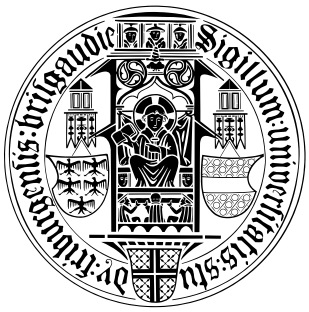
ALBERT-LUDWIGS-UNIVERSITÄT
FREIBURG
INSTITUT FÜR INFORMATIK

Arbeitsgruppe
Grundlagen der Künstlichen Intelligenz

Prof. Dr. Bernhard Nebel



Fluent Merging für klassische Planungsprobleme

Bachelorarbeit

Jendrik Seipp

Betreuer: Dr. Malte Helmert

September 2009

Erklärung

Hiermit erkläre ich, dass ich diese Abschlussarbeit 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 Abschlussarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Freiburg im Breisgau, September 2009 Jendrik Seipp

Inhaltsverzeichnis

1 Einleitung

Das klassische Planen beschäftigt sich mit dem Finden von Plänen in einer voll beobachtbaren Welt. Sowohl der Anfangszustand, die Operatoren, als auch die Beschreibung der Zielzustände sind bekannt. Zustände werden in Form von Variablen repräsentiert, die Werte aus einem bestimmten Wertebereich annehmen können. Jeder Zustand weist jeder Variablen einen Wert aus deren Wertebereich zu.

In der ursprünglichen Problembeschreibung kommen meistens nur boolesche Variablen vor. Diese können entweder wahr oder falsch sein. Es hat sich herausgestellt, dass es sinnvoll ist, solche boolesche Variablen zu Variablen mit größerem Wertebereich zusammenzufassen. So können meist schneller bessere Pläne gefunden werden. Die Kombination mehrerer Variablen bezeichnet man auch als *Fluent Merging*. Gegenwärtig ist die erfolgreichste Strategie beim Zusammenfassen von Variablen das Verschmelzen von booleschen Variablen, die nicht gleichzeitig wahr sein können.

Das Ziel dieser Arbeit ist es, zu untersuchen, ob es aufbauend auf dieser Kombinationsmethode sinnvolle Methoden gibt, mehrwertige Variablen zusammenzufassen.

2 Gliederung

Nach einigen allgemeinen Definitionen wird zunächst das Fluent Merging erklärt und die bisherige Forschung zu dem Thema kurz zusammengefasst. Anschließend wird eine Einführung in das *Fast Downward Planungssystem* [2] gegeben und vorgestellt, wie das Fluent Merging im Rahmen dieser Arbeit dort integriert wurde.

Als nächstes werden die einzelnen Schritte der Variablenkombination dargestellt und es werden die verschiedenen Methoden gezeigt, die für die Auswahl der zu kombinierenden Variablen zur Verfügung stehen.

Im letzten Teil der Arbeit werden die Kombinationsmethoden anhand einiger Planungsprobleme getestet und evaluiert.

3 Allgemeine Definitionen

3.1 SAS$^+$

Eine *SAS$^+$-Planungsaufgabe* [1] ist ein Tupel $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$. Dabei steht $\mathcal{V} = \{v_1, \ldots, v_m\}$ für die Menge der Zustandsvariablen und $\mathcal{O}$ für die Menge der Operatoren. s_0 ist der Anfangszustand und $s_\star$ ist eine *partielle Belegung*,

die für eine Variablenmenge $\mathcal{V}' \subseteq \mathcal{V}$ Werte festlegt, die nach Ausführung des Plans gelten müssen.

- Jede Variable $v \in \mathcal{V}$ hat einen zugeordneten Wertebereich $\mathcal{D}_v$.
- Ein Operator $o \in \mathcal{O}$ hat die Form $\langle \text{pre}, \text{eff} \rangle$, wobei *pre* und *eff* partielle Belegungen sind. Man nennt *pre* die *Vorbedingung* und *eff* den *Effekt* des Operators.

3.2 Domänentransitionsgraph

Sei $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ eine SAS$^+$-Planungsaufgabe. Der *Domänentransitionsgraph (DTG)* einer Variablen $v \in \mathcal{V}$, DTG_v, ist ein gerichteter Graph mit der Knotenmenge $\mathcal{D}_v$ und den Kanten E_v. Der Graph enthält eine Kante von d nach d' für $d, d' \in \mathcal{D}_v$, genau dann wenn eine der folgenden beiden Bedingungen erfüllt ist:

- Es existiert ein Operator $o = \langle \text{pre}, \text{eff} \rangle \in \mathcal{O}$ mit $\text{eff}(v) = d'$ und $\text{pre}(v) = d$ oder $\text{pre}(v)$ undefiniert.
- $d = d'$ und es existiert ein Operator $o = \langle \text{pre}, \text{eff} \rangle \in \mathcal{O}$ mit $\text{eff}(v)$ undefiniert und $\text{pre}(v) = d$ oder $\text{pre}(v)$ undefiniert.

Wir legen fest, dass die Kante mit dem Operator o beschriftet wird.

Diese Festlegung weicht von der generellen Definition eines DTG ab. Normalerweise wird der zweite Fall ausgeschlossen und der DTG enthält keine Knoten mit Kanten auf sich selbst. Man braucht aber den zweiten Fall für die nachfolgende Fluent Merging Definition.

Die Kanten im DTG einer Variablen stehen also für die möglichen Übergänge zwischen den Variablenwerten. Die Intuition ist, dass der DTG eine Kante von d nach d' beschriftet mit Operator o genau dann enthält, wenn es einen Zustand gibt, in dem d gilt, o angewandt wird und dann d' gilt. Nach der Definition kann es vorkommen, dass mehrere Kanten von d nach d' existieren.

3.3 Kausalgraph

Sei $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ eine SAS$^+$-Planungsaufgabe. Der *Kausalgraph* (CG für Causal Graph) von Π, $\text{CG}(\Pi)$, ist ein gerichteter Graph mit der Knotenmenge $\mathcal{V}$. Er enthält eine Kante (v, v') für $v, v' \in \mathcal{V}, v \neq v'$, genau dann wenn ein Operator $o = \langle \text{pre}, \text{eff} \rangle \in \mathcal{O}$ existiert, bei dem sowohl $\text{pre}(v)$ oder $\text{eff}(v)$ als auch $\text{eff}(v')$ definiert ist. Wir definieren die *Stärke* einer Kante durch die Anzahl der Operatoren o, die diese Forderung erfüllen.

Am Kausalgraphen eines Planungsproblems lassen sich die Abhängigkeiten zwischen den Variablen ablesen.

4 Fluent Merging Definition

Als *Fluent Merging* bezeichnet man die Kombination von mehreren Zustandsvariablen zu einer einzigen Variablen. Der Begriff wurde von van den Briel et al. geprägt [7].

Die Kombination wird anhand der Domänentransitionsgraphen der beiden zu kombinierenden Variablen definiert [7]. Diese kompakte Definition soll die generelle Idee des Fluent Merging verdeutlichen. In Abschnitt 7 folgt dann der eigentliche Algorithmus, der zwei Variablen kombiniert.

Definition (Kombination zweier Variablen). Seien a und b zwei Variablen einer SAS$^+$-Planungsaufgabe. Dann entspricht die Kombination ihrer Domänentransitionsgraphen DTG_a und DTG_b dem DTG der kombinierten Variable. Dieser wird mit $\mathrm{DTG}_{a\mathrm{UND}b}$ bezeichnet und wie folgt konstruiert:

Es sei $\mathrm{DTG}_a = (V_a, E_a)$ und $\mathrm{DTG}_b = (V_b, E_b)$ dann gilt:

- $\mathrm{DTG}_{a\mathrm{UND}b} = (V_{a\mathrm{UND}b}, E_{a\mathrm{UND}b})$ mit
- $V_{a\mathrm{UND}b} = V_a \times V_b$
- $E_{a\mathrm{UND}b} = \{((s_1, s_2), (t_1, t_2)) | (s_1, t_1) \in E_a, (s_2, t_2) \in E_b$ und beide Kanten sind mit o beschriftet für einen Operator $o\}$.

Ein Beispiel zu dieser Definition befindet sich in Abschnitt 8.8. Der Domänentransitionsgraph $\mathrm{DTG}_{a\mathrm{UND}b}$ in Abb. 6 ist die Kombination von DTG_a und DTG_b aus den Abb. 4 und 5.

5 Bisherige Forschung

Ausgangspunkt dieser Arbeit ist ein 2007 veröffentlichter Beitrag von van den Briel et al. [7]. Darin beschreiben die Autoren, dass es für das Planen nützlich sein könne, Variablen zu verschmelzen, zwischen denen „starke Abhängigkeiten bestehen“. Die Autoren geben zu, dass die Idee der Variablenkombination nicht neu ist und führen als Beispiel für einen Planer, der bereits Variablen kombiniert, das Fast Downward Planungssystem an. Hier werden allerdings nur solche boolesche Variablen kombiniert, die zueinander *mutex* sind, d.h. nicht gleichzeitig wahr sein können.

Sie behaupten, dass dieser Ansatz zu „zurückhaltend“ sei und schlagen vor, dass man auch nach anderen Kombinations-Methoden suchen sollte. Eine vorgeschlagene Methode ist das Finden und Verschmelzen von Zweier-Zyklen im Kausalgraphen.

Van den Briel et al. kommen zu dem Schluss, dass ihre getesteten Kombinations-Methoden bereits gute Ergebnisse lieferten, jedoch noch weitere Forschung auf diesem Gebiet notwendig sei.

6 Fluent Merging im Fast Downward Planungssystem

Aufbauend auf der theoretischen Kombination soll nun beschrieben werden, wie das Fluent Merging in das Fast Downward Planungssystem [2] integriert wurde. Zunächst wird jedoch kurz die normale Funktionsweise des Fast Downward Planungssystems erläutert.

6.1 Das Fast Downward Planungssystem

Als Eingabe liegt dem Planungssystem ein Problem in PDDL-Syntax [6] vor. Dafür wird in drei Schritten ein Plan erzeugt, falls ein solcher existiert [3].

- Als erstes wird das PDDL-Problem normalisiert, mit Invarianten versehen, instanziiert und in ein SAS^+-Planungsproblem übersetzt. Dieser Schritt wird als *Übersetzung* bezeichnet.
- Im zweiten Schritt, der *Wissenskompilation*, wird die Ausgabe des ersten Schritts so aufbereitet, dass wichtige Informationen über das Problem direkt für den dritten Schritt, die Suche, verfügbar sind.
- Im dritten und letzten Schritt wird dann die tatsächliche Plansuche durchgeführt. Dafür stehen mehrere Algorithmen und Optionen zur Verfügung.

6.2 Fluent Merging Integration

Für das Fluent Merging wird direkt nach der Übersetzung ein neuer Schritt eingefügt. Der *Fluent Merging Algorithmus*, der unten vorgestellt wird, bekommt das SAS^+-Planungsproblem als Eingabe. Er liest die Repräsentation ein, kombiniert Variablen nach einer auswählbaren Methode (Abschnitt 8) und gibt wiederum ein SAS^+-Planungsproblem aus. Diese Ausgabe kann dann zur Wissenskompilation weitergegeben werden. Es findet also nur eine

Umformulierung der Übersetzungsausgabe statt, die sich zwischen die ersten beiden Schritte einfügen lässt. Der Code der ursprünglichen Schritte muss nicht geändert werden.

Wir nennen das SAS$^+$-Planungsproblem, das der Fluent Merging Algorithmus zurückgibt, auch die *kombinierte Planungsaufgabe*.

6.3 Fluent Merging Algorithmus

Der Fluent Merging Algorithmus besteht aus drei Teilen.

- Bewertung
- Selektion
- Kombination

Eine *Bewertungsfunktion* bewertet die *Kombinationsgüte* für jedes Variablenpaar der Planungsaufgabe. Anschließend wählt der *Selektions-Algorithmus* die zu kombinierenden Variablengruppen auf der Basis dieser Information aus. Die eigentliche Kombination der Variablen wird dann vom *Kombinations-Algorithmus* übernommen.

7 Der Kombinations-Algorithmus

Es soll nun zunächst der *Kombinations-Algorithmus* erläutert werden. Wir werden sehen, wie man zwei Variablen kombiniert und den Algorithmus dann auf Variablengruppen erweitern. Die vorhergehende Variablenauswahl lässt sich für die Betrachtung leicht ausklammern, indem wir annehmen, dass die Auswahl der zu kombinierenden Variablen bereits getroffen ist.

7.1 Kombination zweier Variablen

Die Kombination vollzieht sich nacheinander in mehreren Abschnitten. Für alle Teile nehmen wir an, dass $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ eine SAS$^+$-Planungsaufgabe ist. Weiterhin seien $a, b \in \mathcal{V}$ mit den Wertebereichen $\mathcal{D}_a$ und $\mathcal{D}_b$ die zu kombinierenden Variablen.

7.1.1 Neue Variable

Die kombinierte Variable bekommt den neuen Namen $a\text{UND}b$ und den Wertebereich $\mathcal{D}_a \times \mathcal{D}_b$. Wir setzen $\mathcal{D}_a \times \mathcal{D}_b = \{a_i\text{UND}b_i | a_i \in \mathcal{D}_a, b_i \in \mathcal{D}_b\}$ und erweitern $\mathcal{V}$ um $a\text{UND}b$.

7.1.2 Anfangszustand

Da in jedem wohldefinierten SAS$^+$-Planungsproblem der Anfangswert für alle Variablen definiert ist, können wir den Anfangswert der kombinierten Variable leicht bestimmen. Sei $s_0(a) = a_0$ und $s_0(b) = b_0$, dann ist $s_0(a\text{UND}b) = a_0\text{UND}b_0$.

7.1.3 Operatoren

Für jeden Operator $o = \langle \text{pre}, \text{eff} \rangle \in \mathcal{O}$ wird überprüft, ob er aufgrund der Variablenkombination geändert oder gelöscht werden muss und ob neue Operatoren zu $\mathcal{O}$ hinzukommen. Die Überprüfung wird durch Algorithmus 1 beschrieben.

Eingabe:
Operator $o = \langle \text{pre}, \text{eff} \rangle \in \mathcal{O}$
Zu kombinierende Variablen $a, b \in \mathcal{V}$

if o erwähnt weder a noch b **then**
{Nichts zu tun}
return
else if $\text{eff}(a) = a_{\text{eff}}$ und $\text{eff}(b) = b_{\text{eff}}$ und $\text{pre}(a)$, $\text{pre}(b)$ undefiniert **then**
{Die Operatoren setzen a und b und fragen weder a noch b ab}
{Füge den offensichtlichen Effekt ein}
$\text{eff}(a\text{UND}b) = a_{\text{eff}}\text{UND}b_{\text{eff}}$
else
{Füge neue Operatoren hinzu}
$\text{poss}_a = \{\text{pre}(a)\}$ falls definiert, sonst $\mathcal{D}_a$
$\text{poss}_b = \{\text{pre}(b)\}$ falls definiert, sonst $\mathcal{D}_b$
for each $a_{\text{pre}} \in \text{poss}_a$ **do**
for each $b_{\text{pre}} \in \text{poss}_b$ **do**
$a_{\text{eff}} = \text{eff}(a)$ falls definiert, sonst a_{pre}
$b_{\text{eff}} = \text{eff}(b)$ falls definiert, sonst b_{pre}
Operator $o_{\text{neu}} = \langle \text{pre}_{\text{neu}}, \text{eff}_{\text{neu}} \rangle \leftarrow \langle \text{pre}, \text{eff} \rangle = o$
$\text{pre}_{\text{neu}}(a\text{UND}b) = a_{\text{pre}}\text{UND}b_{\text{pre}}$
$\text{eff}_{\text{neu}}(a\text{UND}b) = a_{\text{eff}}\text{UND}b_{\text{eff}}$
$\mathcal{O} \leftarrow \mathcal{O} \cup \{o_{\text{neu}}\}$
$\mathcal{O} \leftarrow \mathcal{O} \setminus \{o\}$
return

Algorithmus 1: Überprüfung eines Operators während der Variablenkombination

7.1.4 Zielbeschreibung

Wir unterscheiden wieder drei Fälle:

- $s_\star(a)$ und $s_\star(b)$ undefiniert:
  Wenn keine der beiden Variablen in der Zielbeschreibung vorkommt, ist nichts zu tun.

- $s_\star(a) = a_\star$ und $s_\star(b) = b_\star$:
  Wenn beide Variablen vorkommen, ist der Wert der kombinierten Variable eindeutig bestimmt. Wir setzen $s_\star(a\text{UND}b) = a_\star\text{UND}b_\star$.

- $s_\star(a) = a_\star$ und $s_\star(b)$ undefiniert oder $s_\star(a)$ undefiniert und $s_\star(b) = b_\star$:
  Eine der beiden Variablen kommt in der Zielbeschreibung vor. Nehme o.B.d.A. an, dass $s_\star(a) = a_\star$ und $s_\star(b)$ undefiniert ist. Der andere Fall wird analog behandelt.
  Das Ziel ist bereits erreicht, wenn $a = s_\star(a)$. Der Wert von b ist für das Erreichen des Ziels irrelevant. Um diesen Sachverhalt wiederzuspiegeln, fügen wir den Wert $a_\star$ zu $\mathcal{D}_{a\text{UND}b}$ hinzu und setzen $s_\star(a\text{UND}b) = a_\star$. Damit dieser Wert auch erreicht werden kann, fügen wir für alle Werte $b_i \in \mathcal{D}_b$ einen *Pseudooperator* o_i zu $\mathcal{O}$ hinzu, der ausschließlich für die Variable $a\text{UND}b$ die Wertänderung von $a_\star\text{UND}b_i$ zu $a_\star$ bewirkt:
  $o_i = \langle\{a\text{UND}b \mapsto a_\star\text{UND}b_i\}, \{a\text{UND}b \mapsto a_\star\}\rangle$

7.1.5 Abschluss der Kombination

Die Variablen a und b werden anschließend aus der Planungsaufgabe gelöscht. Der Algorithmus 2 zeigt die Prozedur, die eine Variable aus der Planungsaufgabe entfernt. Diese Methode wird zum Abschluss des Merge-Vorgangs für beide Variablen aufgerufen.

Eingabe: $v \in \mathcal{V}$
  $\mathcal{V} \leftarrow \mathcal{V} \setminus \{v\}$
  for each $o \in \mathcal{O}$ **do**
    $\text{pre}_o(v) \leftarrow$ undefiniert
    $\text{eff}_o(v) \leftarrow$ undefiniert
  $s_0(v) \leftarrow$ undefiniert
  $s_\star(v) \leftarrow$ undefiniert

Algorithmus 2: Löschen einer Variablen

7.1.6 Mutexe

Der Übersetzungsschritt des Fast Downward Planungssystems gibt nicht nur die SAS$^+$-Planungsaufgabe aus, sondern auch eine Datei mit Gruppen von Variablenwerten, die paarweise *mutex* sind, also nicht gleichzeitig von den jeweiligen Variablen angenommen werden können. Diese Information wird für die Variablenkombination genutzt, indem Variablenwerte $a_x\text{UND}b_y$ aus dem Wertebereich der neuen Variablen gelöscht werden, wenn a_x und b_y mutex sind.

Da nun möglicherweise in den Wertebereichen Werte „fehlen“, kann es in den obigen Schritten vorkommen, dass $a\text{UND}b$ einen Wert $a_x\text{UND}b_y \notin \mathcal{D}_{a\text{UND}b}$ annimmt. Deshalb werden nie solche Operatoren o erzeugt, bei denen $\text{pre}_o(a\text{UND}b)$ oder $\text{eff}_o(a\text{UND}b)$ undefiniert ist.

7.2 Beispiel

Die Kombination zweier Variablen soll an einem vereinfachten Beispiel aus der *Gripper* Domäne erläutert werden.

In der Beispielwelt gibt es einen Roboter mit einem Greifarm (*arm*), zwei Räume, *raum1* und *raum2*, und einen Ball *ball*. Am Anfang befinden sich der Roboter und der Ball in *raum1* und das Ziel soll sein, den Ball in *raum2* zu bringen.

Lässt man das Fast Downward Planungssystem den Übersetzungsschritt für dieses Problem durchführen, so ergibt sich das SAS$^+$-Planungsproblem $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ in Abb. 1.

Kombiniert werden sollen nun die Variablen a und c.

Die neue Variable

Die neue Variable bekommt den Namen $a\text{UND}c$ und den zugehörigen Wertebereich $\mathcal{D}_{a\text{UND}c}$.

- $\mathcal{D}_{a\text{UND}c} =$

$$\left\{ \begin{array}{ll} \textit{in(ball, raum1)UNDfrei(arm)}, & \textit{in(ball, raum1)UNDvoll(arm)}, \\ \textit{in(ball, raum2)UNDfrei(arm)}, & \textit{in(ball, raum2)UNDvoll(arm)}, \\ \textit{trage(ball, arm)UNDfrei(arm)}, & \textit{trage(ball, arm)UNDvoll(arm)} \end{array} \right\}$$

Wir sehen jedoch, dass der Wert *trage(ball, arm)UNDfrei(arm)* nie angenommen und deshalb aus dem Wertebereich entfernt werden kann. Der Kombinations-Algorithmus würde dies der Mutex-Datei entnehmen.

- $\mathcal{V} = \{a, b, c\}$
  - $\mathcal{D}_a = \{$in(ball, raum1), in(ball, raum2), trage(ball, arm)$\}$
  - $\mathcal{D}_b = \{$roboter-in(raum1), roboter-in(raum2)$\}$
  - $\mathcal{D}_c = \{$frei(arm), voll(arm)$\}$
- $\mathcal{O} = \{$laufe-raum1-raum2, laufe-raum2-raum1, greife-ball-in-raum1, greife-ball-in-raum2, lege-ball-in-raum1, lege-ball-in-raum2$\}$
  - laufe-raum1-raum2 = $\langle\{b \mapsto$ roboter-in(raum1)$\}, \{b \mapsto$ roboter-in(raum2)$\}\rangle$
  - greife-ball-in-raum1 = $\langle\{a \mapsto$ in(ball, raum1), $b \mapsto$ roboter-in(raum1), $c \mapsto$ frei(arm)$\}$, $\{a \mapsto$ trage(ball, arm), $c \mapsto$ voll(arm)$\}\rangle$
  - lege-ball-in-raum1 = $\langle\{a \mapsto$ trage(ball, arm), $b \mapsto$ roboter-in(raum1)$\}$, $\{a \mapsto$ in(ball, raum1), $c \mapsto$ frei(arm)$\}\rangle$
  - ... (für die 3 anderen Operatoren ersetze raum1 durch raum2)
- $s_0(a) =$ roboter-in(raum1), $s_0(b) =$ frei(arm), $s_0(c) =$ in(ball, raum1)
- $s_\star(a) =$ in(ball, raum2)

Abbildung 1: Einfaches Gripper Problem nach Übersetzung ohne Operatoren

Zwei andere Werte, die auch nicht angenommen werden können, sind *in(ball, raum1)UNDvoll(arm)* und *in(ball, raum2)UNDvoll(arm)*. Der Algorithmus erkennt sie aber nicht als Mutexe, da sie nicht in der Mutex-Datei stehen. Das hat jedoch keinen Einfluss auf den weiteren Algorithmus oder die späteren Schritte im Fast Downward Planungssystem.

Intuitiv steht die neue Variable für die Position des Balls: *raum1*, *raum2* oder *arm*. Man sieht, dass bereits die Variable a allein genau diese Information gespeichert hat. Dadurch, dass der Roboter nur einen Arm hat, ist die Variable c überflüssig. Die Information, ob der Arm gerade den Ball greift, ist bereits in a enthalten.

Anfangszustand

$s_0(a\text{UND}c) =$ *in(ball, raum1)UNDfrei(arm)*

Operatoren

Wir wollen die Veränderung von $\mathcal{O}$ nur für die drei Operatoren zeigen, die auch in Abb. 1 erwähnt werden. Der Kombinations-Algorithmus funktioniert bei den anderen drei Operatoren analog.

laufe-raum1-raum2 Weder a noch c kommen in der Vorbedingung oder im Effekt des Operators vor. Er muss also nicht verändert werden.

greife-ball-in-raum1 Die Variablen a und c kommen sowohl in der Vorbedingung als auch dem Effekt vor. Die Werte von $a\mathit{UND}c$ sind also klar definiert. Wir setzen für den Operator

- pre(aUNDc) = *in(ball, raum1)UNDfrei(arm)*
- eff(aUNDc) = *trage(ball, arm)UNDvoll(arm)*

lege-ball-in-raum1 Variable c kommt nicht in der Vorbedingung vor. Wir müssten also normalerweise für jede Kombination des Wertes von a, *trage(ball, arm)*, mit jedem Wert von c, *frei(arm)* und *voll(arm)*, einen neuen Operator einfügen. Oben haben wir jedoch gesehen, dass die Kombination *trage(ball, arm)UNDfrei(arm)* kein Wert ist, der auch angenommen werden kann. Da der Wert von $a\mathit{UND}c$ im Effekt wiederum eindeutig ist, wird also nur ein neuer Operator erzeugt:

- lege-ball-in-raum1-mit-ball =
  $\langle\{a$UND$c \mapsto$ trage(ball, arm)UNDvoll(arm), $b \mapsto$ roboter-in(raum1)$\}$,
  $\{a$UND$c \mapsto$ in(ball, raum1)UNDfrei(arm)$\}\rangle$

  Der Operator *lege-ball-in-raum1* kann nun gelöscht werden.

Zielbeschreibung

Hier tritt der Fall ein, dass genau eine der beiden zu kombinierenden Variablen in der Zielbeschreibung vorkommt. Wenn c den Wert *in(ball, raum2)* hat, ist das Problem gelöst, egal was der Wert von a ist. Wir erweitern also $\mathcal{D}_{a\text{UND}c}$ um *in(ball, raum2)*, setzen $s_\star(a\text{UND}c)$ = *in(ball, raum2)* und fügen für jeden Wert von a einen Pseudo Operator hinzu, der den Übergang von jedem Wert aus aUNDc zu *in(ball, raum2)* ermöglicht, wenn c bereits den gewünschten Wert hat.

- pseudo-op-mit-arm-frei =
  $\langle\{a\text{UND}c \mapsto \text{in(ball, raum2)UNDfrei(arm)}\},$
  $\{a\text{UND}c \mapsto \text{in(ball, raum2)}\}\rangle$

- pseudo-op-mit-arm-voll =
  $\langle\{a\text{UND}c \mapsto \text{in(ball, raum2)UNDvoll(arm)}\},$
  $\{a\text{UND}c \mapsto \text{in(ball, raum2)}\}\rangle$

Dabei ist zu beachten, dass die Vorbedingung von *pseudo-op-mit-arm-voll* nie erfüllt werden kann. Das kann der Algorithmus – wie oben bereits bemerkt – aber nicht herausfinden.

Am Ende der Kombination von a und c werden die beiden Variablen aus der Planungsaufgabe gelöscht.

7.3 Kombination einer Variablengruppe

Bei der Kombination einer Gruppe von Variablen werden immer wieder zwei Variablen der Gruppe kombiniert und anschließend das Resultat mit den übrigen Variablen vereint.

Der Algorithmus 3 verdeutlicht die Prozedur.

Eingabe: {SAS$^+$-Planungsaufgabe $\Pi, \mathcal{V}' \subseteq \mathcal{V}$}
  $v_1 \leftarrow \mathcal{V}'$.pop() {Nehme eine Variable aus $\mathcal{V}'$ und lösche sie aus $\mathcal{V}'$}
  while $\mathcal{V}'$ nicht leer **do**
    $v_2 \leftarrow \mathcal{V}'$.pop()
    $v_1 \leftarrow$ kombinations_algorithmus(Π, v_1, v_2)

  {Da der Kombinations-Algorithmus direkt auf Π arbeitet, muss nichts zurückgegeben werden}

Algorithmus 3: Kombination einer Variablengruppe

8 Die Bewertungsfunktionen

Bevor Variablengruppen kombiniert werden können, muss zuerst entschieden werden, welche Gruppen sich für eine Kombination eignen. Die Auswahl der Gruppen erfolgt jedoch nicht direkt, sondern teilt sich in zwei Schritte auf.

Für die SAS$^+$-Planungsaufgabe $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ weist eine *Bewertungsfunktion* $B : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$ zunächst jedem Variablenpaar $(a, b) \in \{(a, b) | a, b \in \mathcal{V}, a \neq b\}$ eine *Kombinationsgüte* zu, d.h. einen numerischen Wert, der angibt

wie *gut* es ist, a und b zu kombinieren. Die *Güte* wird von jeder Bewertungsfunktion anders definiert, sie induziert jedoch immer eine Präferenzordnung auf der Menge der Variablenpaare.

Der *Selektions-Algorithmus* wählt dann anhand der Kombinationsgüte Variablengruppen aus, die kombiniert werden sollen.

Für den Fluent Merging Algorithmus wurden im Rahmen dieser Arbeit mehrere Bewertungsfunktionen implementiert. Sie sollen in diesem Abschnitt vorgestellt werden.

8.1 Zufällige Bewertung

Dieser Algorithmus kontrolliert, wie nützlich gezielte Kombinationen im Vergleich zu willkürlichen Verschmelzungen sind. Er weist jedem Paar eine zufällige Güte zu.

$$B(a, b) = \text{random}()$$

8.2 Maximal Mutex

Hier wird für jedes Variablenpaar (a, b) die Anzahl der Mutexe berechnet. Die Methode definiert dann die Kombinationsgüte eines Paares durch diese Anzahl.

$$B(a, b) = |\{(d_a, d_b) | d_a \in \mathcal{D}_a, d_b \in \mathcal{D}_b, d_a \text{ und } d_b \text{ mutex}\}|$$

Es wird angenommen, dass es sinnvoll ist, Variablen mit einer hohen Anzahl von Mutexen zu kombinieren, da so der Wertebereich der neuen Variable klein gehalten wird und auch die Anzahl der Operatoren nicht so stark steigt.

8.3 Minimaler Wertebereich

Dieser Ansatz berechnet für alle Variablenpaare $a, b \in \mathcal{V}, a \neq b$, wie viele neue Variablenwerte durch die Kombination von a und b hinzukämen. Die Güte eines Paares ist die negative Anzahl der neuen Variablenwerte.

$$\text{Anzahl der neuen Variablenwerte } = |\mathcal{D}_{a\text{UND}b}| - (|\mathcal{D}_a| + |\mathcal{D}_b|)$$

$$B(a, b) = -\text{Anzahl der neuen Variablenwerte}$$

Variablenpaare, deren Kombination den Gesamtwertebereichs $\sum_{v \in \mathcal{V}} |\mathcal{D}_v|$ am wenigsten vergrößert, bekommen die höchste Güte, da man davon ausgeht, dass man mit einem kleineren Wertebereich schneller gute Pläne finden kann.

8.4 Verbundene Variablen

Diese und die nächste Bewertungsmethode basieren auf der Analyse des Kausalgraphen CG(Π) (siehe Abschnitt 3.3).

Die erste Herangehensweise definiert die Kombinationsgüte zweier Variablen a und $b \in \mathcal{V}$ durch ihre *Verbundenheit* im Kausalgraphen CG(Π). Die Verbundenheit von a und b wird definiert als

$$\text{Verbundenheit}(a, b) = \text{Stärke der Kante } (a, b) + \text{Stärke der Kante } (b, a)$$

$$B(a, b) = \text{Verbundenheit}(a, b)$$

Wir nehmen an, dass die Kombination von Variablen mit einer stärkeren Bindung zu besseren Ergebnissen führt.

8.5 Zweierzyklen

Dies ist ein Spezialfall der vorangegangenen Methode. Wiederum wird die Kombinationsgüte eines Paares durch die Verbundenheit definiert. Bildet aber das Paar (a, b) einen *Zweierzyklus* im Kausalgraphen, so wird die Güte höher gesetzt als für alle Paare, die keinen Zyklus bilden. Für den Kausalgraphen $\text{CG}(\Pi) = (\mathcal{V}, E)$ und ein genügend großes x ist also die Bewertungsfunktion

$$B(a, b) = \begin{cases} \text{Verbundenheit}(a, b) + x, & \text{falls } (a, b) \in E \text{ und } (b, a) \in E \\ \text{Verbundenheit}(a, b), & \text{sonst} \end{cases}$$

Diese Methode wurde aus der Arbeit von van den Briel et al. [7] übernommen.

8.6 Zielvariablen

Auch diese Methode verfeinert die Strategie der *Verbundenen Variablen*. In diesem Fall werden diejenigen Paare mit einer höheren Güte versehen, die mindestens eine in der Zielbeschreibung des Problems vorkommende Variable enthalten. Alle anderen Paare erhalten kleinere Kombinationsgüten.

Für ein genügend großes x ist also die Bewertungsfunktion

$$B(a, b) = \begin{cases} \text{Verbundenheit}(a, b) + x, & \text{falls } s_\star(a) \text{ und/oder } s_\star(b) \text{ definiert} \\ \text{Verbundenheit}(a, b), & \text{sonst} \end{cases}$$

8.7 Geringste Operatorenzahl

Bei diesem Ansatz ist die Kombinationsgüte eines Variablenpaares umso größer, je weniger Operatoren durch die Verschmelzung der Variablen neu eingeführt werden. Sei $\Pi' = \langle \mathcal{V}', \mathcal{O}', s'_0, s'_\star \rangle$ das Planungsproblem, das durch die Kombination der Variablen $a, b \in \mathcal{V}$ in $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ entsteht. Dann ist die Anzahl der neu eingeführten Operatoren

$$\#_{\mathcal{O}_{\text{neu}}} = |\mathcal{O}'| - |\mathcal{O}|$$

und die Bewertungsfunktion wird definiert als

$$B(a, b) = -\#_{\mathcal{O}_{\text{neu}}}$$

Es wird ein besseres Ergebnis erwartet, wenn $\#_{\mathcal{O}_{\text{neu}}}$ klein ist, denn dann ist die Heuristikberechnung im Suchschritt des Fast Downward Planungssystems effizienter.

8.8 Gleiches Objekt

Diese Bewertungsmethode gibt ausschließlich solche Variablenpaare zurück, die über das gleiche Objekt in der Problemwelt sprechen. Es wird davon ausgegangen, dass man sehr gute Ergebnissen erhält, wenn man nur diese Art von Variablenpaaren kombiniert, da so eine gesamtheitlichere Betrachtung des jeweiligen Objekts möglich wird.

Ein stark vereinfachtes Beispiel aus der *Schedule* Planungsdomäne soll zunächst die Idee verdeutlichen. In der Schedule Welt müssen verschiedene Objekte mehreren Operationen unterzogen werden, sodass jedes Objekt am Ende die gewünschte Form, Farbe und Oberfläche hat. In unserem Beispiel gibt es nur ein Objekt, einen Stuhl *stuhl*. Dieser soll geschliffen und gestrichen werden.

Die Repräsentation der Aufgabe nach dem Übersetzungsschritt ist in Abb. 2 dargestellt.

Wir sehen sofort, dass sich die beiden Operatoren *polieren* und *streichen* auf den gleichen Stuhl *stuhl* beziehen und schlussfolgern, dass wir ihn zuerst polieren und anschließend streichen müssen, um die Aufgabe zu lösen. Wenn wir mit dem Streichen anfingen, ginge durch das Polieren die Farbe wieder ab und wir müssten den Stuhl erneut streichen.

Die Repräsentation in Abbildung 2 wird jedoch nur für den Menschen z.B. zur Fehlersuche generiert. Die Variablenwerte und Operatorbezeichnungen aus der eingelesenen PDDL-Datei [6] werden eigentlich nach der Übersetzung vom Fast Downward Planungssystem nicht mehr benutzt. Stattdessen wird

- $\mathcal{V} = \{a, b\}$
- $\mathcal{D}_a = \{$nicht-poliert(stuhl), poliert(stuhl)$\}$
- $\mathcal{D}_b = \{$nicht-gestrichen(stuhl), gestrichen(stuhl)$\}$
- $\mathcal{O} = \{$polieren, streichen$\}$
  - polieren $= \langle\{\}, \{a \mapsto$ poliert(stuhl), $b \mapsto$ nicht-gestrichen(stuhl)$\}\rangle$
  - streichen $= \langle\{b \mapsto$ nicht-gestrichen$\}, \{b \mapsto$ gestrichen(stuhl)$\}\rangle$
- $s_0(a) =$ nicht-poliert
- $s_0(b) =$ nicht-gestrichen
- $s_\star(a) =$ poliert
- $s_\star(b) =$ gestrichen

Abbildung 2: Lesbare Repräsentation des Schedule Beispiels

alles zur höheren Effizienz nur durch Zahlen repräsentiert. Unser Beispiel sieht für das Planungssystem daher ungefähr aus wie in Abb. 3. Damit wir die Operatoren von Variablen unterscheiden können, stehen die Zahlen immer als Index bei einem v oder o.

Das Fast Downward Planungssystem ohne eingebautes Fluent Merging hat also keine Möglichkeit, die Verbindung zwischen den beiden Variablen v_0 und v_1 zu erkennen.

Die vorgestellte Bewertungsmethode aber stellt diese Information zur Verfügung. Dazu wird die menschenlesbare Beschreibung des Problems herangezogen, die die Namen der Operatoren und Variablenwerte enthält. Anhand der Beschreibung findet die Methode für jede Variable v des Planungsproblems das Objekt (oder die Objekte), über die v spricht. Eine Variable v spricht über ein Objekt x, wenn $\mathcal{D}_v$ einen Wert enthält, in dessen Klammer x vorkommt.

Im Schedule Beispiel spricht also Variable a über das Objekt *stuhl* und Variable b über das Objekt *stuhl*.

Alle Variablenpaare, die sich auf das gleiche Objekt beziehen, werden von der Bewertungsmethode mit einer höheren Kombinationsgüte versehen als die restlichen Paare. Formal ist die Bewertungsfunktion

$$B(a, b) = \begin{cases} 1, & \text{falls } a \text{ und } b \text{ sich auf das gleiche Objekt beziehen} \\ 0, & \text{sonst} \end{cases}$$

- $\mathcal{V} = \{v_0, v_1\}$
- $\mathcal{D}_{v_0} = \{0, 1\}$, $\mathcal{D}_{v_1} = \{0, 1\}$
- $\mathcal{O} = \{o_0, o_1\}$
  - $o_0 = \langle\{\}, \{v_0 \mapsto 1, v_1 \mapsto 0\}\rangle$
  - $o_1 = \langle\{v_1 \mapsto 0\}, \{v_1 \mapsto 1\}\rangle$
- $s_0(v_0) = 0$, $s_0(v_1) = 0$
- $s_\star(v_0) = 1$, $s_\star(v_1) = 1$

Abbildung 3: Interne Repräsentation des Schedule Beispiels

Wir nehmen an, dass es sinnvoll ist, Variablen zu kombinieren, die sich auf Zustände oder Eigenschaften des gleichen Objekts beziehen, da wir so eine ganzheitlichere Betrachtung des Objektes erzielen können. Es wird dann in weniger Variablen der Zustand dieses Objekts gespeichert und wir können leichter sehen, wie wir von einem Zustand des Objekts in den nächsten kommen können. Dies kann zu kürzeren Plänen führen, da möglicherweise Operatoren nicht mehr angewandt werden, die das Objekt zunächst in einen Zustand führen, der für die Erreichung des Zielzustands nicht dienlich ist. Diese Idee soll anhand unseres Beispiels erläutert werden.

Der Zustand des Stuhls wird in der Ursprungsbeschreibung durch zwei Variablen beschrieben. Die Variable a speichert, ob der Stuhl poliert, Variable b, ob der Stuhl gestrichen ist. Wenn wir die Variablen a und b kombinieren, erhalten wir eine Variable $a\text{UND}b$, die sich beide Qualitäten gleichzeitig merkt.

Um den Unterschied zwischen der Repräsentation des Stuhlzustands mit einer und zwei Variablen zu sehen, betrachten wir die DTGs der drei Variablen in den Abbildungen 4, 5 und 6.

In der Ursprungsrepräsentation kann es leicht vorkommen, dass zuerst der Operator *streichen* angewandt wird, da er ja Variable b auf ihren Zielwert $s_\star(b)$ setzt. Die resultierende Mehrarbeit kann leicht „übersehen“ werden.

Auf der anderen Seite ist es unwahrscheinlich, dass dieser Fehler mit der kombinierten Variable eintritt. Hier wird anhand des DTGs deutlich, dass die Anwendung von *streichen* am Anfang der Lösung den Plan nur verlängert, da dies einen Umweg auf dem Weg von *nicht-poliertUNDnicht-gestrichen* nach *poliertUNDgestrichen* bedeutet.

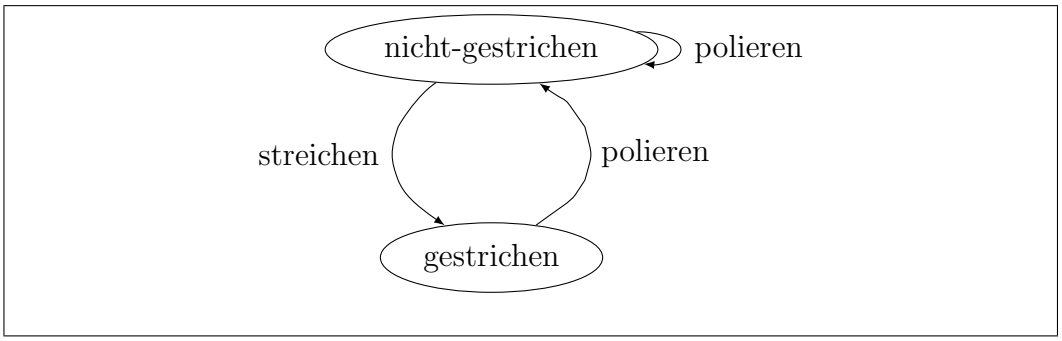


Abbildung 4: DTG_a

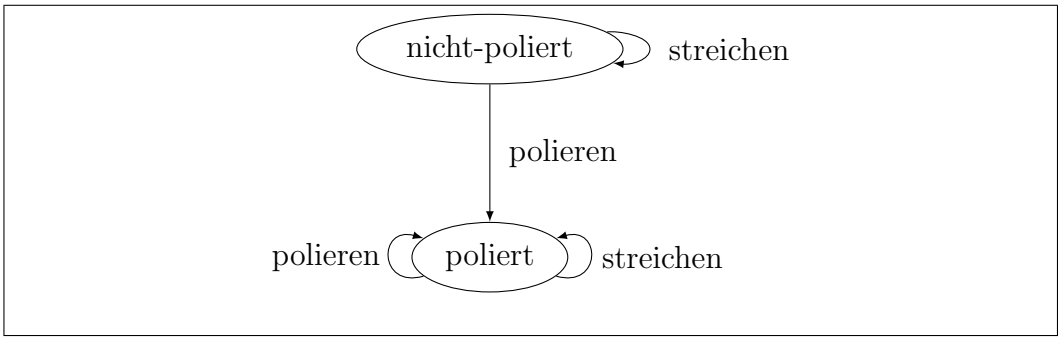


Abbildung 5: DTG_b

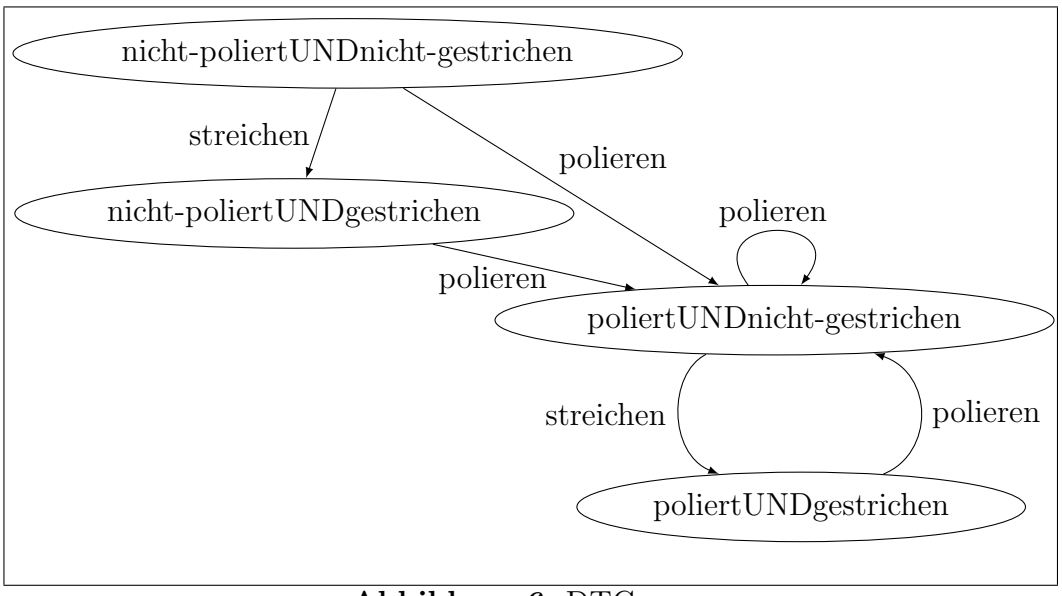


Abbildung 6: $\mathrm{DTG}_{a\mathrm{UND}b}$

9 Der Selektions-Algorithmus

Nachdem die Variablen*paare* von einer Bewertungsfunktion mit Kombinationsgüten versehen wurden, wählt der *Selektions-Algorithmus* die tatsächlich zu kombinierenden Variablen*gruppen* aus. Er erhält als Parameter neben der Menge der Variablenpaare und den zugehörigen Kombinationsgüten die gewünschte Gesamtanzahl der Variablenverschmelzungen n und die maximale Größe einer Variablengruppe m.

Die Rückgabe des Algorithmus ist eine Menge von Variablengruppen G_{komb}, für die gilt:

- Die Variablengruppen in G_{komb} sind paarweise disjunkt
  (Jede Variable kann nur einmal kombiniert werden, da sie anschließend gelöscht wird)
- $\forall g \in G_{\text{komb}} : (|g| \leq m)$
  (Die Gruppengröße wird beschränkt durch m)
- $\sum_{g \in G_{\text{komb}}} (|g| - 1) \leq n$
  (Insgesamt sollen höchstens n-mal zwei Variablen kombiniert werden)

Der Algorithmus gibt diejenigen Variablengruppen zurück, die die obigen Eigenschaften erfüllen und sich aus Variablenpaaren mit möglichst hoher Kombinationsgüte zusammensetzen. Da das Bestimmen der besten Gruppen NP-vollständig ist, wird hierfür ein Greedy-Algorithmus angewandt. Die zurückgegebene Lösung ist also nicht immer optimal.

Wenn $m = 2$ gilt, also nur Paare kombiniert werden sollen, wäre eine zum Algorithmus äquivalente Vorgehensweise, mit dem Greedy-Algorithmus eine möglichst große Menge G_{komb} mit $|G_{\text{komb}}| \leq n$ zu bestimmen, deren Elemente paarweise disjunkte Variablenpaare mit möglichst hoher Kombinationsgüte sind.

Die Selektions-Prozedur wird durch den Algorithmus 4 beschrieben.

Die zurückgegebenen Gruppen werden anschließend mit dem in Abschnitt 7.3 vorgestellten Verfahren jeweils zu einer Variable kombiniert.

9.1 Beispiel

Ein kurzes Beispiel soll die Idee hinter dem Algorithmus verdeutlichen.

Sei $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ eine Planungsaufgabe mit $\mathcal{V} = \{1, \ldots, 10\}$. Angenommen die ausgewählte Bewertungsfunktion gibt für die Variablenpaare folgende Bewertung zurück:

$$B(7,9) \mapsto 3, B(8,3) \mapsto 4, B(8,4) \mapsto 6, B(7,10) \mapsto 8, B(2,5) \mapsto 9$$

Eingabe:

- Menge der Variablenpaare *pairs*
- Gesamtzahl Kombinationen n, Maximale Variablengruppengröße m

Kombiniere Paare aus *pairs* mit gemeinsamen Variablen zu Gruppen g sodass $|g| \leq m$.
$G \leftarrow$ Liste aller $(2, \ldots, m)$-elementigen Teilmengen von $\mathcal{V}$

{Kombiniere möglichst „volle“ Gruppen}
Sortiere die $g \in G$ absteigend nach ihrer Größe $|g|$.

{Kombiniere möglichst „gute“ Gruppen}
Es sei $B(g)$ die Greedy-Approximation der maximalen Bewertungssumme für Paare aus g.
Sortiere die $g_i, g_j \in G$ mit $|g_i| = |g_j|$ so, dass g_i vor g_j, wenn $B(g_i) \geq B(g_j)$.

{Die besten Gruppen stehen jetzt am Anfang von G}

{Jede Variable kann nur einmal kombiniert werden $\Rightarrow$ Mache die g_i paarweise disjunkt}
$G_{\text{disjunkt}} = \emptyset$
for each $g_i \in G$ **do**
  if disjunkt(g_i, g_j) für alle $g_j \in G_{\text{disjunkt}}$ **then**
    $G_{\text{disjunkt}} = G_{\text{disjunkt}} \cup \{g_i\}$

{Selektiere die besten Gruppen, sodass die Anzahl der Merges möglichst n entspricht}
$G_{\text{komb}} \leftarrow \emptyset$ {Die zu kombinierenden Gruppen}
for each Gruppe $g \in G$ **do**
  if Gesamtzahl Kombinationen in $G_{\text{komb}} + |g| - 1 \leq n$ **then**
    $G \leftarrow G_{\text{komb}} \cup \{g\}$
return G_{komb}

Algorithmus 4: Selektions-Algorithmus

Sei $B(x, y) = 0$ für alle Variablenpaare $(x, y) \in \mathcal{V} \times \mathcal{V}, x \neq y$, denen oben kein Wert zugewiesen wurde.

Wir rufen nun den Selektions-Algorithmus mit den Parametern $n = 5$ und $m = 3$ auf und betrachten der Übersichtlichkeit halber nur die 2- und 3-elementigen Teilmengen von $\mathcal{V}$, die Obermengen von Paaren mit einer Kombinationsgüte > 0 sind.

$$\{7, 9\}, \{8, 3, 4\}, \{8, 3\}, \{8, 4\}, \{7, 10\}, \{7, 10, 9\}, \{2, 5\}, \ldots$$

Anschließend werden die Gruppen nach ihrer Größe sortiert:

$$\{8, 3, 4\}, \{7, 10, 9\}, \{7, 9\}, \{8, 3\}, \{8, 4\}, \{7, 10\}, \{2, 5\}, \ldots$$

Wir möchten gerne die größten Gruppen kombinieren. Ist aber die Größe identisch, so soll die Qualität der Gruppe entscheiden. Die Gruppen mit gleicher Größe werden also so sortiert, dass eine Gruppe weiter nach vorn sortiert wird, wenn sie sich aus Paaren mit höherer Güte zusammensetzt.

Im Beispiel lässt sich die Gruppe $\{8, 3, 4\}$ aus den Paaren $(8, 3)$ und $(8, 4)$ zusammensetzen. Die Summe der Kombinationsgüten ist $4 + 6 = 10$. Die andere Dreiergruppe $\{7, 10, 9\}$ kann man durch die Kombination von $(7, 9)$ und $(7, 10)$ herstellen. Die Summe beträgt hier $3 + 8 = 11$. Die Gruppe $\{7, 10, 9\}$ wird dementsprechend vor die Gruppe $\{8, 3, 4\}$ eingeordnet.

Die Zweierpaare werden auch absteigend nach ihrer Kombinationsgüte geordnet und man erhält die Reihenfolge:

$$\{7, 10, 9\}, \{8, 3, 4\}, \{2, 5\}, \{7, 10\}, \{8, 4\}, \{8, 3\}, \{7, 9\}, \ldots$$

Da jede Variable nur einmal kombiniert werden kann, müssen wir garantieren, dass die zurückgegebenen Gruppen paarweise disjunkt sind. Wir löschen also immer die schlechtere von zwei sich überschneidenden Gruppen.

$$\{7, 10, 9\}, \{8, 3, 4\}, \{2, 5\}, \ldots$$

Im letzten Schritt werden nun absteigend für jede Gruppengröße so viele Gruppen zurückgegeben, bis die Anzahl der Kombinationen $n = 5$ erreicht. Im Beispiel resultieren aus der Kombination von $\{7, 10, 9\}$ und $\{8, 3, 4\}$ $2 + 2 = 4$ Kombinationen, es kann also keine Dreiergruppe mehr kombiniert werden. Stattdessen wird die nächstbeste Zweiergruppe zur Rückgabemenge hinzugefügt. Auf die Variablenpaare mit Güte 0 muss nicht zurückgegriffen werden, da die Anzahl der Kombinationen bereits jetzt 5 ist. Die zurückgegebenen Gruppen, die im nächsten Schritt des Fluent Merging Algorithmus vom Kombinationsalgorithmus verschmolzen werden können, sind also

$$\{7, 10, 9\}, \{8, 3, 4\}, \{2, 5\}$$

10 Ergebnisse

Um die oben vorgestellten Bewertungsmethoden zu evaluieren, wird der Fluent Merging Algorithmus auf einigen ICAPS-Planungsbenchmarks ausprobiert.

Jedes Planungsproblem wird dazu zuerst von der PDDL-Repräsentation in eine SAS$^+$-Planungsaufgabe übersetzt. Darauf wird anschließend der Fluent Merging Algorithmus angewandt.

Für den Algorithmus stehen wie oben folgende Optionen zur Verfügung:

- Die Bewertungsfunktion B
- Die Zahl der Variablenkombinationen n
- Die Größe der Variablengruppen m

In der Wissenskompilation wird die resultierende, kombinierte SAS$^+$-Planungsaufgabe vom Planungssystem aufbereitet und anschließend wird für das Problem ein Plan gesucht. Für die Suche wurden zwei Konfigurationen verwendet:

- Die kontext-erweiterte additive Heuristik mit bevorzugten Operatoren [5] zusammen mit gieriger Bestensuche
- Die Landmarken-Schnitt-Heuristik [4] zusammen mit A*

Wir kürzen die beiden Konfigurationen durch die Kürzel ihrer Heuristiken, $h^{\mathbf{cea}}$ und $h^{\mathbf{LM\text{-}cut}}$ ab.

Die erste Konfiguration h^{cea} versucht, einen möglichst guten Plan in möglichst kurzer Zeit zu finden. Die zweite Konfiguration $h^{\text{LM-cut}}$ findet immer einen kürzesten Plan.

Um eine Bewertung der ausgewählten Fluent Merging Optionen zu erhalten, werden die resultierenden Pläne immer mit dem Plan verglichen, den die gleiche Suchkonfiguration für das ursprüngliche Planungsproblem ausgibt. Interessant sind für den Vergleich hauptsächlich drei Werte:

- Wurde ein Plan gefunden?
- Wie lang ist der Plan (Anzahl der Operatoren)?
- Wie lange hat die Suche gedauert?

Die Übersetzung und die Plansuche werden jeweils durch ein 30 minütiges Zeitlimit beschränkt. Wenn einer der beiden Schritte seine Arbeit nicht in dieser Zeit beendet, gilt das Problem als nicht gelöst. Die beiden Schritte dürfen außerdem jeweils höchstens zwei Gigabyte Arbeitsspeicher beanspruchen. Besonders für das Fluent Merging ist diese Beschränkung wichtig, da durch die Verschmelzung die Anzahl der Operatoren sehr groß werden kann.

Abkürzungen Zur Vereinfachung werden die Bewertungsfunktionen im Folgenden abgekürzt.

- **r**: Zufällige Bewertung (*random*)
- **mm**: Maximal Mutex
- **md**: Minimaler Wertebereich (*minimize domain*)
- **mc**: Verbundene Variablen (*most connected*)
- **tc**: Zweierzyklen (*two-cycles*)
- **gv**: Zielvariablen (*goal variables*)
- **fo**: Geringste Operatorenzahl (*fewest operators*)
- **so**: Gleiches Objekt (*same object*)

10.1 $h^{\mathbf{cea}}$ - Die kontext-erweiterte additive Heuristik

Es sollen zunächst die verschiedenen Bewertungsfunktionen bezüglich der Anzahl mit ihnen gelöster Probleme verglichen werden.

10.1.1 Anzahl gelöster Probleme

In Tabelle 1 sieht man die Planungsdomänen auf denen die Bewertungsfunktionen mit der Suchkonfiguration h^{cea} getestet wurden.

Gelöst	$h^{\mathbf{cea}}$	**r**	**mm**	**md**	**mc**	**tc**	**gv**	**fo**
blocks (35)	35	35	35	35	31	31	31	35
driverlog (20)	20	17	13	16	19	14	18	15
grid (5)	5	1	1	5	0	1	1	2
gripper (20)	20	20	15	20	20	20	20	20
logistics00 (28)	28	28	28	28	28	28	28	28
logistics98 (35)	35	28	35	35	20	20	21	11

Gelöst	h^{cea}	**r**	**mm**	**md**	**mc**	**tc**	**gv**	**fo**
miconic (150)	150	150	150	150	150	150	150	150
mprime (35)	35	30	35	35	34	29	35	20
psr-small (50)	50	49	48	48	47	48	47	49
zenotravel (20)	20	20	16	16	20	20	19	15
depot (22)	17	11	14	12	**15**	**15**	13	14
freecell (80)	78	75	**77**	76	72	72	57	37
pathways (30)	15	14	**16**	**17**	14	14	13	15
pipes-nt (50)	38	5	8	**16**	14	14	9	**16**
pipes-t (50)	24	9	3	**17**	11	8	9	15
rovers (40)	34	31	34	**35**	34	34	34	24
schedule (150)	60	58	59	59	54	52	39	**60**
tpp (30)	28	20	**24**	**24**	22	**24**	23	16
trucks (30)	17	15	14	**16**	14	14	**16**	6
total (880)	709	616	625	**660**	619	608	583	548

Tabelle 1: $n = 5, m = 2$, h^{cea}

In der oberen Hälfte der Tabelle (bis zu Zenotravel) stehen die ICAPS-Planungsdomänen, die von h^{cea} ohne Kombinationen vollständig gelöst werden. Bei den Domänen in der zweiten Hälfte gibt es Probleme, die h^{cea} nicht löst. Wir nennen die Domänen aus den beiden Hälften *leicht* und *schwer*.

Betrachten wir zunächst die Gesamtzahl der gelösten Probleme. Keine der Bewertungsfunktionen, schafft es genau so viele Probleme zu lösen wie der originale Algorithmus. Dennoch findet man insgesamt dreimal in einer Domäne mehr Pläne als h^{cea}, wenn man diejenigen Variablen kombiniert, deren Wertebereiche die meisten Mutexpaare beinhalten (mm) oder die den Gesamtwertebereich gering halten (md). Diese Fälle sind unterstrichen und fett markiert.

Eine andere Beobachtung ist, dass es zwischen den Fluent Merging Funktionen zwar deutliche Unterschiede in der Gesamtzahl der gelösten Probleme gibt, dennoch aber jede Bewertungsfunktion in mindestens einer Domäne die meisten Probleme löst, selbst wenn man die leichten Domänen nicht betrachtet (fett hervorgehoben). Ausgenommen davon ist die Zufallsfunktion (r), die aber dennoch insgesamt mehr Probleme löst als die Ansätze, Zweierzyklen (tc) und Zielvariablen (gv) zu kombinieren oder die Operatormenge klein zu halten (fo).

In Tabelle 2 sehen wir, dass die Performanz für $n = 10$ weiter abnimmt und durchschnittlich 10% weniger Aufgaben gelöst werden. Am besten „verkraftet“ die Strategie der kleinsten Operatormenge (fo) den Zuwachs an Verschmelzungen: Hier sinkt die Lösungsrate nur um 4%.

Gelöst	h^{cea}	**r**	**mm**	**md**	**mc**	**tc**	**gv**	**fo**
blocks (35)	35	32	33	35	27	24	25	35
driverlog (20)	20	12	11	12	18	14	17	14
grid (5)	5	1	0	1	0	0	1	0
gripper (20)	20	20	13	20	20	20	20	20
logistics00 (28)	28	28	28	28	28	28	28	28
logistics98 (35)	35	21	35	35	15	14	15	11
miconic (150)	150	150	150	150	150	150	150	150
mprime (35)	35	17	30	31	32	22	35	20
psr-small (50)	50	48	47	47	46	46	47	47
zenotravel (20)	20	14	15	14	20	20	19	15
depot (22)	17	5	**14**	13	**14**	**14**	**14**	12
freecell (80)	78	62	**77**	**77**	68	68	58	36
pathways (30)	15	12	15	12	14	14	13	**16**
pipes-nt (50)	38	3	0	2	**10**	**10**	9	**10**
pipes-t (50)	24	6	2	**15**	6	6	7	8
rovers (40)	34	34	34	34	32	34	**35**	23
schedule (150)	60	35	35	35	6	6	10	**60**
tpp (30)	28	20	**24**	23	22	23	23	16
trucks (30)	17	15	15	**17**	13	13	14	6
total (880)	709	535	578	**601**	541	526	540	527

Tabelle 2: $n = 10, m = 2$, h^{cea}

10.1.2 Planlänge

Planlänge	**r**	**mm**	**md**	**mc**	**tc**	**gv**	**fo**
Punkte	1.05	1.04	1.09	1.15	1.13	1.07	1.11

Tabelle 3: $n = 5, m = 2$, h^{cea}

Punkte einer Bewertungsfunktion Für die Kriterien „Planlänge“ und „Suchzeit“ werden den Bewertungsfunktionen Punkte gegeben. Die Punktzahl bewertet die Leistung der Bewertungsfunktion auf den Domänen aus Tabelle 1. Dabei hängt die Punktzahl davon ab, wie lang die erzeugten Pläne sind, verglichen mit den Plänen, die von h^{cea} erzeugt werden oder wie lange das Suchen gedauert hat, verglichen mit der Zeit, die das Planungssystem für die Lösung der Originalprobleme gebraucht hat. Es werden nur diejenigen Probleminstanzen miteinander verglichen, für die die Suchkonfiguration sowohl für das Original-, als auch das kombinierte Problem eine Lösung gefunden hat. Um die Punktzahl für eine Bewertungsfunktion zu errechnen, werden für jede Domäne drei relative Werte gespeichert, die angeben, wie oft die Funktion schlechter, besser oder genauso gut war wie h^{cea}. Anschließend wird der Durchschnitt der drei Werte über allen Domänen berechnet und die Punktzahl ist:

$$\text{relative Niederlagen} * 0 + \text{relative Unentschieden} * 1 + \text{relative Siege} * 2$$

Da sich die drei relativen Werte zu 1 addieren, ist die Punktzahl einer Funktion größer als 1, wenn öfter gegen h^{cea} gewonnen als verloren wurde.

Dies ist in der obigen Tabelle 3 für alle Funktionen der Fall. Wir sehen also, dass das Kombinieren von Variablen mit großer Wahrscheinlichkeit die Planlänge verkürzt. Dabei nützt die Strategie, Variablen zu kombinieren, die viele Mutexpaare haben, weniger als zufällig Variablen zu kombinieren. Alle anderen Verfahren sind aber besser als die Zufallsmethode. Für die kürzeste Planlänge ist es sinnvoll, die verbundendsten Variablen (mc) oder Zielvariablen zu kombinieren (tc).

Planlänge	**r**	**mm**	**md**	**mc**	**tc**	**gv**	**fo**
Punkte	1.05	1.01	1.03	1.09	1.02	1.06	1.13

Tabelle 4: $n = 10, m = 2$, h^{cea}

Auch für die Planlänge macht es wenig Sinn, 10 Variablenpaare zu kombinieren. In Tabelle 4 sieht man, dass für alle Bewertungsfunktionen die Punktzahl abnimmt oder gleich bleibt im Vergleich zum jeweiligen Wert für $n = 5$. Einzig mit der Methode, die Operatormenge klein zu halten (fo), schafft man noch eine Verbesserung auf 1.13 Punkte.

10.1.3 Suchzeit

Suchzeit	**r**	**mm**	**md**	**mc**	**tc**	**gv**	**fo**
Punkte	0.55	0.60	0.66	0.60	0.57	0.58	0.84

Tabelle 5: $n = 5, m = 2$, h^{cea}

Bezüglich der Suchzeit schneidet die h^{cea}-Suche ohne Kombinationen wieder besser ab als die Kombinationsmethoden (alle Werte sind kleiner 1). Es fällt auf, dass diesmal die Zufallskombination (r) die schlechteste Performanz hat. Vier Funktionen versammeln sich im Mittelfeld und die Methode, den Wertebereich der Variablen gering zu halten (md), ist für eine kurze Suchdauer sinnvoll. Die kürzesten Suchzeiten werden jedoch erreicht, indem man die Operatorenzahl minimiert. Dies deckt sich mit der Annahme, dass die Heuristik effizienter berechnet werden kann, wenn weniger Operatoren zur Verfügung stehen.

Suchzeit	**r**	**mm**	**md**	**mc**	**tc**	**gv**	**fo**
Punkte	0.44	0.53	0.51	0.58	0.55	0.57	0.74

Tabelle 6: $n = 10, m = 2$, h^{cea}

Aus Tabelle 6 wird ersichtlich, dass sich auch bezüglich der Suchzeit die Ergebnisse aller Bewertungsfunktionen verschlechtern, wenn n erhöht wird.

Größere Gruppen Für alle drei betrachteten Werte Anzahl gelöster Probleme, Planlänge und Suchdauer, verschlechtert sich die Performanz der Bewertungsfunktionen, wenn m erhöht wird. Schon bei einem Wert von 3, also maximal drei Variablen in einer Kombinationsgruppe, sind die Ergebnisse sehr viel schlechter als bei $m = 2$.

Zusammenfassung Mit den vorgestellten Kombinationsmethoden lassen sich leider in den meisten Domänen bezüglich der Suchdauer und der Anzahl

der gelösten Probleme keine Verbesserungen erzielen. Lediglich die Planlänge verkürzt sich für alle Methoden im Vergleich zur Länge eines Plans für das jeweilige Originalproblem. Mit zunehmendem n und m werden diese Effekte verstärkt.

Auf der anderen Seite konnten mit der Fluent Merging Methode aber auch in einigen Domänen Verbesserungen erzielt werden. So wurde die Anzahl der gelösten Probleme in den Domänen *Rovers* und *Pathways* gesteigert. Dafür wurden die Methoden *Maximal Mutex*, *Minimaler Wertebereich*, *Zielvariablen*, *Geringste Operatorenzahl* mit guten Werten für n und m benutzt. Will man insgesamt möglichst viele Probleme lösen, so empfiehlt sich die Methode des minimalen Wertebereichs. Wenn es auf die Planlänge ankommt, so ist es sinnvoll, Variablen zu kombinieren, die im Kausalgraphen stark miteinander verbunden sind.

10.2 $h^{\text{LM-cut}}$ - Die Landmarken-Schnitt Heuristik

10.2.1 Anzahl gelöster Probleme

Gelöst	$h^{\text{LM-cut}}$	**r**	**mm**	**md**	**mc**	**tc**	**gv**	**fo**
blocks (35)	28	20	21	21	18	18	17	**26**
depot (22)	7	3	6	4	**7**	**7**	5	**7**
driverlog (20)	14	7	7	9	7	7	7	**10**
grid (5)	2	1	0	2	0	0	0	**2**
gripper (20)	6	5	4	5	4	5	5	**6**
rovers (40)	7	**7**	**7**	**7**	**7**	**7**	**7**	**7**
zenotravel (20)	12	**8**	**8**	7	**8**	**8**	**8**	**8**
total (162)	76	51	53	55	51	52	49	**66**

Tabelle 7: $n = 5, m = 2$, $h^{\text{LM-cut}}$

Lässt man das Fast Downward Planungssystem nach Plänen mit $h^{\text{LM-cut}}$, dem optimalen Planer suchen, so zeigt sich, dass das Fluent Merging hier nicht dazu beiträgt, eine größere Menge an Planungsaufgaben zu lösen. Es werden für die originalen Aufgaben in jeder Domäne mehr Pläne gefunden als für die kombinierten Aufgaben, egal welche Bewertungsfunktion benutzt wird. Am besten schneidet noch die Methode *Geringste Operatorenzahl* (*fo*) ab, was zeigt, dass es für das Finden eines Plans am besten ist, wenn die Operatorenmenge klein ist. Da haben natürlich die originalen Aufgaben einen Vorteil, denn bei ihnen ist die Anzahl der Operatoren immer am geringsten.

Weitere Versuche mit $n > 5$ und $m > 2$ haben gezeigt, dass sich die Performanz des Fluent Merging Algorithmus nur verschlechtert, wenn die Größe der Kombinationsgruppen oder die Anzahl der Kombinationen steigt.

10.2.2 Suchzeit

Suchzeit	**r**	**mm**	**md**	**mc**	**tc**	**gv**	**fo**
Punkte	0.28	0.29	0.39	0.37	0.37	0.29	0.47

Tabelle 8: $n = 5, m = 2$, $h^{\text{LM-cut}}$

Auch in Bezug auf die Suchzeit ist die Fluent Merging Methode mit der $h^{\text{LM-cut}}$-Konfiguration wenig erfolgreich. Nur die Methode, die Operatorenzahl gering zu halten (*fo*), schafft es, annähernd halb so gut zu sein wie der Originalalgorithmus. Alle anderen Bewertungsfunktionen schneiden noch schlechter ab.

10.3 Gleiches-Objekt-Funktion

Die *Gleiches-Objekt-Funktion* wurde auf den Domänen pathways, rovers, schedule getestet, da diese Variablen besitzen, die sich auf das gleiche Objekt beziehen. Hier wurde wiederum die h^{cea}-Heuristik mit gieriger Bestensuche verwendet.

Im Rahmen der Testreihe wurde zuerst die in Abschnitt 8.8 vorgestellte Methode mit dem oben genannten Selektions-Algorithmus ausprobiert. Da die Ergebnisse zwar nicht schlechter waren als die Performanz von h^{cea} auf dem Originalproblem, aber auch nicht ganz die Erwartungen erfüllten, wurde die Gleiches-Objekt-Funktion leicht abgeändert. Anstatt allen Variablenpaaren eine Kombinationsgüte zuzuweisen und die Auswahl der Kombinationsgruppen dem Selektions-Algorithmus zu überlassen, werden die Gruppen direkt an den Kombinations-Algorithmus übergeben. Der Algorithmus findet die Variablengruppen, die sich auf das gleiche Objekt beziehen und teilt diese in Gruppen der Größe $\leq m$ auf.

10.3.1 Anzahl gelöster Probleme

Gelöst	h^{cea}	**5, 3**	**5, 5**	**10, 3**	**10, 5**	**20, 3**
pathways (30)	15	**18**	**<u>20</u>**	**18**	**18**	15
rovers (40)	34	34	34	34	34	34
schedule (150)	60	**<u>81</u>**	**75**	22	**69**	18
total (220)	109	**<u>133</u>**	129	74	121	67

Tabelle 9: Gleiches-Objekt-Funktion, h^{cea}, Gelöste Probleme

Die Zahlenpaare in den Spaltentiteln von Tabelle 9 sind das jeweilige Testpaar n und m. Man sieht, dass es sehr nützlich sein kann, wenn man diese Fluent Merging Methode anwendet. Die Anzahl der gelösten Probleme steigt in der Pathways Domäne von 15 auf 20 und in der Schedule Domäne von 60 auf 81. Lediglich für die Rovers Domäne stellt sich mit keiner der getesteten Kombinationen von n und m eine Verbesserung ein.

10.3.2 Planlänge

Planlänge	**5, 3**	**5, 5**	**10, 3**	**10, 5**	**20, 3**
pathways	4-4-5	4-6-5	2-4-8	2-4-8	3-7-3
rovers	6-18-10	6-18-10	5-19-10	5-19-10	7-14-13
schedule	9-21-29	13-16-30	1-12-9	16-14-29	2-11-5
Punkte	1.18	1.16	**1.31**	1.27	1.11

Tabelle 10: Gleiches-Objekt-Funktion, h^{cea}, Planlänge

In Tabelle 10 sind zusätzlich zu der Punktzahl der Methode die Anzahl der einzelnen Niederlagen, Unentschieden und Siege gegen die Originalaufgabe aufgeführt. Wie schon für die oben besprochenen Methoden, wird die Planlänge durch die Gleiches-Objekt-Funktion in mindestens 10% der Fälle verkürzt. Am besten ist der Wert für 10 Kombinationen von Gruppen mit Maximalgröße 3. Hier wird durchschnittlich in 30% der Fälle ein besseres Ergebnis erreicht.

10.3.3 Suchzeit

Suchzeit	**5, 3**	**5, 5**	**10, 3**	**10, 5**	**20, 3**
Punkte	0.81	0.75	0.89	0.76	0.89

Tabelle 11: Gleiches-Objekt-Funktion, h^{cea}, Suchzeit

Wie bei den anderen Bewertungsfunktionen lässt sich auch für diese Methode eine durchschnittlich längere Suchzeit beobachten. Es fällt auf, dass die Suche besonders für größere Kombinationsgruppen mehr Zeit benötigt und deshalb eine niedrigere Punktzahl bekommt.

11 Fazit

Zusammenfassend lässt sich sagen, dass die Fluent Merging Methode die Suche nach einem guten Plan leider nicht in jedem Fall verbessert. Vielmehr muss auf die Wahl der richtigen Bewertungsfunktion, Gruppengröße und Anzahl der Kombinationen geachtet werden.

Für den optimalen Planer, $h^{\text{LM-cut}}$ mit A*, sind die getesteten Bewertungsfunktionen eher hinderlich, weil es hier scheinbar darauf ankommt, dass die Operatormenge möglichst klein gehalten wird. Da haben natürlich die Originalaufgaben einen Vorteil, denn ihre Operatormenge ist nie größer als die einer kombinierten Aufgabe.

Einige der Funktionen bewirken eine Verbesserung der Suchperformanz mit der h^{cea}-Heuristik. Es werden zum Teil mehr Probleme gelöst und durchschnittlich werden kürzere Pläne gefunden. Dies wird mit einer durchschnittlich längeren Suchzeit bezahlt.

Die besten Ergebnisse werden mit der Gleiches-Objekt-Methode erzielt. Hier konnte für einige ausgewählte Domänen sowohl eine Steigerung der Zahl der gelösten Probleme, als auch eine durchschnittlich kürzere Planlänge festgestellt werden.

Interessant wäre es, herauszufinden, ob das Auslassen des Selektions-Algorithmus positive Auswirkungen auf die anderen Bewertungsfunktionen hätte. Dies ist aber unwahrscheinlich, da die anderen Funktionen alle schon für eine Gruppengröße von 2 nicht mehr die gleiche Performanz zeigen wie das Planungssystem für die jeweilige Originalaufgabe. Die Gleiches-Objekt-Methode schneidet hingegen für Zweierpaare noch genau so gut ab wie die Originalaufgabe. Eine Vergrößerung der Gruppen hat wahrscheinlich immer eine schlechtere Leistung zur Folge, auch ohne den Selektions-Algorithmus.

Wünschenswert für weitere Forschung wäre die Überprüfung, ob sich die guten Ergebnisse der Gleiches-Objekt-Funktion auf mehr Domänen als den bereits getesteten reproduzieren lassen.

Literatur

[1] Bäckström, Christer und Bernhard Nebel: *Complexity Results for SAS^+ Planning.* Computational Intelligence, 11:625–655, 1995.

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

[3] Helmert, Malte: *Concise finite-domain representations for PDDL planning tasks.* Artif. Intell., 173(5–6):503–535, 2009.

[4] Helmert, Malte und Carmel Domshlak: *Landmarks, Critical Paths and Abstractions: What's the Difference Anyway?* In: *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 2009.

[5] Helmert, Malte und Héctor Geffner: *Unifying the Causal Graph and Additive Heuristics.* In: Rintanen, Jussi, Bernhard Nebel, J. Christopher Beck und Eric Hansen (Herausgeber): *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 2008)*, Seiten 140–147. AAAI Press, 2008.

[6] McDermott, Drew, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld und David Wilkins: *PDDL — The Planning Domain Definition Language.* Technischer Bericht CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.

[7] van den Briel, Menkes H. L., Subbarao Kambhampati und Thomas Vossen: *Fluent merging: A general technique to improve reachability heuristics and factored planning.* Proceedings of the ICAPS Workshop on Heuristics for Domain-independent Planning: Progress, Ideas, Limitations, Challenges, 2007.