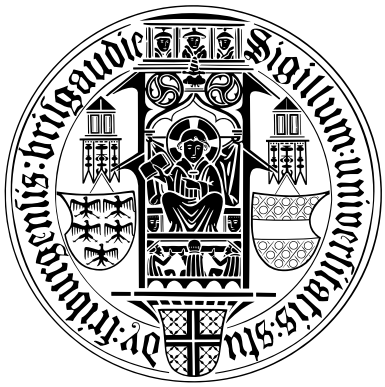
Eine automatentheoretische Heuristik für klassische Planungsprobleme

Diplomarbeit
von
Dennis Jung

Juni 2007



Albert-Ludwigs-Universität Freiburg
Fakultät für Angewandte Wissenschaften
Institut für Informatik
Lehrstuhl Grundlagen der Künstlichen Intelligenz

Danksagung

Ein großes Dankeschön geht an alle Mitarbeiter des Lehrstuhls, die zum Gelingen dieser Arbeit beigetragen haben. Prof. Dr. Nebel danke ich dafür, dass er diese Arbeit ermöglicht hat. Mein besonderer Dank gilt außerdem Malte Helmert für die ausgezeichnete Betreuung und die unzähligen Ratschläge in allen Bereichen.

Erklärung

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig 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 Juni 2007 Dennis Jung

Inhaltsverzeichnis

Kapitel 1

Einleitung

Das Verifizieren von Invarianten in der Modellprüfung ist eine Aufgabenstellung, die sehr nah mit der klassischen Handlungsplanung verwandt ist. Daher ist es oft möglich, Ideen aus dem einen Gebiet auf das andere zu übertragen. In dieser Arbeit wird eine automatentheoretische Heuristik aus der Modellprüfung auf Probleme der Handlungsplanung angewandt.

Ausgehend von einer Implementation des Fast-Downward-Planers von Malte Helmert [1] wird das von Dräger, Finkbeiner und Podelski [2] entwickelte Verfahren zum Berechnen von Fehlerabständen in die Planungsdomäne übersetzt, implementiert und getestet.

Nach der Motivation der Arbeit werden in Kapitel 2 kurz einige notwendige Begriffe definiert. In Kapitel 3 werden die einzelnen Komponenten des zu Grunde liegenden Fast-Downward-Planers und deren Zusammenhang untereinander betrachtet, bevor es dann in Kapitel 4 um die für die Handlungsplanung übersetzte Dräger-Finkbeiner-Heuristik (DF-Heuristk) geht. Nachdem der Versuchsaufbau in Kapitel 5 für die folgenden Experimente beschrieben wurde, wird in Kapitel 6 ein wesentlicher Parameter der angewandten Heuristik näher in Augenschein genommen. Abschließend soll die lauffähige Implementation der DF-Heuristik mit den sehr erfolgreichen Fast-Forward- und Fast-Downward-Heuristiken verglichen werden.

Kapitel 2

Grundbegriffe

Hier soll ein kurzer Überblick über die für diese Arbeit wichtigen Grundbegriffe gegeben werden.

2.1 Klassisches Planen

Klassische Planungsumgebungen zeichnen sich dadurch aus, dass sie deterministisch, finit, statisch und vollständig beobachtbar sind. Konkret bedeutet das, dass sich ein Problemzustand nicht ohne das Zutun des Planers ändert, und dass jede Aktion den Zustand auf eine festgelegte Art und Weise beeinflusst. Die Problembeschreibung und auch ein möglicher Plan sind immer in endlich vielen Schritten beschreibbar.

Planen ist schwer

Die Notwendigkeit für ein spezielles Planungssystem resultiert aus der Überlegung, dass ein allgemeiner Problemlösungsagent leicht durch die schiere Größe eines Realweltproblems verloren gehen kann. So würde ein Agent, der den Planungsraum mit Hilfe von Breiten- oder Tiefensuche traversiert eine Fülle von unnützen Aktionen berücksichtigen, die es ihm unmöglich machen einen vernünftigen Plan in annehmbarer Zeit zu finden. Um eine zielgerichtete Suche wie zum Beispiel mit dem Algorithmus A^* benutzen zu können bedarf es einer möglichst guten, zulässigen (engl. admissible) Heuristik.

Eine Abstandsheuristik ist zulässig, wenn sie den tatsächlichen Abstand niemals überschätzt. Eine noch stärkere Bedingung ist Konsistenz (engl. consistency). Eine Funktion f ist konsistent, wenn für jeden Zustand q und jeden

Nachfolger q' von q gilt: $f(q) \leq f(q') + 1$. Konsistente Abstandsfunktionen verbessern die Laufzeit von A^*, weil es nie nötig ist einmal verworfene Zustände wider zu öffnen. Ein Zustand muss erneut betrachtet werden, wenn er auf einem kürzeren Pfad ein weiteres Mal entdeckt wird. Eine konsistente Abstandsheuristik sorgt dafür, dass der Zustand *immer* zuerst auf dem kürzesten Pfad entdeckt wird. Jede konsistente Funktion ist auch zulässig [4].

Mit Hilfe einer solchen Suchfunktion kann man die erwartete Laufzeit drastisch herabsetzen, auch wenn im schlimmsten Fall eine exponentielle Laufzeit nicht zu verhindern ist. Dies gilt jedoch nur für optimales Planen, also für Planungssysteme, die den Anspruch erheben immer den bestmöglichen Plan zu finden, falls es einen gibt.

2.2 PDDL

Klassische Planungsprobleme werden in der Regel in PDDL beschrieben. PDDL (Planing Domain Definition Language) ist eine Beschreibungssprache für Planungsaufgaben. Sie wurde ursprünglich von Drew McDermott und dem AIPS-98 Competition Committee entwickelt um Problemräume und Aufgaben zu beschreiben. Mittlerweile stellt sie den de-facto Standard für die Repräsentation und den Austausch von Planungsmodellen dar. PDDL hat für einen erheblichen Fortschritt im Bereich der Planungsforschung gesorgt und ist verantwortlich dafür, dass Systeme, die den Standard benutzen, leicht untereinander verglichen werden können. Damit hat PDDL zu einem enormen Anstieg an verfügbaren Planungsbenchmarks geführt. Die Einführung von PDDL hat die wissenschaftliche Entwicklung im Bereich der Planung vorangetrieben. PDDL wurde stark beeinflusst vom STRIPS[1]-Formalismus, der nun eine Untermenge von PDDL bildet. Nach mehreren Erweiterungen liegt seit 2006 die Version PDDL3.0 [5] vor.

Eine PDDL-Definition besteht aus zwei Teilen: Der Domäne (engl. Domain) und der Problembeschreibung (engl. Task).

Die *Domain* enthält die Prädikate und Operatoren (in PDDL Aktionen genannt) und beschreibt den zu Grunde liegenden Problemraum der Planungsaufgabe. Sie kann auch Typen, Konstanten, statische Fakten und eine Reihe anderer Dinge enthalten, je nach Mächtigkeit des verwendeten PDDL-Fragments.

Ein *Task* beschreibt den Anfangs- und Endzustand eines zu lösenden Problems auf Grunde des Problemraums. Eine Lösung des Problems ist ein

[1] STRIPS steht für STanford Research Institute Problem Solver.

Plan, der den Initialzustand in den Endzustand überführt.

PDDL ist eine derart mächtige Sprache, dass kein Planer alle Features komplett unterstützt. Jede Domain definiert deshalb mit Hilfe von so genannten *requirements*, welche Features ein Planer unterstützen muss, um das Problem zu lösen. Die am häufigsten benutzten Features sind: `:strips`, `:equality`, `:typing` und `:adl`. Eine vollständige Liste aller in PDDL enthaltenen *requirements* findet sich unter [3].

Eine PDDL-Planungsaufgabe *erster Ordnung* (*Level 1*) benutzt ausschließlich die nicht-numerischen Sprachfeatures von PDDL. Numerische Zustandsvariablen werden in PDDL *zweiter Ordnung* eingeführt. Der temporale Teil der PDDL-Sprache ist ab der *dritten Ordnung* verfügbar.

Definition

Eine PDDL-Planungsaufgabe erster Ordnung ist ein 4-Tupel $\langle L, I, O, G\rangle$ mit den folgenden Komponenten:

- L ist eine endliche prädikatenlogische Sprache, bestehend aus Konstantensymbolen (Objekten), Relationssymbolen (Prädikaten) und Variablensymbolen.
- I Der Initialzustand (initial state) ist eine Liste von Atomen über Objekten und Prädikaten.
- O ist eine endliche Menge an Operatoren über L. Ein Operator $\langle c, e\rangle$ besteht aus einer prädikatenlogischen Formel c über L, genannt Vorbedingung (precondition) und einem Effekt e. Ein Effekt ist rekursiv definiert als:
  - Ein Literal l über L ist ein *einfacher Effekt*.
  - Wenn $e_1, \ldots, e_n$ Effekte sind, dann ist $e_1 \wedge \cdots \wedge e_2$ ein *konjunktiver Effekt*.
  - Wenn X eine prädikatenlogische Formel über L ist, dann ist $X \triangleright e$ ein *konditionaler Effekt*.
  - Wenn $x_1, \ldots, x_k$ Variablensymbole und e ein Effekt ist, dann ist $\forall x_1, \ldots, x_k : e$ ein *universell quantifizierter Effekt*.
- G Der Zielzustand (goal state) ist eine prädikatenlogische Formel über L.

2.3 Mehrwertige Planungsaufgaben

Wegen der Vorherrschaft des PDDL-Formalismus [5] (und davor STRIPS) werden im Bereich der klassischen Planung Zustandsvariablen allgemeinhin als binär aufgefasst, d.h. sie nehmen nur die Werte *wahr* oder *falsch* an.
Der von Bäckström et al. eingeführte SAS^+-Formalismus [6] erlaubt es, Variablen mit mehr als nur zwei Werten zu definieren. Zustandsvariablen, die mehr als zwei Zustände annehmen können, so wie sie im Fast-Downward Planer benutzt werden, nennt man *mehrwertige Zustandsvariablen* (engl. *multi-valued state variables*).

Definition

Eine mehrwertige Planungsaufgabe (engl. multi-valued planning task, kurz MPT) ist ein 5-Tupel $\Pi = \langle \mathcal{V}, s_0, s_*, \mathcal{A}, \mathcal{O} \rangle$ mit den folgenden Komponenten:

- $\mathcal{V}$ ist eine endliche Menge von Zustandsvariablen. Jede Variable $v \in \mathcal{V}$ hat eine eigene, endliche Domäne $dom(v) = D_v$.
- s_0 ist ein Zustand über $\mathcal{V}$, genannt Initialzustand.
- s_* ist eine partielle Variablenbelegung über $\mathcal{V}$, genannt Zielzustand.
- $\mathcal{A}$ ist eine endliche Menge von (MPT-) Axiomen über $\mathcal{V}$.
- $\mathcal{O}$ ist eine endliche Menge von (MPT-) Operatoren über $\mathcal{V}$. Ein Operator $\langle pre, \mathit{eff} \rangle$ besteht aus einer partiellen Variablenbelegung pre über $\mathcal{V}$, genannt Vorbedingung (engl. precondition) und einer endlichen Menge an Effekten eff. Effekte sind Tripel $\langle cond, v, d \rangle$, wobei $cond$ eine (möglicherweise leere) partielle Variablenbelegung, genannt Effektbedingung (engl. effect condition) ist, v ist die betroffene Variable und $d \in D_v$ ist der neue Wert für v.

```
(define (problem MIC1)
  (:domain miconic)
  (:objects p0 f0 f1 )
  (:init (passenger p0) (floor f0) (floor f1) (above f0 f1)
         (origin p0 f1) (destin p0 f0) (lift-at f0) )
  (:goal (and (served p0) ))
)
```

Abbildung 2.1: Ein MICONIC-Problem, beschrieben in PDDL.

```
(define (domain miconic)
  (:requirements :strips)
  (:predicates (origin ?person ?floor )
    (floor ?floor) (passenger ?passenger)
    (destin ?person ?floor ) (lift-at ?floor )
    (above ?floor1 ?floor2 ) (served ?person )
    (boarded ?person )
(:action board
  :parameters (?f ?p)
  :precondition (and (floor ?f) (passenger ?p)(lift-at ?f)
                     (origin ?p ?f))
  :effect (boarded ?p))
(:action depart
  :parameters (?f  ?p)
  :precondition (and (floor ?f) (passenger ?p) (lift-at ?f)
                     (destin ?p ?f) (boarded ?p))
  :effect (and (not (boarded ?p))
               (served ?p)))
(:action up
  :parameters (?f1 ?f2)
  :precondition (and (floor ?f1) (floor ?f2) (lift-at ?f1)
                     (above ?f1 ?f2))
  :effect (and (lift-at ?f2) (not (lift-at ?f1))))
(:action down
  :parameters (?f1 ?f2)
  :precondition (and (floor ?f1) (floor ?f2) (lift-at ?f1)
                     (above ?f2 ?f1))
  :effect (and (lift-at ?f2) (not (lift-at ?f1))))
)
```

Abbildung 2.2: MICONIC als einfache Beispieldomäne.

Kapitel 3

Aufbau des zugrunde liegenden Fast-Downward-Planers

In diesem Kapitel wird beschrieben, aus welchen Komponenten der Fast Downward Planer aufgebaut ist und welche Teile für diese Arbeit wichtig sind.

Der Fast-Downward-Planer kommt mit allen "rein logischen" Aspekten der PDDL-Sprache zurecht. Dazu zählen beliebige prädikatenlogische Formeln in Vorbedingungen und Effekten (einfach oder konditional) von Aktionen und abgeleiteten Prädikaten (Axiome). Das heißt er kann alle Planungsaufgaben, welche in PDDL erster Ordnung beschrieben sind, bearbeiten.

Der Fast-Downward-Planer besteht aus drei Komponenten: Der Übersetzungseinheit (*Translation*), der Einheit für Wissensaufbereitung (*Knowledge Compilation*) und der Sucheinheit (*Search*). Translation und Knowledge Compilation sind Vorverarbeitungsschritte, die der Sucheinheit eine Heuristik zur Verfügung stellen. Search führt dann die eigentliche Suche im Problemraum aus.

Translation

Diese Komponente übersetzt eine gegebene Planungsdomäne von PDDL erster Ordnung in eine voll-instantiierte Repräsentation basierend auf dem SAS^+-Formalismus [6]. Nach dem Übersetzungsschritt liegt also eine mehrwertige Planungsaufgabe vor und die Zustandsvariablen sind nicht länger auf binäre Domänen beschränkt.

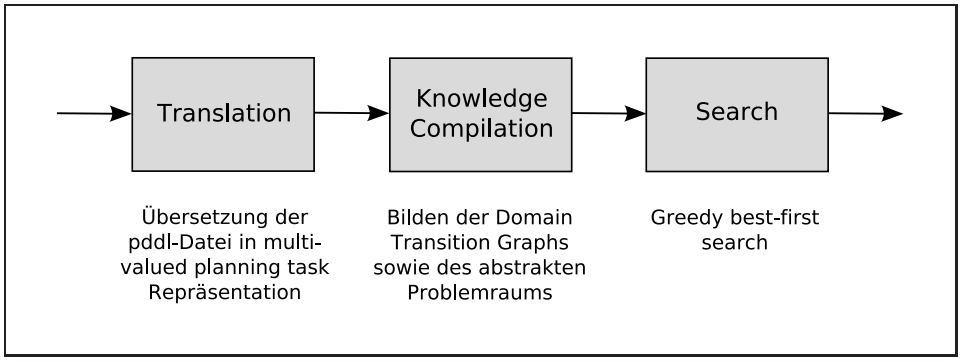


Abbildung 3.1: Die drei Ausführungsstufen des Fast-Downward-Planers.

Knowledge Compilation

Diese Komponente generiert eine Reihe effizienter Datenstrukturen, unter anderem auch die Domain Transition Graphs. Diese werden benutzt, um Abstraction Graphs zu erstellen, die eine zentrale Rolle beim Berechnen der Dräger-Finkbeiner-Heuristik spielen werden. Für jede (mehrwertige) Zustandsvariable wird genau ein Domain Transition Graph erstellt.

Domain Transition Graphs

Ein Domain Transition Graph einer Zustandsvariable kodiert, unter welchen Umständen die Variable ihren Zustand ändern kann. Das heißt, von welchen Werten in ihrer Domäne zu welchen anderen Werten ein Übergang stattfinden kann, welche Operatoren oder Axiome dafür verantwortlich sind und welche Bedingungen anderer Zustandsvariablen mit dem Übergang zusammenhängen. Da ein Domain Transition Graph für alle denkbaren Zustände einer Variable einen Knoten zur Verfügung stellen muss, kann es vorkommen, dass ein Zustandsknoten erzeugt wird, der durch keine legale Aktion erreicht werden kann. Abbildung 3.2 zeigt die erstellten Domain Transition Graphs für das MICONIC-Problem aus Abbildung 2.1.

Search

Nach den vorbereitenden Schritten (Übersetzen der PDDL-Planungsaufgabe in die SAS^+-Repräsentation, dem Erstellen der Domain Transition Graphs und des abstrakten Problemraums) kann nun die Such-Komponente die eigentliche Suche nach einem Plan beginnen. Dafür stellt der Fast Downward Planer drei Modi zur Verfügung: Greedy best-first search, Multi-heuristic best-first search, Focused iterative-broadening search, wobei nur die ersten

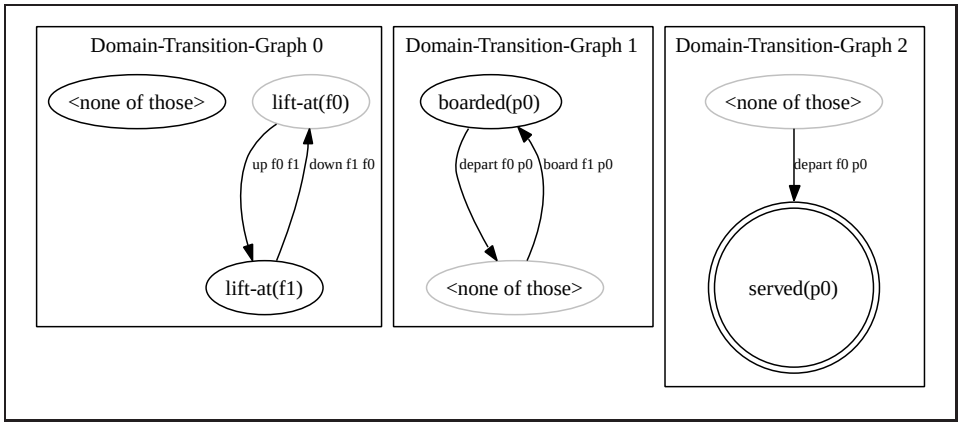


Abbildung 3.2: Alle Domain Transition Graphs für das MICONIC-Problem MIC1. Initialknoten sind grau dargestellt. Der Zielknoten ist doppelt umrandet.

beiden Suchverfahren heuristische Suchalgorithmen sind. Da für diese Arbeit nur greedy best-first search Verwendung findet, wird nur es näher erläutert. Für Näheres zu den anderen Suchverfahren siehe [1].

Greedy best-first search ist ein Standardalgorithmus aus dem Repertoire der Künstlichen Intelligenz. In jedem Schritt expandiert der Algorithmus den Knoten mit dem besten heuristischen Wert, bis er einen Zielknoten gefunden hat oder die Suche abbrechen muss, weil alle Knoten expandiert wurden. Damit derselbe Knoten nicht zweimal in Betracht gezogen wird, unterhält greedy best-first search eine Liste mit bereits besuchten Knoten. Weil dabei, anders als bei A^*, der bereits zurückgelegte Weg außer Acht gelassen wird, findet greedy best-first search nicht notwendigerweise den kürzesten Weg, auch wenn die verwendete Heuristik zulässig (engl. admissible) oder gar konsistent ist. Im Allgemeinen ist greedy best-first search schneller als A^*, weswegen dieser Algorithmus hier auch Verwendung findet.

Kapitel 4

Planen mit Dräger-Finkbeiner Heuristik

Eine möglichst gute Heuristik ist die Grundvoraussetzung, um schnell in einem Problemraum suchen zu können. "Gut" bedeutet in diesem Zusammenhang: schnell zu berechnen und möglichst genaue Distanzangaben liefernd. Dies ist klar, denn ein schneller Algorithmus hat einen schnellen Planer zur Folge und wenn der Algorithmus stets die richtige Entfernung zum nächsten Zielzustand angeben würde, fände z.B. greedy best-first search immer den kürzesten Plan in linearer Zeit. (Eine perfekte Heuristik würde tatsächlich das Suchen im Problemraum überflüssig machen.)

Eine perfekte Heuristik zu berechnen ist aber nur bei sehr kleinen Problemräumen praktikabel. Problemräume, die mehrere Millionen oder gar Milliarden paarweise verschiedene Problemzustände haben, komplett zu berechnen würde zu viel Zeit und Platz in Anspruch nehmen um einen praktischen Nutzen zu haben.

Abbildung 4.1 zeigt den vollständig aufgespannten Problemraum des zweitkleinsten MICONIC-Problems. Vergleicht man die Abbildung mit der Abbildung 4.4 auf Seite 16, welche einen Abstraktionsgraphen zeigt, der den Problemraum des leichtesten MICONIC Problems repräsentiert, so erkennt man, dass eine Verdopplung der Variablen (siehe Tabelle auf Seite 21) eine Quadratur der Knoten des Problemraums nach sich zog. (Acht Zustandsknoten vs. Vierundsechzig.) Dies verdeutlicht wie schnell die Anzahl der Zustandsknoten schon bei "leichten" Problemen wie MICONIC wächst.

Es bleibt also die Frage wie man eine möglichst präzise Einschätzung der Zieldistanz erreichen kann ohne den gesamten Problemraum modellieren zu müssen. Die von Dräger und Finkbeiner vorgeschlagene Lösung sieht so aus: Man berechnet eine abstrahierten, kleineren Problemraum und zieht

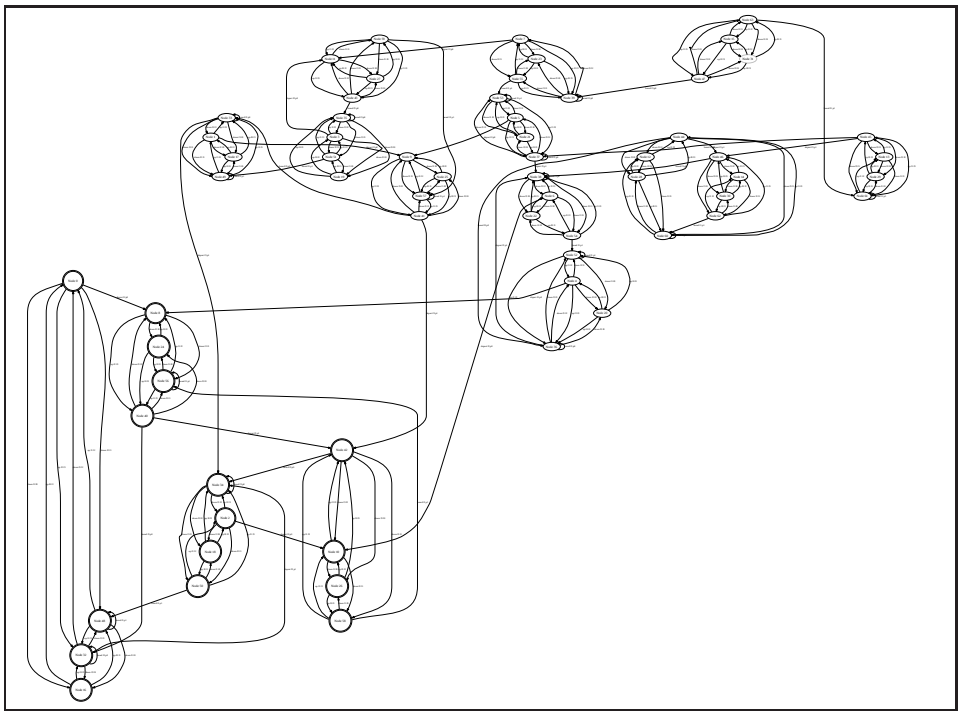


Abbildung 4.1: Der vollständige Problemraum des MICONIC-Problems MIC2. Jeder Weg vom Initialknoten rechts oben zu einem Zielknoten (links unten) kodiert einen validen Plan.

Rückschlüsse von den Distanzen der abstrakten Zustände auf die Distanzen der tatsächlichen Zustände.

4.1 Schreibweise

Sei $\Pi = \langle \mathcal{V}, s_0, s_*, \mathcal{A}, \mathcal{O} \rangle$ ein multi-valued planning task und $v \in V$ eine Zustandsvariable von Π. Dann ist $AG(v)$ der Abstraction Graph von v.

Der Abstraction Graph von v ist nun gegeben als:
$AG(v) = \langle \Sigma, Q, q_0, Q_g, \rightarrow \rangle$ mit den folgenden Komponenten:

- $\Sigma \subseteq \mathcal{O}$ ist eine endliche Menge von Aktionen.
- $Q \subseteq D_v$ ist eine endliche Menge von Zuständen, die die Variable v annehmen kann, in diesem Zusammenhang auch Variablenzustand oder Zustandsknoten genannt.
- $q_0 \in Q$ ist ein Zustand (Zustandsknoten), der im Initialzustand enthalten ist.
- $Q_g \subseteq Q$ ist eine Menge von Zuständen, die in einem Endzustand vertreten sind.
- $\rightarrow \subseteq (Q \times \Sigma \times Q)$ ist eine Transitionsrelation.
  Eine Transition $(p, a, p') \in \rightarrow$ wird geschrieben als $p \xrightarrow{a} p'$.

Ein Zielzustand des Problems ist erreicht, wenn alle Variablen in einem Zielzustand sind. Die *Distanz zu einem Zielzustand* $d_p(q) \in \mathbb{N} \cup \{\infty\}$ eines Variablenzustandes q in einem Graphen D ist die Länge des kürzesten Weges von q zu einem Zustand $q' \in Q_g$ (oder ∞, wenn ein solcher Weg nicht existiert).

Die Vereinigung von zwei Abstraktionsgraphen ist eine einfache Kombinationsaufgabe, bei der je zwei Zustandsknoten miteinander verschmolzen werden. Betrachten wir zwei Graphen $G_i = (\Sigma_i, Q_i, q_{0i}, Q_{g_i}, \rightarrow_i)$, $i = 1, 2$. Die *Verschmelzung*

$$G_1 || G_2 = (\Sigma_1 \cup \Sigma_2, Q_1 \times Q_2, (q_{01}, q_{02}), Q_{g1} \times Q_{g2}, \rightarrow)$$

vereint die zwei Graphen über alle Aktionssymbole $(\Sigma_1 \cup \Sigma_2)$:

$$(p, q) \xrightarrow{a} (p', q') \text{ gdw. } \begin{cases} p \xrightarrow{a}_1 p', q = q', \text{ und } a \in (\Sigma_1 \setminus \Sigma_2) \\ p = p', q \xrightarrow{a}_2 q', \text{ und } a \in (\Sigma_2 \setminus \Sigma_1) \\ p \xrightarrow{a} p', q \xrightarrow{a} q', \text{ und } a \in (\Sigma_1 \cap \Sigma_2) \end{cases}$$

Die Verschmelzung zweier Graphen ist assoziativ und kommutativ. Daraus folgt, dass die Reihenfolge der Verschmelzungen einer Menge von Graphen keinen Einfluss auf den resultierenden Ergebnisgraphen hat.

Die Verschmelzung einer Menge von Graphen wird geschrieben als:

$$\mathcal{G} = \{G_1, \ldots, G_k\} \text{ mit } ||_{G \in \mathcal{G}} G = G_1 || \ldots || G_k$$

4.2 Berechnung des abstrakten Problems

Die von Dräger und Finkbeiner vorgeschlagene Funktion zur Berechnung der Fehlerdistanz (lies: Distanz zum nächsten Zielzustand) beruht auf einer Abstraktion des Problemraums. Damit nicht der komplette Zustandsraum über alle Variablen aufgespannt wird, wird die Abstraktion inkrementell berechnet. Jeder Verschmelzungsschritt wird direkt gefolgt von einem Abstraktionsschritt.

Algorithmus 1: AbstractProblem beschreibt so einen "verschmelze-und-abstrahiere"-Schritt. Wie die Abstraktion erstellt wird, wird in Abschnitt 4.3.1 beschrieben. Entscheidend ist auch, in welcher Reihenfolge die Graphen verschmolzen werden. Algorithmus 1 wird parametrisiert mit der *Verschmelzungsstrategie* S, welche ein Paar aus der Menge an zu verschmelzenden Graphen wählt. Verschiedene Verschmelzungsstrategien werden in Kapitel 6 besprochen.

Der Algorithmus AbstractProblem bearbeitet eine Graphenmenge $\mathcal{D}'$, die anfangs gleich der Menge der gegebenen Domain Transition Graphs $\mathcal{D}$ ist. In einem ersten Schritt werden die von der Knowledge-Compilation-Komponente (s. Kapitel 3) zur Verfügung gestellten Domain Transition Graphs in Abstraction Graphs umgewandelt.

Danach wird $\mathcal{D}'$ Schritt für Schritt reduziert, bis nur noch ein einziges Element $\{A\}$ übrig bleibt. Der Abstraktionsgraph A repräsentiert den Zustandsraum des zu lösenden Problems. Zusammen mit jedem Graphen D' in $\mathcal{D}'$ wird die Funktion $\alpha_{D'} : \prod_{D \in \mathcal{D}'} Q_D \rightarrow (Q'_D \cup \{\square\})$ berechnet. Sie bildet jeden tatsächlichen Zustandsknoten q auf einen Knoten q' eines Abstraktionsgraphen oder auf $\square$ ab. Das Ergebnis $\alpha : D'(p) = \square$ bedeutet, dass der Knoten q Teil einer Sackgasse ist. Das heißt, er ist entweder nicht vom Initialknoten aus erreichbar, oder es kann kein Zielknoten von q erreicht werden. Für die Abstraktionsgraphen der anfänglichen Domain Transition Graphs in $\mathcal{D}$ verweist α_D auf die Zustände der Domain Transition Graphs, aus denen sie entstanden sind.

In jeder Iteration der Schleife "Verschmelze und vereinfache" werden zwei Graphen D und D' aus der Menge $\mathcal{D}'$ von der Verschmelzungsstrategie S

Algorithmus 1 : AbstractProblem berechnet eine Menge von abstrakten Graphen für ein gegebene Menge konkreter Graphen.

Eingabe : konkretes Problem, gegeben als Menge von DTGs
$\mathcal{D} = \{D_1, \ldots, D_n\}$
Ausgabe : abstraktes Problem, gegeben als Abstraktionsgraph A
Ausgabe : Abbildung von konkreten zu abstrakten Zuständen:
$\alpha : \prod_{D \in \mathcal{D}} Q_D \to (Q_A \cup \{\square\})$

begin
  `// Initialisierung`
  $\mathcal{D}' := AG(\mathcal{D})$;
  for $i = 1, \ldots, n$ **do** $\alpha_{P_i}(q_1, \ldots, q_n) = q_i$;
  `// Verschmelze und vereinfache`
  while $|\mathcal{D}'| > 1$ **do**
    $(D, D') := S(\mathcal{D}')$;
    $(C, \gamma) :=$AbstractGraph$(D||D')$;
    $\mathcal{D}' := \mathcal{D}' \cup \{C\} \setminus \{D, D'\}$;
    $\alpha_C(q) := \begin{cases} \square & \text{falls } \alpha_D(q) = \square \text{ oder } \alpha'_D(q) = \square, \\ \gamma(\alpha_D(q), \alpha_{D'}(q)) & \text{sonst} \end{cases}$
  end
  $A :=$ das letzte Element in $\mathcal{D}'$;
  return A, α_A
end

ausgewählt. Zuerst wird ihr Graphenprodukt $D||D'$ explizit berechnet und dann sofort mit AbstractGraph zu C vereinfacht. In der neuen Graphenmenge $\mathcal{D}'$ ersetzt C die beiden Graphen D und D'. Zusammen mit C wird die neue Abbildung α_C aus den Abbildungen von α_D und $\alpha_{D'}$ berechnet.

Das Ergebnis von AbstractGraph ist der Abstraktionsgraph A und die Abbildung α_A, die konkrete Zustände auf abstrakte Zustände oder auf $\square$ abbildet. Hiervon wird die Abstandsfunktion abgeleitet:

$$f(q) = \begin{cases} \infty & \text{falls } \alpha(q) = \square, \\ d_A(\alpha(q)) & \text{sonst} \end{cases}$$

Weil die Abbildung α die tatsächlichen Zustände in Äquivalenzklassen teilt ($p \sim q \Leftrightarrow \alpha(p) = \alpha(q)$), ist diese Abstandsheuristik konsistent (engl. consistent) für jede Wahl der Verschmelzungsstrategie.

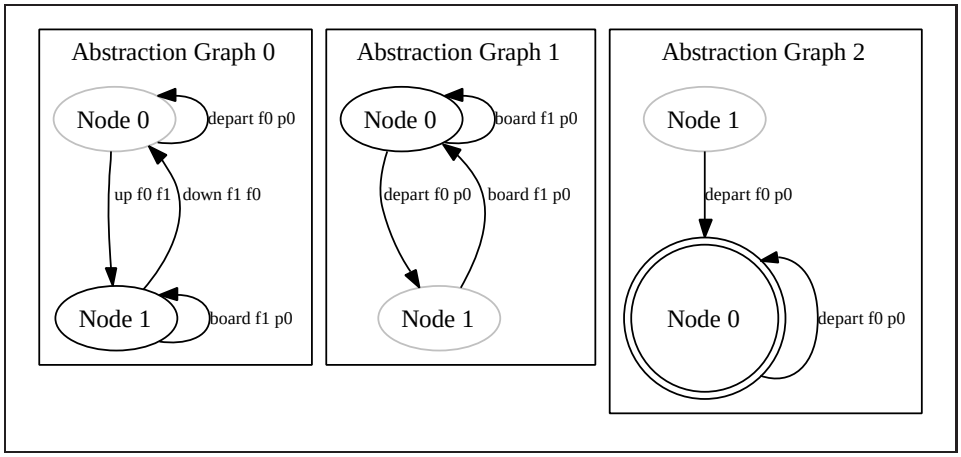


Abbildung 4.2: Die Abstraktionsgraphen, die aus den Domain Transition Graphs des MICONIC-Problems aus Abbildung 3.2 abgeleitet wurden. Initialknoten sind grau dargestellt. Der Zielknoten ist doppelt umrandet.

4.2.1 Aufbau der Abstraktionsgraphen

$AG(\mathcal{D})$ nimmt als Parameter die Menge der gegebenen Domain Transition Graphs und liefert eine (gleichmächtige) Menge von Abstraktionsgraphen. Abstraction Graphs unterscheiden sich von Domain Transitions Graphs auf zwei Arten:

1. *Abstraction Graphs haben keine unnötigen Knoten.* Zustandsknoten, die vom Initialknoten nicht erreichbar sind und Knoten, die keine Zielknoten erreichen können, sind nicht Teil eines Abstraktionsgraphen. Da sie bei der Verschmelzung von Graphen keinen Mehrwert an Information liefern, werden sie von der Funktion $AG()$ entfernt. Zustandsknoten, die durch die Verschmelzung von unnützen Knoten mit "nützlichen" Knoten entstanden sind, sind selbst wieder unnütz. Sie würden beim Abstraktionsschritt ohnehin entfernt werden.

2. *Abstraction Graphs können "Self Loops" haben.* Self Loops sind Transitionen, deren Zielknoten ihr Startknoten ist. Im Gegensatz zu Domain Transition Graphs, die diese Transitionen nicht explizit kodieren, haben Abstraction Graphs diese für die Verschmelzung wichtigen Kanten. Wichtig sind sie deshalb, weil die Zuordnung der Knoten an Vorbedingungen bei einem Verschmelzungsschritt verloren geht.

Das Eliminieren der überflüssigen Knoten und das Hinzufügen der Self Loops reicht also aus, um aus einem Domain Transition Graph einen Abstraction Graph zu machen.

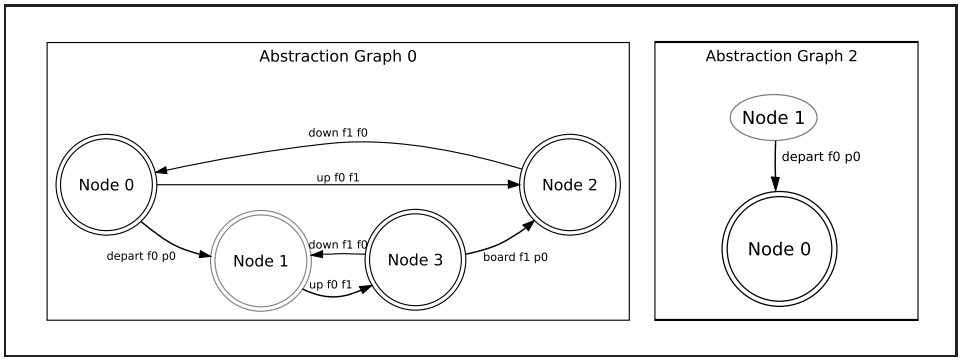


Abbildung 4.3: Das MICONIC-Problem nach dem ersten Abstraktionsschritt. AG 0 ist aus AG 0 und AG 1 aus der Abbildung 4.2 hervorgegangen.

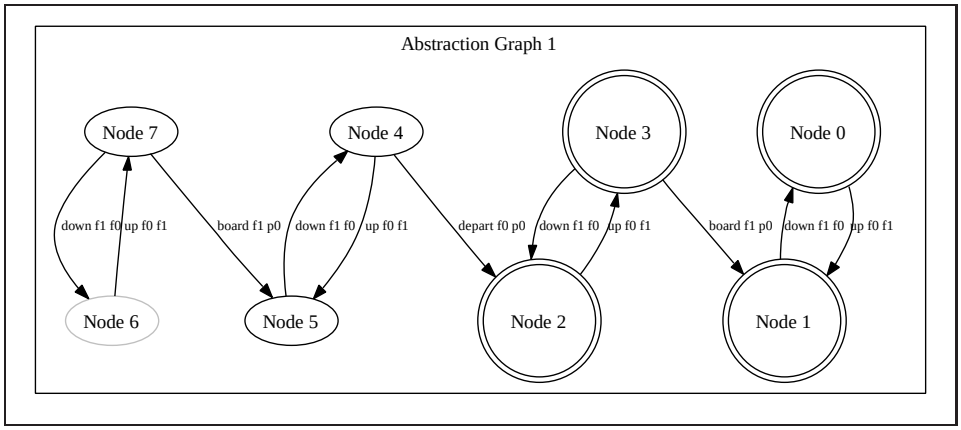


Abbildung 4.4: Das MICONIC-Problem nach dem zweiten Abstraktionsschritt, hervorgegangen aus den beiden AGs in Abbildung 4.3.

4.3 Teile des Abstraktionsprozesses

4.3.1 Berechnung des abstrakten Problems

Wie kann man sicherstellen, dass der Zielabstand eines abstrahierten Zustandes eine gute Näherung für den Abstand des assoziierten konkreten Zustandes ist? Eine Idee ist, für jeden konkreten Zustandsknoten einen entsprechenden abstrakten Zustandsknoten mit dem gleichen Abstand zum Ziel zu berechnen. Obwohl das den Zielabstand für die Knoten eines betrachteten Graphen nicht verändert, so ändert sich doch dessen Verhalten im nächsten "verschmelze und abstrahiere" Schritt. Das hat zur Folge, dass sich für den resultierenden Graphen die Zielabstände ändern.

Dräger, Finkbeiner und Podelski [2] schlagen eine Lösung vor, die auf zwei Ansätzen beruht.
Als erstes wird eine obere Grenze N für die maximale Anzahl von Zuständen eines Abstraktionsgraphen festgelegt. Sollte ein Abstraction Graph durch eine Verschmelzung mehr Zustandsknoten als N haben, und damit verkleinert werden müssen, so besteht die erste Priorität darin, sicherzustellen, dass nur Zustände mit dem gleichen Abstand zum nächsten Ziel vereinigt werden. Zweitens soll das Verhalten beim nächsten "Verschmelze und abstrahiere"-Schritt möglichst erhalten bleiben. Siehe dazu Algorithmus 2: AbstractGraph.

Als Teil des Initialisierungsprozesses entfernt der Algorithmus AbstractGraph irrelevante Knoten. Graph D' enthält nur noch Knoten, die vom Initialknoten zu erreichen sind und solche, von denen man einen Zielknoten erreichen kann.

Die Berechnung der Äquivalenzrelation $\sim$ beginnt damit Zustände paarweise zu bestimmen, wenn sie den gleichen Abstand zu ihrem nächsten Zielknoten haben. Während der gesamten Bearbeitung werden nur Verfeinerungen dieser Äquivalenzklassen in Betracht gezogen. Zuerst werden die Zustände in Teilmengen $B_0, \ldots, B_{N-1}$ aufgeteilt, entsprechend ihrer Abstände zu den Zielknoten. Über jede Teilmenge wird eine eigene Äquivalenzrelation definiert: $R_i = \sim \cap (B_i \times B_i)$, für $i = 1, \ldots, N-1$. Die anschließende Schleife unterteilt $\sim$, bis ein Fixpunkt erreicht ist. Für jede Relation R_i werden die Äquivalenzklassen in R_i entsprechend der Äquivalenzklassen ihrer Nachfolger in $\sim$ geteilt. Falls die verfeinerte Äquivalenz R_i^* die Anzahl der Äquivalenzklassen nicht über die obere Grenze N hebt, wird $\sim$ gemäß R_i^* geteilt. Die Teilmengen werden in aufsteigender Reihenfolge durchgegangen, angefangen mit B_0. Dies wurde von Dräger & Finkbeiner so festgelegt, weil sie der Ansicht waren, dass Zustände mit höheren Abständen zum Ziel Zustände mit niedrigeren Abständen auf ihrem Weg zum Ziel durchlaufen

Algorithmus 2 : AbstractGraph berechnet einen abstrakten Graphen für ein gegebenen, konkreten Graphen.

Eingabe : Abstraktionsgraph $D = (\Sigma, Q, q_0, Q_g, \rightarrow)$
Ausgabe : Abstraktionsgraph A
Ausgabe : Abbildung von konkreten zu abstrakten Zuständen:
$\alpha : Q \rightarrow (Q_A \cup \{\square\})$

begin
  `// Initialisierung`
  $Q' := \{q \in Q | d_D(q) < \infty$ und q ist von q_0 erreichbar$\}$;
  $D' := (\Sigma, Q', q_0 \cap Q', Q_g \cap Q', \rightarrow \cap (Q' \times \Sigma \times Q'))$;
  $\sim := \{(q, q') \in Q' \times Q' | min(d_D(q), N-1) = min(d_D(q'), N-1)\}$;
  $K := |Q'/\sim|$;
  `// Bearbeitungsschritt`
  repeat
    $\sim' := \sim$;
    for $i = 0, \ldots, K-1$ **do**
      $R_i^* := \{(q, q') \in R_i | \forall a \{[r]_\sim | q \xrightarrow{a} r\} = \{[r']_\sim | q' \xrightarrow{a} r'\}\}$;
      if $|Q'/(R_1 \cup \cdots \cup R_i^* \cup \cdots \cup R_K)| \leq N$ **then**
        $R_i := R_i^*$;
        $\sim := \bigcup_{i=0}^{K-1} R_i$;
      end
    end
  until $\sim = \sim'$;
  $A := P'/\sim$;
  $\alpha(q) := \begin{cases} [q]_\sim & \text{falls } q \in Q', \\ \square & \text{sonst} \end{cases}$
  return A, α
end

würden. Ungenauigkeiten bei Zuständen mit größerem Abstand zu einem Zielknoten würden folglich weniger Zustände beeinflussen als Ungenauigkeiten bei Zuständen mit geringerer Entfernung zum Ziel.
Ein Fixpunkt wird nach höchstens N Iterationen erreicht. Die Abstraktion wird dann als Quotient $P'/\sim$ berechnet. Die Funktion α bildet jeden relevanten tatsächlichen Zustand q auf seine Äquivalenzklasse $[q]_\sim$ ab.

4.3.2 Verschmelzung der Abstraktionsgraphen

Der Algorithmus AbstractGraph garantiert, dass die Abstände zu den Zielknoten für alle Knoten bei einem betrachteten Aufruf erhalten bleiben. Allerdings können sich Ungenauigkeiten ergeben, sobald der resultierende

Abstraktionsgraph mit weiteren Graphen verschmolzen wird. Das Ziel einer Verschmelzungsstrategie ist es, diese Ungenauigkeiten auf ein Minimum zu beschränken. Außerdem muss eine gute Strategie mit möglichst wenigen Reduktionen der Abstraktionsgraphen auskommen. Dadurch wird nicht nur der Verschmelzungsschritt beschleunigt, sondern es treten gleichzeitig auch weniger Ungenauigkeiten auf, die im später Verlauf Probleme bereiten könnten.

Es kann davon ausgegangen werden, dass die Wahl der zu verschmelzenden Graphen eine entscheidende Auswirkung auf die Performanz der Heuristik hat. In Kapitel 6 werden deswegen eine Reihe von verschiedenen Verschmelzungsstrategien miteinander verglichen.

Kapitel 5

Experimente

Alle Experimente wurden auf einem Rechner mit einer Intel® XEON™ 3.06 GHz CPU und 3GB Hauptspeicher ausgeführt. Alle Experimente haben eine Zeitbeschränkung von 20 Minuten.

Um die Güte einer Heuristik als Teil eines Planungssystems zu bestimmen bieten sich zwei Größen an:

1. Die Geschwindigkeit, mit der das Planungssystem eine repräsentative Auswahl an Benchmarkproblemen löst. Wobei in diesem Fall zu unterscheiden ist zwischen der Gesamtbearbeitungszeit, und der Zeit die benötigt wird um die Funktion $d(v)$ zu berechnen.

2. Als Gütekriterium für die heuristischen Werte selbst bietet es sich an, die Anzahl der untersuchten Knoten der Suchkomponente zu betrachten. Daraus lassen sich Rückschlüsse ziehen, wie genau die Werte sind, die die Heuristik liefert. Je geringer die Anzahl der untersuchten Knoten, desto genauer sind die heuristischen Werte.

Wenn nicht anders erwähnt, wurde als Obergrenze der Graphengröße $N = 128$ gewählt. Erste Betrachtungen ergaben, dass kleinere Zahlen für N im Allgemeinen keine Graphen liefern, die stabil genug für eine aussagekräftige Betrachtung sind. Größere Werte erhöhen im Gegensatz die Genauigkeit der heuristischen Werte nur wenig, so dass die Ausführungszeit leidet.

5.1 Problemstellungen

Um die Leistung der Implementation empirisch zu untersuchen, werden fünf ausgewählte Problemdomänen herangezogen. Diese sollen im Folgenden kurz

betrachtet und deren Parameter identifiziert werden.

Miconic

Diese Problemdomäne gibt es in drei Ausführungen: MICONIC, MICONIC-SIMPLEADL und MICONIC-FULLADL. Wir betrachten nur die erste der drei Domains.

Bei MICONIC muss ein Aufzug eine Menge von Passagieren von ihrem Startstockwerk zu einem Zielstockwerk befördern. Ist der Aufzug auf der Etage A, so können alle Passagiere, die auf der Ebene A auf den Aufzug warten einsteigen und alle Passagiere, deren Zieletage A ist, können aussteigen und sind damit *bedient* (engl. served).

Anfangs befindet sich der Aufzug auf einer beliebigen Etage und alle Passagiere befinden sich auf ihren Startstockwerken.

Das Ziel ist es, die Stockwerke mit dem Aufzug so anzufahren, dass alle Passagiere am Ende bedient sind. Hierbei variieren die Anzahl der Passagiere und die Anzahl der Stockwerke. Je höher die Anzahl beider, desto länger der Plan und desto schwieriger ist es eine Lösung zu finden.

Problem	Dateiname	Passagiere	Etagen
MIC1	s1-0.pddl	1	2
MIC2	s2-0.pddl	2	4
MIC3	s3-0.pddl	3	6
MIC4	s4-0.pddl	4	8
MIC5	s5-0.pddl	5	10
MIC6	s6-0.pddl	6	12
MIC7	s7-0.pddl	7	14
MIC8	s8-0.pddl	8	16
MIC9	s9-0.pddl	9	18
MIC10	s10-0.pddl	10	20

Gripper

Die GRIPPER-Domäne wurde ursprünglich erstellt, um die Grenzen von Graphplan aufzuzeigen.

In GRIPPER kann sich ein Roboter zwischen Raum A und Raum B hin und her bewegen. Er hat zwei Greifarme G_1 und G_2 mit denen er jeweils einen Ball, der sich im selben Raum befindet wie der Roboter greifen kann. Hat er einen Ball in G_1 oder G_2, so kann er ihn hinlegen. Der Ball ist fortan in dem Raum, indem der Roboter war, als er ihn hingelegte.

Anfänglich ist der Roboter in Raum A zusammen mit einer Anzahl von Bällen $B_1 \dots B_n$. Die Aufgabe ist nun diese Bälle nach Raum B zu transportieren. Die Anzahl der Bälle, die von Raum A nach Raum B gebracht werden müssen ist hierbei variabel. Je mehr Bälle bewegt werden müssen, desto länger ist der benötigte Plan.

Problem	Dateiname	Bälle
GRIP1	prob01.pddl	4
GRIP2	prob02.pddl	6
GRIP3	prob03.pddl	8
GRIP4	prob04.pddl	10
GRIP5	prob05.pddl	12
GRIP6	prob06.pddl	14
GRIP7	prob07.pddl	16
GRIP8	prob08.pddl	18
GRIP9	prob09.pddl	20
GRIP10	prob10.pddl	22

Zenotravel

ZENOTRAVEL wurde erstmals vorgestellt während der International Planning Competition 2002.

Diese Domäne hat Aktionen, mit denen man Passagiere in Flugzeuge ein- und aussteigen lassen kann, welche mit zwei unterschiedlichen Geschwindigkeiten ("fly" und "zoom") zwischen einzelnen Orten hin und her fliegen können.

Dabei variiert die Anzahl der zur Verfügung stehenden Flugzeuge, die Anzahl der verschiedenen Städte und die Anzahl der zu befördernden Fluggäste.

Die betrachtete STRIPS-Variante ist nicht so fordernd wie ihre numerischen oder temporalen Schwestern, denn es bietet nur Nachteile, nicht jedoch Vorteile die Passagiere mit höherer Geschwindigkeit ("zoom") zu befördern. Kluge Planer sollten diese Operatoren komplett vermeiden. Ansonsten bildet ZENOTRAVEL ein typisches Transportproblem ab.

Problem	Dateiname	Flugzeuge	Personen	Städte
ZENO1	pfile1	1	2	3
ZENO2	pfile2	1	3	3
ZENO3	pfile3	2	4	3
ZENO4	pfile4	2	5	3
ZENO5	pfile5	2	4	4
ZENO6	pfile6	2	5	4
ZENO7	pfile7	2	6	4
ZENO8	pfile8	3	6	4
ZENO9	pfile9	3	7	5
ZENO10	pfile10	3	8	5

Logistics

LOGISTICS ist eine Standardbenchmark im Bereich der Handlungsplanung. In dieser Domain gibt es eine Anzahl von Städten und in jeder Stadt eine Reihe von Plätzen. Das Ziel besteht darin, eine Menge von Paketen von ihrem Aufenthaltsort zu ihrem jeweiligen Zielort zu transportieren. Dazu stehen zwei verschiedene Arten von Fahrzeugen zur Verfügung: Lastwagen und Flugzeuge. Lastwagen können sich nur zwischen verschiedenen Orten in derselben Stadt bewegen, während Flugzeuge nur von Flughafen zu Flughafen fliegen können.
Hierbei variieren sowohl die Anzahl der Plätze je Ort, die Anzahl der unterschiedlichen Fahrzeuge, als auch die Anzahl der zustellbaren Pakete und die Anzahl der Flughäfen.

Problem	Dateiname	Locations	Fahrzeuge	Pakete
LOG1	probLOGISTICS-4-0.pddl	6	3	6
LOG2	probLOGISTICS-5-0.pddl	6	3	6
LOG3	probLOGISTICS-6-0.pddl	6	3	6
LOG4	probLOGISTICS-7-0.pddl	9	4	9
LOG5	probLOGISTICS-8-0.pddl	9	4	9
LOG6	probLOGISTICS-9-0.pddl	9	4	9
LOG7	probLOGISTICS-10-0.pddl	12	5	12
LOG8	probLOGISTICS-11-0.pddl	12	5	12
LOG9	probLOGISTICS-12-0.pddl	12	5	12
LOG10	probLOGISTICS-13-0.pddl	15	7	15

Satellite

SATELLITE entsprang ursprünglich einer Diskussion von David E. Smith und Jeremy Franks vom NASA Ames Research Center.

Die Probleme dieser Domäne sind vielschichtig und beinhalten, einen (oder auch mehrere) Satelliten zu benutzen, um eine Reihe von Beobachtungen zu machen, Daten zu sammeln und diese an eine Bodenstation zu übermitteln. Dazu haben die Satelliten eine Reihe von (möglicherweise redundanten) Instrumenten zur Verfügung. Jedes Instrument ist mit unterschiedlichen Charakteristika hinsichtlich der Datenermittlung, der Kalibrierung, des Energieverbrauchs und möglichen Ansprüchen für Vorwärm- und Abkühlzeiten ausgestattet. Die Satelliten können auf verschiedene Ziele ausgerichtet werden, jedoch können nicht notwendigerweise alle Satelliten auf alle Ziele ausgerichtet werden. Alle gesammelten Daten müssen vom Satelliten selbst gespeichert werden und zu gegebener Zeit (wenn sich ein Zeitfenster öffnet) zur Bodenstation übermittelt werden.

Hierbei können sich die Anzahl der Satelliten, die Menge und Art der Instrumente je Satellit, die Anzahl und Art der gewünschten Daten, sowie die Anzahl der Bodenstationen und zu untersuchenden Objekte ändern.

Die Schwierigkeit der STRIPS-Version des Problems liegt darin zu entscheiden wie die Datenermittlung am effizientesten in Angriff genommen wird, hinsichtlich der Fähigkeiten der einzelnen Satelliten.

Problem	Dateiname	Variablen
SATE1	p01-pfile1.pddl	12
SATE2	p02-pfile2.pddl	14
SATE3	p03-pfile3.pddl	17
SATE4	p04-pfile4.pddl	18
SATE5	p05-pfile5.pddl	25
SATE6	p06-pfile6.pddl	23
SATE7	p07-pfile7.pddl	28
SATE8	p08-pfile8.pddl	33
SATE9	p09-pfile9.pddl	36
SATE10	p10-pfile10.pddl	38

Kapitel 6

Verschmelzungsstrategien

6.1 Zufällige Verschmelzung

Diese Verschmelzungsart kann man kaum als "Strategie" bezeichnen. S wählt aus der Menge an Abstraktionsgraphen $\mathcal{D}'$ zwei zufällige Kandidaten aus, welche dann verschmolzen werden.

Funktion `S-random(`$\mathcal{D}$`)` wählt zwei Abstraction Graphs aus einer Menge von Abstraction Graphs.

Daten : Eine Menge von AGs $\mathcal{D} = \{D_1, \ldots, D_n\}$
Ergebnis : Ein Paar AGs gemäß der Strategie
begin
  G_1, G_2 =wähle zwei AGs zufällig aus D;
  return G_1, G_2
end

Interessant ist die zufällige Verschmelzung als Referenz für die anderen Verschmelzungsstrategien. Diese sollten nach Möglichkeit bessere Ergebnisse liefern als eine zufällige Auswahl der Graphen. Auch kann durch Zufall eine geschickte Reihenfolge der zu verschmelzenden Graphen zustande kommen und somit eine gute Laufzeit erzielt werden. Deshalb werden die Ergebnisse der benötigten Zeit für die Suche und die Gesamtzeit sowie für die zu öffnenden Zustände in jeweils drei Spalten angegeben. Eine für die beste Zeit bzw. geringste Anzahl der zu untersuchenden Knoten, eine für den Median über alle Durchläufe, und eine Spalte für die jeweils schlechtesten Werte. Es werden zehn Durchläufe pro Problem ausgewertet.

6.1.1 Ergebnisse

In die Tabelle aufgenommen wurden nur Probleme, die mindestens in 50% der Fälle lösbar waren. Sollte also der Planer bei ein und demselben Problem öfters als fünf Mal nicht zu einem Lösungsweg gelangen (durch timeout, oder aufgrund eines nicht gefundenen Lösungsweges) wurden die Ergebnisse verworfen. Die Probleme gelten als "nicht gelöst" und wurden nicht in die Tabelle eingetragen. Sollten weniger als 50% (jedoch mehr als 0%) der Lösungsversuche gescheitert sein, so wurde dies vermerkt.

Zu allererst sticht ins Auge, dass die Zeit, die die Suchkomponente mit der Suche nach einem Plan verbringt, nur einen sehr kleinen Anteil der Gesamtlaufzeit ausmacht, und das über alle Domänen hinweg. Das bedeutet, dass der Löwenanteil der Planungszeit für den Aufbau des abstrakten Problems benötigt wird. Die Berechnung der Zieldistanz eines Zustandes fällt hier nicht weiter ins Gewicht, da dies in dieser speziellen Implementierung durch einfaches "Nachschlagen" in zuvor angelegten Tabellen realisiert ist und daher in konstanter Zeit machbar ist. Ferner ist zu sehen, dass die zeitliche Beschränkung zwar eine Rolle spielt, jedoch oft dominiert wird von der Tatsache, dass der heuristische Algorithmus bei größeren Problemen keine hinreichenden Werte mehr liefert, so dass kein Zielzustand mehr gefunden werden kann. Die Abstraktionsgraphen "zerbrechen" an zu vielen Abstraktionsschritten. Eine Distanzberechnung kann nicht mehr stattfinden und der Algorithmus liefert nur noch unnütze Werte. Hierbei ist wichtig zu wissen, dass die Suchkomponente und speziell der Suchalgorithmus greedy best-first search keine Evaluation des betrachteten Problemzustands unternimmt. Das heißt, es wird nicht überprüft, ob der Zustand ein Zielzustand ist oder nicht, weil eine solche Bewertung im Allgemeinen von der Heuristik ohnehin durchgeführt wird. Ist die Heuristik allerdings nicht mehr im Stande eine solche Evaluation verlässlich durchzuführen, so kann die Suchkomponente ihren Dienst auch nicht mehr verlässlich erfüllen und sie geht davon aus, dass überhaupt kein Plan existiert, auch wenn das nicht zutreffend ist.

Wie zu erwarten war, steigt die Bearbeitungszeit mit steigendem Schwierigkeitsgrad der Probleme an. Überraschend ist in dieser Hinsicht nur die teilweise enorme Diskrepanz zwischen der schnellsten und langsamsten Planfindung eines Problems. Schon bei dem zweitleichtesten LOGISTICS-Problem liegt die schlechteste Bearbeitungszeit um ein fünfzig-faches höher als die beste Bearbeitungszeit. Dies bestärkt die Vermutung, dass eine gute Strategie zur Auswahl der zu verschmelzenden Abstraction Graphs unabdingbar für eine annehmbare Laufzeit ist.

Erfreulich ist, dass die gelieferten heuristischen Werte im Allgemeinen recht genau zu sein scheinen. Wie schon erwähnt verbringt der Planer nicht viel Zeit damit einen Lösungsweg zu suchen (für gewöhnlich $<5\%$ der Gesamt-

	T_{Suche} (Sekunden)			T_{Gesamt} (Sekunden)			Zustände		
Problem	min	med	max	min	med	max	min	med	max
LOG1	0.01	0.01	0.01	0.63	2.00	72.34	47	50	72
LOG2	0.01	0.01	0.01	2.46	19.49	132.48	93	106	137
LOG3	0.01	0.01	0.01	4.99	49.14	216.33	117	128	163
LOG4	0.01	0.01	0.12	8.72	99.30	328.90	210	248	2373
LOG5	0.01	0.01	0.02	95.31	220.80	389.04	169	223	456
LOG6	0.01	0.02	0.02	66.59	249.51	437.47	265	361	502
LOG7	0.02	0.03	0.05	156.09	409.30	890.31	434	642	996
MIC1	0.01	0.01	0.01	0.01	0.01	0.01	5	5	5
MIC2	0.01	0.01	0.01	0.01	0.01	0.01	14	14	14
MIC3	0.01	0.01	0.01	0.06	0.07	0.08	37	37	37
MIC4	0.01	0.01	0.01	0.40	2.80	3.09	42	42	111
MIC5	0.01	0.01	0.02	0.43	4.5	95.08	65	156	271
MIC6**	0.01	0.02	0.07	1.06	12.34	316.19	231	375	750
MIC7**	0.03	0.04	0.07	1.96	10.13	332.19	467	612	1039
MIC8**	0.02	0.05	0.09	6.16	12.26	94.74	413	651	953
MIC9**	0.07	0.11	1.67	13.52	277.49	500.86	799	1276	9074
MIC10**	0.16	0.23	1.25	15.23	132.04	429.02	1640	2463	6901
GRIP1	0.01	0.01	0.01	0.07	0.30	1.31	19	19	70
GRIP2	0.01	0.01	0.01	0.78	4.66	20.99	146	156	284
GRIP3	0.01	0.01	0.02	7.70	66.02	346.16	268	328	538
GRIP4	0.02	0.02	0.04	4.06	41.96	405.94	517	579	1199
GRIP5	0.03	0.04	0.05	11.90	69.68	598.95	928	937	1460
GRIP6	0.05	0.07	0.55	34.05	71.16	680.95	1255	1583	12057
GRIP7**	0.01	0.09	0.78	10.12	360.49	604.12	1756	1996	15446
GRIP8*/**	0.13	0.14	0.30	145.89	424.68	680.68	2743	2743	6942
GRIP9*/**	0.18	0.20	0.44	151.55	484.78	862.26	3652	3905	9033
GRIP10*/**	0.26	0.27	0.71	274.30	553.96	1148.41	4743	4745	12668
ZENO1	0.01	0.01	0.01	0.06	0.07	0.08	3	3	3
ZENO2	0.01	0.01	0.01	0.06	0.08	0.09	22	22	22
ZENO3	0.01	0.01	0.01	2.24	96.01	392.20	59	137	363
ZENO4	0.01	0.02	0.05	8.45	370.43	408.38	53	564	1503
ZENO5	0.01	0.01	0.04	9.39	168.41	1105.03	125	248	1119
ZENO6	0.01	0.01	0.21	77.62	193.71	662.11	183	408	4116
ZENO7	0.04	0.05	0.12	14.20	115.61	332.45	1041	1402	2680
ZENO8*/**	0.03	0.30	11.13	162.26	506.44	964.52	666	5663	166533
SATE1	0.01	0.01	0.01	0.04	0.05	0.05	36	36	36
SATE2	0.01	0.01	0.01	0.06	0.58	7.54	58	65	75
SATE3	0.01	0.02	0.06	1.76	22.09	156.02	209	367	917
SATE4*/**	0.01	0.21	0.46	11.27	257.77	652.51	251	2666	6775
SATE5*/**	2.38	3.10	3.91	28.08	350.42	798.40	29607	39628	45100
SATE6*/**	1.76	5.13	107.12	59.45	356.53	1121.78	20987	33027	249705

* Zeitüberschreitung; ** keine Lösung gefunden (bei manchen Instanzen)

Tabelle 6.1: Ergebnisse der Experimente für zufällige Verschmelzung. Vergleich der Zeit für die Plansuche (T_{Suche}), der Gesamtlaufzeit (T_{Gesamt}) und der Anzahl der expandierten Knoten (*Zustände*); jeweils Minimum (min), Median (med) und Maximum (max).

laufzeit). Betrachtet man die Anzahl der expandierten Zustände, so lässt sich erkennen, dass die Suchkomponente selbst für mittelschwere Probleme wie GRIP7 nur eine moderate Anzahl von möglichen Lösungswegen in Betracht zieht, was ein Indikator für recht genaue heuristische Werte ist.

6.2 Aufsteigende Sortierung der Graphengröße

Da der entscheidende Faktor für die Schnelligkeit des Aufbaus des abstrakten Problems die Graphgröße während des "verschmelze und abstrahiere"-Schritts zu sein scheint, scheint es nur natürlich zu versuchen die Graphen in den einzelnen Zwischenschritten möglichst klein zu halten. So sollte ein langsamer Anstieg der Graphengröße dafür sorgen, dass das abstrakte Problem schnell berechnet werden kann.

Die Idee hinter dieser Strategie ist, die Graphengröße möglichst langsam ansteigen zu lassen. Um das zu erreichen, werden immer die zwei Abstraktionsgraphen mit der geringsten Knotenanzahl miteinander verschmolzen. Der Ergebnisgraph wird dann wenn nötig verkleinert, um die Obergrenze N nicht zu überschreiten und anschließend wieder als kleinster möglicher Graph berücksichtigt.

Funktion `S-ascending(`$\mathcal{D}$`)` wählt zwei Abstraction Graphs aus einer Menge von Abstraction Graphs.

Daten : Eine Menge von AGs $\mathcal{D} = \{D_1, \ldots, D_n\}$
Ergebnis : Ein Paar AGs gemäß der Strategie
begin
  sort($\mathcal{D}$) ; `// sortiere aufsteigend gemäß Knotenanzahl`
  $G_1 = D_1 \in \mathcal{D}$;
  $G_2 = D_2 \in \mathcal{D}$;
  return G_1, G_2
end

6.2.1 Ergebnisse

Tabelle 6.2 zeigt die Ergebnisse dieser Strategie. Man kann deutlich erkennen, dass das Ziel die Berechnungszeit des abstrakten Problems zu minimieren nicht erreicht werden konnte. Die Laufzeiten des Planers sind über alle Probleme hinweg schlecht. Deutlich zu sehen ist das zum Beispiel an der LOGISTICS-Domäne, wo viele Laufzeiten sogar schlechter sind als die schlechtesten Laufzeiten der entsprechenden Probleme unter der randomisierten Verschmelzungsstrategie.

Problem	T_{such}	T_{gesamt}	# Knoten	Lösungsschritte
LOG1	0.01	13.29	56	20
LOG2	0.01	31.35	142	29
LOG3	0.01	190.47	161	25
LOG4	0.02	283.42	331	38
LOG5	0.01	359.49	323	35
LOG6	0.02	484.38	466	41
LOG7	0.03	1003.64	628	47
MIC1	0.01	0.01	5	4
MIC2	0.01	0.01	14	7
MIC3	0.01	0.09	29	10
MIC4	0.01	3.15	42	14
MIC5	0.01	24.41	279	20
MIC6	0.02	228.87	252	22
MIC7	0.05	385.11	810	32
MIC8	0.09	94.75	953	33
MIC9	0.07	75.54	795	36
MIC10	0.12	98.92	1119	42
GRIP1	0.01	1.18	34	13
GRIP2	0.01	35.29	156	19
GRIP3	0.01	308.49	249	25
GRIP4	0.02	544.10	413	29
GRIP5	0.03	633.97	930	47
GRIP6	**	**	**	**
GRIP7	0.09	1119.31	2023	61
GRIP8	0.13	326.51	16771	71
GRIP9	45.77	1168.12	677095	61
GRIP10	*	*	*	*
ZENO1	0.01	0.08	3	1
ZENO2	0.01	0.09	22	6
ZENO3	0.01	110.55	59	6
ZENO4	0.02	154.08	696	11
ZENO5	0.01	186.83	255	18
ZENO6	0.02	815.61	520	12
ZENO7	0.22	731.78	4433	20
ZENO8	**	**	**	**
SATE1	0.01	0.04	36	9
SATE2	0.01	7.69	58	13
SATE3	0.02	101.27	330	13
SATE4	0.06	842.52	1291	22
SATE5	3.30	164.99	42117	24
SATE6	3.46	557.56	19452	20
SATE7	6.81	481.15	70204	22
SATE8	*	*	*	*

* Zeitüberschreitung; ** keine Lösung gefunden

Tabelle 6.2: Ergebnisse der Experimente für aufsteigende Graphengröße.

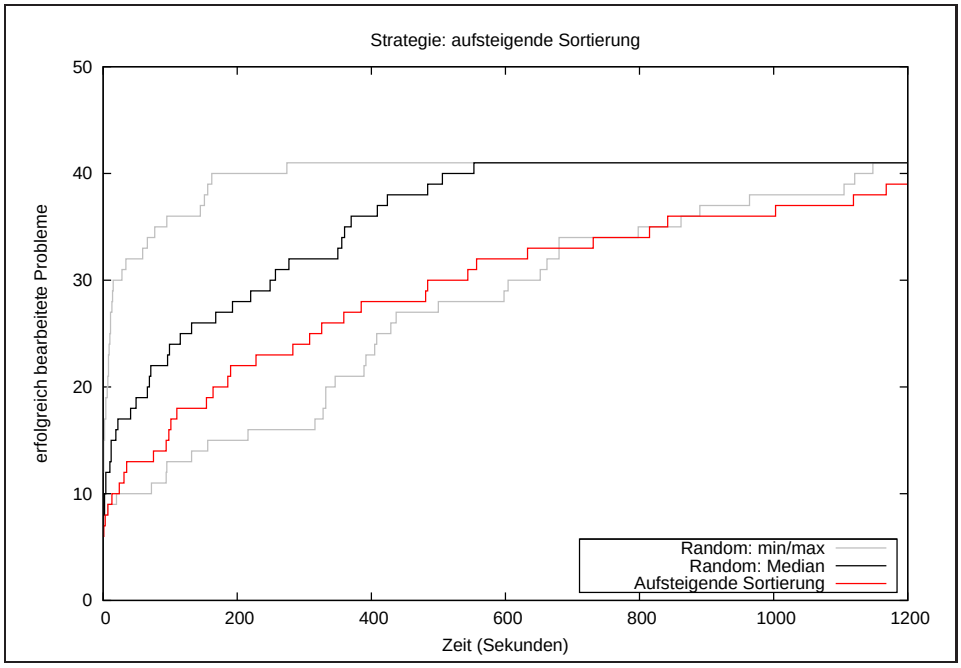


Abbildung 6.1: Anzahl der gelösten Probleme in Abhängigkeit der Zeit für die Strategie aufsteigende Sortierung.

Obgleich die Laufzeiten der einzelnen Probleme zu wünschen übrig lassen, so ist doch die Menge an gelösten Problemen positiv zu bewerten. Alles in allem konnten mehr Probleme gelöst werden als während der Zufallsstrategie. Allerdings ist auch hier ein Wermutstropfen zu finden. Das Problem GRIP6 konnte nicht erfolgreich bearbeitet werden. Aus keinem erkennbaren Grund versagte die Heuristik ihren Dienst, so dass greedy best-first search keinen Weg zu einem Zielknoten finden konnte.

Alle anderen heuristischen Werte scheinen vergleichsweise gut zu sein. Es wird im Schnitt kaum mehr Zeit mit der Suche verbracht als bei der randomisierten Verschmelzung der Abstraktionsgraphen.

6.3 Verschmelzung als “Liste”

Auf Grund der enttäuschenden Ergebnisse für die wiederholte Verschmelzung der kleinsten Graphen soll nun ein anderer Ansatz mit der gleichen Zielgebung betrachtet werden. Auch bei dieser Strategie ist die Idee, die Abstraktionsgraphen möglichst klein zu halten um eine schnelle Verschmelzung zu gewährleisten.

Bei dieser Verschmelzungsart werden zuerst zwei Abstraktionsgraphen, die

beide unmittelbar aus Domain Transition Graphs hervorgegangen sind, miteinander verschmolzen. Der resultierende Graph wird dann wiederholt mit einem noch nicht verschmolzenen Graphen gepaart, bis alle Graphen verschmolzen sind. Die einzelnen Abstraction Graphs werden wie in einer Liste nacheinander abgearbeitet. Da die anfänglichen Domain Transition Graphs im Allgemeinen keine große Anzahl an Zuständen haben wird damit erreicht, dass niemals zwei große Graphen (mit Anzahl der Zustände nahe N) miteinander kombiniert werden müssen. Damit werden Abstraktionsschritte, die einen Graphen mit zehntausend oder mehr Knoten verkleinern müssen, umgangen.

Funktion `S-list(`$\mathcal{D}$`)` wählt zwei Abstraction Graphs aus einer Menge von Abstraction Graphs.

Daten : Eine Menge von AGs $\mathcal{D} = \{D_1, \ldots, D_n\}$
Ergebnis : Ein Paar AGs gemäß der Strategie
begin
  A = get_complexAG($\mathcal{D}$) ; `// suche zusammengesetzten AG`
  if *exists(A)* **then**
    $G_1 = A$;
    $G_2 = D_1 \in \mathcal{D} \setminus A$;
  else
    $G_1 = D_1 \in \mathcal{D}$;
    $G_2 = D_2 \in \mathcal{D}$;
  end
  return G_1, G_2
end

6.3.1 Ergebnisse

Die Ergebnisse der empirischen Untersuchung wurden in Tabelle 6.3 eingetragen. Der Vergleich mit den Werten aus der Liste für die zufällige Verschmelzung zeigt deutlich wie leistungsstark diese Herangehensweise ist. Die Bearbeitungszeit der einzelnen Probleme unterbietet die besten Werte für die randomisierte Verschmelzung meist deutlich. Teilweise sind die Bearbeitungszeiten um den Faktor fünf kleiner als bei den anderen Strategien.

Dabei bleiben die außergewöhnlich guten heuristischen Abstandswerte der anderen Strategien erhalten. Berücksichtigt man die theoretische Anzahl der Zustände[1] von $1.79 * 10^8$ für das GRIPPER Problem GRIP5, muten die 1076 betrachteten Zustände verschwindend gering an. Und das zu Recht, bilden sie doch gerade mal 0.0006 % aller möglichen Zustände.

[1] Nicht alle Zustände sind vom Initialzustand aus erreichbar.

Problem	T_{such}	T_{gesamt}	# Knoten	Lösungsschritte
LOG1	0.01	0.72	56	20
LOG2	0.01	1.35	103	27
LOG3	0.01	1.54	120	25
LOG4	0.01	6.79	287	38
LOG5	0.01	7.67	301	35
LOG6	0.01	10.13	284	36
LOG7	0.02	21.30	657	48
LOG8	**	**	**	**
MIC1	0.01	0.01	5	4
MIC2	0.01	0.01	14	7
MIC3	0.01	0.08	37	12
MIC4	0.01	0.27	120	16
MIC5	0.01	0.45	181	18
MIC6	0.02	0.96	332	21
MIC7	**	**	**	**
GRIP1	0.01	0.09	24	11
GRIP2	0.01	0.29	231	19
GRIP3	0.02	1.64	399	31
GRIP4	0.02	4.40	679	39
GRIP5	0.04	14.06	1076	47
GRIP6	0.06	34.05	1397	55
GRIP7	0.09	36.62	2251	63
GRIP8	0.14	43.74	3074	71
GRIP9	0.20	52.78	4066	79
GRIP10	0.27	37.69	5250	87
ZENO1	0.01	0.07	3	1
ZENO2	0.01	0.08	22	6
ZENO3	0.01	1.98	71	8
ZENO4	0.01	1.08	136	12
ZENO5	**	**	**	**
SATE1	0.01	0.06	36	9
SATE2	0.01	0.28	72	13
SATE3	0.02	1.64	362	11
SATE4	0.08	35.17	1397	20
SATE5	**	**	**	**

* Zeitüberschreitung; ** keine Lösung gefunden

Tabelle 6.3: Ergebnisse der Experimente für "Verschmelzung als Liste" (N=128).

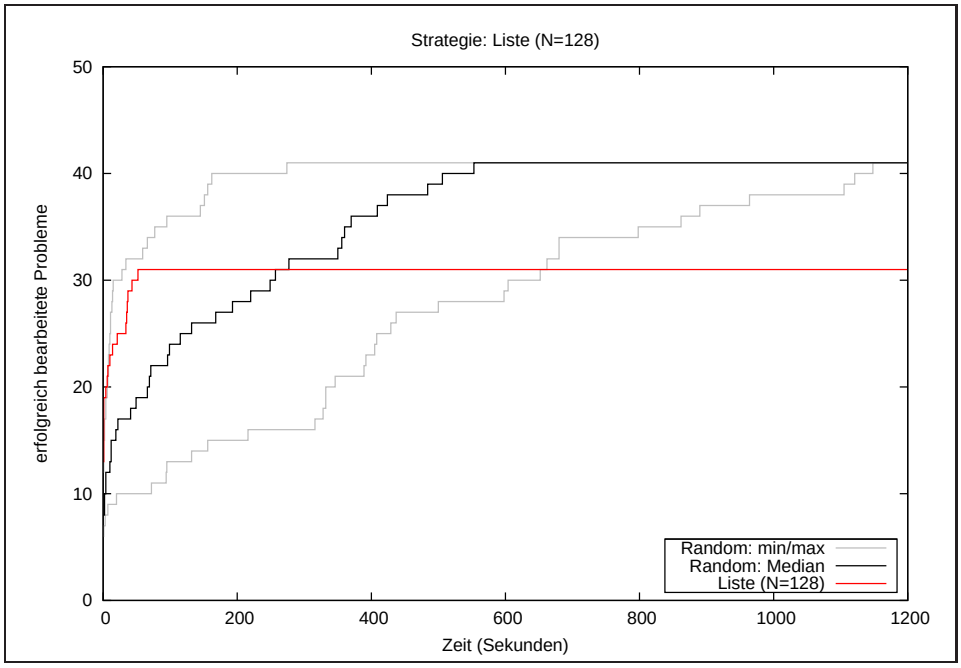


Abbildung 6.2: Anzahl der gelösten Probleme in Abhängigkeit der Zeit für die Strategie Liste (N-Wert 128).

Leider fällt auch auf, dass bei einigen Problemen (z.B. ZENOTRAVEL und SATELLITE) schon die mittelschweren Probleme nicht mehr erfolgreich gelöst werden konnten. Schuld daran ist die sehr asymmetrische Problemgeometrie dieser Domänen. Da dieser Ansatz die aus den Domain Transition Graphs abgeleiteten Abstraction Graphs in der Reihenfolge berücksichtigt, in der sie erschaffen wurden, kommt es bei diesen Problemen dazu, dass Graphen mit einander verschmolzen werden, deren Informationsgehalte so weit auseinander liegen, dass der anschließende Abstraktionsschritt den Graphen fast komplett demontiert. Dadurch gehen sehr viele Informationen verloren, so dass der resultierende Graph nicht mehr genug Knoten enthält um eine sinnvolle Abstandsheuristik zu liefern. Dem kann eventuell durch eine Anhebung von N entgegengesteuert werden.

6.4 Größenanpassung für N

Um es der Strategie Verschmelzung als “Liste” zu ermöglichen, eine größere Anzahl an Problemen erfolgreich zu bearbeiten, wird hier die Obergrenze der maximalen Graphengröße nach einem Abstraktionsschritt N angehoben. Es werden zwei weitere Durchläufe der Strategie begangen, einmal mit $N = 256$ und einmal mit $N = 512$. Da durch die Anhebung von N nun größere Zwi-

schenergebnisse zu erwarten sind, muss mit einer Verschlechterung der Laufzeit aller Probleme (mit Ausnahme der aller einfachsten) gerechnet werden. Hoffentlich wiegen die Vorteile diesen Nachteil auf. Es wird auch interessant zu sehen sein, wieviele Probleme durch eine Verdoppelung von N zusätzlich gelöst werden können.

6.4.1 Ergebnisse

Die Ergebnisse für $N = 256$ und $N = 512$ finden sich in Tabelle 6.4 und Tabelle 6.5, respektive. Wie zu erkennen ist, leidet die Ausführungszeit beinahe aller Probleme unter der Anhebung der maximalen Graphengröße N. So wurden für die zehn GRIPPER Probleme bei $N = 128$ etwas weniger als 4 Minuten benötigt, für $N = 256$ wurden schon etwa 13 Minuten benötigt und für $N = 512$ wurde mehr als 50 Minuten mit dem Finden einer Lösung verbracht.

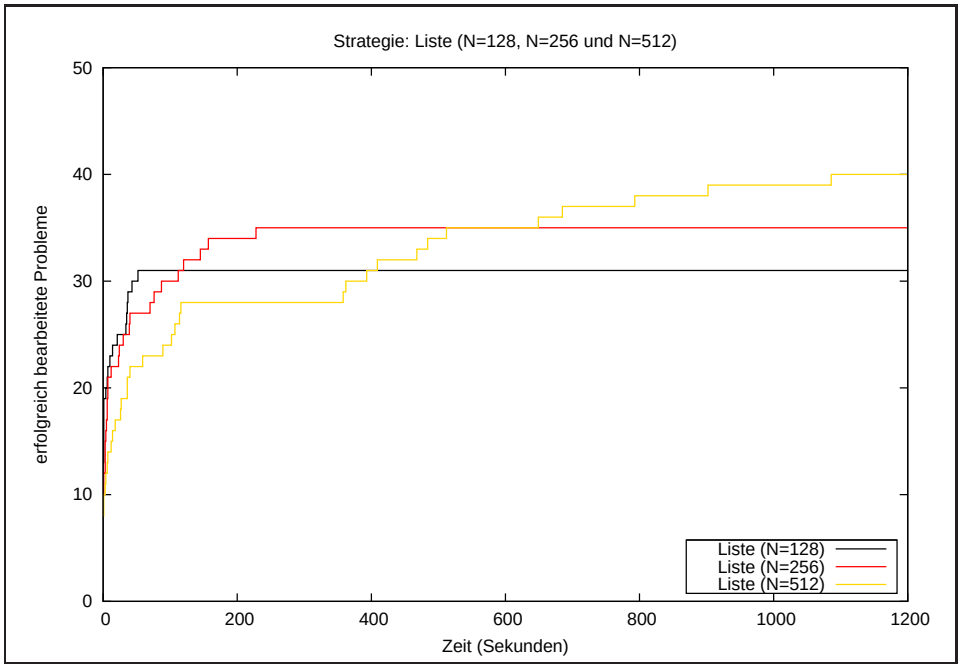


Abbildung 6.3: Anzahl der gelösten Probleme in Abhängigkeit der Zeit für die Strategie Liste mit den N-Werten 128, 256 und 512.

Problem	T_{such}	T_{gesamt}	# Knoten	Lösungsschritte
LOG1	0.01	3.30	47	20
LOG2	0.01	4.18	98	27
LOG3	0.01	6.13	120	25
LOG4	0.01	23.38	384	40
LOG5	0.01	24.59	392	36
LOG6	0.01	30.68	390	42
LOG7	0.02	76.61	761	47
LOG8	0.04	87.91	832	54
LOG9	**	**	**	**
MIC1	0.01	0.01	5	4
MIC2	0.01	0.01	14	7
MIC3	0.01	0.08	29	10
MIC4	0.01	0.49	77	16
MIC5	0.01	1.68	191	21
MIC6	0.02	3.34	359	21
MIC7	0.04	5.46	519	27
MIC8	0.09	6.54	1157	31
MIC9	0.17	12.64	1702	37
MIC10	**	**	**	**
GRIP1	0.01	0.08	19	11
GRIP2	0.02	0.71	384	17
GRIP3	0.17	1.65	3654	25
GRIP4	1.59	3.78	28072	29
GRIP5	0.04	40.10	1076	47
GRIP6	0.03	112.70	857	41
GRIP7	0.10	120.81	2251	63
GRIP8	0.14	145.90	2743	71
GRIP9	0.20	228.01	4066	79
GRIP10	0.27	157.36	5250	87
ZENO1	0.01	0.05	3	1
ZENO2	0.01	0.04	22	6
ZENO3	0.01	7.38	105	6
ZENO4	0.01	7.19	203	11
ZENO5	**	**	**	**
SATE1	0.01	0.04	36	9
SATE2	0.01	0.75	66	13
SATE3	0.04	70.76	603	11
SATE4	0.19	39.27	2029	20
SATE5	**	**	**	**

* Zeitüberschreitung; ** keine Lösung gefunden

Tabelle 6.4: Ergebnisse der Experimente für “Verschmelzung als Liste” (N=256).

Problem	T_{such}	T_{gesamt}	# Knoten	Lösungsschritte
LOG1	0.01	6.78	47	20
LOG2	0.01	14.40	99	27
LOG3	0.01	27.22	117	25
LOG4	0.01	107.23	238	38
LOG5	0.01	114.94	251	34
LOG6	0.02	116.89	440	43
LOG7	0.03	358.32	615	47
LOG8	0.04	362.09	718	52
LOG9	0.03	409.60	539	43
LOG10	**	**	**	**
MIC1	0.01	0.01	5	4
MIC2	0.01	0.01	14	7
MIC3	0.01	0.03	29	10
MIC4	0.01	0.86	63	15
MIC5	0.01	4.57	136	19
MIC6	0.01	12.21	276	24
MIC7	0.04	18.96	531	27
MIC8	**	**	**	**
MIC9	**	**	**	**
MIC10	0.22	36.21	2096	38
GRIP1	0.01	0.06	19	11
GRIP2	0.01	1.22	123	19
GRIP3	0.09	3.22	2232	25
GRIP4	2.08	7.02	34899	29
GRIP5	0.04	40.10	1076	47
GRIP6	76.72	89.25	1006165	45
GRIP7	0.09	512.30	2251	63
GRIP8	0.14	685.49	3074	71
GRIP9	0.20	902.42	4066	79
GRIP10	0.27	793.70	5250	87
ZENO1	0.01	0.02	3	1
ZENO2	0.01	0.02	22	6
ZENO3	0.01	26.91	170	9
ZENO4	0.01	36.17	72	9
ZENO5	**	**	**	**
ZENO7	0.02	102.00	723	18
ZENO8	0.02	393.34	546	16
ZENO9	0.69	468.18	11675	31
ZENO10	0.06	484.27	1434	31
SATE1	0.01	0.02	36	9
SATE2	0.01	1.68	65	13
SATE3	0.04	59.86	793	11
SATE4	**	**	**	**
SATE5	3.61	649.12	20366	17
SATE6	5.92	1086.21	21916	20
SATE7	*	*	*	*

* Zeitüberschreitung; ** keine Lösung gefunden

Tabelle 6.5: Ergebnisse der Experimente für “Verschmelzung als Liste” (N=512).

Die folgende Tabelle zeigt die Anzahl der gelösten Probleme in Abhängigkeit der gewählten Größe N:

N	LOG	MIC	GRIP	ZENO	SATE	Σ
128	7	6	10	4	4	31
256	8	9	10	4	4	35
512	9	8	10	8	5	40

Es wird deutlich, dass die Erhöhung von N tatsächlich den gewünschten Effekt hat und bei zunehmender Größe mehr Probleme erfolgreich gelöst werden können. Die Kosten sind allerdings recht hoch. So steigt die Bearbeitungszeit für alle Probleme an, was sich stark bemerkbar macht. Auch wurde N bei jeder Erhöhung verdoppelt, was aber leider keine Verdoppelung der gemeisterten Probleme nach sich zog. Die Summe der gelösten Probleme steigt nur linear. Es ist sogar so, dass manche Probleme, die unter $N = 256$ noch gelöst werden konnten, unter $N = 512$ nicht mehr gelöst wurden. Vermutlich ist dafür eine durch die höhere Anzahl von Teilmengen $B_1 \dots B_n$, ungünstige Einteilung der Zustandsknoten gemäß der Äquivalenzrelation R_i verantwortlich.

Die Anzahl der expandierten Knoten bleibt zum letzten Durchlauf mit $N = 256$ beinahe konstant. Es gibt eine Ausnahme: Bei dem Problem GRIP6 wurde quasi der gesamte Zustandsraum erforscht, um eine Lösung zu finden.

6.5 Aktionsbezogene Verschmelzungsstrategie

Bisher wurden Verschmelzungsstrategien betrachtet, die die Aufgabe hatten, die Größe der verschmolzenen Abstraktionsgraphen möglichst klein zu halten. Damit sollte eine geringe Laufzeit des Vorverarbeitungsschritts *Knowledge Compilation* erreicht werden. Um stabilere heuristische Werte zu erlangen und das "Zerbrechen" der Graphen zu verhindern, wurde dann die maximale Knotenzahl N erhöht. Eine alternative Herangehensweise ist, die Graphen nicht nach ihrer Knotenanzahl zu wählen, sondern hinsichtlich dessen, welche Information sie repräsentieren.

Bei dieser Strategie wird versucht immer diejenigen zwei Graphen miteinander zu verschmelzen, die die meisten gemeinsamen Aktionen aufweisen. Dadurch soll verhindert werden, dass Graphen verschmolzen werden, die von ihrer repräsentierten Information orthogonal zu einander stehen. Dadurch erhält der resultierende Abstraktionsgraph den höchstmöglichen Informationsgehalt.

Zu erwarten sind also nicht unbedingt schnellere Laufzeiten, sondern vielmehr viele gelöste Probleme.

Funktion `S-Aktion(`$\mathcal{D}$`)` wählt zwei Abstraction Graphs aus einer Menge von Abstraction Graphs.

Daten : Eine Menge von AGs $\mathcal{D} = \{D_1, \ldots, D_n\}$
Ergebnis : Ein Paar AGs gemäß der Strategie
begin
  max = 0;
  for $i = 0; i < |\mathcal{D}| - 1$ **do**
    for $j = i + 1; j < |\mathcal{D}|$ **do**
      if $max < |gemeinsame\ Aktionen(D_i, D_j)|$ **then**
        $max = |$gemeinsame Aktionen$(D_i, D_j)|$;
        $G_1 = D_i$;
        $G_2 = D_j$;
      end
    end
  end
  return G_1, G_2
end

6.5.1 Ergebnisse

Die Werte in Tabelle 6.6 bestätigen die Voraussagen teilweise. So konnten viele Probleme erfolgreich bearbeitet werden, bei recht ordentlicher Gesamtzeit. Die Abbildung 6.4 zeigt auch das recht ordentliche Abschneiden dieser Strategie.

Kurios ist das Verhalten des Planers bei der MICONIC-Domäne. Sie konnte nur zu einem kleinen Teil gelöst werden.

Erklären lässt sich das durch die spezielle Auswahl der Graphen durch die angewandte Strategie. Sie sorgt dafür, dass der Graph, der den Zustand des Aufzugs kodiert nacheinander mit allen anderen Graphen verschmolzen wird, weil er mehr, oder zumindest gleichviele, Aktionen mit den Graphen teilt, wie diese untereinander. Dies ähnelt er Strategie "Verschmelzung als Liste", die es in dieser Domäne nicht einmal mit einem N-Wert von 512 geschafft hat, alle MICONIC-Probleme zu lösen.

6.6 Vergleiche

Um die einzelnen Strategien sinnvoll miteinander vergleichen zu können, ist es zunächst wichtig festzuhalten, unter welchen Gesichtspunkten die Strategien miteinander verglichen werden sollen.

Problem	T_{such}	T_{gesamt}	# Knoten	Lösungsschritte
LOG1	0.01	0.74	68	20
LOG2	0.01	0.98	107	27
LOG3	0.01	1.66	120	25
LOG4	0.01	6.08	289	39
LOG5	0.01	6.78	290	33
LOG6	0.02	7.83	398	41
LOG7	0.03	20.80	642	47
LOG8	0.04	22.86	631	49
LOG9	0.05	28.23	881	52
LOG10	**	**	**	**
MIC1	0.01	0.01	5	4
MIC2	0.01	0.01	14	7
MIC3	0.01	0.06	37	12
MIC4	0.01	0.16	155	16
MIC5	**	**	**	**
GRIP1	0.01	0.07	24	11
GRIP2	0.01	0.26	145	17
GRIP3	0.01	1.58	399	31
GRIP4	0.03	4.48	687	39
GRIP5	0.04	13.48	1076	47
GRIP6	0.03	33.19	857	41
GRIP7	0.09	35.82	2251	63
GRIP8	0.15	46.66	3074	71
GRIP9	0.19	52.31	4066	79
GRIP10	0.26	36.57	5250	87
ZENO1	0.01	0.05	3	1
ZENO2	0.01	0.05	22	6
ZENO3	0.01	11.48	123	9
ZENO4	0.02	11.44	629	11
ZENO5	0.01	17.72	267	13
ZENO6	0.02	20.34	523	13
ZENO7	0.02	31.48	643	20
ZENO8	0.02	157.31	666	16
ZENO9	**	**	**	**
ZENO10	5.92	171.98	86462	33
SATE1	0.01	0.04	36	9
SATE2	0.01	0.24	180	17
SATE3	0.03	74.28	793	11
SATE4	**	**	**	**
SATE5	**	**	**	**
SATE6	18.40	37.61	57284	23
SATE7	27.30	804.01	266302	26
SATE8	610.30	695.60	1268128	28

* Zeitüberschreitung; ** keine Lösung gefunden

Tabelle 6.6: Ergebnisse der Experimente für die aktionsbezogene Verschmelzungsstrategie.

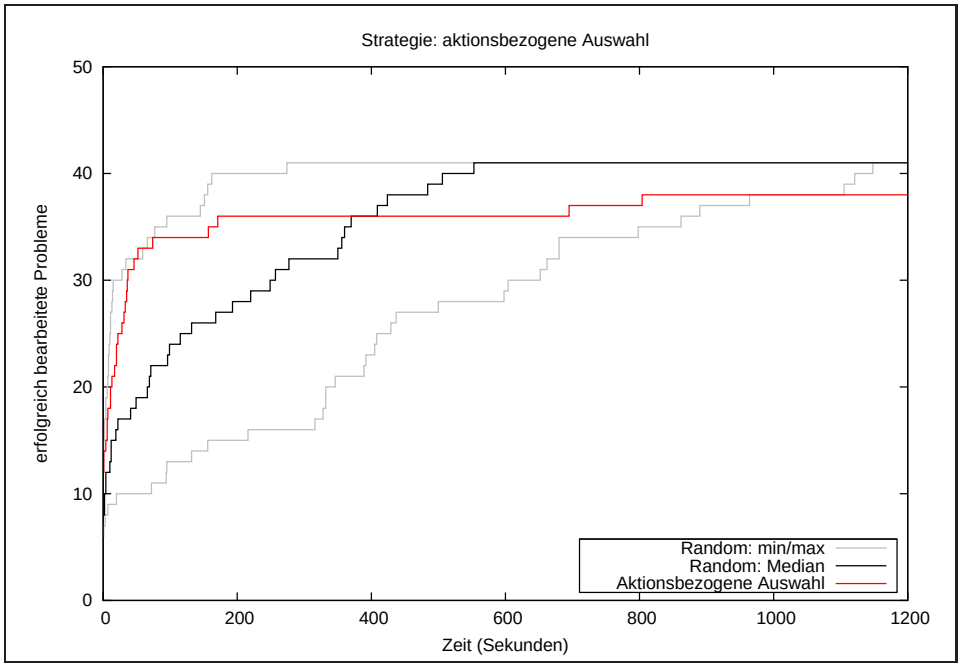


Abbildung 6.4: Anzahl der gelösten Probleme in Abhängigkeit der Zeit für die Strategie aktionsbezogene Auswahl.

- *Die Gesamtlaufzeit.* Dieses Kriterium ist natürlich wichtig wenn man einen möglichst schnellen Planer haben will. Je geringer die Laufzeit, desto besser sorgt die Verschmelzungsstrategie dafür, dass nur Graphen miteinander verschmolzen werden, die keine extremen Abstraktionsschritte nach sich ziehen.

- *Die Anzahl der erfolgreich bearbeiteten Probleme.* Natürlich sind Planer gefragt, die möglichst viele verschiedene Problemdomänen beherrschen und möglichst viele Probleme in jeder Domäne lösen können.

- *Güte der heuristischen Werte.* Wie genau eine Heuristik den Abstand eines Zustandes zum nächsten Zielzustand schätzt, ist ausschlaggebend dafür, wie schnell die Suchkomponente einen validen Plan findet. Gute heuristische Werte sind also die Grundvoraussetzung für einen schnellen heuristischen Planer. Allerdings zeigten die empirischen Untersuchungen, dass die heuristischen Werte über alle Strategien und Problemdomänen hinweg hervorragend zu sein scheinen. Ein Vergleich der minimalen Abweichungen scheint daher nicht sehr sinnvoll und wird aus diesem Grund hier nicht weiter betrachtet.

Gesamtlaufzeit

Hier sollen die Laufzeiten des Planers über die einzelnen Verschmelzungsstrategien betrachtet werden. Dafür wird der Wert der durchschnittlichen Laufzeit für eine gegebene Problemdomäne $\mathcal{D}$ wie folgt berechnet:

$$T_{\varnothing}(\mathcal{D}) = \frac{\sum^{d_i \in \mathcal{D}} T_{gesamt}(d_i)}{\text{Anzahl der gelösten Probleme in } \mathcal{D}}$$

Die Ergebnisse wurden in folgende Tabelle eingetragen. Der Vollständigkeit halber werden die Ergebnisse der Strategie der zufälligen Verschmelzung mit den Medianwerten ebenfalls aufgeführt.

	$T_{\varnothing}(LOG)$	$T_{\varnothing}(MIC)$	$T_{\varnothing}(GRIP)$	$T_{\varnothing}(ZENO)$	$T_{\varnothing}(SATE)$	Σ
Rand_{med}	149.93	45.17	207.77	181.35	164.57	748.79
Sort	338.01	91.09	517.12	285.57	307.89	1539.68
$\text{List}_{N=128}$	7.07	0.30	22.54	0.80	9.29	40.00
$\text{List}_{N=256}$	32.10	3.36	70.31	3.67	27.71	137.15
$\text{List}_{N=512}$	168.61	9.11	303.48	188.86	359.38	1029.44
Aktion	10.66	0.06	22.44	64.87	273.25	371.28

Es ist zu erkennen, dass die Strategie "Sort" die mit Abstand schlechtesten Laufzeiten liefert. Die Durchschnittslaufzeiten der Listen-Strategien steigen mit steigendem N stark an. Dies hat zwei Gründe: zum einen steigt mit größerem N die Laufzeit der kleinen Probleme an, zum anderen aber lassen sich mit höheren N-Werten auch schwierigere Probleme lösen, die von sich aus eine höhere Bearbeitungszeit erfordern. Beides zusammen drückt die durchschnittliche Ausführungszeit nach unten. Für sich alleine genommen sind diese Werte also nicht sehr aussagekräftig, da sie die Strategien bevorzugen, die nur die leichteren Probleme lösen können, nicht aber die schwereren, die natürlich mehr Zeit benötigen.

Anzahl der gelösten Probleme

Wie viel Probleme eine Strategie zu lösen im Stande war, wird auf der folgenden Tabelle festgehalten. Für die zufällige Verschmelzung wurden alle Ergebnisse herangezogen, die auch in Tabelle 6.1 eingeflossen sind.

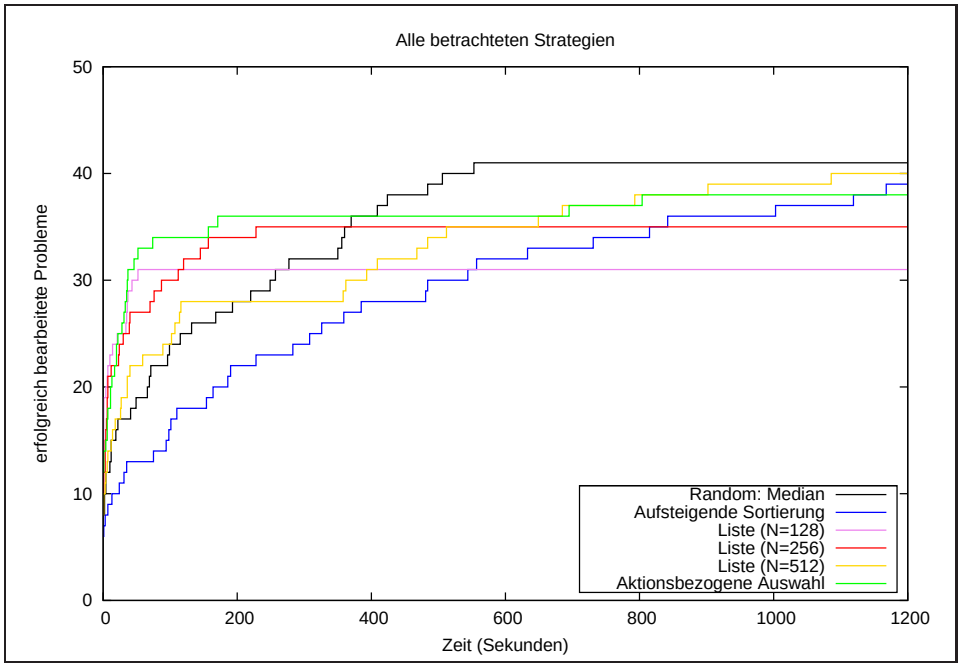


Abbildung 6.5: Anzahl der gelösten Probleme in Abhängigkeit der Zeit für alle betrachteten Strategien.

	LOG	MIC	GRIP	ZENO	SATE	Σ
Rand_{med}	7	10	10	8	6	41
Sort	7	10	8	7	7	39
$\text{List}_{N=128}$	7	6	10	4	4	31
$\text{List}_{N=256}$	8	9	10	4	4	35
$\text{List}_{N=512}$	9	8	10	8	5	40
Aktion	9	4	10	9	6	38

Obgleich Sort die schlechtesten Laufzeiten von allen betrachteten Strategien hat, so konnte mit ihr doch eine erkleckliche Anzahl an Problemen gelöst werden. Die schlechten Laufzeiten rechtfertigen jedoch keine weiteren Untersuchungen dieser Strategie.

Besser sieht es bei einer listenhaften Verschmelzung aus. Obwohl ein hohes N benötigt wird, um viele Probleme zu lösen, so können doch die Laufzeiten im Vergleich zur Konkurrenz überzeugen. So ist $\text{List}_{N=512}$ bei allen Domains (mit Ausnahme der SATELLITE Domain) wesentlich schneller als die Strategie der aufsteigenden Sortierung.

Der beste Kompromiss zwischen guter Laufzeit und der Menge an bearbeiteten Problemen scheint jedoch die aktionsbezogene Strategie zu sein. Sie liefert vernünftige Laufzeiten und bearbeitet sehr viele Probleme erfolgreich.

Kapitel 7

Die Konkurrenz

7.1 Vergleich mit Fast Forward & Fast Downward

Wie ist die heuristische Berechnung mit Hilfe eines automatentheoretischen Konzepts im Hinblick auf die Forschungslage einzuordnen? Ein Vergleich mit den beiden sehr erfolgreichen Fast-Forward-[1] und Fast-Downward-[2]Planern soll diese Frage beantworten.

Problem	FF	FD	DF
MIC1	0.01	0.01	0.01
MIC2	0.01	0.03	0.01
MIC3	0.01	0.05	0.08
MIC4	0.01	0.06	0.49
MIC5	0.01	0.06	1.68
MIC6	0.01	0.05	3.34
MIC7	0.01	0.07	5.46
MIC8	0.01	0.10	6.54
MIC9	0.01	0.11	12.64

Tabelle 7.1: Die ernüchternden Vergleiche mit der Weltspitze. Lösungszeit in Sekunden für Fast Forward (FF), Fast Downward mit Causal-Graph-Heuristik (FD) und der Implementierung der Dräger-Finkbeiner-Heuristik mit Verschmelzungsstrategie "Liste" und $N = 128$ (DF).

Da die Implementierung der Heuristikberechnung im Rahmen dieser Arbeit keine besonders hohe Optimierung erfahren hat, ist ein Vergleich mit der

[1] Gewinner der 2^{nd} *International Planning Competition.*
[2] Gewinner der 4^{th} *International Planning Competition.*

Spitze der heutigen Entwicklung nur als Referenz des Möglichen zu betrachten.

Stellvertretend für viele seien hier die Ausführungszeiten für die MICONIC-Probleme MIC1 bis MIC9 angegeben.

Wie nicht anders zu erwarten war, schlagen die beiden Spitzenplaner das in dieser Arbeit betrachtete Verfahren um Längen. Anzumerken ist hier noch, dass die Causal-Graph-Heuristik bei den MICONIC-Problemen nicht die beste Performanz liefert, die der Fast-Downward-Planer zu liefern imstande ist.

Obwohl die vorgestellte Heuristik in diesem ersten Vergleich von der Konkurrenz in ihre Schranken gewiesen wird, so ist sie dennoch nicht so weit abgeschlagen, dass mit etwas mehr Feintuning und einer etwas optimierteren Implementation nicht doch noch etwas näher an die Spitze heranzukommen wäre. Im nächsten Kapitel werden noch ein paar Ideen zur Verbesserung der Laufzeit angeschnitten.

Kapitel 8

Zusammenfassung und Ausblick

Es konnte gezeigt werden, dass es möglich ist die von Dräger und Finkbeiner entwickelte Heuristik zur Berechnung von Fehlerabständen effektiv für die Abschätzung der Zieldistanz eines gegebenen Zustandes in Planungsproblemen zu benutzen. Mit der Heuristik kann ein Suchalgorithmus wie greedy best-first search effizient im Problemraum nach einem Plan suchen und schnell eine Lösung finden. Die herausragende Stellung der Planer Fast Forward und Fast Downward ist nicht gefährdet.

Abschließend kann man sagen, dass die betrachtete automatentheoretische Heuristik einen äußerst interessanten Ansatz zur Berechnung von Zielabständen darstellt. Obgleich noch nicht konkurrenzfähig mit Spitzenheuristiken wie der Causal-Graph-Heuristik, welche im Fast Downward Planer Verwendung findet, lassen sich – mal abgesehen von einer optimierteren Implementation – einige Teile isolieren, die noch weiter verfeinert werden können. Zwei dieser Ideen werden im Folgenden kurz angerissen.

Dynamische Werte für N

Während der Experimente wurde nur ein fester Wert für N in Betracht gezogen. Höhere Werte hatten Vorteile bei der Gesamtzahl der bearbeitbaren Probleme, was jedoch auf Kosten der Bearbeitungszeit aller Probleme erkauft wurde. Das legt die Vermutung nahe, dass ein dynamisches N sinnvoll wäre. So könnte man N abhängig machen von der Anzahl der übergebenen Domain Transition Graphs. Je mehr Graphen zu bearbeiten sind, desto größer muss man N wählen, um einen hinreichend großen abstrakten Problemraum zu erstellen. Eine andere Möglichkeit wäre, N graduell während

der “abstrahiere und verschmelze”-Schritte zu erhöhen. Für jeden Durchlauf der Schleife könnte man N um einen festen Wert erhöhen, so dass die Genauigkeit des abstrakten Problems mit der Anzahl der Abstraktionsschritte “mitwächst”.

Aktionsbezogene Verschmelzungsstrategien

Die hier vorgestellte aktionsbezogene Verschmelzungsstrategie war die Strategie mit der ausgewogensten Performanz. Eine weitere Strategie, die in diese Kategorie fällt, wird von Dräger und Finkbeiner in [2] vorgeschlagen. Sie schlagen vor, die zu verschmelzenden Graphen so zu wählen, dass ihre gemeinsamen Aktionen nahe der Zielzustände liegen. Dadurch wird ein Bereich um die Zielknoten geschaffen, in dem keine Abstraktion nötig ist und deshalb die Zieldistanz akkurat wiedergegeben wird.

Literaturverzeichnis

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

[2] Klaus Dräger, Bernd Finkbeiner, Andreas Podelski. 13th International SPIN Workshop on Model Checking of Software (SPIN), 2006.
Directed Model Checking with Distance-Preserving Abstractions

[3] http://cs-www.cs.yale.edu/homes/dvm/
Homepage von Drew McDermott

[4] J. Pearl. Morgan Kaufman, San Francisco, CA, 1983.
Heuristics

[5] http://zeus.ing.unibs.it/ipc-5/bnf.pdf
Definition der Sprache PDDL3.0

[6] Christer Bäckström and Bernhard Nebel. Computational Intelligence, 11(4):625-655, 1995.
Complexity results for SAS^+ planning