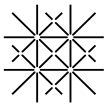

Universität
Basel

P^m-Kompilierung von Planungsaufgaben im Fast-Downward Planungssystem als alternative Berechnung der h^m-Heuristik

Bachelorarbeit

Philosophisch-Naturwissenschaftliche Fakultät der Universität Basel
Departement Mathematik und Informatik
Künstliche Intelligenz
http://ai.cs.unibas.ch

Beurteiler: Prof. Dr. Malte Helmert
Zweitbeurteiler: Gabriele Röger

Raphael Imahorn
raphael.imahorn@stud.unibas.ch
2010-057-826

28. Januar 2016

Danksagung

An dieser Stelle möchte ich mich bei Gabriele Röger bedanken. Sie hat mich beim Erstellen dieser Arbeit stets mit Rat und Tat unterstützt.
Ich möchte mich auch bei Prof. Malte Helmert bedanken. Ich erhielt die Chance, im Bereich der künstlichen Intelligenz eine Bachelor-Arbeit zu schreiben und viel zu lernen.

Zusammenfassung

In einer Planungsaufgabe geht es darum einen gegebenen Wertezustand durch sequentielles Anwenden von Aktionen in einen Wertezustand zu überführen, welcher geforderte Zieleigenschaften erfüllt. Beim Lösen von Planungsaufgaben zählt Effizienz. Um Zeit und Speicher zu sparen verwenden viele Planer heuristische Suchen. Dabei wird mittels einer Heuristik abgeschätzt, welche Aktion als nächstes angewendet werden soll um möglichst schnell in einen gewünschten Zustand zu gelangen.
In dieser Arbeit geht es darum, die von Haslum vorgeschlagene P^m - Kompilierung für Planungsaufgaben zu implementieren und die h^{max} - Heuristik auf dem kompilierten Problem gegen die h^m - Heuristik auf dem originalen Problem zu testen. Die Implementation geschieht als Ergänzung zum Fast-Downward Planungssystem.
Die Resultate der Tests zeigen, dass mittels der Kompilierung die Zahl der gelösten Probleme erhöht werden kann. Das Lösen eines kompilierten Problems mit der h^{max} - Heuristik geschieht im allgemeinen mit selbiger Informationstiefe schneller als das Lösen des originalen Problems mit der h^m - Heuristik. Diesen Zeitgewinn erkauft man sich mit einem höheren Speicherbedarf.

Inhaltsverzeichnis

1 Einführung

In dieser Arbeit geht es darum Planungsaufgaben zu lösen. Das heisst, wir wollen einen gegebenen Wertzustand durch eine Folge von Aktionsanwendungen in einen Zustand bringen, der gegebene Zieleigenschaften aufweist. Dabei geben die Vorbedingungen einer Aktion an, in welchen Zuständen die Aktion angewandt werden kann und die Effekte einer Aktion geben an, wie das Anwenden der Aktion den Wertzustand verändert.
Eine Planungsaufgabe wird gelöst, indem durch sukzessives Anwenden von Aktionen, vom Anfangszustand zu einem Zielzustand gewechselt wird. Diese Folge von Aktionen nennt sich *Plan*. Die *Plankosten* ergeben sich aus der Summe aller Aktionen, die im Plan enthalten sind. Ein Plan heisst *optimal*, wenn es keinen anderen Plan für dasselbe Problem (synoym zu Planungsaufgabe) mit kleineren Plankosten gibt. Kann man einen Plan zu einer Planungsaufgabe bilden, ist diese *lösbar*. Falls kein solcher Plan existiert, ist die Planungsaufgabe *unlösbar*.
Ein *Suchalgorithmus* ist ein Algorithmus, der versucht eine Planungsaufgabe zu lösen. Ziel der meisten Suchalgorithmen ist primär einen Plan zu finden, falls das Problem lösbar[1] ist und sekundär zu garantieren, dass der gefundene Plan optimal[2] ist. Ziel der Implementierung eines Suchalgorithmus ist, dass dieser möglichst wenig Speicher und Zeit zur Lösung eines Planungsproblems benötigt.
Für diese Arbeit verwenden wir heursitische Suche. Das bedeutet, dass der Suchalgorithmus mittels einer Heuristikfunktion abschätzt, mit welcher im aktuellen Zustand anwendbaren Aktion wir am günstigsten in einen Zielzustand gelangen.
Eine Heuristik ist eine Funktion, welche einen Zustand auf eine reelle Zahl abbildet. Die Idee dabei ist, dass die Heuristik abschätzt, wie die Plankosten eines optimalen Planes vom angeschauten Zustand zu einem Zielzustand ausfallen. Eine Heuristik sollte für einen Zielzustand null zurückgeben und für einen Zustand, von dem aus die Zielbeschreibung nicht erreicht werden kann, ∞.
Wenn eine Heuritik für keinen Zustand einen Wert zurückliefert, der grösser als die Plankosten eines optimalen Planes von diesem Zustand zu einem Zielzustand ist, ist die Heuristik

[1] Falls der Algorithmus für eine lösbare Planungsaufgabe garantiert einen Plan findet und andernfalls terminiert, ist der Suchalgorithmus *vollständig*.
[2] Falls jeder vom Algorithmus gefundene Plan garantiert optimal ist, ist der Suchalgorithmus *optimal*.

zulässig. Wir verwenden für die Experimente dieser Arbeit A* [HNR68] als Suchalgorithmus. Dieser liefert für lösbare Planungsaufgaben mit einer zulässigen Heuristik immer einen optimalen Plan.
Wir verwenden in unserer Arbeit die Familie der h^m - *Heuristiken* [HG00]. Der Parameter m ist eine natürliche Zahl grösser 0. Die Idee dieser Heuristiken ist für eine Teilmenge von maximal m Werten die geforderten Zieleigenschaften zu erreichen. Es gilt dabei die Plankosten für einen optimalen Plan zu einem angepasseten Zielzustand derjenigen maximal m Werte zurückzugeben, welche diese Kosten maximieren. Ein Spezialfall dieser Familie ist $m = 1$. Die h^1 - Heuristik entspricht der bereits früher publizierten h^{max} - *Heuristik* [BG01]. Alle Heuristiken dieser Familie sind zulässig.
Wenn wir eine Planungsaufgabe P mittels der P^m - Kompilierung transformieren, erhalten wir mit der h^{max} - Heuristik auf der transformierten Planungsaufgabe dieselben Werte, wie wenn wir mit demselben m die h^m - Heuristik für das originale Problem berechnen [Has09]. In dieser Arbeit wollen wir analysieren, wie sich h^m_P zu $h^{max}_{P^m}$ hinsichtlich Zeit und Speicherbedarf tatsächlich verhält.
Sämtliche Implementationen wurden im Fast-Downward Planungssystem[3] [Hel06] durchgeführt.
Im Weiteren ist die Arbeit folgendermassen aufgegliedert: Wir erläutern zunächst die theoretischen Grundlagen im Zusammenhang mit der P^m - Kompilierung. Es folgt die Auswertung der durchgeführten Experimente. Zum Abschluss werden die Erkenntnisse dieser Arbeit zusammengefasst aufgeführt.

[3] Weitere Informationen zum Fast-Downward Planungssystem sind zu finden unter: http://www.fast-downward.org/.

2
Hintergrund

In diesem Kapitel wird der Hintergrund der Arbeit vorgestellt. Ziel ist es, die Grundlagen aufzuzeigen und die verwendeten Schreibweisen vorzustellen.

2.1 Planungsformalismen

Damit ein Suchalgorithmus für eine Planungsaufgabe eine Lösung finden kann, muss diese in einer für diesen verständlichen Formulierung niedergeschrieben werden. Im Folgenden werden zwei für diese Arbeit relevante Formalismen für Planungsaufgaben eingeführt. Es handelt sich dabei um den *STRIPS* - und den SAS$^+$ - Formalismus.

2.1.1 *STRIPS*

STRIPS (**St**anford **R**esearch **I**nstitute **P**roblem **S**olver) [FN71] ist die formale Sprache, welche heute die Grundlage zum Beschreiben vieler Problemdomänen bildet. Eine *STRIPS*-Planungsaufgabe besteht aus einem *Anfangszustand*, einer *Zielbeschreibung*, einer Menge von *Atomen* und einer Menge von *Aktionen*. Jede Aktion a beinhaltet eine Menge Atome, die *Vorbedingungen*, welche erfüllt sein müssen, damit die Aktion in einem Zustand s *anwendbar* ist, und eine Menge Atome, die *Effekte*. Die Effekte teilen sich in die *Add-Effekte*, also Atome, welche nach dem Anwenden von a erfüllt sind, und in die *Delete-Effekte*, Atome, welche Teil der Vorbedingungen von a sind und nach der Anwendung von a nicht mehr erfüllt sind. Ein *Atom* hat genau eine aus zwei möglichen Belegungen: `wahr` oder `falsch`. Formal sieht das in dieser Arbeit verwendete *STRIPS* wie folgt aus:

Definition 1 (*STRIPS* - Planungsaufgabe)**.** *Eine STRIPS- Planungsaufgabe ist ein 4-Tupel* $\Pi = \langle V, A, I, G \rangle$ *mit:*

- V *einer endlichen Menge von Atomen,*
- A *einer endlichen Menge von Aktionen. Jede Aktion* $a \in A$ *ist gegeben als Tupel* $a = \langle pre(a), add(a), del(a), cost(a) \rangle$ *mit* Vorbedingung $pre(a) \subseteq V$*,* Add-Effekten $add(a) \subseteq V$*,* Del-Effekten $del(a) \subseteq V$ *und* Kosten $cost(a) \in \mathbb{R}_0^+$*.*
- $I \subseteq V$*, dem* Anfangszustand*, und*

- $G \subseteq V$, *der* Zielbeschreibung.

Ein Zustand $s \subseteq V$ *ist eine endliche Menge von Atomen, welche* `wahr` *sind. Mit einem* Zielzustand $g \subseteq V$ *ist eine ist eine Obermenge der Zielbeschreibung* G *gemeint. Eine Aktion* $a \in A$ *ist* anwendbar *in einem Zustand* s*, falls* $pre(a) \subseteq s$*. Die Anwendung von* a *in* s *führt in den* Nachfolgezustand $s' = (s \setminus del(a)) \cup add(a)$*. Ein* Plan *für* Π *ist eine Folge von Aktionen* $\rho = \langle a_1, \ldots, a_n \rangle$*, so dass es eine Folge von Zuständen* $\langle s_0, s_1, \ldots, s_n \rangle$ *gibt mit* $s_0 = I$*,* $G \subseteq s_n$ *und für alle* $1 \leq i \leq n$ *ist Aktion* a_i *anwendbar in* s_{i-1} *und führt zu Nachfolgezustand* s_i*. Plan* ρ *hat Kosten* $cost(\rho) = \sum_{i=1}^{n} cost(a_i)$ *und ist* optimal*, falls er minimale Kosten unter allen Plänen für* Π *hat. Eine Planungsaufgabe heisst* lösbar*, falls ein Plan existiert, und andernfalls* unlösbar.

Machen wir hierzu ein Beispiel:

Beispiel 1 (*STRIPS* - Blocksworld)**.** *Wir haben drei verschiedene Blöcke A, B und C. Die Blöcke können auf dem Tisch oder auf einem anderen Block stehen. Es darf in jedem Zug ein Block bewegt werden. Ein Block kann nur bewegt werden, wenn kein Block auf ihm steht. Am Anfang steht A auf B, welcher auf C steht, welcher sich wiederum auf dem Tisch befindet. Wir wollen die Reihenfolge umkehren. Wir verwenden die Atome aufTisch(X), das angibt ob ein Block X auf dem Tisch steht, obenOhne(X), welches angibt, ob ein Block X freistehend ist und somit bewegt werden kann, und auf(X, Y), welches angibt, ob auf einem Block X ein Block Y steht. Wir verwenden die Aktionen legeAufTisch(X, Y), mit welcher ein auf Y freistehender Block X auf den Tisch bewegt wird; legeAufBlock(X, Y), mit welcher ein auf dem Tisch freistehender Block Y auf einen freistehenden Block X platziert wird; und legeAufBlock(X, Y, Z), mit welcher ein auf einem Block Z freistehender Block Y auf einen freistehenden Block X platziert wird. Somit sieht das Problem in STRIPS wie folgt aus, wobei die Aktionen aus Platzgründen generisch angegeben werden:*

$\Pi = \langle V, A, I, G \rangle$ *mit:*

Die Atome sind:	*Der Startzustand ist:*	*Die Aktionen, wobei X, Y und Z als Variablen für A, B und C eingesetzt werden, sind:*
aufTisch(A)	*aufTisch*(C)	*legeAufTisch(X, Y)*
aufTisch(B)	*auf*(C, B)	*pre: obenOhne(X), auf(Y, X)*
aufTisch(C)	*auf*(B, A)	*add: aufTisch(X), obenOhne(Y)*
obenOhne(A)	*obenOhne*(A)	*del: auf(Y, X)*
obenOhne(B)	$\uparrow$	*legeAufBlock(X, Y)*
obenOhne(C)	I	*pre: obenOhne(X), obenOhne(Y), aufTisch(Y)*
auf(A, B)	$\leftarrow V \qquad A \rightarrow$	*add: auf(X, Y)*
auf(A, C)	G	*del: obenOhne(X), aufTisch(Y)*
auf(B, A)	$\downarrow$	*legeAufBlock(X, Y, Z)*
auf(B, C)	*Das Ziel ist:*	*pre: obenOhne(X), obenOhne(Y), auf(Z, Y)*
auf(C, A)	*auf*(A, B)	*add: auf(X, Y), obenOhne(Z)*
auf(C, B)	*auf*(B, C)	*del: obenOhne(X), auf(Z, Y)*

Wir gehen davon aus, dass alle Aktionen einheitliche Kosten $cost(a) = 1$ *haben. Ein*

möglicher Plan ist $\rho_1 = \langle legeAufTisch(A, B),\ legeAufTisch(B, C),\ legeAufBlock(A, B),\ legeAufBlock(B, C)\rangle$. *Die Plankosten sind* $cost(\rho_1) = 1 + 1 + 1 + 1 = 4$. *Die Zustände sehen dann wie folgt aus, wobei Atome, welche Teil der Zielbeschreibung G sind, fett dargestellt werden:*

$$\begin{array}{l} aufTisch(C),\\ obenOhne(A),\\ auf(B, A),\\ auf(C, B) \end{array} \rightarrow \begin{array}{l} aufTisch(A),\\ aufTisch(C),\\ obenOhne(A),\\ obenOhne(B),\\ auf(C, B) \end{array} \rightarrow \begin{array}{l} aufTisch(A),\\ aufTisch(B),\\ aufTisch(C),\\ obenOhne(A),\\ obenOhne(B),\\ obenOhne(C) \end{array} \rightarrow \begin{array}{l} aufTisch(A),\\ aufTisch(C),\\ obenOhne(B),\\ obenOhne(C),\\ \boldsymbol{auf(A,\ B)} \end{array} \rightarrow \begin{array}{l} aufTisch(A),\\ obenOhne(C),\\ \boldsymbol{auf(A,\ B)},\\ \boldsymbol{auf(B,\ C)} \end{array}$$

Dieser Plan ist jedoch nicht optimal. Fasst man die zweite und die dritte Aktion im Plan zusammen, legt man also den Block B direkt von Block C auf Block A, spart man sich eine Aktion und deren Kosten. Wir sehen uns einen zweiten Plan ρ_2 *an, welcher für dieses Problem optimal ist. Dieser sieht so aus:* $\rho_2 = \langle legeAufTisch(A, B),\ legeAufBlock(A, B, C),\ legeAufBlock(B, C)\rangle$ *und hat Kosten* $cost(\rho_2) = 1 + 1 + 1 = 3$. *Die Zustände sehen dann wie folgt aus, wobei Atome, welche Teil der Zielbeschreibung G sind, fett dargestellt werden:*

$$\begin{array}{l} aufTisch(C),\\ obenOhne(A),\\ auf(B, A),\\ auf(C, B) \end{array} \rightarrow \begin{array}{l} aufTisch(A),\\ aufTisch(C),\\ obenOhne(A),\\ obenOhne(B),\\ auf(C, B) \end{array} \rightarrow \begin{array}{l} aufTisch(A),\\ aufTisch(C),\\ obenOhne(B),\\ obenOhne(C),\\ \boldsymbol{auf(A,\ B)} \end{array} \rightarrow \begin{array}{l} aufTisch(A),\\ obenOhne(C),\\ \boldsymbol{auf(A,\ B)},\\ \boldsymbol{auf(B,\ C)} \end{array}$$

2.1.2 SAS$^+$

Eine Planungsaufgabe kann auch als SAS$^+$ - Planungsaufgabe [BN95] beschrieben werden. Eine SAS$^+$ - Planungsaufgabe besteht aus einem *Anfangszustand*, einer *Zielbeschreibung*, einer Menge *Variablen* und einer Menge von *Aktionen*. Formal sieht eine SAS$^+$ - Planungsaufgabe wie folgt aus:

Definition 2 (SAS$^+$ - Planungsaufgabe)**.** *Eine* SAS$^+$ *- Planungsaufgabe ist ein 4-Tupel* $\Pi^+ = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G}\rangle$ *mit:*

- $\mathcal{V}$ *einer endlichen Menge von mehrwertigen Variablen, wobei jede Variable* $v \in \mathcal{V}$ *Werte aus einer endlichen Domäne* D_v *annehmen kann. Eine* partielle Variablenbelegung *b bildet eine Teilmenge der Variablen auf Werte aus der jeweiligen Domäne ab, wobei wir den Definitionsbereich von b mit* $vars(b) \subseteq \mathcal{V}$ *bezeichnen. Ein* Zustand *ist eine (partielle) Variablenbelegung s mit* $vars(s) = \mathcal{V}$.
- $\mathcal{A}$ *einer endlichen Menge von Aktionen. Jede Aktion* $a \in \mathcal{A}$ *ist gegeben als Tripel* $a = \langle pre(a), \mathit{eff}(a), cost(a)\rangle$, *wobei die* Vorbedingung $pre(a)$ *und die* Effekte $\mathit{eff}(a)$ *partielle Variablenbelegungen sind und die* Kosten $cost(a) \in \mathbb{R}_0^+$ *eine nicht-negative reelle Zahl darstellen,*
- $\mathcal{I}$, *dem* Anfangszustand, *und*

- $\mathcal{G}$, *der* Zielbeschreibung, *eine partielle Variablenbelegung.*

Für eine partielle Variablenbelegung b bezeichnet $b[v]$ den Wert einer Variable $v \in vars(b)$. Eine Aktion $a \in \mathcal{A}$ ist anwendbar *in Zustand s, wenn $pre(a)[v] = s[v]$ für alle $v \in vars(pre(a))$. Die Anwendung von a in s führt in den* Nachfolgezustand s' *mit*

$$s'[v] = \begin{cases} \mathit{eff}(a)[v] & v \in vars(\mathit{eff}(a)) \\ s[v] & \text{sonst.} \end{cases}$$

Ein Plan *für Π^+ ist eine Folge von Aktionen $\rho = \langle a_1, \ldots, a_n \rangle$, so dass es eine Folge von Zuständen $\langle s_0, s_1, \ldots, s_n \rangle$ gibt mit $s_0 = \mathcal{I}$, $s_n[v] = \mathcal{G}[v]$ für alle $v \in vars(\mathcal{G})$, und für alle $1 \leq i \leq n$ ist Aktion a_i anwendbar in s_{i-1} und führt zu Nachfolgezustand s_i. Plan ρ hat Kosten $cost(\rho) = \sum_{i=1}^{n} cost(a_i)$ und ist* optimal, *falls er minimale Kosten unter allen Plänen für Π^+ hat.*

Sehen wir uns ein Beispiel an und modellieren dabei dasselbe Problem wie in Beispiel 1 als SAS$^+$ - Planungsaufgabe:

Beispiel 2 (SAS$^+$ - Planungsaufgabe)**.** *Wir haben sechs Variablen wovon drei angeben, wo sich ein Block befindet und drei, wie die Situation direkt über einem Block aussieht. Wir verwenden wieder Aktionen um einen Block auf den Tisch oder einen anderen Block zu stellen. Die Aktionen haben andere Effekte und Vorbedingungen.*
Die SAS$^+$ *- Planungsaufgabe sieht wie folgt aus, wobei die Aktionen aus Platzgrüden wiederum generisch dargestellt werden:*
$\Pi^+ = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ *mit* $\mathcal{V} = \{pos(A), pos(B), pos(C), auf(A), auf(B), auf(C)\}$ *und Domänen*

- $D_{pos(A)} = \{\mathbf{B}, \mathbf{C}, \textbf{TISCH}\}$
- $D_{pos(B)} = \{\mathbf{C}, \mathbf{A}, \textbf{TISCH}\}$
- $D_{pos(C)} = \{\mathbf{A}, \mathbf{B}, \textbf{TISCH}\}$
- $D_{auf(A)} = \{\mathbf{B}, \mathbf{C}, \textbf{FREI}\}$
- $D_{auf(B)} = \{\mathbf{C}, \mathbf{A}, \textbf{FREI}\}$
- $D_{auf(C)} = \{\mathbf{A}, \mathbf{B}, \textbf{FREI}\}$

Anfangszustand $\mathcal{I} = \{pos(A) \mapsto \mathbf{B}, pos(B) \mapsto \mathbf{C}, pos(C) \mapsto \textbf{TISCH}, auf(A) \mapsto \textbf{FREI}, auf(B) \mapsto \mathbf{A}, auf(C) \mapsto \mathbf{B}\}$

Zielbeschreibung $\mathcal{G} = \{pos(B) \mapsto \mathbf{A}, pos(C) \mapsto \mathbf{B}, auf(A) \mapsto \mathbf{B}, auf(B) \mapsto \mathbf{C}\}$

Die Aktionen sehen wie folgt aus:
Aktionsgruppe 0: $legeAufTisch(X, Y) = \langle$
$pre(legeAufTisch(X, Y)) = \{pos(X) \mapsto \mathbf{Y}, auf(X) \mapsto \textbf{FREI}, auf(Y) \mapsto \mathbf{X}\}$,
$\mathit{eff}(legeAufTisch(X, Y)) = \{pos(X) \mapsto \textbf{TISCH}, auf(Y) \mapsto \textbf{FREI}\}$,
$cost(legeAufTisch(X, Y)) = 1\rangle$

Aktionsgruppe 1: $legeAufBlock(X, Y) = \langle$
$pre(legeAufBlock(X, Y))=\{pos(Y) \mapsto \textbf{Tisch}, auf(X) \mapsto \textbf{Frei}, auf(Y) \mapsto \textbf{Frei}\}$,
$eff(legeAufBlock(X, Y))=\{pos(Y) \mapsto \mathbf{X}, auf(X) \mapsto \mathbf{Y}\}$,
$cost(legeAufBlock(X, Y))=1\rangle$

Aktionsgruppe 2: $legeAufBlock(X, Y, Z) = \langle$
$pre(legeAufBlock(X, Y, Z))=\{pos(Y) \mapsto \mathbf{Z}, auf(X) \mapsto \textbf{Frei}, auf(Y) \mapsto$
$\textbf{Frei}, auf(Z) \mapsto \mathbf{Y}\}$,
$eff(legeAufBlock(X, Y, Z))=\{pos(Y) \mapsto \mathbf{X}, auf(X) \mapsto \mathbf{Y}, auf(Z) \mapsto \textbf{Frei}\}$,
$cost(legeAufBlock(X, Y, Z))=1\rangle$

Ein Plan für Π^+ *könnte wie folgt aussehen:* $\rho = \langle legeAufTisch(A, B), legeAufBlock(A, B, C), legeAufBlock(B, C)\rangle$ *und hat Kosten* $cost(\rho_2) = 1 + 1 + 1 = 3$. *Die Zustände sehen dann wie folgt aus:*

$pos(A)\mapsto\mathbf{B}$		$pos(A)\mapsto\textbf{Tisch}$		$pos(A)\mapsto\textbf{Tisch}$		$pos(A)\mapsto\textbf{Tisch}$
$pos(B)\mapsto\mathbf{C}$		$pos(B)\mapsto\mathbf{C}$		$pos(B)\mapsto\mathbf{A}$		$pos(B)\mapsto\mathbf{A}$
$pos(C)\mapsto$ **Tisch**	$\rightarrow$	$pos(C)\mapsto\textbf{Tisch}$	$\rightarrow$	$pos(C)\mapsto\mathbf{B}$	$\rightarrow$	$pos(C)\mapsto\textbf{Tisch}$
$auf(A)\mapsto\textbf{Frei}$		$auf(A)\mapsto\textbf{Frei}$		$auf(A)\mapsto\mathbf{B}$		$auf(A)\mapsto\mathbf{B}$
$auf(B)\mapsto\mathbf{A}$		$auf(B)\mapsto\textbf{Frei}$		$auf(B)\mapsto\mathbf{C}$		$auf(B)\mapsto\textbf{Frei}$
$auf(C)\mapsto\mathbf{B}$		$auf(C)\mapsto\mathbf{B}$		$auf(C)\mapsto\textbf{Frei}$		$auf(C)\mapsto\textbf{Frei}$

2.1.3 Von *STRIPS* nach SAS$^+$ und zurück

Die unten behandelte P^m - Kompilierung wurde von Haslum für *STRIPS* - Planungsaufgaben formuliert. Das Fast-Downward Planungssystem [Hel06] verwendet jedoch SAS$^+$ - Planungsaufgaben. Somit wird es nötig, eine *STRIPS* - Planungsaufgabe in eine SAS$^+$ - Planungsaufgabe umwandeln zu können und umgekehrt. Dies geschieht in der Implementierung der P^m - Kompilierung implizit.
Wie wir bereits gesehen haben, unterscheiden sich die beiden Formulierungen primär durch die Menge der Variablen respektive Atome V und $\mathcal{V}$. Während ein Atom $atom \in V$ genau zwei mögliche Belegungen hat, nähmlich `wahr` und `falsch`, kann die Domäne D_v einer Variable $v \in \mathcal{V}$ mehr als zwei Elemente enthalten.

2.1.3.1 *STRIPS* nach SAS$^+$

Eine *STRIPS* - Planungsaufgabe $\Pi = \langle V, A, I, G\rangle$ induziert eine SAS$^+$ - Planungsaufgabe $\Pi^+ = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G}\rangle$ wie folgt:

- $\mathcal{V} = V$, wobei $D_v = \{\texttt{wahr}, \texttt{falsch}\}$ für $v \in \mathcal{V}$,
- $\mathcal{A}$ enthält für jede Aktion $a = \langle pre(a), add(a), del(a), cost(a)\rangle \in A$ eine Aktion $a' = \langle pre(a'), eff(a'), cost(a)\rangle$, wobei $pre(a') = \{v \mapsto \texttt{wahr} \mid v \in pre(a)\}$, und $eff(a') = \{v \mapsto \texttt{wahr} \mid v \in add(a)\} \cup \{v \mapsto \texttt{falsch} \mid v \in del(a) \setminus add(a)\}$.
- $\mathcal{I} = \{v \mapsto \texttt{wahr} \mid v \in I\} \cup \{v \mapsto \texttt{falsch} \mid v \notin I, v \in V\}$
- $\mathcal{G} = \{v \mapsto \texttt{wahr} \mid v \in G\}$

Mit dieser Tranformation induziert jeder Plan $\rho = \langle a_1, \ldots, a_n \rangle$ von Π einen Plan $\rho' = \langle a'_1, \ldots, a'_n \rangle$ von Π^+ (wobei a'_i die für a_i eingeführte Aktion ist) und umgekehrt (ohne Beweis).

2.1.3.2 SAS$^+$ nach *STRIPS*

Eine SAS$^+$ - Planungsaufgabe $\Pi^+ = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ induziert eine *STRIPS* - Planungsaufgabe $\Pi = \langle V, A, I, G \rangle$ wie folgt:

- $V = \{(v, w) \mid v \in \mathcal{V}, w \in D_v\}$
- A enthält für jede Aktion $a = \langle pre(a), \mathit{eff}(a), cost(a) \rangle \in \mathcal{A}$ eine Aktion $a' = \langle pre(a'), add(a'), del(a'), cost(a) \rangle$, wobei
  - $pre(a') = \{(v, w) \mid v \in vars(pre(a)), pre(a)[v] = w\}$
  - $add(a') = \{(v, w) \mid v \in vars(\mathit{eff}(a)), \mathit{eff}(a)[v] = w\}$
  - $del(a') = \{(v, w) \mid v \in vars(\mathit{eff}(a)), w \in D_v, \mathit{eff}(a)[v] \neq w\}$
- $I = \{(v, w) \mid \mathcal{I}[v] = w\}$
- $G = \{(v, w) \mid v \in vars(\mathcal{G}), \mathcal{G}[v] = w\}$

Mit dieser Tranformation induziert jeder Plan $\rho = \langle a_1, \ldots, a_n \rangle$ von Π einen Plan $\rho' = \langle a'_1, \ldots, a'_n \rangle$ von Π^+ (wobei a'_i die für a_i eingeführte Aktion ist) und umgekehrt (ohne Beweis).
Ein kleines Beispiel, basierend auf Beispiel 2:

Beispiel 3 (SAS$^+$ zu *STRIPS*)**.** *Wir haben eine* SAS$^+$ *- Planungsaufgabe* $\Pi^+ = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ *wie in Beispiel 2.*
Wir wollen eine STRIPS - Planungsaufgabe $\Pi = \langle V, A, I, G \rangle$ *kreieren.*
Mit $V = \{(v, w) \mid v \in \mathcal{V}, w \in D_v\}$ *erhalten wir:*
$V = \{(pos(A), \mathbf{B}), (pos(A), \mathbf{C}), (pos(A), \textsc{Tisch}), (pos(B), \mathbf{C}), (pos(B), \mathbf{A}), (pos(B), \textsc{Tisch})$
$(pos(C), \mathbf{A}), (pos(C), \mathbf{B}), (pos(C), \textsc{Tisch}), (\mathit{auf}(A), \mathbf{B}), (\mathit{auf}(A), \mathbf{C}), (\mathit{auf}(A), \textsc{Frei})$
$(\mathit{auf}(B), \mathbf{C}), (\mathit{auf}(B), \mathbf{A}), (\mathit{auf}(B), \textsc{Frei}), (\mathit{auf}(C), \mathbf{A}), (\mathit{auf}(C), \mathbf{B}), (\mathit{auf}(C), \textsc{Frei})\}$
Mit $I = \{(v, w) \mid \mathcal{I}[v] = w\}$ *erhalten wir:*

$I = \{(pos(A), \mathbf{B}), (pos(B), \mathbf{C}), (pos(C), \textsc{Tisch}), (\mathit{auf}(A), \textsc{Frei}), (\mathit{auf}(B), \mathbf{A}), (\mathit{auf}(C), \mathbf{B})\}$
Mit $G = \{(v, w) \mid v \in vars(\mathcal{G}), \mathcal{G}[v] = w\}$ *erhalten wir:*
$G = \{(pos(B), \mathbf{A}), (pos(C), \mathbf{B}), (\mathit{auf}(A), \mathbf{B}), (\mathit{auf}(B), \mathbf{C})\}$

Und mit A *enthält für jede Aktion* $a = \langle pre(a), \mathit{eff}(a), cost(a) \rangle \in \mathcal{A}$ *eine Aktion* $a' = \langle pre(a'), add(a'), del(a'), cost(a) \rangle$, *wobei*
$pre(a') = \{(v, w) \mid v \in vars(pre(a)), pre(a)[v] = w\}$,
$add(a') = \{(v, w) \mid v \in vars(\mathit{eff}(a)), \mathit{eff}(a)[v] = w\}$ *und*
$del(a') = \{(v, w) \mid v \in vars(\mathit{eff}(a)), w \in D_v, \mathit{eff}(a)[v] \neq w\}$
erhalten wir folgende Aktionen.

Aktionsgruppe 0: legeAufTisch$(X, Y) = \langle$
$pre(legeAufTisch(X, Y))=\{pos(X) \mapsto \mathbf{Y}, auf(X) \mapsto$ **Frei**, $auf(Y) \mapsto \mathbf{X}\}$,
$eff(legeAufTisch(X, Y))=\{pos(X) \mapsto$ **Tisch**, $auf(Y) \mapsto$ **Frei**$\}$,
$cost(legeAufTisch(X, Y))=1\rangle$
verändert sich zu:
Aktionsgruppe 0': legeAufTisch$(X, Y) =\langle$
$pre(legeAufTisch(X, Y))=\{ (pos(X), \mathbf{Y}), (auf(X),$ **Frei**$), (auf(Y), \mathbf{X})\}$,
$add(legeAufTisch(X, Y))=\{(pos(X),$ **Tisch**$), (auf(Y),$ **Frei**$)\}$,
$del(legeAufTisch(X, Y))=\{ (pos(X), \mathbf{Y}), (pos(X), \mathbf{Z}), (auf(Y), \mathbf{Z}),$
$(auf(Y), \mathbf{X})\}$,
$cost(legeAufTisch(X, Y))=1\rangle$
Aktionsgruppe 1: legeAufBlock$(X, Y) = \langle$
$pre(legeAufBlock(X, Y))=\{pos(Y) \mapsto$ **Tisch**, $auf(X) \mapsto$ **Frei**, $auf(Y) \mapsto$ **Frei**$\}$,
$eff(legeAufBlock(X, Y))=\{pos(Y) \mapsto \mathbf{X}, auf(X) \mapsto \mathbf{Y}\}$,
$cost(legeAufBlock(X, Y))=1\rangle$
verändert sich zu:
Aktionsgruppe 1': legeAufBlock$(X, Y) =\langle$
$pre(legeAufBlock(X, Y))=\{ (pos(Y),$ **Tisch**$), (auf(X),$ **Frei**$), (auf(Y),$ **Frei**$)\}$,
$add(legeAufBlock(X, Y))=\{(pos(Y), \mathbf{X}), (auf(X), \mathbf{Y})\}$,
$del(legeAufBlock(X, Y))=\{$
$(pos(Y), \mathbf{Z}), (pos(Y),$ **Tisch**$), (auf(X), \mathbf{Z}), (auf(X),$ **Frei**$)\}$,
$cost(legeAufBlock(X, Y))=1\rangle$
Aktionsgruppe 2: legeAufBlock$(X, Y, Z) = \langle$
$pre(legeAufBlock(X, Y, Z))=\{pos(Y) \mapsto \mathbf{Z}, auf(X) \mapsto$ **Frei**, $auf(Y) \mapsto$
Frei, $auf(Z) \mapsto \mathbf{Y}\}$,
$eff(legeAufBlock(X, Y, Z))=\{pos(Y) \mapsto \mathbf{X}, auf(X) \mapsto \mathbf{Y}, auf(Z) \mapsto$ **Frei**$\}$,
$cost(legeAufBlock(X, Y, Z))=1\rangle$
verändert sich zu:
Aktionsgruppe 2': legeAufBlock$(X, Y, Z) =\langle$
$pre(legeAufBlock(X, Y, Z))=\{ (pos(Y), \mathbf{Z}), (auf(X),$ **Frei**$), (auf(Y),$ **Frei**$),$
$(auf(Z), \mathbf{Y})\}$,
$add(legeAufBlock(X, Y, Z))=\{(pos(Y), \mathbf{X}), (auf(X), \mathbf{Y}), (auf(Z),$ **Frei**$)\}$,
$del(legeAufBlock(X, Y, Z))=\{ (pos(Y), \mathbf{Z}), (pos(Y),$ **Tisch**$), (auf(X), \mathbf{Z}),$
$(auf(X),$ **Frei**$)\}$,
$cost(legeAufBlock(X, Y, Z))=1\rangle$

2.2 Heuristiken

Damit der Agent nicht blind seinen Weg ins Ziel finden muss, kann er versuchen die Distanz ins Ziel abzuschätzen und so an einer Verzweigung den kürzeren Weg nehmen. Was hier bildlich dargestellt wurde, findet in den Algorithmen der küstlichen Intelligenz Eingang. Viele Suchalgorithmen verwenden nämlich sogenannte Heuristiken [HNR68]. Formal lässt sich eine Heuristik wie folgt definieren:

Definition 3 (Heuristik)**.** *Sei* $\Pi = \langle V, A, I, G\rangle$ *eine STRIPS - oder* SAS$^+$ *- Planungsaufgabe mit Zustandsmenge* S*. Eine* Heuristikfunktion *für* Π *ist eine Funktion*

$$h : S \to \mathbb{R}_0^+ \cup \{\infty\}$$

die jedem Zustand eine nicht-negative Zahl oder ∞ *zuordnet.*
Wollen wir verdeutlichen, dass die Heuristik h *für die Planungsaufgabe* Π *berechnet wird, schreiben wir auch* h_Π*.*

Das Ziel dabei ist, dass eine Heuristik $h(s)$ die Plankosten eines optimalen Planes für das Problem $\Pi' = \langle V, A, s, G\rangle$ abschätzt. Im Idealfall ist $h(s)$ möglichst nahe an den Plankosten eines optimalen Planes in Π'.
Um Heuristiken klassifizieren zu können, führen wir zunächst die *perfekte Heuristik* ein:

Definition 4 (Perfekte Heuristik)**.** *Sei* $\Pi = \langle V, A, I, G\rangle$ *eine STRIPS - oder* SAS$^+$ *- Planungsaufgabe mit Zustandsmenge* S*. Die* perfekte Heuristik h^* *für* Π *bildet jeden Zustand* $s \in S$ *auf die Plankosten eines optimalen Pfades von* $\Pi' = \langle V, A, s, G\rangle$*. Es gilt:* $h^*(s) = \infty$ *falls die Planungsaufgabe* Π' *unlösbar ist.*

Für die Experimente dieser Arbeit, verwenden wir den A* - Suchalgorithmus [HNR68]. Damit eine von ihm gefundene Lösung garantiert optimal ist, muss die verwendete Heuristik zulässig sein.

Definition 5 (Zulässigkeit von Heuristiken)**.** *Sei* $\Pi = \langle V, A, I, G\rangle$ *eine STRIPS - oder* SAS$^+$ *- Planungsaufgabe mit Zustandsmenge* S*. Eine Heuristik* h *für* Π *heisst* zulässig, *falls* $h(s) \leq h^*(s)$ *für alle* $s \in S$*.*

Je grösser ein von einer zulässigen Heuristik zurückgegebener Wert für einen Zustand s ist, umso besser ist dieser, da dieser Wert nicht grösser als das Optimum $h^*(s)$ sein kann.

2.3 h^m-Heuristik

Eine Familie von zulässigen Heuristiken wollen wir hier genauer betrachten. Es ist die Familie der h^m - Heuristiken [HG00], wobei $m \in \mathbb{N}_1$ und typischerweise $m \leq 3$. Der Spezialfall dieser Heuristiken ist die h^{max} - Heuristik [BG01], welche der h^1 - Heuristik entspricht. Die Idee der h^m - Heuristiken für eine *STRIPS* - Planungsaufgabe $\Pi = \langle V, A, I, G\rangle$ ist es die maximalen Kosten abzuschätzen, um einem Teilmenge von maximal m relevanten Atome aus V kumulativ zu erfüllen. Die h^{max} - Heuristik für Π schätzt für einen Zustand s also die Kosten für das am teuersten zu erreichende Atom aus der Zielbeschreibung.
In der formalen Definition der h^m - Heuristiken orientieren wir uns an der Version von Patrik Haslum, Blai Bonet und Hector Geffner [HBG05].

Definition 6 (h^m - Heuristik)**.** *Sei* $\Pi = \langle V, A, I, G\rangle$ *eine STRIPS - Planungsaufgabe. Für* $m \in \mathbb{N}_1$ *ist die* h^m *- Heuristik definiert als*

$$h^m(s) = \begin{cases} 0 & \text{falls } s \subseteq I \\ min_{s' : s \xrightarrow{a} s' \in R(P)} h^m(s') + cost(a) & \text{falls } |s| \leq m \\ max_{s' \subseteq s, |s'| \leq m} h^m(s') & \text{sonst} \end{cases}$$

Hierbei bezeichnet $R(P)$ den Regressionsraum von Π, d.h. für $a \in A$ und $s, s' \subseteq V$ ist $s \xrightarrow{a} s' \in R(P)$ genau dann, wenn $s \cap del(a) = \emptyset$ und $s' = (s \setminus add(a)) \cup pre(a)$.

Die Komplexität der Berechnung einer h^m - Heurisitk wächst exponentiell mit m. Mit wachsendem m nähert sich h^m immer mehr an h^* an. Für $m = |V|$ gilt: $h^m(s) = h^*(s)$ [KHH14].

2.4 P^m-Kompilierung

Patrik Haslum [Has09] schlägt 2009 als Alternative für die h^m - Heuristik auf einer *STRIPS*-Planungsaufgabe P vor, P zu verändern und die h^{max} - Heuristik über diesem *transformierten Problem* P^m zu berechnen. Er definiert die Transformation von P nach P^m wie folgt:

Definition 7 (P^m - Kompilierung)**.** *Sei $P = \langle V, A, I, G\rangle$ eine STRIPS - Planungsaufgabe. Wir definieren für $X \subseteq V$ die Menge $X^m = \{\pi_c \,|\, c \subseteq X, 1 \leq |c| \leq m\}$. Die Elemente der Menge V^m des transformierten Problemes $P^m\langle V^m, A^m, I^m, G^m\rangle$ nennen wir Meta-Atome π_c.*
Für jede Aktion $a \subseteq A$ und für jede Menge $f \subseteq V$ mit $0 \leq |f| \leq m-1$, so dass f disjunkt von $add(a)$ und von $del(a)$ ist, enthält A^m eine Meta-Aktion $\alpha_{a,f}$ mit:

$$\begin{aligned} pre(\alpha_{a,f}) &= \{\pi_c \,|\, c \subseteq (pre(a) \cup f), 1 \leq |c| \leq m\} \\ add(\alpha_{a,f}) &= \{\pi_c \,|\, c \subseteq (add(a) \cup f), (c \cap add(a)) \neq \emptyset, 1 \leq |c| \leq m\} \\ del(\alpha_{a,f}) &= \emptyset \\ cost(\alpha_{a,f}) &= cost(a) \end{aligned}$$

Der Startzustand von P^m, I^m, setzt alle Meta-Atome $\pi_c\,(|c| \leq m)$ auf `wahr`*, für welche c eine Teilmenge des Startzustandes I von P ist, und alle anderen Meta-Atome sind* `falsch`*.*

Haslum zeigt, dass $h^m_P(s) = h^{max}_{P^m}(s^m)$. Des Weiteren legt er dar, dass die Grösse von P^m polynomiell zur Grösse von P ist, wenn auch exponentiell zu m. Er zeigt, dass $h^*_{P^m}$ grösser als h^*_P sein kann. Dies lässt folgern, dass eine auf P^m zulässige Heuristik auf P nicht zwingend zulässig sein muss.
Haslum postulierte bereits, dass die Berechnung von $h^{max}_{P^m}$ im Vergleich zu h^m_P typischerweise weder einen Speicher noch einen Zeitvorteil liefert. Wir wollen in dieser Arbeit analysieren, wie sich $h^{max}_{P^m}$ im Vergleich zu h^m_P tatsächlich verhält.
Machen wir zur P^m - Kompilierung ein Beispiel mit $m = 2$. Wir bedienen uns wieder an Beispiel 1.

Beispiel 4 (P^2)**.** *Wir haben eine STRIPS - Planungsaufgabe $P = \langle V, A, I, G\rangle$ wie in Beispiel 1. Wir wollen eine P^2 - kompilierte STRIPS - Planungsaufgabe $P^2 = \langle V^2, A^2, I^2, G^2\rangle$ berechnen. Beginnen wir mit den Meta-Atomen: Die Menge der Meta-Atome bildet sich hier aus allen Kombinationen von maximal $m = 2$ Atomen aus P. P enthielt noch 12 Atome. P^2 enthält diese 12 Atome und alle $\binom{12}{2}$ Kombinationen dieser Atome als Meta-Atome. Ergo erhalten wir $12 + \binom{12}{2} = 12 + 66 = 78$ Meta-Atome für P^2. Diese sehen wie folgt aus:*
$V^2 = \{\pi_{\{aufTisch(A)\}}, \pi_{\{aufTisch(B)\}}, \pi_{\{aufTisch(C)\}}, \cdots, \pi_{\{auf(C,B)\}}, \pi_{\{aufTisch(A)aufTisch(B)\}}$,
$\pi_{\{aufTisch(A),aufTisch(C)\}}$, $\pi_{\{aufTisch(A),obenOhne(A)\}}$, $\cdots$, $\pi_{\{auf(B,C),auf(C,A)\}}$,
$\pi_{\{auf(B,C),auf(C,B)\}}$, $\pi_{\{auf(C,A),auf(C,B)\}}\}$

Die Anzahl der Aktionen nimmt noch stärker zu. Während wir in P jeweils sechs Aktionen in den Aktionsgruppen legeAufTisch(X, Y), legeAufBlock(X, Y) und legeAufBlock(X, Y, Z) also insgesamt 18 Aktionen haben, entstehen in P^2 jeweils zehn Meta-Aktionen pro Aktion der Gruppen legeAufTisch(X, Y) und legeAufBlock(X, Y) und jeweils neun Meta-Aktionen pro Aktion der Gruppe legeAufBlock(X, Y, Z) also insgesamt $10 \times 6 + 10 \times 6 + 9 \times 6 = 174$ Meta-Aktionen. Sehen wir uns eine Aktion $a = legeAufTisch(B, C)$ und die daraus abgeleitenen Meta-Aktionen an:
Diese hat die Vorbedingungen $pre(a) = \{obenOhne(B), auf(C, B)\}$ und die Effekte: $add(a) = \{aufTisch(B), obenOhne(C)\}$ sowie $del(a) = \{auf(C, B)\}$. Die Kandidaten für f entsprechen den Meta-Atomen von $P^{m-1} = P^{2-1} = P^1$ also den Atomen von P, da die Kandidaten maximal aus $m - 1$ Atomen bestehen dürfen, und der leeren Menge $\emptyset$. Für diese müssen wir überprüfen, ob Sie zu add(a) und del(a) disjunkt sind. Daher fallen Kandidaten, die $aufTisch(B)$, $obenOhne(C)$ oder $auf(C, B)$ enthalten weg. Somit erhalten wir für jedes $f \in \{aufTisch(A), aufTisch(C), obenOhne(A), obenOhne(B), auf(A, B), auf(A, C), auf(B, A), auf(B, C), auf(C, A), \emptyset\}$ eine Meta-Aktion.
Wir erhalten also folgende zehn Meta-Aktionen:

$f = aufTisch(A)$ $pre(\alpha_{a,f}) =$ $\{\pi_{\{aufTisch(A)\}}, \pi_{\{aufTisch(A),obenOhne(B)\}}, \pi_{\{aufTisch(A),auf(C,B)\}}, \pi_{\{obenOhne(B)\}}, \pi_{\{obenOhne(B),auf(C,B)\}}, \pi_{\{auf(C,B)\}}\},$ $add(\alpha_{a,f}) =$ $\{\pi_{\{aufTisch(A),aufTisch(B)\}}, \pi_{\{aufTisch(A),obenOhne(C)\}}, \pi_{\{aufTisch(B)\}}, \pi_{\{aufTisch(B),obenOhne(C)\}}, \pi_{\{obenOhne(C)\}}\},$ $del(\alpha_{a,f}) = \emptyset$	$f = aufTisch(C)$ $pre(\alpha_{a,f}) =$ $\{\pi_{\{aufTisch(C)\}}, \pi_{\{aufTisch(C),obenOhne(B)\}}, \pi_{\{aufTisch(C),auf(C,B)\}}, \pi_{\{obenOhne(B)\}}, \pi_{\{obenOhne(B),auf(C,B)\}}, \pi_{\{auf(C,B)\}}\},$ $add(\alpha_{a,f}) =$ $\{\pi_{\{aufTisch(C),aufTisch(B)\}}, \pi_{\{aufTisch(C),obenOhne(C)\}}, \pi_{\{aufTisch(B)\}}, \pi_{\{aufTisch(B),obenOhne(C)\}}, \pi_{\{obenOhne(C)\}}\},$ $del(\alpha_{a,f}) = \emptyset$
$f = obenOhne(A)$ $pre(\alpha_{a,f}) =$ $\{\pi_{\{obenOhne(A)\}}, \pi_{\{obenOhne(A),obenOhne(B)\}}, \pi_{\{obenOhne(A),auf(C,B)\}}, \pi_{\{obenOhne(B)\}}, \pi_{\{obenOhne(B),auf(C,B)\}}, \pi_{\{auf(C,B)\}}\},$ $add(\alpha_{a,f}) =$ $\{\pi_{\{obenOhne(A),aufTisch(B)\}}, \pi_{\{obenOhne(A),obenOhne(C)\}}, \pi_{\{aufTisch(B)\}}, \pi_{\{aufTisch(B),obenOhne(C)\}}, \pi_{\{obenOhne(C)\}}\},$ $del(\alpha_{a,f}) = \emptyset$	$f = obenOhne(B)$ $pre(\alpha_{a,f}) =$ $\{\pi_{\{obenOhne(B)\}}, \pi_{\{obenOhne(B),auf(C,B)\}}, \pi_{\{auf(C,B)\}}\},$ $add(\alpha_{a,f}) =$ $\{\pi_{\{obenOhne(B),aufTisch(B)\}}, \pi_{\{obenOhne(B),obenOhne(C)\}}, \pi_{\{aufTisch(B)\}}, \pi_{\{aufTisch(B),obenOhne(C)\}}, \pi_{\{obenOhne(C)\}}\},$ $del(\alpha_{a,f}) = \emptyset$
$f = auf(A, B)$ $pre(\alpha_{a,f}) =$ $\{\pi_{\{auf(A,B)\}}, \pi_{\{auf(A,B),obenOhne(B)\}}, \pi_{\{auf(A,B),auf(C,B)\}}, \pi_{\{obenOhne(B)\}}, \pi_{\{obenOhne(B),auf(C,B)\}}, \pi_{\{auf(C,B)\}}\},$ $add(\alpha_{a,f}) =$ $\{\pi_{\{auf(A,B),aufTisch(B)\}}, \pi_{\{auf(A,B),obenOhne(C)\}}, \pi_{\{aufTisch(B)\}}, \pi_{\{aufTisch(B),obenOhne(C)\}}, \pi_{\{obenOhne(C)\}}\},$ $del(\alpha_{a,f}) = \emptyset$	$f = auf(A, C)$ $pre(\alpha_{a,f}) =$ $\{\pi_{\{auf(A,C)\}}, \pi_{\{auf(A,C),obenOhne(B)\}}, \pi_{\{auf(A,C),auf(C,B)\}}, \pi_{\{obenOhne(B)\}}, \pi_{\{obenOhne(B),auf(C,B)\}}, \pi_{\{auf(C,B)\}}\},$ $add(\alpha_{a,f}) =$ $\{\pi_{\{auf(A,C),aufTisch(B)\}}, \pi_{\{auf(A,C),obenOhne(C)\}}, \pi_{\{aufTisch(B)\}}, \pi_{\{aufTisch(B),obenOhne(C)\}}, \pi_{\{obenOhne(C)\}}\},$ $del(\alpha_{a,f}) = \emptyset$

$f = auf(B, A)$ $pre(\alpha_{a,f}) =$ $\{\pi_{\{auf(B,A)\}}, \pi_{\{auf(B,A),obenOhne(B)\}}, \pi_{\{auf(B,A),auf(C,B)\}}, \pi_{\{obenOhne(B)\}}, \pi_{\{obenOhne(B),auf(C,B)\}}, \pi_{\{auf(C,B)\}}\},$ $add(\alpha_{a,f}) =$ $\{\pi_{\{auf(B,A),aufTisch(B)\}}, \pi_{\{auf(B,A),obenOhne(C)\}}, \pi_{\{aufTisch(B)\}}, \pi_{\{aufTisch(B),obenOhne(C)\}}, \pi_{\{obenOhne(C)\}}\},$ $del(\alpha_{a,f}) = \emptyset$	$f = auf(B, C)$ $pre(\alpha_{a,f}) =$ $\{\pi_{\{auf(B,C)\}}, \pi_{\{auf(B,C),obenOhne(B)\}}, \pi_{\{auf(B,C),auf(C,B)\}}, \pi_{\{obenOhne(B)\}}, \pi_{\{obenOhne(B),auf(C,B)\}}, \pi_{\{auf(C,B)\}}\},$ $add(\alpha_{a,f}) =$ $\{\pi_{\{auf(B,C),aufTisch(B)\}}, \pi_{\{auf(B,C),obenOhne(C)\}}, \pi_{\{aufTisch(B)\}}, \pi_{\{aufTisch(B),obenOhne(C)\}}, \pi_{\{obenOhne(C)\}}\},$ $del(\alpha_{a,f}) = \emptyset$
$f = auf(C, A)$ $pre(\alpha_{a,f}) =$ $\{\pi_{\{auf(C,A)\}}, \pi_{\{auf(C,A),obenOhne(B)\}}, \pi_{\{auf(C,A),auf(C,B)\}}, \pi_{\{obenOhne(B)\}}, \pi_{\{obenOhne(B),auf(C,B)\}}, \pi_{\{auf(C,B)\}}\},$ $add(\alpha_{a,f}) =$ $\{\pi_{\{auf(C,A),aufTisch(B)\}}, \pi_{\{auf(C,A),obenOhne(C)\}}, \pi_{\{aufTisch(B)\}}, \pi_{\{aufTisch(B),obenOhne(C)\}}, \pi_{\{obenOhne(C)\}}\},$ $del(\alpha_{a,f}) = \emptyset$	$f = \emptyset$ $pre(\alpha_{a,f}) =$ $\{\pi_{\{obenOhne(B)\}}, \pi_{\{obenOhne(B),auf(C,B)\}}, \pi_{\{auf(C,B)\}}\},$ $add(\alpha_{a,f}) =$ $\{\pi_{\{aufTisch(B)\}}, \pi_{\{aufTisch(B),obenOhne(C)\}}, \pi_{\{obenOhne(C)\}}\},$ $del(\alpha_{a,f}) = \emptyset$

Eine Meta-Aktion hat dieselben Aktionskosten $cost(\alpha_{a,f}) = cost(a)$ *wie die Aktion* a*, aus welcher sie gebildet wird. Es zeigt sich, dass für den Fall* $m = 2$ *Anzahl der Kandidaten, welche für eine Aktion* a *als* f *in Frage kommen, der Anzahl der Atome in* P *abzüglich der Anzahl der Effekte in* $add(a) \cup del(a)$ *zuzüglich 1 für die leere Menge ist.*
Der Startzustand I^m *besteht aus allen Meta-Atomen* π_c*, bei denen die Atommenge* $c \subseteq V$*.*
Mit $I = \{aufTisch(C), obenOhne(A), auf(B, A), auf(C, B)\}$ *erhalten wir:*
$I^2 = \{\pi_{\{aufTisch(C)\}}, \pi_{\{obenOhne(A)\}}, \pi_{\{auf(B,A)\}}, \pi_{\{auf(C,B)\}}, \pi_{\{aufTisch(C),obenOhne(A)\}}, \pi_{\{aufTisch(C),auf(B,A)\}}, \pi_{\{aufTisch(C),auf(C,B)\}}, \pi_{\{obenOhne(A),auf(B,A)\}}, \pi_{\{obenOhne(A),auf(C,B)\}}, \pi_{\{auf(B,A),auf(C,B)\}}\}$
Der Zielzustand G^m *ist so über den Zielzustand* G *von* P *definiert:* $G^m = \{\pi_c \,|\, c \subseteq G, 1 \leq |c| \leq m\}$*. Somit erhalten wir für* $G = \{auf(A, B), auf(B, C)\}$ *folgenden Zielzustand:*
$G^2 = \{\pi_{\{auf(A,B)\}}, \pi_{\{auf(A,B),auf(B,C)\}}, \pi_{\{auf(B,C)\}}\}$.

3
Ergebnisse

Um zu analysieren, wie sich h^m_P zu $h^{max}_{P^m}$ verhält, wurden einige Experimente durchgeführt. Es wurde das Fast-Downward Planungssystem [Hel06] und die bereits darin implementierten h^m - und h^{max} - Heuristiken verwendet. Neu implementiert wurde die P^m - Kompilierung. Diese berechnet vor dem ersten Aufruf der Heuristik das transformierte Problem und behält dieses im Speicher. Die Umwandlung von SAS$^+$ nach *STRIPS* und zurück geschieht implizit während der P^m - Kompilierung.

Die Experimente wurden mit LAB[4] generiert, durchgeführt und zusammengefasst. Für die Experimente wurde der A* - Suchalgorithmus [HNR68] benutzt. Als Benchmark wurden sämtliche *STRIPS* - Planungsaufgaben aus den Benchmarks für optimales Planen der **I**nternational **P**lanning **C**ompetition (IPC) von 1998 bis 2011 verwendet. Es wurden für $m = \{1, 2, 3\}$ jeweils h^m_P und $h^{max}_{P^m}$ getestet. Als Limiten wurden jeweils 2048 MB Speicher und 30 Minuten Rechenzeit eingesetzt. Die Binaries wurden für 32-Bit Wortbreite erstellt.

3.1 Erfolgsrate

Wir wollen uns zunächst ansehen, für welche Konfiguration der Planer am erfolgreichsten ist, also am meisten Probleminstanzen in den gegebenen 30 Minuten und 2048 MB Speicher löst. Wir nehmen für die Stufe $m = 1$ die Konfigurationen h^{max}_P, $h^{max}_{P^1}$ sowie h^1_P, für die Stufe $m = 2$ die Konfigurationen $h^{max}_{P^2}$ sowie h^2_P sowie für die Stufe $m = 3$ die Konfigurationen $h^{max}_{P^3}$ sowie h^3_P.
Generell ist zu erwarten, dass Konfigurationen mit kleinerem m besser abschliessen, da die Komplexität der Berechnung der Heuristik mit wachsendem m stark zunimmt. Diese Komplexität zeigt sich besonders im Speicherbedarf für die P^m - kompilierten Tasks. Daher ist auch anzunehmen, dass für gleiches m die Variante h^m_P mehr Instanzen löst als $h^{max}_{P^m}$. Auf Stufe $m = 1$ erwartet man, dass h^{max}_P am meinsten Instanzen löst, da diese Koniguration keine P^1 - Komplierung durchführt, wofür Zeit und Speicher benötigt würde, und die Implementierung von h^{max} im Fast-Downward Planungssystem schneller als diejenige für h^1

[4] Weitere Informationen zum LAB finden sich unter http://lab.rtfd.org

ist. Das Fast-Downward Planungssystem warnt for der Nutzung der h^m - Heuristik, da diese besonders langsam sei.

Erfolgsrate	h_P^{max}	$h_{P^1}^{max}$	h_P^1	$h_{P^2}^{max}$	h_P^2	$h_{P^3}^{max}$	h_P^3
airport (50)	**22**	**22**	17	11	7	2	2
barman-opt11-strips (20)	**8**	**8**	0	0	0	0	0
blocks (35)	**18**	**18**	17	16	10	9	8
depot (22)	**6**	**6**	2	2	2	1	1
driverlog (20)	**9**	**9**	7	7	2	4	2
elevators-opt08-strips (30)	**16**	**16**	6	9	0	1	0
elevators-opt11-strips (20)	**13**	**13**	4	7	0	0	0
floortile-opt11-strips (20)	**6**	**6**	2	2	0	0	0
freecell (80)	**15**	**15**	7	7	1	0	0
grid (5)	**2**	**2**	1	1	0	0	0
gripper (20)	**7**	**7**	6	5	3	3	2
logistics00 (28)	**10**	**10**	**10**	**10**	6	**10**	6
logistics98 (35)	**2**	**2**	**2**	**2**	0	**2**	0
miconic (150)	**50**	**50**	44	45	30	35	25
movie (30)	**30**	**30**	**30**	**30**	**30**	**30**	**30**
mprime (35)	**24**	**24**	16	15	4	1	1
mystery (30)	**17**	**17**	13	11	8	2	2
nomystery-opt11-strips (20)	**8**	**8**	7	**8**	6	6	4
openstacks-opt08-strips (30)	**21**	**21**	12	10	5	5	1
openstacks-opt11-strips (20)	**16**	**16**	7	5	1	1	0
openstacks-strips (30)	**7**	**7**	6	5	5	5	0
parcprinter-08-strips (30)	**15**	**15**	12	12	8	7	6
parcprinter-opt11-strips (20)	**11**	**11**	8	8	4	3	2
parking-opt11-strips (20)	**0**	**0**	**0**	**0**	**0**	**0**	**0**
pathways-noneg (30)	**4**	**4**	**4**	**4**	3	3	2
pegsol-08-strips (30)	**27**	**27**	26	26	9	2	6
pegsol-opt11-strips (20)	**17**	**17**	16	16	1	0	1
pipesworld-notankage (50)	**18**	17	11	8	3	0	0
pipesworld-tankage (50)	**11**	**11**	6	6	2	0	0
psr-small (50)	**49**	**49**	46	48	40	39	30
rovers (40)	**6**	**6**	5	5	4	4	4
satellite (36)	**6**	**6**	4	4	3	3	2
scanalyzer-08-strips (30)	**9**	**9**	6	6	3	3	3
scanalyzer-opt11-strips (20)	**6**	**6**	3	3	1	1	1
sokoban-opt08-strips (30)	**28**	**28**	18	13	4	1	0
sokoban-opt11-strips (20)	**20**	**20**	15	10	1	0	0
storage (30)	**15**	**15**	12	12	7	5	6
tidybot-opt11-strips (20)	**13**	**13**	3	0	1	0	0
tpp (30)	**6**	**6**	5	5	5	5	4
transport-opt08-strips (30)	**11**	**11**	10	10	6	6	4
transport-opt11-strips (20)	**6**	**6**	5	5	1	1	0
trucks-strips (30)	**8**	**8**	4	5	2	3	1
visitall-opt11-strips (20)	**9**	**9**	**9**	**9**	8	8	7
woodworking-opt08-strips (30)	**9**	**9**	7	6	4	2	1
woodworking-opt11-strips (20)	**4**	**4**	2	1	0	0	0
zenotravel (20)	**8**	**8**	7	**8**	5	6	4
Total (1456)	**623**	622	460	438	245	219	168

Wir sehen in der Tabelle, dass für kleinere m mehr Instanzen gelöst wurden. Am effizientesten ist h_P^{max}, jedoch nur mit einer Instanz mehr als $h_{P^1}^{max}$. Aufgrund dieser ähnlichen Ergebnisse wird in den folgenden Auwertungen auf h_P^{max} verzichtet. Dass h_P^{max} die erfolgreichste Konfiguration ist, haben wir bereits erwartet. Dass $h_{P^1}^{max}$ jedoch nur eine Instanz weniger löst und somit die zweit erfolgreichste Konfiguration ist, war nicht absehbar. Dies liegt, wie wir später sehen werden, daran, dass $h_{P^1}^{max}$ viel weniger Zeit benötigt als h_P^1, jedoch nur minim mehr Speicher. Ebenfalls löst für $m = 2$ und $m = 3$ $h_{P^m}^{max}$ mehr Instanzen als h_P^m für jeweils gleiches m. Dies folgt daraus, dass die Zeiteinsparungen von $h_{P^m}^{max}$ wesentlich deutlicher ausfallen, als der zusätzlich benötigte Speicher.

3.2 Plankosten, $h(I)$ und Expansionen

Da die verwendeten Heuristiken zulässig sind, findet der A* - Suchalgorithmus jeweils die optimalen Plankosten. Laut der Theorie $h^m_P = h^{max}_{P^m}$ muss dies auch für den jeweiligen Startzustand I gelten. Da aus dem Fall $h(I) = \infty$ folgt, dass das Problem unlösbar ist, ist dieser Wert ein Indikator für unlösbare Probleme. Wir erwarten, dass mit wachsendem m die Werte $h(I)$ anwachsen, was für zulässige Heuristiken wünschenswert ist. Dies ist in der Abbildung 3.1, in welcher die Heuristikwerte für den Startzustand einer Probleminstanz für $m = 2$ und $m = 3$ gegen denjenigen von $m = 1$ dargestellt werden, ersichtlich.
Mit wachsendem m sind die Heuristiken besser informiert, da sie mehr Variablen betrachten. Dies führt dazu, dass die Anzahl der Expansionen, also die untersuchten Zustände, mit wachsendem m abnimmt. Das Einzige was sich für A* durch die P^m - Kompilierung ändert, ist der Heuristikwert. Da aber laut Theorie $h^m_P = h^{max}_{P^m}$ gilt, bleibt der Heuristikwert desselben Zustandes für gleiches m gleich. In der Abbildung 3.2 sieht man die Zahl der Expansionen einer Probleminstanz für $m = 2$ und $m = 3$ gegen diejenige von $m = 1$ dargestellt. Man kann erkennen, dass mit $m = 3$ weniger Expansionen benötigt werden, was an den genaueren Heuristikabschätzungen liegt.

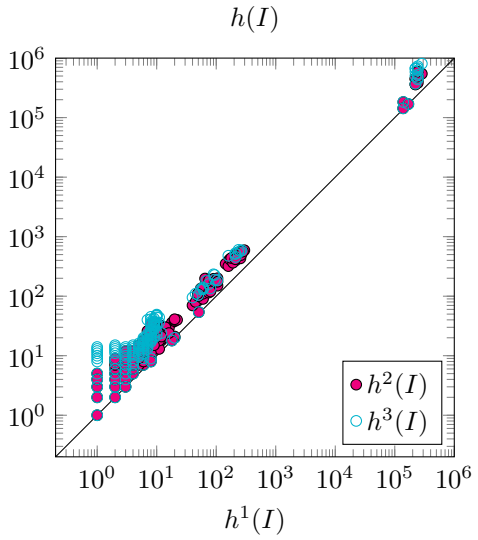

Abbildung 3.1: Vergleich der Heuristikwerte für den Startzustand I $h^2(I)$ und $h^3(I)$ mit $h^1(I)$

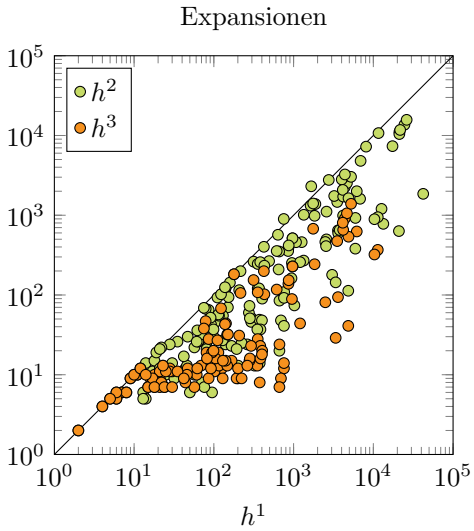

Abbildung 3.2: Vergleich der Expansionen für h^2 und h^3 mit h^1

Man kann auf den Graphiken erkennen, dass generell gilt $h^3(I) > h^2(I)$ und dass die Zahl der Expansionen mit für $m = 3$ zumeist kleiner ist als für $m = 2$.

3.3 Anzahl der Variablen und Aktionen

3.3.1 Variablen

Wie bereits erwähnt, verwendet das Fast-Downward Planungssystem intern die SAS$^+$ - Repräsentation. Somit werden Variablen und nicht Atome betrachtet. Für transformierte Probleme P^m haben alle Variablen v Domänengrösse $|D_v| = 2$, da sie als mögliche Belegungen jeweils genau ein Atom und dessen Negierung haben.
Die Theorie von Haslum [Has09] sagt die Anzahl der Variablen $|V^m|$ einer transformierten Planungsaufgabe $P^m = \langle V^m, A^m, I^m, G^m \rangle$ als Funktion der Anzahl der Variablen $|V|$ im originalen *STRIPS* - Problem $P = \langle V, A, I, G \rangle$ voraus.

$$|V^m| = \sum_{i=1}^{m} \binom{|V|}{i}$$

Diese Korrelation wurde in den Experimenten bestätigt und wird nebenan graphisch dargestellt.

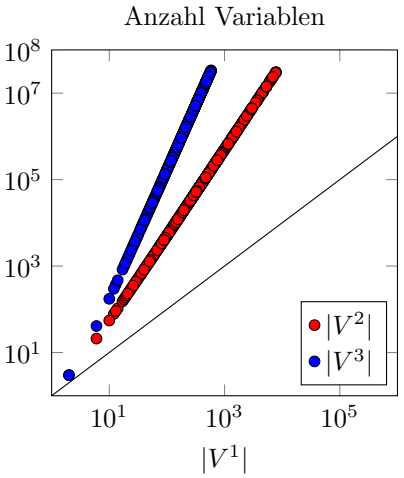

Abbildung 3.3: Vergleich der Anzahl Variablen in den Problemen P^2 und P^3 mit derjenigen in P^1

3.3.2 Aktionen

Aufgrund der Definition der P^m - Kompilierung können wir erwarten, dass die Anzahl der Aktionen $|A^m|$ in einer aus *STRIPS* - Planungsaufgabe $P = \langle V, A, I, G \rangle$ transformierten *STRIPS* - Planungsaufgabe $P^m = \langle V^m, A^m, I^m, G^m \rangle$ instanzenabhängig ist. Die Anzahl M_a der Meta-Aktionen $\alpha_{a,f}$ welche aus einer Aktion $a \in A$ entstehen, entspricht der Anzahl der Teilmengen von Atomen aus P, welche aus maximal m Atomen bestehen und disjunkt zu den Effekten von a sind, also

$$M_a = \sum_{i=0}^{m-1} \binom{|V| - |add(a) \cup del(a)|}{i}.$$

Somit erwarten wir

$$|A^m| = \sum_{a \in A} \sum_{i=0}^{m-1} \binom{|V| - |add(a) \cup del(a)|}{i}.$$

Dies wurde überprüft und verifiziert.
In Abbildung 3.4 sehen wir die Anzahl der Aktionen in den P^2 - und P^3 - kompilierten Probleminstanzen gegen die Anzahl der Aktionen im originalen Problem P. Es ist erkennbar, dass mit $m = 3$ deutlich mehr Aktionen als für $m = 2$ entstehen.
Anhand zweier Domänen wollen wir ein wenig genauer analysieren, wie sich die Anzahl der Aktionen in transformierten Problemen für unterschiedliche Domänen unterscheidet. In Abbildung 3.5 sehen wir die Anzahl der Aktionen für P^2 - kompilierte Probleminstanzen im Vergleich zur Anzahl in der P^1 - kompilierten Instanz, nach Domänen aufgeschlüsselt. Wir betrachten insbesondere die Domänen `airport` und `movie`.

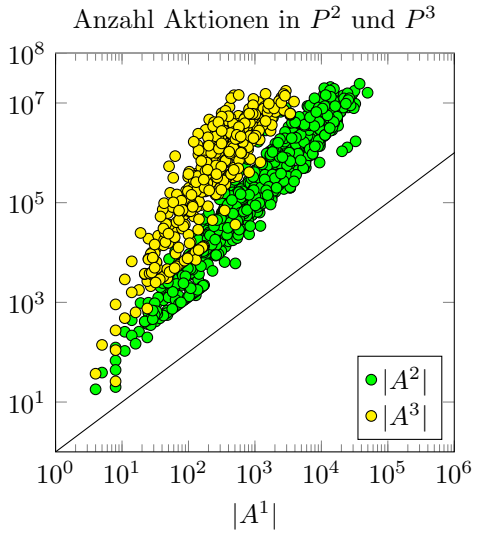

Abbildung 3.4: Vergleich der Anzahl Aktionen in den Problemen P^2 und P^3 mit der Anzahl der Aktionen in P^1

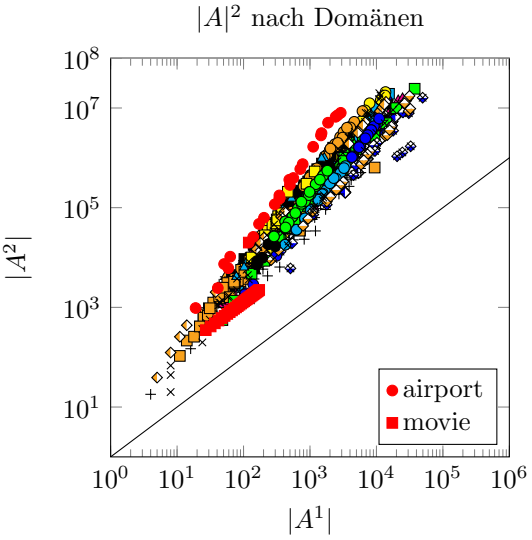

Abbildung 3.5: Vergleich der Anzahl Aktionen in den Problemen P^2 mit der Anzahl der Aktionen in P^1, nach Domänen aufgeschlüsselt

In der Domäne `airport` ist eine Streuung der Datenpunkte wahrzunehmen. Die Domäne mit den Meisten neuen Aktionen scheint `airport` zu sein. Die Domäne `movie` scheint eine Linie parallel zu $x = y$ zu bilden, es besteht also kaum eine Streuung innerhalb der Domäne. Innerhalb der Domäne `movie` sind nur Probleme, welche der SAS$^+$ - Repräsentation einer *STRIPS* - Planungsaufgabe entsprechen. Also alle Variablen v dieser Probleme haben $|D_v| = 2$ und beinhalten im Wertebereich ein Atom und dessen Negierung. Zudem beinhalten alle Probleme dieselben Gruppen von Aktionen, weshalb wir kaum eine Streuung für diese Domäne in der Abbildung 3.5 sehen. Die Aktionen haben zudem sehr wenige Vorbedingungen und Effekte.
Innerhalb der Domäne `airport` existieren in einigen Problemen Variablen v mit grossen Wertebereichen $|D_v|$. Es wurden Variablen mit $|D_v| = 257$ gefunden. Die Aktionen der Probleme unterscheiden sich stark. Daher sehen wir in der Abbildung 3.5 eine Streuung für diese Domäne.

3.4 Transformations-Zeit

Die Transformationszeit ist die Zeit, die für die P^m - Kompilierung benötigt wird. Diesen Wert gibt es nur für die Experimente, welche mit der $h^{max}_{P^m}$ - Heuristik laufen. Da die Anzahl von Atomen und Aktionen bei der Transformation mit wachsendem m stark anwachsen, ist die Transformationszeit eine Funktion von m.

Transformations Zeit	$h^{max}_{P^1}$	$h^{max}_{P^2}$	$h^{max}_{P^3}$
airport (7)	**0.00**	0.01	0.47
barman-opt11-strips (4)	**0.00**	0.25	49.71
blocks (28)	**0.00**	**0.00**	0.06
depot (4)	**0.00**	0.01	0.46
driverlog (12)	**0.00**	0.01	0.15

elevators-opt08-strips (29)	**0.00**	0.04	1.93
elevators-opt11-strips (20)	**0.00**	0.06	3.72
floortile-opt11-strips (14)	**0.00**	0.03	1.97
freecell (7)	**0.00**	0.16	13.33
gripper (20)	**0.00**	**0.00**	0.03
logistics00 (22)	**0.00**	**0.00**	0.08
logistics98 (5)	**0.00**	0.01	0.41
miconic (135)	**0.00**	**0.00**	**0.00**
movie (30)	**0.00**	**0.00**	0.01
mprime (4)	**0.00**	0.05	2.24
mystery (8)	**0.00**	**0.00**	**0.00**
nomystery-opt11-strips (10)	**0.00**	0.01	0.24
openstacks-opt08-strips (20)	**0.00**	**0.00**	0.09
openstacks-opt11-strips (15)	**0.00**	0.03	1.68
openstacks-strips (7)	**0.00**	0.02	0.61
parcprinter-08-strips (18)	**0.00**	**0.00**	0.07
parcprinter-opt11-strips (14)	**0.00**	0.01	0.76
pathways-noneg (6)	**0.00**	**0.00**	0.14
pegsol-08-strips (30)	**0.00**	0.01	0.60
pegsol-opt11-strips (20)	**0.00**	0.06	4.50
pipesworld-notankage (6)	**0.00**	0.06	7.17
pipesworld-tankage (6)	**0.00**	0.03	2.78
psr-small (46)	**0.00**	**0.00**	**0.00**
rovers (16)	**0.00**	**0.00**	0.02
satellite (10)	**0.00**	**0.00**	0.01
scanalyzer-08-strips (9)	**0.00**	0.03	0.36
scanalyzer-opt11-strips (6)	**0.00**	0.03	0.37
sokoban-opt08-strips (17)	**0.00**	0.02	1.44
sokoban-opt11-strips (14)	**0.00**	0.06	6.66
storage (14)	**0.00**	**0.00**	**0.00**
tpp (10)	**0.00**	**0.00**	**0.00**
transport-opt08-strips (12)	**0.00**	0.01	0.09
transport-opt11-strips (8)	**0.00**	0.06	2.35
trucks-strips (9)	**0.00**	0.02	0.68
visitall-opt11-strips (20)	**0.00**	**0.00**	**0.00**
woodworking-opt08-strips (11)	**0.00**	0.03	1.65
woodworking-opt11-strips (6)	**0.00**	0.13	14.58
zenotravel (13)	**0.00**	**0.00**	0.03
Minimum (722)	**0.00**	**0.00**	**0.00**
Maximum (722)	**0.00**	0.25	49.71
Summe (722)	**0.53**	94.06	11477.06
Geometrisches Mittel (722)	**0.00**	**0.00**	**0.00**

Wie sich zeigt, nehmen die Transformationskosten exponentiell mit m zu. Dies ist der Grund, weshalb die P^m - Kompilierung keine Anwendung für grosse m findet.
Vergleichen wir noch die jeweilige totale Zeit mit der Transformationszeit für alle Probleme, für die der Planer eine Lösung gefunden hat. Es fällt auf, dass die Transformationszeit nicht gross ins Gewicht fällt. In der folgenden Tabelle können die Geometrischen Mittel von $h^{max}_{P^1}$, $h^{max}_{P^2}$ und $h^{max}_{P^3}$ nicht verglichen werden, da über unterschiedlich viele Instanzen gemittelt wurde.

Zeit	$h^{max}_{P^1}$	$h^{max}_{P^2}$	$h^{max}_{P^3}$
Transformationszeit Geometrisches Mittel (622, 438, 219)	**0.00**	0.04	0.33
Totale Zeit Geometrisches Mittel (622, 438, 219)	**1.57**	5.54	8.23
Prozent der totalen Zeit (622, 438, 219)	0%	0.72%	8.92%

Dass der Wert für $h^{max}_{P^m}$ in der unmittelbar obigen Tabelle kleiner ist als in der ersten Tabelle dieses Abschnitts, liegt an den verwendeten Daten. In beiden Tabellen werden die Werte als geometrischen Durchschnitt über die einzelnen Probleme dargestellt. Während in der zweiten Tabelle nur für diejenigen Probleme die Transformationszeiten genommen wurden, für die der Planer in 1800 Sekunden und mit weniger als 2048 MB Speicher eine Lösung gefunden hat, berücksichtigt die erste Tabelle die Transformationszeiten für alle Probleme, die

in dieser Zeit und mit diesem Speicher die Transformation abgeschlossen haben. Je länger die Transformation dauert, umso komplexer stellt sich auch die Suche dar. Somit fallen die grösseren Transformationszeiten in der unteren Tabelle hinaus.

Dass die totale Zeit in der obigen Tabelle andere Werte annimmt als in derjenigen im Abschnitt 3.5, liegt an den verglichenen Heuristiken. In der ersten Tabelle werden nur Probleme miteinbezogen, die von $h^{max}_{P^m}$ und von h^m_P für $m = \{1, 2, 3\}$ komplett gelöst wurden. In der hier gezeigten Tabelle werden in jeder Spalte sämtliche Probleminstanzen betrachtet, für welche die entsprechende $h^{max}_{P^m}$ - Konfiguration einen Plan gefunden hat.

Erfolgreiche Transformationen	$h^{max}_{P^1}$	$h^{max}_{P^2}$	$h^{max}_{P^3}$
Summe(1456)	**1456**	1316	722

Gelöste Probleme	$h^{max}_{P^1}$	$h^{max}_{P^2}$	$h^{max}_{P^3}$
Summe(1456)	**463**	251	164

In den obigen Tabellen sehen wir zum einen, wie viele Transformationen abgeschlossen wurden, zum anderen, wie viele Probleme durch dieselben Konfigurationen gelöst wurden. Wie zu erwarten, nehmen diese Werte mit wachsendem m ab. Die Zahl der abgeschlossenen Transformationen ist um einiges höher, als die Erfolgrate für diese Konfigurationen.

3.5 Zeitaufwand

Der Zeitaufwand einer Plansuche kann auf verschiedene Weisen abgeschätzt werden. Wir haben bereits in Abschnitt 3.2 die Anzahl der expandierten Zustände angeschaut. Diese ist für gleiches m für h^m_P und $h^{max}_{P^m}$ gleich. Im Abschnitt 3.4 haben wir für die P^m - kompilierten Probleme die Transformationszeit verglichen. Diese nimmt mit wachsendem m zu. Im Folgenden sehen wir uns die Zeit um die Probleminstanzen zu lösen an.

Das Fast-Downward Planungssystem [Hel06] löst ein Problem in drei Schritten. Zuerst läuft der Übersetzer gefolgt vom Preprozessor. In diesen beiden Schritten wird grundsätzlich ein Problem aus einem `.pddl` - File erstellt. Diese beiden Schritte sind für jedes Problem über alle Suchkonfigurationen konstant und zumeist vernachlässigbar klein. Aus diesem Grund wird in dieser Arbeit nicht weiter darauf eingegangen. Es folgt als dritter Schritt die Suche. Wenden wir die P^m - Kompilierung an, findet zwischen dem Preprozessor und der Suche die Transformation statt. Somit ist die *Suchzeit* diejenige Zeit, welcher der Planer für die Suche benötigt. Die *totale Zeit* ist die gesamte Zeit, die benötigt wurde um, das Problem zu lösen. Dies beinhaltet die Suchzeit, die Übersetzungszeit, die Preprozessor-Zeit und für die P^m - kompilierten Probleme aus der Transformationszeit.

In den Tabellen sieht man jeweils die totale, respektive die Suchzeit für $h^{max}_{P^m}$ und h^m_P für gleiches m. Wie zu erwarten, ist $h^{max}_{P^m}$ jeweils schneller fertig als h^m_P für gleiches m. Dies liegt zum einen daran, dass wie im Abschnitt 3.4 erwähnt, die Transformationszeit kaum ins Gewicht fällt, zum anderen, dass die Implementation der h^m - Heuristik im Fast-Downward relativ langsam ist.

Suchzeit	$h^{max}_{P^1}$	h^1_P
Minimum (463)	**0.01**	**0.01**
Maximum (463)	**4.26**	270.60
Geometrisches Mittel (463)	**0.02**	0.16

totale Zeit	$h^{max}_{P^1}$	h^1_P
Minimum (463)	**0.01**	**0.01**
Maximum (463)	**4.27**	270.60
Geometrisches Mittel (463)	**0.03**	0.17

Suchzeit	$h^{max}_{P^2}$	h^2_P
Minimum (251)	**0.01**	**0.01**
Maximum (251)	**8.17**	1267.06
Geometrisches Mittel (251)	**0.06**	1.77

totale Zeit	$h^{max}_{P^2}$	h^2_P
Minimum (251)	**0.01**	**0.01**
Maximum (251)	**8.21**	1267.07
Geometrisches Mittel (251)	**0.07**	1.79

Suchzeit	$h^{max}_{P^3}$	h^3_P
Minimum (164)	**0.01**	**0.01**
Maximum (164)	**14.51**	1029.06
Geometrisches Mittel (164)	**0.31**	4.23

totale Zeit	$h^{max}_{P^3}$	h^3_P
Minimum (164)	**0.01**	**0.01**
Maximum (164)	**16.75**	1029.07
Geometrisches Mittel (164)	**0.37**	4.25

Da die h^m- Heuristik im Fast-Downward Planungssystem nicht optimal implementiert ist, benötigt h^1_P mehr Zeit als h^{max}_P. Dies sieht man auch in der nebenstehenden Graphik. In dieser wird h^{max}_P und h^1_P miteinander verglichen. Man sieht, dass h^{max}_P besser abschneidet als h^1_P. Unten links in der Graphik scheint h^{max}_P über beinahe 2 Grössenordnungen von h^1_P zu sein. Dies sind die Werte, für die h^{max}_P eine Suchzeit von 0.00 angibt. Da die Zeitdaten auf 2 Nachkommastellen gerundet werden, liegen alle Suchzeiten < 0.005 Sekunden bei 10^{-2}. Die Suchzeiten für diese Probleme sind für h^m_P zu grossen Teilen bereits ≥ 0.005 Sekunden.

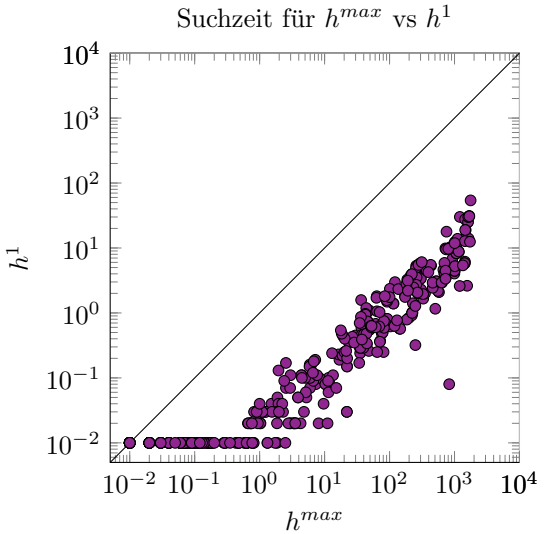

Abbildung 3.6: Vergleich der Suchzeiten für h^{max}_P und h^1_P

3.6 Speicherbedarf

Während die Heuristiken h^m und $hmax$ ihre Heuristikwert sukzessive berechnen, speichert die P^m - Kompilierung das ganze transformierte Problem. Daher ist die Transformation viel speicheraufwändiger als die Suche. Wir erwarten daher, dass $h^{max}_{P^m}$ für gleiches m mehr Speicher als h^m_P benötigt. Bei der Transformation wächst die Anzahl der Variablen und Aktionen in P^m - kompilierten Problemen sehr stark mit wachsendem m. Es wird erwartet, dass $h^{max}_{P^m}$ mit wachsendem m mehr Speicher benötigt.

Speicherbedarf	h^2_P	$h^{max}_{P^2}$
airport (7)	**42784**	510484
blocks (10)	**52948**	132732
depot (2)	**11036**	73188
driverlog (2)	**10508**	15284
freecell (1)	**5720**	118192
gripper (3)	**16224**	21388
logistics00 (6)	**32060**	38372
miconic (30)	**161012**	174296
movie (30)	**155640**	164224
mprime (4)	**23256**	344516
mystery (8)	**49764**	1437100
nomystery-opt11-strips (6)	**34132**	182848
openstacks-opt08-strips (5)	**27652**	57492
openstacks-opt11-strips (1)	**5712**	19120
openstacks-strips (5)	**27392**	62480
parcprinter-08-strips (8)	**43864**	93296
parcprinter-opt11-strips (4)	**22584**	62076
pathways-noneg (3)	**16352**	28780
pegsol-08-strips (9)	**50520**	431700
pegsol-opt11-strips (1)	**5716**	57396
pipesworld-notankage (3)	**17020**	269212
pipesworld-tankage (2)	**10904**	74728
psr-small (40)	**211116**	356516
rovers (4)	**20752**	23272
satellite (3)	**15832**	18936
scanalyzer-08-strips (3)	**16356**	63552
scanalyzer-opt11-strips (1)	**5452**	21184
sokoban-opt08-strips (4)	**22468**	136040
sokoban-opt11-strips (1)	**5716**	52668
storage (7)	**37244**	112420
tpp (5)	**26500**	30328
transport-opt08-strips (6)	**32832**	95052
transport-opt11-strips (1)	**5708**	31456
trucks-strips (2)	**10780**	24796
visitall-opt11-strips (8)	**42036**	47460
woodworking-opt08-strips (4)	**22052**	110456
zenotravel (5)	**26592**	53588
Durchschnitt (244)	**5427.20**	22732.08
Summe (244)	**1324236**	5546628

Speicherbedarf	h^3_P	$h^{max}_{P^3}$
airport (2)	**18164**	3241524
blocks (8)	**47444**	3080136
depot (1)	**6244**	631556
driverlog (2)	**11432**	197196
gripper (2)	**10904**	90780
logistics00 (6)	**34300**	328296
miconic (25)	**131812**	341580
movie (30)	**155640**	247612
mprime (1)	**6640**	1256916
mystery (2)	**13016**	1043952
nomystery-opt11-strips (4)	**26872**	2629524
openstacks-opt08-strips (1)	**5452**	70800
parcprinter-08-strips (6)	**52512**	2630952
parcprinter-opt11-strips (2)	**22520**	1664000
pathways-noneg (2)	**14732**	421852
pegsol-08-strips (2)	**14352**	1803848
psr-small (30)	**165936**	2651440
rovers (4)	**21680**	101788
satellite (2)	**10508**	27952
scanalyzer-08-strips (3)	**16752**	1078240
scanalyzer-opt11-strips (1)	**5584**	359416
storage (5)	**28316**	1639072
tpp (4)	**21280**	60832
transport-opt08-strips (4)	**22204**	995216
trucks-strips (1)	**5852**	277364
visitall-opt11-strips (7)	**38560**	151048
woodworking-opt08-strips (1)	**6904**	1332568
zenotravel (4)	**22212**	562752
Durchschnitt (162)	**5789.04**	178507.48
Summe (162)	**937824**	28918212

Speicherbedarf	h^1_P	$h^{max}_{P^1}$
airport (17)	**121712**	125508
⋮	⋮	⋮
movie (30)	**155640**	**155640**
⋮	⋮	⋮
zenotravel (7)	**60224**	60796
Durchschnitt (460)	**13436.43**	13725.23
Summe (460)	**6180756**	6313608

In den obigen Tabellen sieht man den Speicherbedarf für verschiedene m. Für $m = 1$ haben wir ähnliche Werte, wobei h^1_P leicht im Vorteil ist. Da sich hier die Werte nur sehr wenig unterscheiden, wurde die Tabelle gekürzt abgebildet. Die Domäne `movie` ist jedoch hervorgehoben. In dieser Domäne sind, wie schon in Abschnitt 3.3.2 erwähnt, alle Variablen bereits die Repräsentierungen von Atomen. Somit wird durch die P^1 - Kompilierung, wo lediglich aus der SAS$^+$ - Repräsentation via der *STRIPS* - Repräsentation wieder eine SAS$^+$ - Repräsentation gemacht wird, das Problem nicht verändert.
Für $m = 2$ und $m = 3$ benötigen die P^m - kompilierten Probleme deutlich mehr Speicher. Für $m = 2$ benötigen wir über die gelösten Instanzen gut vier mal, für $m = 3$ gut 30 mal so viel Speicher für $h^{max}_{P^m}$ als für h^m_P.

3.7 Ursachen für ungelöste Instanzen

Wir erwarten, dass für h^{max}_{Pm} mit wachsendem m tendenziell mehr Instanzen aufgrund von Speichermangel anstelle von Zeitmangel ungelöst bleiben, da der Speicher gefüllt wird, bevor die Suche beginnt. Das Transformieren ist speicheraufwändig, das Suchen zeitaufwändig.

Anzahl Fehler	h^{max}_{P1}	h^{1}_{P}	h^{max}_{P2}	h^{2}_{P}	h^{max}_{P3}	h^{3}_{P}
Zeitmangel (1456)	480	993	602	1198	**82**	1039
Speichermangel (1456)	350	**0**	409	3	1073	245
Kein Plan gefunden (1456)	4	3	7	10	**2**	4
Summe (1456)	**834**	996	1018	1211	1237	1288

3.7.1 Zeit- und Speichermangel

Wir stellen fest, dass für gleiches m h^{max}_{Pm} und h^{m}_{P} ähnlich viele Experimente mit einem Fehler terminieren. Dies geschieht fast ausschliesslich, da die 2048 MB Speicher oder die 1800 Sekunden Rechenzeit nicht zur Lösung des Problems ausreichen.

Für beide Heuristiken zeichnet sich ab, dass mit grösserem m der Anteil an Speicherfehlern zunimmt. Für h^{max}_{Pm} nehmen die Zeitfehler mit grösserem m stark ab. Dies hat damit zu tun, dass h^{max}_{Pm} viel Speicher und Zeit für die Transformation benötigt. Ist diese jedoch beendet, läuft die Suche schnell und ohne viel Speicherbedarf ab. Dies ist auch in den Nachfolgenden Tabellen ersichtlich.

Zeitmangel	h^{max}_{P1}	h^{1}_{P}	h^{max}_{P2}	h^{2}_{P}	h^{max}_{P3}	h^{3}_{P}
airport (50)	28	33	4	41	**0**	13
barman-opt11-strips (20)	8	20	20	20	**0**	20
blocks (35)	**0**	18	19	25	**0**	27
depot (22)	16	20	8	20	**0**	15
driverlog (20)	6	13	9	18	**4**	14
elevators-opt08-strips (30)	**0**	24	21	30	**0**	30
elevators-opt11-strips (20)	**0**	16	13	20	**0**	20
floortile-opt11-strips (20)	**0**	18	18	20	2	20
freecell (80)	65	73	7	79	**0**	80
grid (5)	3	4	1	5	**0**	2
gripper (20)	**0**	14	15	17	5	18
logistics00 (28)	**0**	18	18	22	6	22
logistics98 (35)	25	33	15	34	**0**	12
miconic (150)	**0**	106	105	120	40	125
movie (30)	**0**	**0**	**0**	**0**	**0**	**0**
mprime (35)	11	19	1	31	**0**	22
mystery (30)	9	14	**0**	12	**0**	15
nomystery-opt11-strips (20)	11	13	12	14	**0**	16
openstacks-opt08-strips (30)	4	18	20	25	**2**	29
openstacks-opt11-strips (20)	**0**	13	15	19	1	20
openstacks-strips (30)	23	24	11	25	**0**	26
parcprinter-08-strips (30)	4	18	18	22	**0**	24
parcprinter-opt11-strips (20)	1	12	12	16	**0**	18
parking-opt11-strips (20)	20	20	**0**	20	**0**	10
pathways-noneg (30)	**0**	26	26	27	**0**	17
pegsol-08-strips (30)	**0**	4	4	21	**0**	24
pegsol-opt11-strips (20)	**0**	4	4	19	**0**	19
pipesworld-notankage (50)	33	39	5	47	**0**	18
pipesworld-tankage (50)	39	44	5	48	**0**	30
psr-small (50)	**0**	4	2	10	3	20
rovers (40)	20	35	22	36	**5**	19
satellite (36)	19	32	18	33	**3**	21
scanalyzer-08-strips (30)	21	24	9	27	**3**	27
scanalyzer-opt11-strips (20)	14	17	7	19	**2**	19
sokoban-opt08-strips (30)	2	12	14	26	**0**	27
sokoban-opt11-strips (20)	**0**	5	9	19	**0**	19
storage (30)	15	18	5	23	**0**	14
tidybot-opt11-strips (20)	7	17	**0**	19	**0**	8
tpp (30)	18	25	10	25	**0**	12
transport-opt08-strips (30)	17	20	10	24	**0**	25
transport-opt11-strips (20)	12	15	15	19	**0**	20
trucks-strips (30)	12	26	17	28	**0**	27
visitall-opt11-strips (20)	**0**	11	11	12	5	13
woodworking-opt08-strips (30)	7	23	22	26	**0**	29
woodworking-opt11-strips (20)	3	18	19	20	**0**	20
zenotravel (20)	7	13	6	15	**1**	13
Summe (1456)	480	993	602	1198	**82**	1039

Speichermangel	$h^{max}_{P^1}$	h^1_P	$h^{max}_{P^2}$	h^2_P	$h^{max}_{P^3}$	h^3_P
airport (50)	**0**	**0**	35	2	27	35
barman-opt11-strips (20)	4	**0**	**0**	**0**	20	**0**
blocks (35)	17	**0**	**0**	**0**	26	**0**
depot (22)	**0**	**0**	12	**0**	20	6
driverlog (20)	5	**0**	4	**0**	12	4
elevators-opt08-strips (30)	14	**0**	**0**	**0**	29	**0**
elevators-opt11-strips (20)	7	**0**	**0**	**0**	20	**0**
floortile-opt11-strips (20)	14	**0**	**0**	**0**	18	**0**
freecell (80)	**0**	**0**	66	**0**	80	**0**
grid (5)	**0**	**0**	3	**0**	5	3
gripper (20)	13	**0**	**0**	**0**	12	**0**
logistics00 (28)	18	**0**	**0**	**0**	12	**0**
logistics98 (35)	8	**0**	18	1	20	23
miconic (150)	100	**0**	**0**	**0**	75	**0**
movie (30)	**0**	**0**	**0**	**0**	**0**	**0**
mprime (35)	**0**	**0**	19	**0**	31	12
mystery (30)	**0**	**0**	12	**0**	23	9
nomystery-opt11-strips (20)	1	**0**	**0**	**0**	14	**0**
openstacks-opt08-strips (30)	5	**0**	**0**	**0**	23	**0**
openstacks-opt11-strips (20)	4	**0**	**0**	**0**	18	**0**
openstacks-strips (30)	**0**	**0**	14	**0**	25	4
parcprinter-08-strips (30)	11	**0**	**0**	**0**	23	**0**
parcprinter-opt11-strips (20)	8	**0**	**0**	**0**	17	**0**
parking-opt11-strips (20)	**0**	**0**	20	**0**	20	10
pathways-noneg (30)	26	**0**	**0**	**0**	27	11
pegsol-08-strips (30)	3	**0**	**0**	**0**	28	**0**
pegsol-opt11-strips (20)	3	**0**	**0**	**0**	20	**0**
pipesworld-notankage (50)	**0**	**0**	37	**0**	40	32
pipesworld-tankage (50)	**0**	**0**	39	**0**	46	20
psr-small (50)	1	**0**	**0**	**0**	8	**0**
rovers (40)	14	**0**	13	**0**	22	17
satellite (36)	11	**0**	14	**0**	23	13
scanalyzer-08-strips (30)	**0**	**0**	15	**0**	24	**0**
scanalyzer-opt11-strips (20)	**0**	**0**	10	**0**	17	**0**
sokoban-opt08-strips (30)	**0**	**0**	3	**0**	29	3
sokoban-opt11-strips (20)	**0**	**0**	1	**0**	20	1
storage (30)	**0**	**0**	13	**0**	21	10
tidybot-opt11-strips (20)	**0**	**0**	20	**0**	20	12
tpp (30)	6	**0**	15	**0**	20	14
transport-opt08-strips (30)	2	**0**	10	**0**	24	1
transport-opt11-strips (20)	2	**0**	**0**	**0**	19	**0**
trucks-strips (30)	10	**0**	8	**0**	27	2
visitall-opt11-strips (20)	11	**0**	**0**	**0**	7	**0**
woodworking-opt08-strips (30)	14	**0**	2	**0**	28	**0**
woodworking-opt11-strips (20)	13	**0**	**0**	**0**	20	**0**
zenotravel (20)	5	**0**	6	**0**	13	3
Summe (1456)	350	**0**	409	3	1073	245

3.7.2 Kein Plan gefunden

Dass der Planer ohne Lösung terminiert, geschieht nur in der Domäne `mystery`. Der verwendete Suchalgorithmus, A*, ist mit h^m - Heuristiken vollständig, findet also immer einen Plan, wenn einer existiert. Dass der Planer dennoch für bis zu zehn Instanzen keine Pläne gefunden hat, liegt daran, dass in der Domäne `mystery` unlösbare Probleme existieren.

Kein Plan gefunden	$h^{max}_{P^1}$	h^1_P	$h^{max}_{P^2}$	h^2_P	$h^{max}_{P^3}$	h^3_P
airport (50)	**0**	**0**	**0**	**0**	**0**	**0**
⋮	⋮	⋮	⋮	⋮	⋮	⋮
mystery (30)	4	3	7	10	**2**	4
⋮	⋮	⋮	⋮	⋮	⋮	⋮
zenotravel (20)	**0**	**0**	**0**	**0**	**0**	**0**
Summe (1456)	4	3	7	10	**2**	4

4

Schlussbemerkungen

Ziel dieser Arbeit war es die von Haslum [Has09] vorgeschlagene P^m - Kompilierung zu implementieren und die $h^{max}_{P^m}$ Heuristik auf einer kompilierten Planungsaufgabe P^m gegen die h^m_P - Heuristik auf der originalen Planungsaufgabe P empirisch zu testen. Im Folgenden fassen wir die Resultate zusammen.

Mit $h^{max}_{P^m}$ konnten für gleiches m jeweils mehr Probleme gelöst werden, als mit h^m_P. Die Anzahl gelöster Probleminstanzen nimmt zudem mit wachsendem m ab. Die Anzahl der Variablen und der Aktionen in einem P^m - komplierten Problem nehmen mit zunehmendem m stark zu. Die benötigte Zeit für die P^m - Kompilierung nimmt mit wachsendem m stark zu, ist jedoch bis $m = 3$ ein kleiner Bestandteil der gesamten Zeit, welche für die Lösung einer Planungsaufgabe benötigt wird. Der Anteil dieser Transformationszeit an der gesamten Zeit nimmt mit wachsendem m stark zu. Die Suchzeit für erfolgreich gelöste Probleme nimmt mit wachsendem m zu. Zudem benötigt h^m_P merkbar mehr Zeit für die Suche als $h^{max}_{P^m}$ bei gleichem m. Mit der h^{max} - Implementierung ist die Suche deutlich schneller als mit h^1 der h^m - Implementierung für dasselbe Problem. Der Speicherbedarf von $h^{max}_{P^m}$ ist höher als derjenige von h^m_P bei gleichem m. Bei den Tests, scheiterte $h^{max}_{P^m}$ öfters an der Speicherlimite, während h^m_P primär mit der Zeitlimite Probleme hatte.

Die in dieser Arbeit vorgestellten Experimente sind Implementierungsspezifisch. Es wurde die bestehende Implementierung der h^m - Heuristik aus dem Fast-Downward Planungssystem verwendet. Diese weist darauf hin, dass sie langsam sei. Die Experimente mit einer schnelleren Implementierung von h^m zu wiederholen könnte daher qualitativ andere Resultate liefern.

Für diese Arbeit wurde eine erste komplexe Problemtransformation zum Fast-Downward Planungssystem hinzugefügt. In Zukunft werden wohl noch weitere Transformationsimplementierungen folgen, welche selbst zu neuen interessanten Einsichten führen werden.

Literaturverzeichnis

[BG01] Blai Bonet and Hector Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1-2):5–33, 2001.

[BN95] Christer Bäckström and Bernhard Nebel. Complexity Results for SAS+ Planning. *Computational Intelligence*, 11:625–656, 1995.

[FN71] Richard Fikes and Nils J. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2(3-4):189–208, 1971.

[Has09] Patrik Haslum. $h^m(P) = h^1(P^m)$: Alternative Characterisations of the Generalisation From $h^{\max}$ To h^m. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling*, pages 354–357. AAAI Press, 2009.

[HBG05] Patrik Haslum, Blai Bonet, and Hector Geffner. New Admissible Heuristics for Domain-Independent Planning. In *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference*, pages 1163–1168. AAAI Press / The MIT Press, 2005.

[Hel06] Malte Helmert. The Fast Downward Planning System. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[HG00] Patrik Haslum and Hector Geffner. Admissible Heuristics for Optimal Planning. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems*, pages 140–149. AAAI Press, 2000.

[HNR68] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[KHH14] Emil Ragip Keyder, Jörg Hoffmann, and Patrik Haslum. Improving Delete Relaxation Heuristics Through Explicitly Represented Conjunctions. *Journal of Artificial Intelligence Research*, 50:487–533, 2014.

Declaration on Scientific Integrity
Erklärung zur wissenschaftlichen Redlichkeit

includes Declaration on Plagiarism and Fraud
beinhaltet Erklärung zu Plagiat und Betrug

Author — Autor
Raphael Imahorn

Matriculation number — Matrikelnummer
2010-057-826

Title of work — Titel der Arbeit
P^m-Kompilierung von Planungsaufgaben im Fast-Downward Planungssystem als alternative Berechnung der h^m-Heuristik

Type of work — Typ der Arbeit
Bachelorarbeit

Declaration — Erklärung
I hereby declare that this submission is my own work and that I have fully acknowledged the assistance received in completing this work and that it contains no material that has not been formally acknowledged. I have mentioned all source materials used and have cited these in accordance with recognised scientific rules.

Hiermit erkläre ich, dass mir bei der Abfassung dieser Arbeit nur die darin angegebene Hilfe zuteil wurde und dass ich sie nur mit den in der Arbeit angegebenen Hilfsmitteln verfasst habe. Ich habe sämtliche verwendeten Quellen erwähnt und gemäss anerkannten wissenschaftlichen Regeln zitiert.

Basel, 28. Januar 2016

Signature — Unterschrift