Eingaben Für GLPK existiert eine eigene Sprache zur Formulierung der zu lösenden Gleichungen. Eine ausführliche Beschreibung der Sprache findet sich unter [glp]. Die Teile der Sprache die für die Eingabe des GLPK benötigt werden, werden in diesem Abschnitt noch einmal erklärt. In der Eingabe des GLPK werden zunächst alle Variablen die Teil der Gleichung sind deklariert. Dies geschieht mit der Zeile:

$$var\ \ VARIABLE,\ \ TYP \tag{31}$$

Mit *var* wird dabei die Variablendeklaration begonnen. $VARIABLE$ ist der Name der Variable, wobei auch Ausdrücke wie $x\{0..5\}$ zulässig sind, welcher fünf Variablen mit den Namen $x1$ bis $x5$ deklariert. TYP bestimmt die Art der Variable, beispielsweise sind hier Ausdrücke wie *binary* oder $>= 0, integer$ zulässig. Ersteres erzeugt eine Binäre Variable, letzteres eine ganzzahlige Variable mit einem Wertebereich von $[0, \infty)$.

$$DO\ obj:\ X \tag{32}$$

Mittels diesem Ausdruck wird die zu erreichende Bedingung gestellt. DO kann dabei *minimize* oder *maximize* sein, welches die in X gegebenen Variablen entweder minimieren oder maximieren soll. Der letzte wichtige Teil ist die Beschreibung der Bedingungen oder in diesem Fall die Gleichungen die erfüllt werden müssen. Es ist möglich beliebig viele Gleichungen anzugeben. Ein entsprechender Eintrag kann wie folgt aussehen:

$$seq0:\ x[0] + x[1] + x[2] + x[3] <= W \tag{33}$$

Das $seq0:$ steht dabei für eine Art Nummerierung und kann frei gewählt werden. Die Restliche Zeile ist die Gleichung, in diesem Beispiel darf die Summe aus $x0$ bis $x3$ nicht größer als W sein.

Die Eingabe wird von Python erzeugt. Die benötigten Informationen werden direkt aus der Problemstellung extrahiert und an die Funktion übergeben. In Abbildung 4 ist die Funktion zur Generierung der Eingabe abgebildet.

Abbildung 4: Code zur Generierung der Eingabe des GLPK

```
def generate_lp_input(list, limit, output):
  max = len(list)
  nout = output + str(int(limit))
  out = open(nout, 'w')
  out_var = "var w{0..%d}, >= 0, integer;\n" % max
  out.writelines([out_var, 'var W, >= 0, integer;\n', 'minimize obj: W;\n', 's.t.\n'])
  counter = 0
  for subset in powerset(enumerate(list)):
    if subset:
      total_set = sum(item for index, item in subset)
      set_vars = ['w[%d]' % index for index, item in subset]
      set_term = ' + '.join(set_vars)
      if total_set <= limit:
        tem =  'ineq%d: %s <= W;\n' % (counter, set_term)
        out.write(tem)
      else:
        tem = 'ineq%d: %s >= W + 1;\n' % (counter, set_term)
        out.write(tem)
    counter += 1
  out.write('end;\n')
```

Aufruf Da glpk in ANSI C programmiert ist, die Implementation jedoch in Python erfolgte, musste GLPK aus Python heraus aufgerufen werden. Dazu wurde die von Python gestellte Methode *subprocess* verwendet. Um *glpsol*, der Name des Programms, aufzurufen, muss diesem eine Eingabe übergeben werden. Dies erfolgt mit dem Parameter *-m* und dann dem Namen der Eingabedatei. Mit dem Parameter *-o* wird bestimmt wie die Ausgabedatei heißt, wird dieser Parameter nicht angegeben erfolgt die Ausgabe an stdout. In Abbildung 5 ist die Methode zum Aufrufen von glpsol abgebildet. Die Ausgabe erfolgt über den in *file* gegebenen Namen, erweitert um den Wert der Variable und das Suffix .log.

Abbildung 5: Aufruf von glpsolver für optimalen Ansatz

```
def run_glpsolverer(set, limit, file):
  generate_lp_input(set, limit, file)
  file = file + str(int(limit))
  cmd = "glpsol -m %s -o %s.log" % (file, file)
  return_code = call(cmd, shell=True)
  if return_code != 0:
    print "LP didn't run smoothly"
```

Beide Funktionen werden sowohl für Packing- als auch Kontingentvariablen verwendet. Für Probleme mit Kontingentvariablen wird noch eine weiter Funktion benötigt. Wie in Kapitel 3 bereits ausgeführt, müssen für Kontingente die Werte der Kosten in Relation zu dem gegebenen Limit eventuell mehrfach eingefügt werden. Dies geschieht mit der in Abbildung 6 gezeigten Funktion. Dabei wird für jeden Wert der Liste geprüft, wie oft er gewählt werden kann ohne das Limit zu übersteigen. Der Wert wird dann entsprechend oft in die Liste aufgenommen.

Abbildung 6: Generieren der erweiterten Kostenliste für Kontingentvariablen

```
def generate_type2_list(list, limit):
  new_list = []
  for l in list:
    fac = int(limit/l)
    while fac:
      new_list.append(l)
      fac = fac - 1
  return new_list
```

Zum Ausführen der Methoden genügt ein einfacher Aufruf von run_glpsolverer, zusammen mit der Übergabe der Kosten Liste und des Grenzwertes. Diese wiederum werden mit Hilfe der Parserfunktion von Gabi Röger extrahiert. Die Methode erzeugt dann eine Datei in der die optimale Lösung ausgelesen werden kann, siehe Abbildung 7.

```
Problem:    depot220
Rows:       4
Columns:    3 (3 integer, 0 binary)
Non-zeros:  8
Status:     INTEGER OPTIMAL
Objective:  obj = 0 (MINimum)

   No.   Row name        Activity     Lower bound   Upper bound
------ ------------    ------------- ------------- -------------
     1 obj                         0
     2 ineq1                       0                          -0
     3 ineq2                       0                          -0
     4 ineq3                       0                          -0

   No. Column name       Activity     Lower bound   Upper bound
------ ------------    ------------- ------------- -------------
     1 w[0]         *              0             0
     2 w[1]         *              0             0
     3 W            *              0             0

Integer feasibility conditions:

INT.PE: max.abs.err. = 0.00e+00 on row 0
        max.rel.err. = 0.00e+00 on row 0
        High quality

INT.PB: max.abs.err. = 0.00e+00 on row 0
        max.rel.err. = 0.00e+00 on row 0
        High quality

End of output
```

Abbildung 7: Ausgabedatei des glpsolverer am Beispiel der Depot Domäne

Unter dem Bereich *Column name* können die optimalen Werte direkt ausgelesen werden. In diesem Fall sind sowohl die Werte für Kosten als auch den Grenzwert 0. Dieses Ergebnis entspricht einer Eliminierung der numerischen Werte.

4.2 Approximierend

Im Folgenden werden die beiden implementierten Approximationsverfahren beschrieben und im Detail besprochen.

glpsolver Im wesentlichen muss beim approximierenden Ansatz zwischen zwei Varianten unterschieden werden. Zum Einen wurde ein Ansatz mit glpsolver als Grundlage zum Testen zulässiger Reduktionen verwendet. Zum Anderen ein Ansatz der mit Hilfe dynamischer Programmierung

testet ob eine Reduktion zuläßig ist. Beide Varianten wurden jeweils für Tests von Packing- und Kontingentvariablen implementiert. Wie beim Ansatz für optimale Lösungen muss glpsolver dabei von Python extern ausgeführt werden. Die entsprechende Methode dazu findet sich in Abbildung 8.

Abbildung 8: Aufruf von glpsolver für Approximation

```
def run_solver(set, limit, target):
  generate_input(set, limit, target)
  cmd = "bash -c 'glpsol -m input.log -o output.log &> /dev/null'"
  return_code = call(cmd, shell=True)
  if return_code != 0:
    print "LP did not run smoothly"
  temp = open('output.log', 'r')
  for line in temp:
    if re.search("INTEGER OPTIMAL", line):
      return False
  return True
```

Wie schon bei der optimalen Methode wird glpsolver mittels *subprocess* von Python gestartet. Für den Fall das glpsolver eine gültige Belegung findet, und somit ein Gegenbeispiel für eine zulässige Reduktion, wird in der Datei output.log nach einer Zeile **INTEGER OPTIMAL** gesucht. Wird diese gefunden gibt run_solver False zurück und die Reduktion nicht vorgenommen. Genaueres dazu findet sich im unteren Abschnitt. Der Aufruf von *generate_input* findet im eigentlichen Code für Packing- bzw. Kontingentvariablen unterschiedlich statt. Beide Methoden sind in Abbildung 9 zu finden.

Der Unterschied zwischen *generate_cont_input* und *generate_pack_input* besteht nur in der dritten Zeile. Dort werden die Variablen entweder als Binäre Variablen oder Integer Variablen deklariert. Das liegt an der Tatsache das bei Packingproblemen die Kosten nur einmal pro Wert anfallen können, ein Paket das einmal geladen ist kann kein zweites mal geladen werden. Bei Kontingentvariablen hingegen können diese Kosten mehrfach anfallen, weshalb der entsprechende Faktor im ganzzahligen Bereich liegen muss.

Die eigentliche Methode die letztlich für eine Reduktion aufgerufen wird findet sich in Abbildung 10. Als Eingabe werden der Methode die Kosten in Form einer Liste und der Grenzwert als Integer übergeben. Der erste Schritt besteht in einer GGT Suche. Die Gründe dafür wurden bereits in Kapitel 3 ausgeführt. Danach werden die Elemente der Liste nacheinander verarbeitet. Ein solcher Verarbeitungsschritt besteht dabei aus einem Test, ob es zulässig ist, das aktuelle Element um 1 zu verringern und falls dies möglich ist das Element zu verringern und mit geänderter Liste rekursiv die Reduktionsmethode erneut aufzurufen. Sobald keines der Elemente mehr reduziert werden kann, wird gegen den Grenzwert getestet. Falls es möglich ist den Grenzwert um 1 zu verringern, wird wiederum ein neuer Aufruf der Reduktionsmethode gestartet. Dies erfolgt rekursiv so lange bis auch der Grenzwert nicht mehr reduziert werden kann. Am Ende der Methode werden schließlich reduzierte Liste und Grenzwert zurückgegeben. Der Test ob ein Element oder der Grenzwert echt größer 1 ist, hat den Hintergrund, dass sobald ein Element den Wert 0 der Test keine Zuverlässigen Ergebnisse mehr liefert.

Wie in Kapitel 3 bereits motiviert wurde, gibt es eine leicht modifizierte Reduktionsmethode. Das Ziel bei dieser Methode ist es, gezielt gerade Werte zu erzeugen um die Eigenschaften der GGT Suche zu verbessern. Die Änderungen beziehen sich dabei nur auf die beiden if Klauseln. Wie bei der ursprünglichen Methode wird dabei getestet ob ein Element größer 1 ist. Bei der modifizierten Methode wird zusätzlich noch getestet, ob das Element ungerade ist. Dies Erfolg durch einen Vergleich ob das Element Modulo 2 ungleich 0 ist.

Abbildung 9: Methoden zur Generierung der Eingabe von glpsolver für Approximationen

```
def generate_cont_input(set, limit, target):
  max = len(set)
  out_var = "var x{0..%d}, >= 0, integer;\n" % (max - 1)
  out = open('input.log', 'w')
  set_max = set_maximum(max)
  out.writelines([out_var, set_max, 's.t.\n'])
  temp = 's0: '
  index = 0
  for position, item in enumerate(set):
    if position == target:
      item = item - 1
    temp = temp + '+%d*x[%d] ' % (item, index)
    index += 1
  temp = temp + '<= %d;\n' % limit
  out.write(temp)
  temp = 's1: '
  index = 0
  for item in set:
    temp = temp + '+%d*x[%d] ' % (item, index)
    index += 1
  temp = temp + '>= %d;\n' % (limit + 1)
  out.writelines([temp, 'end;\n'])

def generate_pack_input(set, limit, target):
  max = len(set)
  out_var = "var x{0..%d} binary;\n" % (max - 1)
  out = open('input.log', 'w')
  set_max = set_maximum(max)
  out.writelines([out_var, set_max, 's.t.\n'])
  temp = 's0: '
  index = 0
  for position, item in enumerate(set):
    if position == target:
      item = item - 1
    temp = temp + '+%d*x[%d] ' % (item, index)
    index += 1
  temp = temp + '<= %d;\n' % limit
  out.write(temp)
  temp = 's1: '
  index = 0
  for item in set:
    temp = temp + '+%d*x[%d] ' % (item, index)
    index += 1
  temp = temp + '>= %d;\n' % (limit + 1)
  out.writelines([temp, 'end;\n'])
```

dp Bei dieser Variante wird auf Prinzipien der Dynamischen Programmierung [] zurückgegriffen. Die Idee ist dabei, mit dp alle erreichbaren Werte zu ermitteln die kleiner oder Gleich dem Grenzwert sind. Dazu werden zunächst alle Elemente mit Null addiert und in eine Menge aufgenommen. In jedem weiteren Schritt wird ein neues Element der Liste auf jedes Element der neuen Menge addiert und alle neu erreichten Werte aufgenommen. Dies geschieht so lange, bis alle Elemente abgearbeitet wurden. Das dabei die bereits verwendeten erreichten Werte benutzt um die neuen zu erzeugen ohne immer von neuem aufzusummieren ist gerade die Idee der Dynamischen Programmierung. Diese Methode wird zum einen für die Bestimmung eines verkleinerten Grenzwertes verwendet, als auch für den Test ob eine Verringerung eines Listenelements zulässig ist. Für die Bestimmung des Grenzwertes wird die Methode in Abbildung 11 verwendet. Aus der gegebenen Liste an Elementen werden alle erreichbaren Werte bestimmt die kleiner gleich dem Grenzwert sind und dann der größte Wert zurück gegeben.

Die zweite Verwendung zum Testen der erreichbaren Teilmengen ist in Abbildung 12 abgebildet. Ausgehend von einer Gegebenen Liste und einem Grenzwert werden alle Kombinationen übernommen, die nicht gößer als der Grenzwert sind. Im Gegensatz zur ersten Methode wer-

Abbildung 10: Methode für glpsolver Ansatz

```
def reduction(list, limit):
  list, limit = find_gcd(list, limit)
  for target, item in enumerate(list):
    if item > 1:
      tmp = run_solver(list, limit, target)
      if not tmp:
        break
      list[target] -= 1
      list, limit = reduction(list, limit)
  if test_limit(list, limit) and limit > 1:
    limit -= 1
    list, limit = reduction(list, limit)
  return list, limit
```

Abbildung 11: Methode zur Bestimmung des kleinsten zulässigen Grenzwertes

```
def derive_max_limit(elements, limit):
  sums = set([0])
  for e in elements:
    for old_value in list(sums):
      if old_value + e <= limit:
        sums.add(old_value + e)
  res = sorted(sums, reverse=True)
  return res[0]
```

den diesmal alle Kombinationen gezählt auch wenn deren Ergebnisse auf die gleichen Werten abbilden. Um zu testen, ob die Reduktion eines Elements der Liste um 1 zulässig ist, reicht es die Kombinationen aus ursprünglicher Liste und modifizierter Liste zu bestimmen. Ist diese bei beiden identisch ist die Reduktion zulässig. Da pro Schritt nur ein bestimmter Wert reduziert wird ist es nicht möglich eine gleiche Anzahl mit unterschiedlichen Kombinationen zu erhalten.

Abbildung 12: Methode zum zählen zulässiger Kombinationen

```
def count_fitting_sums(list, limit):
    sums = defaultdict(int)
    sums[0] = 1
    for l in list:
        for old_value, counts in sums.items():
            if old_value + l <= limit:
                sums[old_value + l] += counts
    return sum(sums.values())
```

Beide Varianten zur approximierten Methode werden mit der in Abbildung 13 gezeigten Funktion gestartet. Der obere Teil regelt die Generierung der Permutationslisten. Für Listen mit weniger als 7 Elementen wird die komplette Permutationsmenge gebildet, bei Listen mit mehr Elementen eine Menge von 100 zufälligen Permutationen. Der untere Teil ist für die Aufrufe der Testmethoden zur Reduktion zuständig. Dabei wird zwischen glpsolver und count Methode unterschieden. Für glpsolver wird zunächst die Variante für gerade Reduktion und dann die ursprüngliche Methode aufgerufen.

Durch die Abbruchbedingung im Aufruf der Approximation verbesserte sich die Laufzeit, bei Problemen deren numerische Elemente komplett eliminiert werden können, merklich. Sobald ein Reduktionsschritt ein Ergebnis liefert, dass gerade der Summe aus $2 \cdot \sum_0^n x_n$, wobei n die Anzahl der Elemente der Liste ist, entspricht, müßen keine weiteren Reduktionen versucht werden. Diese Zahl muss verdoppelt werden, da der Grenzwert gerade der Summe aller Elemente entspricht, wenn alle gleichzeitig aufgenommen werden können. Eine detailierte Liste mit Beispielen findet

Abbildung 13: Aufruf der Approximation am Beispiel Depot

```
if len(weights) < 7:
  pw = create_perm_list(weights)
else:
  pw = random_shuffle_list(weights)

for t in trucks:
  pwn = copy.deepcopy(pw)
  tmp = []
  print 'new truck %s:' % t, pwn
  for p in pwn:
    print p, int(t), sum(p) + int(t)
    print 'experimental approx for: %s' % p
    if method = count:
      nw, nt = packing_solver_count(p, int(t))
    else:
      nw, nt = packing_solver2(p, int(t))
      nw, nt = packing_solver(nw, nt)
    print nw, nt, sum(nw) + nt
    tmp.append(sum(nw) + nt)
    if len(p) * 2 in tmp:
      break
  print 'list of results: ', sorted(tmp)
```

sich in Abbildung 14.

Abbildung 14: Laufzeiten mit und ohne Abbruchbedingung

Problem	ohne Abb.	mit Abb.
Transport p1	9.80	0.80
Depot p1	63.24	4.60
Depot p2	78.08	13.41
Depot p4	5426.07	73.82

4.3 Hilfsfunktionen

Für die Umsetzung der Implementation war es notwendig einige Hilfsfunktionen zu realisieren. Die wichtigsten sollen an dieser Stelle vorgestellt werden.

Random Shuffle und Permutate List Random Shuffle, genauer gesagt *random.shuffle()*, ist eine von Python bereitgestellte Funktion mit der es möglich ist, eine gegebene Liste zufällig zu permutieren. Die daraus entstandene Funktion zum Generieren der Menge von zufälligen Permutationen findet sich in Abbildung 15. Die zweite Funktion wird in dieser Form nicht von Python bereitgestellt und wird dazu verwendet, aus einer gegebenen Liste alle Permutationen zu berechnen. Diese werden dann als Liste von Permutationen zurückgegeben. Die Methode und ihre Hilfsfunktion findet sich ebenfalls in Abbildung 15.

GGT Suche Für die Suche nach dem GGT aus einer Liste von Elementen und einem Grenzwert wurde die in Abbildung 16 dargestellte Methode verwendet. Die Idee ist, die Liste und den Grenzwert, sofern angegeben, zu vereinen unm dann aus einem bereits bestimmten GGT und dem jeweils nächsten Element aus der Liste wiederum den GGT zu bestimmten. Der zuerst bestimmte GGT wird als erstes Element aus der Liste genommen um die Suche einzuleiten.

Abbildung 15: Struktur von random_shuffle_list und permutate_list.

```
def permutate_list(list):
  if len(list) <= 1:
    yield list
  else:
    for item in permutate_list(list[1:]):
      for i in range(len(item)+1):
        yield item[:i] + list[0:1] + item[i:]

def create_perm_list(list):
  res = []
  for p in permutate_list(list):
    res.append(p)
  return res

def random_shuffle_list(list):
  res = []
  counter = 100
  while counter:
    l = copy.deepcopy(list)
    random.shuffle(l)
    res.append(l)
    counter -= 1
  return res
```

Abbildung 16: Struktur der GGT Suche.

```
def find_gcd(list, limit=False):
  tmp = list[:]
  if limit:
    tmp.append(limit)
  ggt = tmp[0]
  for t in tmp:
    ggt = gcd(ggt, t)
  if ggt != 1:
    list = [x/ggt for x in list]
  limit = limit/ggt
  return list, limit

def gcd(a, b):
  if b == 0:
    return a
  else:
    return gcd(b, a % b)
```

5 Experimenteller Teil

In den folgenden Abschnitten werden die Ergebnisse zur Reduktion der Wertebereiche in den verschiedenen Domänen präsentiert. Dabei wurden jeweils das optimale und die nähernden Verfahren angewendet. Alle experimentellen Ergebnisse wurden auf insgesamt vier Rechnern mit identischer Konfiguration berechnet. Die Konfiguration der Rechner bestand aus Intel Core2Duo Prozessoren mit 3Ghz Taktung und 4Gb Arbeitsspeicher. Als Betriebssystem wurde OpenSUSE 64bit in der Version 11.1 verwendet. Für die Messung der Laufzeiten und Speicherauslastung wurde das Programm *memtime* verwendet [mem]. Es wurden nur Ergebnisse von Tests verwendet, die innerhalb des RAM ausgeführt wurden und nicht auf den Swap zurückgegriffen haben.

Die Darstellung der Werte und der reduzierten Varianten findet als Summe aller Werte statt, die Teil des Reduktionsprozesses waren. Im Detail bedeutet das, Summe aller Kosten summiert mit dem Grenzwert. Es wird jeweils in einer Spalte die Summe der Ausgangswerte, die Summer der Werte nach einer optimalen Lösung und die Summe der Werte nach den genäherten Lösungen angegeben. Zellen die ein - enthalten, stellen Fälle dar in denen entweder keine Lösung aufgrund zu großer Hardwareanforderungen möglich war oder die Berechnung anderweitig nicht möglich war.

Die Spalte Zeit enthält die Laufzeiten jedes Durchlaufs, diese gelten dabei für alle Teile eines Problem. Das bedeutet, befinden sich innerhalb einer Problemstellung mehr als eine Variable für die eine Vereinfachung gesucht wurde, stellt der Zeitwert die gesamte Laufzeit dar. Diese Darstellung wurde gewählt, da diese Arbeit versucht eine Vorverarbeitung eines Problems zu erreichen und im Falle einer Anwendung alle Teile des Problems zunächst verarbeitet werden müssen, bevor der eigentliche Planer Anwendung findet. Der Speicherbedarf wurde nicht explizit angegeben, da er bei allen Reduktionen, immer zwischen 35MB und 60MB lag. Diese Werte wurden wie die Laufzeiten mit *memtime* gemessen.

Einige der ursprünglich gewählten Domänen eignen sich nicht für die in dieser Arbeit vorgestellten Ansätze zur Vereinfachung. Zur Vollständigkeit sollen sie an dieser Stelle erwähnt werden.

DriverLog DriverLog ist ähnlich wie die Depot Domäne. Es sollen Päckchen zwischen zwei Orten transportiert werden. Die Unterschiede sind, dass jedes der Fahrzeuge einen Fahrer benötigt. Ein Fahrer muss, bevor er ein Fahrzeug besteigt, zu diesem laufen sofern er sich nicht am selben Ort befindet. Bei dieser Domäne gibt es keine Kontingente für den Verbrauch von Kraftstoff oder eine Limitierung für Zuladung. Bei DriverLog wird nur die Zeit gemessen, die ein Fahrer braucht um zu seinem Fahrzeug zu laufen oder mit diesem von einem Ort zu einem anderen zu fahren. Ein optimaler Planer minimiert dabei die vergangene Zeit. Für laufen und fahren werden Zeiten unterschiedlich gewertet. Bei den Verbindungen zwischen Orten wird zwischen Strecken die nur zu Fuß und Strecken die nur mit einem Fahrzeug zurückgelegt werden können. Zeiten für Be- und Entsteigen des Fahrzeuges gibt es nicht. Da es bei DriverLog keine obere Schranke für die vergangene Zeit gibt, können die beschriebenen Ansätze zur Reduktion nicht angewandt werden.

Rover Die Rover Domäne wurde von der Mars Rover Mission inspiriert. Das Problem beschreibt ein Szenario bei dem ein oder mehrere Rover die Oberfläche eines Planeten erforschen. Dazu können Proben entnommen und verschiedene Orte angefahren werden. Zusätzlich sind die Rover in der Lage miteinander zu kommunizieren. Jeder Rover besitzt dabei sein eigenes Energiekontingent. Ein optimaler Plan sollte dabei die Anzahl der auflade Zyklen am geringsten halten. Die Kosten für mögliche Aktionen sind dabei in der Domänenbeschreibung und nicht wie bei den anderen Domänen im eigentlichen Problem kodiert. Da in diesen Kosten auch eine Aktion mit Kosten von 1 vorhanden ist, sind Versuche eine Reduktion zu finden erfolglos.

5.1 Depot

Problemstellung Depot ist ein Logistikproblem, bei dem Kisten mit Lastkraftwagen (LKW) von verschiedenen Orten an vorbestimmte Orte gebracht werden müssen. Die Kisten können mittels eines Krans be- oder entladen werden. Das Fahren zwischen zwei Orten verbraucht Kraftstoff. Es gibt jedoch keine Obergrenze oder ein Kraftstoffkontingent für einen LKW. Der Wert dient nur dazu die Effizienz eines Planers zu messen. Ein LKW hat zudem eine maximale Zuladung und jede Kisten ein individuelles Gewicht. Die Gewichte und die maximale Zuladung sind als Konstante an das jeweilige Objekt gebunden, d.h. die Werte können sich nicht im Verlauf des Plans ändern. Das Gewicht der Kisten ist für das Beladen durch die Kräne nicht von Belang. Interessant für eine Reduktion der Werte sind, Gewichte der Kisten und die maximale Zuladung eines LKW. Der einzige variable Wert ist die momentane Beladung eines LKW. Dieser Wert berechnet sich aus der Summe der geladenen Kisten und ändert sich nur mit dem Beladen und Entladen der Kisten.

numerische Variablen Da current_load direkt aus den Gewichten für Kisten und der maximalen Zuladung berechnet wird, muss dieser Wert nicht explizit vereinfacht werden. Der Wertebereich ändert sich mit Änderungen der anderen Werte. Für load_limit und die Menge der Gewichte gelten die Bedingungen für Packing Variablen aus Kapitel 3. Die Summe der Gewichte aller geladenen Kisten muss zu jedem Zeitpunkt kleiner oder gleich der Maximalen Zuladung sein.

Ergebnisse der Reduktion Die Ergebnisse der Experimente für eine optimale und approximierte Vereinfachung finden sich in Tabelle 17 und Tabelle 18. Unter **Problem** werden die Probleme nach Nummer aufgelistet, **org.** steht für die Summe der Ausgangswerte, **Red.** für die prozentuale Verbesserung, **opt.** ist die Summe aller Werte nach dem Vereinfachungsschritt mit der optimalen Lösungsmethode und **glp** ist die Summe aller Werte nach einer Vereinfachung mit der approximierenden Methode mit gplsolver, **dp** die Summe aller Werte mit approximierter Methode mit dynamischer Programmierung.

Abbildung 17: Ergebnisse der Depot Domäne, Teil 1

Problem	**org.**	**opt.**	%	Zeit	**glp**	Red.	Zeit	**dp**	Red.	Zeit
1: truck0	420	0	100	0.30	4	99.05	4.60	4	99.05	0.20
1: truck1	317	0	100	0.30	4	98.74	4.60	4	98.74	0.20
2: truck0	598	0	100	0.30	8	99.06	13.41	8	99.06	0.10
2: truck1	577	0	100	0.30	8	98.61	13.41	8	98.61	0.10
3: truck0	847	0	100	0.20	12	98.58	3314.24	12	98.58	9.10
3: truck1	721	7	99.03	0.20	241	66.57	3314.24	481	33.29	9.10
4: truck0	808	0	100	0.30	16	98.02	73.82	16	98.02	0.70
4: truck1	621	76	87.76	0.30	295	52.50	73.82	596	4.03	0.70
5: truck0	954	217	77.25	0.40	946	0.84	33.61	316	66.88	2.10
5: truck1	910	303	66.7	0.40	907	0.33	33.61	903	0.77	2.10
6: truck0	1150	1150	0	1194.04	1150	0	25.06	1150	0	2.60
6: truck1	1117	905	18.98	1194.04	1117	0	25.06	1117	0	2.60
7: truck0	664	9	98.64	0.40	118	82.23	3653.76	478	28.01	14.20
7: truck1	610	5	99.18	0.40	100	83.61	3653.76	376	38.36	14.20
8: truck0	909	0	100	0.40	20	97.80	26.51	20	97.80	0.80
8: truck1	774	180	76.74	0.40	774	0	26.51	774	0	0.80
9: truck0	1254	1254	0	1260.70	1254	0	22.43	1254	0	2.40
9: truck1	1186	1186	0	1260.70	1186	0	22.43	1186	0	2.40
10: truck0	693	0	100	0.30	12	98.27	892.94	12	98.27	4.10
10: truck1	610	18	97.05	0.30	53	94.21	892.94	229	62.46	4.10
11: truck0	885	314	64.52	0.40	885	0	28.51	885	0	1.20
11: truck1	915	450	50.82	0.40	915	0	28.51	914	0.11	1.20
12: truck0	984	984	0	589.46	984	0	27.21	984	0	3.60
12: truck1	840	840	0	589.46	840	0	27.21	840	0	3.60

Abbildung 18: Ergebnisse der Depot Domäne, Teil 2

Problem	**org.**	**opt.**	%	Zeit	**glp**	Red.	Zeit	**dp**	Red.	Zeit
13: truck0	785	0	100	0.30	12	98.47	1178.46	12	98.47	2.20
13: truck1	594	43	92.76	0.30	283	52.36	1178.46	543	8.59	2.20
14: truck0	893	238	73.35	0.40	887	0.67	28.62	880	1.46	2.40
14: truck1	819	289	64.71	0.40	818	0.12	28.62	818	0.12	2.40
15: truck0	1251	1251	0	671.64	1251	0	21.92	1251	0	4.30
15: truck1	1031	1031	0	671.64	1031	0	21.92	1031	0	4.30
16: truck0	477	0	100	0.40	12	97.48	32.01	12	97.48	0.20
16: truck1	435	0	100	0.40	12	97.24	32.01	12	97.24	0.20
16: truck2	528	0	100	0.40	12	97.73	32.01	12	97.73	0.20
16: truck3	503	0	100	0.40	12	97.61	32.01	12	97.61	0.20
17: truck0	888	0	100	0.80	20	97.75	49.81	20	97.75	1.80
17: truck1	687	282	58.95	0.80	683	0.58	49.81	683	0.58	1.80
17: truck2	680	400	41.18	0.80	680	0	49.81	680	0	1.80
17: truck3	717	203	71.69	0.80	717	0	49.81	717	0	1.80
18: truck0	1226	1226	0	1815.91	1226	0	44.41	1226	0	6.70
18: truck1	995	995	0	1815.91	995	0	44.41	995	0	6.70
18: truck2	1080	1080	0	1815.91	1080	0	44.41	1080	0	6.70
18: truck3	1048	1048	0	1815.91	1048	0	44.41	1048	0	6.70
19: truck0	784	48	93.88	0.50	372	52.55	474.45	746	4.85	3.10
19: truck1	847	52	93.86	0.50	398	53.01	474.45	802	5.31	3.10
19: truck2	820	50	93.9	0.50	783	4.51	474.45	780	4.88	3.10
19: truck3	753	46	93.89	0.50	714	5.18	474.45	712	5.44	3.10
20: truck0	1368	1368	0	1706.02	1368	0	49.44	1368	0	7.80
20: truck1	1310	1310	0	1706.02	1310	0	49.44	1310	0	7.80
20: truck2	1300	1300	0	1706.02	1300	0	49.44	1300	0	7.80
20: truck3	1289	1289	0	1706.02	1289	0	49.44	1289	0	7.80
21: truck0	762	0	100	1.21	20	97.38	69.23	20	97.38	7.60
21: truck1	705	52	92.62	1.21	688	2.41	69.23	556	21.13	7.60
21: truck2	623	253	59.39	1.21	621	0.32	69.23	621	0.32	7.60
21: truck3	629	272	56.76	1.21	629	0	69.23	629	0	7.60
21: truck4	637	99	84.46	1.21	637	0	69.23	637	0	7.60
21: truck5	397	0	100	1.21	20	94.96	69.23	20	94.96	7.60
22: truck0	1479	-	-	-	1479	0	85.63	1479	0	17.70
22: truck1	1258	-	-	-	1258	0	85.63	1258	0	17.70
22: truck2	1303	-	-	-	1303	0	85.63	1303	0	17.70
22: truck3	1337	-	-	-	1337	0	85.63	1337	0	17.70
22: truck4	1270	-	-	-	1270	0	85.63	1270	0	17.70
22: truck5	1255	-	-	-	1255	0	85.63	1255	0	17.70

Abbildung 19: Durchschnittliche Reduktion und Laufzeit

	glp solver	app. glp solver	app. dp
Laufzeit	329.31s	461.35s	4.31s
Reduktion	54.67%	36.08%	29.23%

5.2 Satellite

Problemstellung Satellite beschreibt das Problem eines oder mehrerer Satelliten, die Objekte im Raum aufnehmen sollen und die Daten in einem internen Speicher ablegen, um diese dann versenden zu können. Ein Satellit kann dazu sein Objektiv auf ein Objekt ausrichten und eine Aufnahme davon machen. Ein Planer ist dabei besonders effektiv, wenn er alle vorher bestimmten Objekte Katalogisiert und diese im Speicher abgelegt. Wobei er dabei einen möglichst geringen Verbrauch an Energie haben sollte. Der numerische Teil der Satellite Domäne besteht aus dem verfügbaren Speicher jedes einzelnen Satellit, die Speichermenge die ein aufgenommenes Foto verbraucht, die Kosten für das ausrichten des Objektives und das Kontingent an Energie jedes Satellit. Neben normalen Bildern können auch thermographische und spektrale Aufnahmen gemacht werden. Jede Variante verbraucht unterschiedliche Mengen Speicher. Dies wird durch einen Faktor dargestellt, mit dem die Kosten für eine Aufnahme multipliziert werden. Das Energie- bzw. Speicherkontingent eines Satellit kann nicht zurückgewonnen werden, einmal verbraucht steht es nicht mehr zur Verfügung.

numerische Variablen Bei Speicher und Kosten für Aufnahmen handelt es sich um eine Kontingent Variable. Da jedoch eine Funktion *have_image ?d ?m* existiert, die in der :goal Bedingung des Problems abgefragt wird, kann davon ausgegangen werden, dass es sich eher um eine Variable erster Gattung handelt. Damit kann dieser Teil der Domäne mit Hilfe der Methoden für Packingvariablen vereinfacht werden. Die Werte für Energie und die Kosten für das Ausrichten auf Objekte, verhalten sich dagegen wie Variablen der zweiten Gattung.

Ergebnisse der Reduktion Unter **Problem** werden die Probleme nach Nummer aufgelistet, **org.** steht für die Summe der Ausgangswerte, **Red.** für die prozentuale Verbesserung, **opt.** ist die Summe aller Werte nach dem Vereinfachungsschritt mit der optimalen Lösungsmethode und **glp** ist die Summe aller Werte nach einer Vereinfachung mit der approximierenden Methode mit gplsolver, **dp** die Summe aller Werte mit approximierter Methode mit dynamischer Programmierung.

Im Folgenden sind die Tabellen für die Reduktion von Speicher und Energie getrennt aufgeführt. Die Lösungen des optimalen Ansatzes für den Speicher enden bei Problem 3, da bereits bei Problem 4 zu viele Unbekannte zu beachten sind als dass eine Lösung gefunden werden konnte. Für Kontingente sind bereits bei Problem 1 zu viele Unbekannte zu bestimmen. Bei der Approximation der Energiewerte durch die dp Methode sind die Laufzeiten durch die Menge an Werten zu lang um sinnvolle Ergebnisse zu liefern. Deshalb wurde nach Problem 1 das Programm abgebrochen.

Abbildung 20: Ergebnisse der Satellite Domäne (Speicher)

Problem	**org.**	**opt.**	Red.	Zeit	**glp**	Red.	Zeit	**dp**	Red.	Zeit
problem1	2814	1408	49.96	0.10	2814	0	15.94	2814	0	1.90
problem2	3002	2984	0.60	0.10	3002	0	17.13	3002	0	4.40
problem3	3183	3183	0	0.10	3183	0	16.82	3183	0	4.10
problem4	3838	-	-	-	3838	0	18.11	3838	0	11.80
problem5	4849	-	-	-	4849	0	21.92	4849	0	9.30
problem6	4813	-	-	-	4813	0	24.67	4813	0	27.10
problem7	5589	-	-	-	5589	0	26.88	5589	0	24.40
problem8	6379	-	-	-	6379	0	20.93	6379	0	62.81
problem9	8692	-	-	-	8692	0	21.02	8692	0	109.51
problem10	9462	-	-	-	9462	0	27.04	9462	0	148.02

Bei den Summenwerten ist zu beachten, dass alle Werte zunächst mit 10000 multipliziert wurden um die im Original als Float gegebenen Werte in Integer Werte zu konvertieren, ohne durch

Rundung Information zu verlieren. Da kein Wert mehr als vier Nachkommastellen aufweist, wurde 10000 als ausreichend gesehen. Bei den Laufzeitangaben bedeuten Werte die mit einem h versehen sind Stunden, Werte mit einem d, Tage.

Abbildung 21: Ergebnisse der Satellite Domäne (Energie)

Problem	**org.**	**opt.**	Red.	Zeit	**glp**	Red.	Zeit	**dp**	Red.	Zeit
1: sat0	8810680	-	-	-	440451	95.00	375.85	440451	95.00	19037.05
2: sat0	12281760	-	-	-	614088	95.00	420.04	-	-	-
3: sat0	11540207	-	-	-	11540202	0.0004	901.93	-	-	-
3: sat1	12200207	-	-	-	12200207	0	901.93	-	-	-
4: sat0	14406853	-	-	-	14406853	0	926.34	-	-	-
4: sat1	14296853	-	-	-	14296853	0	926.34	-	-	-
5: sat0	15272540	-	-	-	1527254	90.00	1.61d	-	-	-
5: sat1	15622540	-	-	-	1562254	90.00	1.61d	-	-	-
5: sat2	15662540	-	-	-	1566254	90.00	1.61d	-	-	-
6: sat0	19051988	-	-	-	19051988	0	4.03h	-	-	-
6: sat1	19771988	-	-	-	19771988	0	4.03h	-	-	-
6: sat2	19311988	-	-	-	19311988	0	4.03h	-	-	-
7: sat0	24824080	-	-	-	2482408	90.00	16.83h	-	-	-
7: sat1	24894080	-	-	-	2489408	90.00	16.83h	-	-	-
7: sat2	24744080	-	-	-	2474391	90.00	16.83h	-	-	-
7: sat3	24724080	-	-	-	2472377	90.00	16.83h	-	-	-
8: sat0	31865047	-	-	-	31865047	0	1260.71	-	-	-
8: sat1	31425047	-	-	-	31425047	0	1260.71	-	-	-
8: sat2	31575047	-	-	-	31575047	0	1260.71	-	-	-
8: sat3	31975047	-	-	-	31975047	0	1260.71	-	-	-
9: sat0	41180790	-	-	-	41180790	0	18.64h	-	-	-
9: sat1	40850790	-	-	-	40850790	0	18.64h	-	-	-
9: sat2	41020790	-	-	-	41020790	0	18.64h	-	-	-
9: sat3	41640790	-	-	-	41640790	0	18.64h	-	-	-
9: sat4	41310790	-	-	-	41310790	0	18.64h	-	-	-
10: sat0	47561832	-	-	-	23780916	50.00	2.92d	-	-	-
10: sat1	47391832	-	-	-	23695916	50.00	2.92d	-	-	-
10: sat2	47991832	-	-	-	23995916	50.00	2.92d	-	-	-
10: sat3	47301832	-	-	-	23650916	50.00	2.92d	-	-	-
10: sat4	48221832	-	-	-	24110916	50.00	2.92d	-	-	-

Abbildung 22: Durchschnittliche Reduktion und Laufzeit

	glp solver	app. glp solver	app. dp
Reduktion Speicher	5.06%	0.00%	0.00%
Reduktion Energie	-	35.67%	-
Laufzeit Speicher	0.10s	20.01s	40.33s
Laufzeit Energie	-	14.94h	-

5.3 Zenotravel

Problemstellung Zenotravel beschreibt ein Szenario, bei dem Personen mittels eines Flugzeuges von einem Ort zu einem anderen transportiert werden sollen. Dabei gibt es zwei Möglichkeiten für ein Flugzeug die Strecke zurückzulegen. Ein Flugzeug kann eine Strecke mit normal Geschwindigkeit fliegen oder, vorausgesetzt es sind nur eine vorherbestimmte Menge an Passagieren an Bord, im so genannten *zoom* Modus. Bei letzterem steigt die verbrauchte Menge an Treibstoff um einen festgelegten Faktor. Die Schwierigkeit des Planens steigt dabei mit der Anzahl vorhandener Orte, Personen und verfügbarer Flugzeuge. Um das Verwenden des *zoom* Modus interessant für den Planer zu machen, ist in der *goal*-Bedingung die verbrauchte Zeit und die Menge an verbrauchtem Treibstoff zu minimieren.

numerische Variablen Die numerischen Teile der Domäne sind die Kosten für die Entfernungen zwischen zwei Orten und das Kontingent an Treibstoff. Die Anzahl der Personen die sich in einem Flugzeug befinden kann nicht weiter vereinfacht werden, da eine Person nur einen Platz verbraucht. Ob ein Flugzeug den *zoom* Modus verwenden kann wird dabei durch einen Vergleich zwischen Passagieren an Bord und einem Grenzwert ermittelt. Dieser kann nicht vereinfacht werden. Die Multiplikatoren für normale Reisegeschwindigkeit und *zoom* werden in jeder Problembeschreibung unterschiedlich definiert und erst nachträglich auf die Kosten angewendet.

Ergebnisse der Reduktion Unter **Problem** werden die Probleme nach Nummer aufgelistet, **org.** steht für die Summe der Ausgangswerte, **Red.** für die prozentuale Verbesserung, **opt.** ist die Summe aller Werte nach dem Vereinfachungsschritt mit der optimalen Lösungsmethode und **glp** ist die Summe aller Werte nach einer Vereinfachung mit der approximierenden Methode mit gplsolver, **dp** die Summe aller Werte mit approximierter Methode mit dynamischer Programmierung.

Bereits ab Problem 1 ist die Anzahl der Kosten zu groß um eine optimale Lösung finden zu können.

Abbildung 23: Durchschnittliche Reduktion und Laufzeit

	glp solver	app. glp solver	app. dp
Reduktion	-	30.98%	5.83%
Laufzeit	-	936.93s	2.36h

Abbildung 24: Ergebnisse der Zeno Travel Domäne

Problem	**org.**	**opt.**	Red.	Zeit	**glp**	Red.	Zeit	**dp**	Red.	Zeit
1: plane1	12495	-	-	-	2082	83.34	1.70	12495	0.00	4.10
2: plane1	9086	-	-	-	272	97.01	8.60	9053	0.36	1.50
3: plane1	11124	-	-	-	2777	75.04	3.50	5562	50.00	7.50
3: plane2	10923	-	-	-	5458	50.03	3.50	5461	50.00	7.50
4: plane1	7383	-	-	-	3653	50.52	35.31	7383	0.00	1.70
4: plane2	8227	-	-	-	245	97.02	35.31	8227	0.00	1.70
5: plane1	6641	-	-	-	3274	50.70	619.01	6639	0.03	387.94
5: plane2	8490	-	-	-	8490	0.00	619.01	8490	0.00	387.94
6: plane1	10229	-	-	-	10227	0.02	9485.15	10229	0.00	609.57
6: plane2	7563	-	-	-	816	89.21	9485.15	7508	0.73	609.57
7: plane1	12371	-	-	-	12371	0.00	258.47	12371	0.00	5147.16
7: plane2	10526	-	-	-	10526	0.00	258.47	10526	0.00	5147.16
8: plane1	19316	-	-	-	19316	0.00	251.20	19316	0.00	7650.14
8: plane2	10442	-	-	-	2594	75.16	251.20	10407	0.34	7650.14
8: plane3	12889	-	-	-	12889	0.00	251.20	12889	0.00	7650.14
9: plane1	10672	-	-	-	5238	50.92	320.88	10619	0.50	17277.68
9: plane2	23150	-	-	-	23150	0.00	320.88	23150	0.00	17277.68
9: plane3	14854	-	-	-	14854	0.00	320.88	14854	0.00	17277.68
10: plane1	23302	-	-	-	23302	0.00	139.62	23302	0.00	26808.42
10: plane2	17518	-	-	-	17518	0.00	139.62	17518	0.00	26808.42
10: plane3	16928	-	-	-	16928	0.00	139.62	16928	0.00	26808.42
11: plane1	16913	-	-	-	16913	0.00	632.64	16913	0.00	13083.68
11: plane2	14528	-	-	-	14527	0.01	632.64	14527	0.01	13083.68
11: plane3	21327	-	-	-	21327	0.00	632.64	21327	0.00	13083.68
12: plane1	16500	-	-	-	16500	0.00	131.80	16500	0.00	1317.03
12: plane2	14713	-	-	-	14713	0.00	131.80	14713	0.00	1317.03
12: plane3	13245	-	-	-	13226	0.14	131.80	13226	0.14	1317.03
13: plane1	12719	-	-	-	12690	0.23	580.15	12690	0.23	38331.79
13: plane2	22525	-	-	-	22525	0.00	580.15	22525	0.00	38331.79
13: plane3	15597	-	-	-	15597	0.00	580.15	15597	0.00	38331.79

5.4 Woodworking

Problemstellung Bei woodworking sollen aus Holzrohlingen die eine bestimmte Länge besitzen, kleinere Teile gesägt werden. Die Rohlinge können aus verschiedensten Materialien bestehen so dass es möglich ist bestimmte Teile in bestimmten Materialien herzustellen. Zum Absägen der Teile stehen normale und Hochgeschwindigkeitssägen zur Verfügung. Der Unterschied zwischen den Sägen liegt dabei in der Zeit, die für einen Sägevorgang benötigt wird und das eine Hochgeschwindigkeitssäge mit speziellen Aktionen bestückt werden muss. Nachdem die Teile abgesägt wurden, können diese noch oberflächenbehandelt und farblich Lackiert werden. Ein optimaler Planer minimiert dabei die Zeit die für das Absägen und Bearbeiten der gewünschten Teile benötigt wird.

numerische Variablen Die interessanten numerischen Variablen bei dieser Domäne sind die Länge der Bretter und die daraus zu schneidenden Teile. Dabei handelt es sich um ein Packingproblem. Die Ursprungslänge des Brettes stellt den Grenzwert dar die gewünschten Teilstücke die Verbrauchskosten die anfallen. Die verbrauchte Zeit ist eine globale Variable die durch Aufsummieren der anfallenden Zeiten stetig wächst. Da es keinen Grenzwert für die Zeitliche Bearbeitung gibt, können diese nicht vereinfacht werden.

Ergebnisse der Reduktion Die Ergebnisse finden sich in den Tabellen 26 und 27. Unter **Problem** werden die Probleme nach Nummer aufgelistet, **org.** steht für die Summe der Ausgangswerte, **Red.** für die prozentuale Verbesserung, **opt.** ist die Summe aller Werte nach dem Vereinfachungsschritt mit der optimalen Lösungsmethode und **glp** ist die Summe aller Werte nach einer Vereinfachung mit der approximierenden Methode mit gplsolver, **dp** die Summe aller Werte mit approximierter Methode mit dynamischer Programmierung.

Abbildung 25: Durchschnittliche Reduktion

	glp solver	app. glp solver	app. dp
Reduktion	51.82%	21.83%	19.97%
Laufzeit	0.79s	201.51s	0.58s

Abbildung 26: Ergebnisse der Woodworking Domäne, Teil 1

Problem	**org.**	**opt.**	Red.	Zeit	**glp**	Red.	Zeit	**dp**	Red.	Zeit
1: board0	36	5	86.11	0.20	10	72.22	13.91	23	36.11	0.30
1: board1	45	3	93.33	0.20	22	51.11	13.91	22	51.11	0.30
2: board0	55	11	80.00	0.20	35	36.36	20.92	47	14.55	0.10
2: board1	62	9	85.48	0.20	62	0	20.92	62	0	0.10
3: board0	68	14	79.41	0.20	55	19.12	191.22	58	14.71	0.30
3: board1	46	8	82.61	0.20	13	71.74	191.22	28	39.13	0.30
4: board0	85	23	72.94	0.20	41	51.76	352.05	41	51.76	1.10
4: board1	89	46	48.31	0.20	86	3.37	352.05	86	3.37	1.10
5: board0	96	42	56.25	0.20	95	1.04	36.42	95	1.04	0.30
5: board1	75	46	38.67	0.20	73	2.67	36.42	73	2.67	0.30
6: board0	114	75	34.21	0.30	113	0.88	117.92	113	0.88	0.50
6: board1	86	11	87.21	0.30	40	53.49	117.92	43	50.00	0.50
6: board2	104	57	45.19	0.30	103	0.96	117.92	103	0.96	0.50
7: board0	111	77	30.63	0.40	111	0	37.52	111	0	0.40
7: board1	123	104	15.48	0.40	123	0	37.52	123	0	0.40
7: board2	106	47	55.66	0.40	105	0.94	37.52	105	0.94	0.40
8: board0	156	156	0	0.60	156	0	47.23	156	0	0.60
8: board1	129	46	64.34	0.60	123	4.65	47.23	107	17.05	0.60
8: board2	155	155	0	0.60	155	0	47.23	155	0	0.60
9: board0	140	89	36.43	1.70	140	0	44.82	140	0	0.60
9: board1	160	116	27.50	1.70	160	0	44.82	160	0	0.60
9: board2	152	152	0	1.70	152	0	44.82	152	0	0.60
10: board0	113	23	79.65	1.10	40	64.60	146.27	71	37.17	1.00
10: board1	138	138	0	1.10	138	0	146.27	138	0	1.00
10: board2	152	152	0	1.10	152	0	146.27	152	0	1.00
10: board3	137	137	0	1.10	137	0	146.27	137	0	1.00
11: board0	28	5	82.14	0.20	5	82.14	6.00	17	39.29	0.10
11: board1	30	5	83.33	0.20	5	83.33	6.00	8	73.33	0.10
12: board0	74	5	93.24	0.20	11	20.31	30.41	46	37.84	0.10
13: board0	87	15	82.76	0.20	73	16.09	168.84	76	12.64	0.40
13: board1	77	18	76.62	0.20	56	27.27	168.84	23	70.13	0.40
14: board0	86	16	81.40	0.20	71	17.44	1814.02	78	9.30	1.60
14: board1	56	10	82.14	0.20	16	71.42	1814.02	34	39.26	1.60
15: board0	125	15	88.00	0.30	115	8.00	145.34	115	8.00	0.50
15: board1	101	43	57.43	0.30	95	5.94	145.34	95	5.94	0.50
15: board2	82	11	86.59	0.30	24	70.73	145.34	53	35.37	0.50
16: board0	105	53	49.52	0.20	104	0.95	29.21	104	0.95	0.30
16: board1	99	37	62.63	0.20	98	1.01	29.21	98	1.01	0.30

Abbildung 27: Ergebnisse der Woodworking Domäne, Teil 2

Problem	**org.**	**opt.**	Red.	Zeit	**glp**	Red.	Zeit	**dp**	Red.	Zeit
17: board0	98	20	79.59	0.30	37	62.24	155.24	68	30.61	0.50
17: board1	138	109	21.01	0.30	138	0	155.24	138	0	0.50
17: board2	116	70	39.66	0.30	115	0.86	155.24	115	0.86	0.50
18: board0	146	115	21.23	0.70	145	0.68	95.02	145	0.68	0.70
18: board1	133	29	78.20	0.70	99	25.56	95.02	86	35.34	0.70
18: board2	160	137	14.38	0.70	160	0	95.02	160	0	0.70
19: board0	192	192	0	2.01	192	0	940.05	192	0	1.10
19: board1	140	21	85.00	2.01	35	75.00	940.05	98	30.00	1.10
19: board2	189	157	16.93	2.01	189	0	940.05	189	0	1.10
19: board3	146	12	91.78	2.01	105	28.08	940.05	45	69.18	1.10
20: board0	179	179	0	0.30	179	0	127.03	179	0	1.10
20: board1	146	59	59.60	0.30	141	3.42	127.03	140	4.11	1.10
20: board2	140	16	88.57	0.30	99	29.29	127.03	40	71.43	1.10
20: board3	180	180	0	0.30	180	0	127.03	180	0	1.10
21: board0	47	0	100	0.20	6	87.23	0.60	6	87.23	0.10
22: board0	52	7	86.54	0.20	10	80.77	39.61	7	86.54	0.10
22: board1	50	8	84.00	0.20	8	84.00	39.61	14	72.00	0.10
23: board0	95	9	90.53	0.20	23	75.79	177.54	71	25.26	0.20
24: board0	68	18	73.53	0.20	67	1.47	592.36	67	1.47	1.90
24: board1	96	9	90.63	0.20	70	27.08	592.36	70	27.08	1.90
25: board0	85	17	80.00	0.20	73	14.12	79.12	73	14.12	0.40
25: board1	111	19	82.88	0.20	98	11.71	79.12	98	11.71	0.40
26: board0	127	71	44.10	0.30	127	0	140.33	127	0	0.40
26: board1	129	69	46.51	0.30	129	0	140.33	129	0	0.40
26: board2	90	15	83.33	0.30	30	66.67	140.33	23	74.44	0.40
27: board0	181	97	46.41	0.41	181	0	191.15	181	0	0.60
27: board1	174	60	65.52	0.41	172	1.15	191.15	172	1.15	0.60
27: board2	133	10	92.48	0.41	40	69.92	191.15	46	65.41	0.60
28: board0	186	186	0	0.80	186	0	227.67	186	0	0.80
28: board1	137	13	90.51	0.80	99	27.74	227.67	24	82.48	0.80
28: board2	179	144	19.55	0.80	179	0	227.67	179	0	0.80
28: board3	140	11	92.14	0.80	44	68.57	227.67	11	92.14	0.80
29: board0	154	154	0	1.72	154	0	42.81	154	0	0.60
29: board1	148	148	0	1.72	148	0	42.81	148	0	0.60
29: board2	130	130	0	1.72	130	0	42.81	130	0	0.60
30: board0	151	151	0	9.62	151	0	56.61	151	0	0.80
30: board1	157	157	0	9.62	157	0	56.61	157	0	0.80
30: board2	142	142	0	9.62	142	0	56.61	142	0	0.80
30: board3	138	138	0	9.62	138	0	56.61	138	0	0.80

5.5 Transport

Problemstellung Transport ist von der Aufgabenstellung her ähnlich zu der aus der Depot Domäne. Die Unterschiede bestehen im wesentlichen darin, dass diesmal der Verbrauch von Kraftstoff eine entscheidende Rolle spielt und dieser auch wieder aufgefüllt werden muss sobald das verbleibende Kontingent kleiner ist als die Kosten für eine zurückzulegende Strecke. Ein Tankvorgang setzt das Kontingent wieder auf den ursprünglichen Wert zurück. Wie in der Depot Domäne verursache die zu transportierenden Pakete Kosten, wobei diesmal die Größe eines Paketes und nicht das Gewicht des Paketes angegeben sind. Ebenfalls unterschiedlich zur Depot Domäne benötigt das Fahren zwischen zwei Orten Zeit. Ein optimaler Plan minimiert dabei die anfallende Fahrtzeit

numerische Variablen Wie bei Depot gibt es bei dieser Domäne eine Packing Komponente. Diese besteht aus den Größen der Pakete und dem maximalen Raum den ein Lkw verfügbar hat. Im Gegensatz zu Depot gibt es bei Transport zusätzlich noch den Verbrauch von Kraftstoff. Dieser Teil ist ganz eindeutig eine Kontingentvariable, bestehend aus Kosten für Strecken die mit dem Lkw zurückgelegt werden und dem verfügbarem Kraftstoff.

Ergebnisse der Reduktion Unter **Problem** werden die Probleme nach Nummer aufgelistet, **org.** steht für die Summe der Ausgangswerte, **Red.** für die prozentuale Verbesserung, **opt.** ist die Summe aller Werte nach dem Vereinfachungsschritt mit der optimalen Lösungsmethode und **glp** ist die Summe aller Werte nach einer Vereinfachung mit der approximierenden Methode mit gplsolver, **dp** die Summe aller Werte mit approximierter Methode mit dynamischer Programmierung.

Bei dieser Domäne besitzen fast alle Fahrzeuge die selbe Zuladungsgrenze, aus diesem Grund findet sich bei Problemen jeweils nur ein Eintrag pro Ladungskapazität.

Abbildung 28: Durchschnittliche Reduktion

	glp solver	app. glp solver	app. dp
Reduktion Pakete	85.71%	65.17%	63.77%
Reduktion Treibstoff	-	2.23%	2.43%
Laufzeit Pakete	207.13s	178.26s	0.75s
Laufzeit Treibstoff	-	50.54s	5147.36s

Bei den Ergebnissen zur Laufzeit zwischen glp und dp muss bedacht werden, dass bei dp die Menge der Elemente entsprechend vergrößert wurde um die Möglichen Kombinationen zu erhalten. Eine genaue Erklärung warum dieses Vorgehen wichtig ist findet sich in Kapitel 3. Längere Laufzeiten sind dabei in Bezug auf die Listengröße zu verstehen.

Abbildung 29: Ergebnisse der Transport Domäne (Packete)

Problem	**org.**	**opt.**	Red.	Zeit	**glp**	Red.	Zeit	**dp**	Red.	Zeit
problem1	178	0	100	0.20	4	97.75	0.80	4	97.75	0.20
problem2	346	11	96.82	0.20	26	92.49	23.71	269	22.25	0.20
problem3	345	8	97.68	0.20	112	67.54	1837.95	223	35.36	3.70
problem4	563	13	97.69	0.20	380	32.50	460.72	280	50.27	1.20
problem5	619	50	91.92	0.20	504	18.58	167.05	511	17.45	0.80
problem6	648	35	94.60	0.20	424	34.57	207.65	83	87.19	1.10
problem7	732	165	77.46	2.70	699	4.51	64.82	632	13.66	1.20
problem8	984	321	67.38	5359.53	966	1.83	25.81	956	2.85	1.10
problem9	694	-	-	-	691	0.43	10.20	691	0.43	0.60
problem10	1060	-	-	-	1057	0.28	13.20	1054	0.28	0.80
problem11	219	3	98.63	0.10	15	93.15	1.70	113	48.40	0.10
problem12	346	11	96.82	0.20	26	92.49	19.70	269	22.25	0.30
problem13	348	15	95.69	0.20	236	32.18	2026.75	236	32.18	2.50
problem14	492	66	86.59	0.10	198	59.76	141.84	66	86.59	0.50
problem15	681	56	91.78	0.30	262	61.53	137.32	527	22.61	1.00
problem16	748	173	76.87	2.70	722	3.48	37.11	686	8.29	1.00
problem17	722	205	71.61	2.70	695	3.74	35.21	691	4.29	0.70
problem18	840	111	86.79	13.60	806	4.05	67.62	111	86.79	1.50
problem19	1020	-	-	-	1010	0.98	22.91	976	4.31	1.20
problem20	997	-	-	-	973	2.41	41.41	945	5.22	1.80
problem21: t0	60	0	100	0.20	4	93.33	0.40	4	93.33	0.10
problem21: t1	130	0	100	0.20	4	96.92	0.40	4	96.92	0.10
problem22: t0	60	0	100	0.20	4	93.33	0.40	4	93.33	0.10
problem22: t1	130	0	100	0.20	4	96.92	0.40	4	96.92	0.10
problem23: t0	60	0	100	0.20	4	93.33	0.40	4	93.33	0.10
problem23: t1	130	0	100	0.20	4	96.92	0.40	4	96.92	0.10
problem24: t0	60	0	100	0.20	4	93.33	0.40	4	93.33	0.10
problem24: t1	130	0	100	0.20	4	96.92	0.40	4	96.92	0.10
problem25: t0	60	0	100	0.20	4	93.33	0.40	4	93.33	0.10
problem25: t1	130	0	100	0.20	4	96.92	0.40	4	96.92	0.10
problem26: t0	60	0	100	0.20	4	93.33	0.40	4	93.33	0.10
problem26: t1	130	0	100	0.20	4	96.92	0.40	4	96.92	0.10
problem27: t0	60	0	100	0.20	4	93.33	0.40	4	93.33	0.10
problem27: t1	130	0	100	0.20	4	96.92	0.40	4	96.92	0.10
problem28: t0	60	0	100	0.20	4	93.33	0.40	4	93.33	0.10
problem28: t1	130	0	100	0.20	4	96.92	0.40	4	96.92	0.10
problem29: t0	60	0	100	0.20	4	93.33	0.50	4	93.33	0.10
problem29: t1	130	0	100	0.20	4	96.92	0.50	4	96.92	0.10
problme30: t0	60	0	100	0.20	4	93.33	0.50	4	93.33	0.10
problme30: t1	130	0	100	0.20	4	96.92	0.50	4	96.92	0.10

Abbildung 30: Ergebnisse der Transport Domäne (Treibstoff)

Problem	**org.**	**opt.**	Red.	Zeit	**glp**	Red.	Zeit	**dp**	Red.	Zeit
problem1	805	-	-	-	805	0	142.84	805	0	129.61
problem2	1310	-	-	-	1310	0	27.46	1310	0	384.75
problem3	1756	-	-	-	1756	0	22.11	1756	0	1468.87
problem4	1922	-	-	-	1922	0	24.74	1922	0	9484.78
problem5	1801	-	-	-	1801	0	22.65	1801	0	11251.66
problem6	1713	-	-	-	1713	0	43.43	1713	0	7202.49
problem7	1682	-	-	-	1682	0	33.19	1682	0	9417.08
problem8	1579	-	-	-	1579	0	32.27	1579	0	9825.04
problem9	1464	-	-	-	1464	0	195.38	1464	0	8229.06
problem10	1375	-	-	-	1375	0	199.36	1375	0	7448.85
problem11	1051	-	-	-	1051	0	141.24	1051	0	1062.83
problem12	1417	-	-	-	1417	0	21.52	1417	0	2303.21
problem13	1384	-	-	-	1384	0	21.53	1384	0	3628.73
problem14	1524	-	-	-	1524	0	23.09	1524	0	6969.57
problem15	1651	-	-	-	1651	0	24.05	1651	0	11312.17
problem16	1614	-	-	-	1614	0	30.73	1614	0	12240.99
problem17	1583	-	-	-	1583	0	43.67	1583	0	18117.19
problem18	1384	-	-	-	1384	0	23.06	1384	0	10563.91
problem19	1497	-	-	-	1497	0	26.41	1497	0	12043.78
problem20	1408	-	-	-	1408	0	27.15	1408	0	11097.92
problem21: t0	51	-	-	-	12	86.96	50.11	4	92.16	0.40
problem21: t1	92	-	-	-	92	0	50.11	92	0	0.40
problem22: t0	87	-	-	-	86	1.15	41.91	85	2.30	2.60
problem22: t1	104	-	-	-	104	0	41.91	104	0	2.60
problem23: t0	119	-	-	-	119	0	44.42	119	0	5.40
problem23: t1	87	-	-	-	86	1.15	44.42	85	2.60	5.40
problem24: t0	119	-	-	-	119	0	40.01	119	0	8.30
problem24: t1	111	-	-	-	111	0	40.01	111	0	8.30
problem25: t0	134	-	-	-	134	0	41.41	134	0	12.50
problem25: t1	111	-	-	-	111	0	41.41	111	0	12.50
problem26: t0	134	-	-	-	134	0	40.01	134	0	19.10
problem26: t1	135	-	-	-	135	0	40.01	135	0	19.10
problem27: t0	149	-	-	-	149	0	37.31	149	0	26.50
problem27: t1	135	-	-	-	135	0	37.31	135	0	26.50
problem28: t0	149	-	-	-	149	0	30.81	149	0	41.01
problem28: t1	159	-	-	-	159	0	30.81	159	0	41.01
problem29: t0	164	-	-	-	164	0	31.61	164	0	49.60
problem29: t1	159	-	-	-	159	0	31.61	159	0	49.60
problem30: t0	164	-	-	-	164	0	32.61	164	0	73.01
problem30: t1	183	-	-	-	183	0	32.61	183	0	73.01

5.6 Ergebnisse der Konvertierung

In diesem Kapitel wird die Konvertierung eines numerischen Problems in ein STRIPS Problem erklärt. Die Probleme der woodworking Domäne werden zunächst unverändert nach STRIPS konvertiert und dann mit der reduzierten Variante verglichen die ebenfalls konvertiert wurde.

Konvertieren von numerischen Problemen nach STRIPS

Der Sinn die numerischen Werte zu reduzieren ist in erster Linie, einem Planer kleinere Werte zu übergeben. Eine optimale Reduktion besteht in der Elimination aller Werte. Wenn dies nicht geht wird versucht den Wertebereich so stark zu reduzieren, dass die Werte durch wenige Variablen in STRIPS beschrieben werden können. So kann ein numerisches Problem mit einem Wertebereich von 3 konvertiert werden, indem die Zahl 3 durch vier Variablen $?z0, ?z1, ?z2, ?z3$ ersetzt wird. Dabei repräsentiert $?z0$ den Wert 0, $?z1$ den Wert 1, $?z2$ den Wert 2 und $?z3$ den Wert 3. Diese Art der Repräsentation ist nur bei kleinen Werten realisierbar ist. Es reicht jedoch nicht nur den Zahlenbereich selbst in eine für einen STRIPS Planer verständliche Form zu bringen. Es müssen auch alle arithmetischen Funktionen und Vergleichsoperationen ersetzt werden. In den folgenden zwei Abschnitten werden die notwendigen Änderungen für eine Konvertierung der woodworking Probleme nach STRIPS für jeweils die Domänenbeschreibung und die Problemstellung gegeben.

Domänenbeschreibung

Die Änderungen innerhalb der Domänenbeschreibung umfassen alle Bereiche. Als erstes muss ein neuer Typ definiert werden das die numerischen Werte darstellt. In diesem Fall wird ein neuer Typ *nmbrs* eingeführt. Alle in STRIPS dargestellten numerischen Teile werden diesem Typ zugeordnet. Weiter müssen alle numerischen Erweiterungen entfernt werden. Das bedeutet, der komplette Bereich *functions* wird entfernt. Da woodworking ursprünglich temporale Teile enthält müssen diese ebenfalls ersetzt oder entfernt werden. Aus *durative-action* wird deshalb *action*, *duration* wird komplett entfernt und aus die Zusätze *at start* und *at end*, die sowohl in den Bedingungen als auch in den Effekten vorkommen, werden ebenfalls entfernt. Variablen die innerhalb eines Effektes während *at start* gesetzt werden und dann wiederum *at end* zurückgesetzt werden, können entfernt werden. Zuletzt wird *:condition* noch durch *:precondition* ersetzt. Da bei STRIPS die Aktionen seriell und nicht parallel ablaufen, sind diese Einträge nicht weiter relevant. Die aufwändigste Änderung betrifft die arithmetischen und Vergleichsoperatoren. Im Falle von woodworking werden neue Prädikate für Subtraktions- und Ungleichungsoperatoren gebraucht. Die Subtraktion wird durch ein dreistelliges Prädikat (subs ?val1 ?val2 ?val3 - nmbrs) dargestellt, wobei ?val1 der Minuend, ?val2 der Subtrahend und ?val3 die Differenz ist. Die Ungleichung wird durch (geq ?val1 ?val2 - nmbrs) ersetzt, mit ?val1 als zu Testender Wert und ?val2 als Referenzwert. Dieses Prädikat dient dabei als Vergleich, ob ?val1 größer oder gleich ?val2 ist, stellt damit das äquivalent zu $\geq$ dar. Um diese Prädikate nutzen zu können, müssen noch die Zuweisungen der ehemals numerischen Werte an ein Objekt modelliert werden. Um einem Objekt, im konkreten Fall einem Brett eine Länge zuzuweisen, benötigt man ein Prädikat board-size. Board-size sieht dabei wie folgt aus, (board-size ?board - board ?val - nmbrs). Gleiches gilt für die goal-size, (goal-size ?obj - part ?board - board ?val - nmbrs). Dabei wird ein Teil in Bezug auf ein bestimmtes Brett gesehen. Dies ist notwendig, da bei der Reduktion der Werte die reduzierten Werte immer im Bezug auf einen bestimmtes Brett ermittelt wurden. Zuletzt müssen noch aus den requirements die Hinweise auf die Verwendung von numerischen und temporalen Erweiterungen entfernt werden. Die so konvertierte Domänenbeschreibung kann nun

für die Verwendung mit einem STRIPS Planer genutzt werden. Als nächstes folgt die Anpassung der Problembeschreibungen.

Problembeschreibung

Bei der Problembeschreibung müssen Zuweisungen von numerischen Werten an Variablen durch die bereits in der Domänenbeschreibung ersetzten Prädikate getauscht werden. Im Fall von woodworking sind die Zuweisungen board-size und goal-size zu ersetzen. In der Ursprünglichen Problembeschreibung wird die board-size global definiert. In der STRIPS Variante wird die board-size in Abhängigkeit zu einem board Objekt eingeführt. Aus dem Ausdruck (= (goal-size pm) x) wird nach der Konvertierung (goal-size pm bn val$x$). Dieser Eintrag wird so oft eingefügt, wie board Objekte (n) vorkommen und für jedes part (m). Aus (= (board-size bn) x) wird nach der Konvertierung (board-size bn val$x$). x sind die numerischen Werte des Problems. Angenommen der größte Werte des Problems ist X, so müssen $X + 1$ neue Objekte eingeführt und dem Typ nmbrs zugewiesen werden. Diese repräsentieren die Werte von 0 bis X. Weiter müssen alle möglichen Subtraktionen die kein negatives Ergebnis liefern in der Form (subs x1 x2 x3) in den init Teil der Problembeschreibung eingefügt werden und als letztes der Eintrag metric entfernt werden.

Beispiel Die oben eingeführte Ersetzung des größer-gleich Vergleichs wird durch das subs Prädikat abgedeckt. Da keine negativen Werte zugelassen werden, ist die Differenz bei einer Subtraktion immer positiv. Die Vorbedingungen lassen daher alle Fälle in denen ein Brett kleiner ist als das abzusägende Stück ist gar nicht erst zu.

In den Abbildungen 31 und 32 sind Ausschnitte aus der Ursprünglichen woodworking Domänenbeschreibung und der resultierenden Konvertierung zu sehen. In den Abbildungen 33 und 34 sind die Änderungen in der Problemstellung zu sehen.

Ergebnisse konvertierter Probleme

Zum Testen wurden Fast Forward (FF) von Jörg Hoffmann und Bernhard Nebel [HN01], die Fast Downward Variante von Malte Helmert (FD) [HHH07] und ein zu diesem Zeitpunkt noch unveröffentlichter Planer (LM) verwendet. Die Ergebnisse finden sich in den Tabellen 35 und 36. Alle nicht angegebenen Probleme wurden entweder nicht gelöst oder die Reduktion führte zu keiner Verringerung des Wertebereichs (Probleme 29 und 30). Die Ergebnisse von FD und LM in Tabelle 35 zeigen eine allgemein leicht kürzere Laufzeit bei der Verarbeitung der reduzierten Probleme. Die Zahl der Expandierten Zustände ist bei nicht reduzierten und reduzierten Problemen bis auf zwei Ausnahmen bei beiden Planern gleich geblieben. Dies zeigt, dass keine Verschlechterung durch einen reduzierten Wertebereich entsteht. Zwar ist dies in einem der beiden Ausnahmen (Problem 9 bei LM, Problem 24 bei FD) der Fall, dagegen ist der zweite Fall (Problem 25 bei LM, Problem 7 bei FD) ein starker Hinweis auf das Potenzial das die Reduktion birgt. Im Falle von LM ist die Anzahl der Zustände und die Laufzeit (beide um 99.7%) radikal gesunken, im Fall von FD immerhin noch um ca. 11 Prozent. FD wurde für die Experimente mit dem Parameter *a 100000* gestartet. Bei FF (Tabelle 36) zeigt sich bis auf eine Ausnahme (Problem 19) eine konstante Reduktion der vorhandenen Fakten und relevanten Aktionen eines Problems im Vergleich mit seiner reduzierten Variante.

```
(:requirements :typing :durative-actions :numeric-fluents)
(:types
    ...
    planer saw spray-varnisher - machine
    board part - woodobj)

...

(:predicates
          ...
          (grind-treatment-change ?old ?new - treatmentstatus)
          (is-smooth ?surface - surface))

(:functions
          (board-size ?board - board)
          (goal-size ?obj - part))

...

(:durative-action unload-highspeed-saw
  :parameters (?b - board ?m - highspeed-saw)
  :duration (= ?duration 10)
  :condition (and
          (at start (idle ?m))
          (at start (in-highspeed-saw ?b ?m)))
  :effect (and
          (at start (not (idle ?m)))
          (at end (available ?b))
          (at end (not (in-highspeed-saw ?b ?m)))
          (at end (empty ?m))
          (at end (idle ?m))))

...

(:durative-action do-saw
  :parameters (?b - board ?p - part ?m - saw ?w - awood
               ?surface - surface)
  :duration (= ?duration 30)
  :condition (and
          (at start (idle ?m))
          (at start (unused ?p))
          (at start (available ?b))
          (at start (wood ?b ?w))
          (at start (surface-condition ?b ?surface))
          (at start (>= (board-size ?b) (goal-size ?p))))
  :effect (and
          (at start (not (idle ?m)))
          (at start (not (unused ?p)))
          (at start (not (available ?b)))
          (at end (decrease (board-size ?b) (goal-size ?p)))
          (at end (idle ?m))
          (at end (available ?p))
          (at end (available ?b))
          (at end (wood ?p ?w))
          (at end (surface-condition ?p ?surface))
          (at end (colour ?p natural))
          (at end (treatment ?p untreated))))
```

Abbildung 31: Ausschnitt aus der woodworking Domänenbeschreibung, numerisch

```
(:requirements :typing)
(:types
    ...
    planer saw spray-varnisher - machine
    board part - woodobj
    nmbrs - object)

...

(:predicates
          ...
          (grind-treatment-change ?old ?new - treatmentstatus)
          (is-smooth ?surface - surface)
          (board-size ?board - board ?val - nmbrs)
          (goal-size ?obj - part ?board - board ?val - nmbrs)
          (subs ?val1 ?val2 ?val3 - nmbrs)
          )

...

(:action unload-highspeed-saw
  :parameters (?b - board ?m - highspeed-saw)
  :precondition (and
          (idle ?m)
          (in-highspeed-saw ?b ?m))
  :effect (and
          (available ?b)
          (not (in-highspeed-saw ?b ?m))
          (empty ?m)))

...

(:action do-saw
  :parameters (?b - board ?p - part ?m - saw ?w - awood
               ?surface - surface
               ?old-board-size ?new-board-size ?part-size - nmbrs)
  :precondition (and
          (idle ?m)
          (unused ?p)
          (available ?b)
          (wood ?b ?w)
          (surface-condition ?b ?surface)
          (board-size ?b ?old-board-size)
          (goal-size ?p ?b ?part-size)
          (subs ?old-board-size ?part-size ?new-board-size)
          )
  :effect (and
          (not (unused ?p))
          (not (board-size ?b ?old-board-size))
          (board-size ?b ?new-board-size)
          (available ?p)
          (wood ?p ?w)
          (surface-condition ?p ?surface)
          (colour ?p natural)
          (treatment ?p untreated)))
```

Abbildung 32: Woodworking Domänenbeschreibung, STRIPS

```
(define (problem wood-prob)
  (:domain woodworking)
  (:objects
    ...
    p0 p1 p2 - part
    b0 b1 - board
  )
  (:init
    ...
    (= (goal-size p0) 7)
    (unused p1)
    (= (goal-size p1) 11)
    (unused p2)
    (= (goal-size p2) 9)
    (= (board-size b0) 9)
    (wood b0 cherry)
    (surface-condition b0 smooth)
    (available b0)
    (= (board-size b1) 18)
    ...
  )
  (:goal
    ...
  )
  (:metric minimize (total-time))
)
```

Abbildung 33: Woodworking Problem 1, numerisch

```
(define (problem wood-prob)
  (:domain woodworking)
  (:objects
    p0 p1 p2 - part
    b0 b1 - board
    val18 val17 ... val2 val1 val0 - nmbrs
  )
  (:init
    (subs val18 val18 val0)
    (subs val18 val17 val1)
    ...
    (subs val2 val1 val1)
    (subs val1 val1 val0)
    ...
    (goal-size p0 b0 val7)
    (goal-size p0 b1 val7)
    (unused p1)
    (goal-size p1 b0 val11)
    (goal-size p1 b1 val11)
    (unused p2)
    (goal-size p2 b0 val9)
    (goal-size p2 b1 val9)
    (board-size b0 val9)
    (wood b0 cherry)
    (surface-condition b0 smooth)
    (available b0)
    (board-size b1 val18)
    ...
  )
  (:goal
    ...
  )
)
```

Abbildung 34: Woodworking Problem 1, STRIPS

Abbildung 35: Ergebnisse mit Fast Downward Varianten, expanded states und search time.

Problem	LM	LM_{red}	FD	FD_{red}
p1	4 states, 0s	4 states, 0s	4 states, 0s	4 states, 0s
p2	5 states, 0s	5 states, 0s	5 states, 0s	5 states, 0s
p3	10 states, 0s	10 states, 0s	6 states, 0s	6 states, 0s
p4	7 states, 0.01s	7 states, 0.01s	7 states, 0s	7 states, 0s
p5	8 states, 0.01s	8 states, 0.01s	8 states, 0s	8 states, 0s
p6	36 states, 0.06s	36 states, 0.05s	9 states, 0.03s	9 states, 0.02s
p7	191 states, 0.24s	191 states, 0.24s	4201 states, 0.18s	3734 states, 0.21s
p8	11 states, 0.12s	11 states, 0.12s	11 states, 0.04s	11 states, 0.04s
p9	116 states, 0.06s	928 states, 4.89s	12 states, 0.02s	12 states, 0.02s
p10	1749 states, 10.28s	1749 states, 10.21s	146452 states, 7.35s	146452 states, 7.36s
p11	7 states, 0s	7 states, 0s	7 states, 0.01s	7 states, 0.01s
p12	9 states, 0.04s	9 states, 0.03s	9 states, 0.03s	9 states, 0.03s
p13	11 states, 0.06s	11 states, 0.06s	11 states, 0.02s	11 states, 0.02s
p14	13 states, 0.15s	13 states, 0.14s	13 states, 0.03s	13 states, 0.02s
p15	15 states, 0.28s	15 states, 0.27s	-	-
p16	17 states, 0.65s	17 states, 0.63s	-	-
p17	19 states, 1.26	19 states, 1.22s	-	-
p18	21 states, 2.44s	21 states, 2.39s	-	-
p19	23 states, 3.66s	23 states, 3.82s	-	-
p20	25 states, 5.76s	25 states, 5.70s	-	-
p21	8 states, 0.01s	8 states, 0.01s	8 states, 0s	8 states, 0s
p22	12 states, 0.04s	12 states, 0.03s	12 states, 0.01s	12 states, 0s
p23	14 states, 0.08s	14 states, 0.07s	14 states, 0.02s	14 states, 0.03s
p24	19 states, 0.22s	19 states, 0.21s	1134943 states, 131.58s	1135816 states, 130.12s
p25	9107 states, 87.1s	29 states, 0.24s	16 states, 0.03s	16 states, 0.04s

Abbildung 36: Ergebnisse mit FF, yielding facts und actions

Problem	FF	FF_{red}
p1	41 facts, 26 actions	35 facts, 14 actions
p2	56 facts, 66 actions	45 facts, 40 actions
p3	69 facts, 263 actions	51 facts, 98 actions
p4	96 facts, 416 actions	77 facts, 284 actions
p5	115 facts, 662 actions	95 facts, 395 actions
p6	137 facts, 969 actions	113 facts, 666 actions
p7	153 facts, 1221 actions	135 facts, 990 actions
p8	190 facts, 2322 actions	184 facts, 2310 actions
p9	204 facts, 2842 actions	187 facts, 2382 actions
p10	221 facts, 3248 actions	221 facts, 3248 action
p11	55 facts, 228 actions	53 facts, 228 actions
p12	65 facts, 402 actions	62 facts, 398 actions
p13	132 facts, 759 actions	98 facts, 577 actions
p14	125 facts, 776 actions	103 facts, 626 actions
p15	170 facts, 1171 actions	135 facts, 863 actions
p16	193 facts, 1612 actions	170 facts, 1412 actions
p17	236 facts, 2117 actions	216 facts, 1933 actions
p18	247 facts, 2052 actions	236 facts, 2000 actions
p19	322 facts, 3499 actions	325 facts, 3781 actions
p20	327 facts, 3106 actions	319 facts, 3102 actions
p21	57 facts, 208 actions	41 facts, 124 actions
p22	72 facts, 304 actions	70 facts, 304 actions
p23	91 facts, 436 actions	73 facts, 384 actions
p24	147 facts, 884 actions	103 facts, 516 actions
p25	164 facts, 1212 actions	122 facts, 726 actions
p26	232 facts, 2090 actions	198 facts, 1710 actions
p27	-	-
p28	304 facts, 3499 actions	288 facts, 3361 actions

6 Fazit

Die Ergebnisse der Reduktion zeigen, dass die vorgestellten Methoden in der Lage sind den Wertebereich numerischer Problemstellungen zu verringern. Wie gut diese Reduktion funktioniert, hängt stark von der Aufgabenstellung und den Werten der Domäne und des Problems ab. Gerade bei Domänen mit Packing verwandten Problemen zeigt sich dies. Eine durchschnittliche Reduktion von 54.67% über alle Probleme konnte beispielsweise bei Depot erreicht werden. Wobei gerade die einfacheren Probleme stark reduziert und teilweise die Werte komplett eliminiert werden konnten. Elimination ist der optimale Fall bei Reduktion, alle numerischen Werte sind nicht relevant, was die Konvertierung des Wertebereichs überflüssig macht. Ebenso wurden bei Woodworking (51.82%) und Transport (85.71%) gute Ergebnisse durch Reduktion erzielt. Dass die Ergebnisse der Reduktionen im Bereich der Kontingentvariablen nicht vergleichbar gute Ergebnisse liefern, liegt vornehmlich an ihrer Komplexität. Durch die großen Mengen an Kosten, die zu beachten sind, waren die vorhandenen Systeme überlastet. Mit einem Rechnersystem das über größere Mengen an Arbeitsspeicher verfügt, um die Eingabe des GLPK zu verarbeiten, wären auch diese Probleme reduzierbar.

Dass die Reduktion des Wertebereichs sinnvoll einsetzbar ist, zeigen die Ergebnisse der Tests mit der konvertierten Version der woodworking Domäne. Es zeigt sich beispielsweise bei Fast Forward (FF), dass die Anzahl der benötigten Fakten und die Zahl der relevanten Aktionen bei fast allen Problemen (26 von 27 Problemen) reduziert werden konnte. Tests, bei denen die Werte eines Problem mit einem festen Faktor multipliziert wurden, zeigen zudem, dass FF bei großen Wertebereichen nicht mehr in der Lage ist das Problem zu bearbeiten. Eine Reduktion kann demzufolge ein vorher unlösbares Problem in solchen Fällen lösbar machen. Die Ergebnisse der anderen auf Fast Downward basierenden Planern scheinen zuerst unbedeutend, bei genauerer Betrachtung zeigt gerade der Unterschied von ca. 99.7% bei Problem 25 mit LM als Planer, wie groß der Vorteil durch reduzierte Wertebereiche sein kann.

Im Vergleich mit den Ergebnissen der IPC 2008 zeigt sich außerdem, dass die Zahl der optimal gelösten woodworking Probleme im Bereich *temporal satisficing* vergleichbar mit denen anderer Teilnehmer ist. Wesentlich interessanter ist jedoch, dass die Zahl der gelösten Probleme im Bereich *sequential optimization* sogar höher liegt als die der angetretenen Planer, zumal dort im Vergleich mit den hier verwendeten Problemen nur mit kleinen Werten, vergleichbar mit den reduzierten Problemen, gearbeitet wurde.

Zukünftige Arbeiten Auch wenn die in dieser Arbeit vorgestellten Ergebnisse bereits zeigen, wie die Reduktion von numerischen Planungsproblemen funktioniert, gibt es einige Punkte an denen diese Prozesse optimiert werden können. So konnten wegen der langen Testphasen nach der Implementation nicht alle Ansätze in diese Arbeit aufgenommen werden. Zum anderen wäre es sinnvoll einen Parser in einen der verwendeten Planer zu integrieren, der numerische Probleme zusammen mit den Ergebnissen der Reduktion direkt in ein für den Planer verständliches Format konvertiert. Dazu können bei einer Invariantensuche die in Kapitel 3 vorgestellten Merkmale verwendet werden um numerische Variablen zu erkennen und festzustellen um welche Art es sich dabei handelt. Mit diesen Informationen können die Reduktions- und Konvertierungsmethoden, die in dieser Arbeit entwickelt wurden, angewandt werden.

Literatur

[Ber00] Dimitri P. Bertsekas. *Linear Network Optimization: Algorithms and Codes*. The MIT Press, 2000.

[BFG+00] Robert Bixby, Mary Fenelon, Zonghao Gu, Ed Rothberg, and Roland Wunderling. MIP: Theory and practice — Closing the gap. In Michael J. D. Powell and Stefan Scholtes, editors, *System Modelling and Optimization: Methods, Theory and Applications*, volume 174 of *International Federation for Information Processing*, pages 19–49. Kluwer Academic Press, 2000.

[CB01] Amedeo Cesta and Daniel Borrajo, editors. *Pre-proceedings of the Sixth European Conference on Planning (ECP'01)*, Toledo, Spain, 2001.

[DK01] Minh Binh Do and Subbarao Kambhampati. Sapa: A domain-independent heuristic metric temporal planner. In Cesta and Borrajo [CB01], pages 109–120.

[FL03] Maria Fox and Derek Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

[FN71] Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability — A Guide to the Theory of NP-Completeness*. Freeman, 1979.

[glp] Gnu linear programming kit. Web site: `http://www.gnu.org/software/glpk/`.

[GSS03] Alfonso Gerevini, Alessandro Saetti, and Ivan Serina. Planning through stochastic local search and temporal action graphs in LPG. *Journal of Artificial Intelligence Research*, 20:239–290, 2003.

[HG01] Patrik Haslum and Héctor Geffner. Heuristic planning with time and resources. In Cesta and Borrajo [CB01], pages 121–132.

[HHH07] Malte Helmert, Patrik Haslum, and Jörg Hoffmann. Flexible abstraction heuristics for optimal sequential planning. In Mark Boddy, Maria Fox, and Sylvie Thiébaux, editors, *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS 2007)*, pages 176–183. AAAI Press, 2007.

[HN01] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[Lif87] Vladimir Lifschitz. On the semantics of STRIPS. In M. Georgeff and A. Lansky, editors, *Reasoning about Actions and Plans*, pages 1–9. Morgan Kaufmann, 1987.

[mem] Memtime. Web site: `http://www.update.uu.se/~johanb/memtime/`.

[MGH+98] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL — The planning domain definition language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, Yale University, 1998.

[Ped89] Edwin P. D. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In Ronald J. Brachman, Hector J. Levesque, and Raymond Reiter, editors, *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning (KR'89)*, pages 324–332. Morgan Kaufmann, 1989.

[WC06] Benjamin W. Wah and Yixin Chen. Constraint partitioning in penalty formulations for solving temporal planning problems. *Artificial Intelligence*, 170(3):187–231, 2006.