

Bachelorarbeit an der Technischen Fakultät der
Albert-Ludwigs-Universität Freiburg

Einfluss der
Finite-Domain-Repräsentation
auf die
Performance von Planungssystemen

Jonas Sternisko



Überarbeitete Fassung
Oktober 2009

Lehrstuhl Grundlagen der Künstlichen Intelligenz
Prof. Dr. Bernhard Nebel

Das Ziel der klassischen Handlungsplanung ist die Lösung von deterministischen Planungsaufgaben. Für eine formalisierte Darstellung eines Problems wird dabei eine Sequenz von Aktionen gesucht, mit der vom Ausgangszustand ein Ziel erreicht werden kann. Für diese Berechnung kennt die Handlungsplanung unterschiedlichste Ansätze. Gegenstand dieser Arbeit sind aber zwei unterschiedliche Arten der Problemformalisierung: Sie untersucht die Auswirkungen, die eine Darstellung von Planungsaufgaben durch Variablen endlichen Wertebereichs auf die Leistung von bestehenden Planungssystemen hat. Diese so genannte Finite-Domain-Repräsentation vergleichen wir anhand von Experimenten für optimale und suboptimale Planer mit der klassischen Problemformalisierung durch Aussagenvariablen. Die Aufgabenstellung dieser Arbeit ist, bisher bestehende qualitative Aussagen über die Auswirkungen der Finite-Domain-Repräsentation auf die Performance von Planern durch quantitative Erkenntnisse zu konkretisieren und zu ergänzen.

Im ersten Abschnitt dieser Arbeit fassen wir die theoretischen Grundzüge der Handlungsplanung zusammen und gehen auf die unterschiedlichen Repräsentationen von Planungsaufgaben ein. Im Abschnitt 2 stellen wir eine Planungsdomäne vor, die wir für die Untersuchung der Performance-Unterschiede entwickelt haben. Im dritten Abschnitt erläutern wir eine Alternative zu informierter heuristischer Suche. Diese symbolische Exploration wurde von uns für den Vergleich der verschiedenen Problemformulierungen in Form eines BDD-basierten Planungssystems implementiert. In diesem Text stellen wir die Grundzüge der Implementierung vor und diskutieren Optimierungsmöglichkeiten. Im letzten, überarbeiteten Abschnitt werden die Ergebnisse unserer Experimente präsentiert und interpretiert.

Danksagung

Ich möchte mich bei Professor Nebel für die Betreuung meiner Bachelorarbeit und die Möglichkeit bedanken, diese in der Arbeitsgruppe Grundlagen der Künstlichen Intelligenz zu schreiben. Malte Helmert danke ich für die anschaulichen Tipps zur symbolischen Exploration und dafür, dass er als Zweitgutachter diese Arbeit übernommen hat.

Insbesondere gilt mein Dank Gabriele Röger, die durch ihr Hintergrundwissen, ihre vielfältigen inhaltlichen und formalen Anregungen und durch ihre Unterstützung bei der Fehlersuche einen großen Beitrag dazu gebracht hat, dass ich diese Arbeit erfolgreich abschließen konnte.

Zuletzt möchte ich Aslaug Sternisko und Philip Stahl danken, die mir für die Struktur und Formulierung des Textes wertvolle Anregungen geliefert haben.

Inhaltsverzeichnis

1	Die Finite-Domain-Repräsentation im Kontext der Handlungsplanung	5
1.1	Handlungsplanung	5
1.1.1	Komponenten deterministischer Planungsaufgaben	5
1.1.2	Arten von Planern	8
1.1.3	Über die Schwierigkeit deterministischen Planens	8
1.2	Die Finite-Domain-Repräsentation	9
1.2.1	Planungsproblemen mit Variablen endlichen Wertebereichs	9
1.2.2	Vorteile der Finite-Domain-Repräsentation für Planungssysteme	10
2	Die Testdomäne ANTS	12
3	Symbolische Exploration mit Binary Decision Diagrams	17
3.1	Planerzeugung durch symbolische Exploration	17
3.1.1	Die Transitionsrelation	17
3.1.2	Symbolische Exploration des Zustandsraumes	18
3.1.3	Planextraktion	19
3.2	Binary Decision Diagrams	20
3.3	Implementierung eines BDD-basierten Planers	23
3.3.1	Implementationsdetails	24
3.3.2	Diskussion von Optimierungen	26
3.3.3	Über den entwickelten Planer	27
4	Ergebnisse der Experimente	30
4.1	Suboptimale Planer	30
4.1.1	Versuchsaufbau	30
4.1.2	Auswertung und Interpretation der Ergebnisse	30
4.1.3	Verhalten von suboptimalen Planern in der Domäne ANTS	40
4.1.4	Zusammenfassung	42
4.2	Optimale Planer	43
4.2.1	Versuchsaufbau	43
4.2.2	Auswertung und Interpretation der Ergebnisse	44
4.2.3	Verhalten von optimalen Planern in der Domäne ANTS	49
4.2.4	Zusammenfassung	50
5	Fazit	52

Einleitung

Stellen Sie sich vor, Sie sind eine Ameise und leben in einem großen Raum voller Tische. Eines Tages sehen Sie, dass auf einem anderen Tisch ein süßer Pfirsich vergessen wurde. Wenn Sie kein Obst mögen, können Sie sich auch gerne ein Stück Schokolade oder ein beliebiges anderes für eine Ameise begehrenswertes Objekt vorstellen. Sie beschließen jedenfalls, irgendwie auf den anderen Tisch zu kommen. Leider sind Sie erstmal ziemlich ratlos, denn als Ameise können Sie sich nur entlang der Straßen ihres Ameisenstaats bewegen und wissen nicht, wie Sie am schnellsten zu dem Leckerli kommen oder ob es überhaupt einen Pfad gibt, über den Sie ihr Ziel erreichen können. Bevor Sie einfach drauf los marschieren und vielleicht nie ankommen, nehmen Sie sich erstmal diese Bachelorarbeit über Handlungsplanung zur Hand. Darin werden Sie erfahren, dass ihr Problem auf zwei unterschiedliche Arten formalisiert werden kann. Nach der Lektüre werden Sie wissen, warum Sie ihr Problem in der Finite-Domain-Repräsentation darstellen sollten.

1 Die Finite-Domain-Repräsentation im Kontext der Handlungsplanung

In diesem Abschnitt werden wir Definitionen vornehmen, die wir im Laufe dieses Textes benötigen. Zuerst werden wir die klassische Handlungsplanung und anschließend die Finite-Domain-Repräsentation einführen.

1.1 Handlungsplanung

1.1.1 Komponenten deterministischer Planungsaufgaben

Das in der Einleitung geschilderte Problem entspricht einer typischen Situation aus der Domäne ANTS, welche wir in Abschnitt 2 noch genauer vorstellen werden. Bevor es jedoch soweit ist, müssen wir noch ein paar theoretische Grundlagen klären. Zu Beginn möchten wir einen Überblick über klassische Planungsprobleme geben.

Ziel des klassischen Planens ist es, eine Folge von Aktionen zu finden, mit der ein bestimmtes System von einem Ausgangszustand in einen Zielzustand überführt werden kann. Dafür formalisiert man das Problem zunächst in eine logische Darstellung, bevor dann mit Hilfe unterschiedlicher Algorithmen ein Plan gesucht wird. Im Detail definiert man eine deterministische Planungsaufgabe Π in aussagenlogischer Darstellung als 4-Tupel:

$$\Pi = \langle A, I, O, G \rangle$$

Sie besteht aus den folgenden Komponenten: Einer Menge von aussagenlogischen Zustandsvariablen A , einem über A definierten Startzustand I , einer Menge von Operatoren O und einer durch eine logische Formel G beschriebene Menge von Zielzuständen. Die Lösung einer Planungsaufgabe ist eine Sequenz von Operatoren, mit der der Startzustand

in einen Zustand aus der Zielmenge überführt werden kann. Diese Operatorenfolge wird als Plan π bezeichnet.

Zustandsvariablen A Dies ist die Menge von Variablen, die relevante Aspekte des Systems beschreiben. In unserer Definition sind dies aussagenlogische Variablen. Diese können die Werte \top (“wahr”) und \perp (“falsch”) annehmen. Betrachten wir dazu ein Beispielproblem aus der Domäne ANTS: Nehmen wir einmal einen Tisch t_1 und eine Ameise Willy a_{Willy} an, die sich auf t_1 befindet. In dieser Planungsaufgabe sind unsere Aussagenvariablen $(table\ t_1)$, $(ant\ a_{Willy})$ und $(at\ a_{Willy}\ t_1)$. Diese können die aussagenlogischen Werte “wahr” und “falsch” annehmen. Beispielsweise ist die Variable $(at\ a_{Willy}\ t_1)$ genau dann wahr, wenn die Aussage “Die Ameise Willy befindet sich auf dem Tisch t_1 .” stimmt.

Wir werden später sehen, dass die Finite-Domain-Repräsentation eine Generalisierung propositionaler Planungsaufgaben ist, die auf Variablen mit endlichem Wertebereich $dom(a)$ basiert.

Initialzustand I Als Zustände bezeichnet man eine Belegung über eine Menge von Variablen, also eine Funktion $s : a \in A \rightarrow dom(a)$, die jeder Variable einen Wert aus ihrem Wertebereich zuweist. Der Startzustand $s_o = I$ ist dann einfach die Belegung zum Zeitpunkt 0. Für die aussagenlogische Darstellung von Planungssystemen lässt sich der Startzustand als eine Menge $I \subseteq A$ angeben. Dann gilt $s(a) = 1$, wenn $a \in I$ und sonst ist $s(a) = 0$. Dazu wieder ein Beispiel aus ANTS: Der Startzustand mit der Ameise Willy auf dem Tisch t_1 wird durch die folgende Formel ausgedrückt:

$$I = \{(table\ t_1), (ant\ a_{Willy}), (at\ a_{Willy}\ t_1)\}$$

Jetzt erweitern wir diesen Zustand: Wir nehmen noch einen zweiten Tisch t_2 hinzu und verbinden diesen mit dem ersten Tisch durch eine Ameisenstraße. Dann wird die Formel schon etwas größer:

$$I = \{(table\ t_1), (ant\ a_{Willy}), (at\ a_{Willy}\ t_1), (table\ t_2), (path\ t_1\ t_2), (path\ t_2\ t_1), \neg(at\ a_{Willy}\ t_2)\}$$

Weil jede Variable zwei Werte annehmen kann, gibt es für eine Planungsaufgabe mit n Aussagenvariablen 2^n syntaktisch mögliche Zustände.

Operatoren O Durch Operatoren $o \in O$ geht das System von einem Zustand s in einen anderen s' über. Man kann sie sich als binäre Relationen vorstellen: Die (s, s') erfüllt die Relation, wenn der Zustand s durch Anwendung des Operators o in den Zustand s' übergeht. Man bezeichnet diesen Übergang als Transition. Alle möglichen Zustände bilden durch die Menge der Operatoren O ein Transitionssystem, welches wir im Laufe dieses Abschnitts noch definieren werden.

Wir werden die Funktion von Operatoren wieder an einem Beispiel verdeutlichen: In der Welt von ANTS gibt es einen Operator $(move\ a_{Willy}\ t_1\ t_2)$. Die Ausführung dieses Operators führt dazu, dass die Ameise Willy sich auf Tisch t_2 befindet. Allerdings lässt

sich dieser Operator nicht immer anwenden: Einerseits muss sich Willy auf Tisch t_1 befinden, andererseits muss eine Verbindung zwischen den Tischen t_1 und t_2 bestehen.

Wie wir im Beispiel gesehen haben, bestehen Operatoren aus zwei Teilen: Einer Bedingung c , die erfüllt sein muss, bevor der Operator ausgeführt werden kann und einem Effekt e , welcher die Auswirkungen des Operators beschreibt und damit den Folgezustand festlegt. Für unser Beispiel würde das formal wie folgt aussehen:

$$\begin{aligned} o &= \langle c, e \rangle \\ &= \langle (\text{path } t_1 \ t_2) \wedge (\text{at } a_{\text{Willy}} \ t_1), (\text{at } a_{\text{Willy}} \ t_2) \wedge \neg(\text{at } a_{\text{Willy}} \ t_1) \rangle \end{aligned}$$

In dieser Formel fehlen Variablen *ant* und *table*, die Objekte unseres Systems als Ameisen oder Tische identifizieren. Wir lassen diese Variablen auch im Folgenden aus Gründen der Übersichtlichkeit weg, weil ihre Werte sich nie ändern werden. Das Resultat der Anwendung eines Operators o auf einen Zustand s schreiben wir als $\text{app}_o(s)$.

Menge der Zielzustände G Ausgehend vom Startzustand soll durch wiederholtes Anwenden von Operatoren ein Zustand aus der Menge von Zielzuständen erreicht werden. Häufig ist diese Menge durch eine Eigenschaft charakterisiert, die sich durch eine aussagenlogische Formel implizit ausdrücken lässt. Eine explizite Angabe aller Zielzustände ist nicht erforderlich.

In der Domäne ANTS ist das Ziel in der Regel, alle Ameisen auf einen bestimmten Tisch zu bewegen. Wollte man aber lediglich Willy zum Zieltisch t_G bringen, so wäre die entsprechende Zielformel $G = (\text{at } a_{\text{Willy}} \ t_G)$. Diese wird nicht nur durch einen Zustand erfüllt:

$$\begin{aligned} (\text{at } a_{\text{Willy}} \ t_G) \wedge (\text{at } a_{\text{Garry}} \ t_G) &\models G, \text{ aber auch} \\ (\text{at } a_{\text{Willy}} \ t_G) \wedge \neg(\text{at } a_{\text{Garry}} \ t_G) &\models G \end{aligned}$$

Transitionssystem Jede Planungsaufgabe induziert ein Transitionssystem. Wir definieren ein solches als ein Tupel $\langle S, I_T, T, G_T \rangle$. S ist dabei die endliche Menge von Zuständen oder auch der Zustandsraum. $I_T \subseteq S$ und $G_T \subseteq S$ sind endliche Mengen von Start- beziehungsweise Zielzuständen. Die Transitionsrelation $T \subseteq S \times S$ beschreibt die Übergänge des Transitionssystems. Das Transitionssystem zu einer Planungsaufgabe $\Pi = \langle A, I, O, G \rangle$ kann wie folgt gebildet werden:

$$\begin{aligned} S &= \{s(A) \mid s \text{ ist eine vollständige Belegung über } A\} \\ I_T &= s_0 \\ T &= \{(s, s') \in S \times S \mid \exists o \in O : \text{app}_o(s) = s'\} \\ G_T &= \{s(A) \mid s \models G\} \end{aligned}$$

Ein Plan entspricht einem Pfad zwischen einem Start- und Zielzustand in einem Transitionssystem.

PDDL Um eine Planungsaufgabe für einen Computer zu definieren, gibt es mehrere Möglichkeiten. Wir verwenden in dieser Arbeit die Planning Domain Description Language (PDDL). Diese ist der Versuch, die Beschreibung von Planungsdomänen zu vereinheitlichen, damit sich unterschiedliche Ansätze zur Lösung von Planungsproblemen vergleichen lassen. Dabei werden in einer Domänenbeschreibung die Relationen für Objekte und die zur Verfügung stehenden Aktionen anhand schematischer Operatoren deklariert. Getrennt davon werden Probleminstanzen erstellt: Die Problembeschreibungen legen fest, welche Objekte es in einem Problem geben soll, welche Eigenschaften diese haben, wie der Startzustand aussieht und unter welchen Voraussetzungen das Ziel erreicht ist. PDDL ist weit verbreitet und wird zum Beispiel zur Spezifikation von Problemen für die regelmäßig stattfindende International Planning Competition (IPC) genutzt. Ein Beispiel für eine PDDL Beschreibung kann in Listing 1 eingesehen werden.

1.1.2 Arten von Planern

Man unterscheidet für klassische deterministische Planungsaufgaben zwischen Suboptimalen und Optimalen Planern.

Optimales Planen Wie der Name vermuten lässt, suchen diese Planungssysteme einen optimalen Plan π_{opt} . Um zu garantieren, dass es wirklich keine kürzere Aktionssequenz als die berechnete gibt, wenden diese Planer zum Beispiel eine A^* -Suche mit zulässiger Heuristik an. Alternativ kommen auch BDD-basierte Suchen zum Einsatz. Diese werden wir später vorstellen. Beispiele für solche Planungssysteme sind MIPS [9] oder HSP* [2].

Suboptimale Planer Diese Planungssysteme versuchen einen Plan zu finden, wobei dieser nicht notwendigerweise optimal sein muss. Die auch als suboptimal bezeichneten Planer führen meist eine gierige Bestensuche im Zustandsraum durch, beispielsweise unter Verwendung von Relaxierungsheuristiken. Beispiele sind Fast Forward von Jörg Hoffmann und Bernhard Nebel [21] oder Fast Downward von Malte Helmert [16].

1.1.3 Über die Schwierigkeit deterministischen Planens

Wie wir oben gesehen haben, kann man sich einen Plan als Pfad in einem Transitionssystem zwischen einem Start- und einem von mehreren Zielknoten vorstellen. Die Suche in solchen gerichteten Graphen aus n Knoten und m Kanten könnte man mit Dijkstras Algorithmus mit einer Komplexität von $O(n \log n + m)$ bewerkstelligen. Die Größe des Transitionssystems entspricht aber der Anzahl möglicher Zustände. Für ein Planungssystem mit $|A|$ aussagenlogischen Zustandsvariablen gibt es $2^{|A|}$ mögliche Zustände. Dadurch ist die Durchführung von Dijkstras Algorithmus für große Probleme nicht machbar. Es müssen also andere Ansätze verfolgt werden. Einer davon ist, mittels einer Heuristik den Abstand von Zuständen zu einem Zielzustand zu schätzen und so nur einen Teil des Transitionsgraphens zu durchsuchen. Andere Planer betrachten gleichzeitig mehrere Zustände und führen eine so genannte symbolische Exploration durch.

Die Frage, ob es für ein Planungssystem einen Plan gibt und das Problem, ob es für ein Planungssystem einen Plan bestimmter Länge gibt, werden in der Komplexitätstheorie mit PLANEX beziehungsweise PLANLEN bezeichnet. Für beide Probleme wurde PSPACE-Vollständigkeit nachgewiesen [6].

1.2 Die Finite-Domain-Repräsentation

1.2.1 Planungsproblemen mit Variablen endlichen Wertebereichs

Die Finite-Domain-Repräsentation (FDR) entstand aus einer kompakten Darstellung von Planungsvariablen für den BDD-basierten Planer MIPS [10]. Wie der Name schon sagt, stellt man mit ihr Planungsaufgaben durch Variablen endlichen Wertebereichs dar. Die Intuition dafür ist, dass man Wissen über die Domäne ausnutzt und Variablen zusammen fasst, die sich gegenseitig ausschließen. Wir möchten das wieder an einem Beispiel verdeutlichen:

In der ANTS Domäne gibt es das Prädikat (*at ant table*), welches genau dann wahr ist, wenn sich die Ameise *ant* auf dem Tisch *table* befindet. Betrachten wir folgende Situation:

$$(at\ a_{Willy}\ t_1) \wedge (at\ a_{Willy}\ t_2) \wedge (at\ a_{Willy}\ t_3)$$

Man sieht sofort, dass dieser Zustand keinen Sinn macht, denn die Ameise Willy kann nicht gleichzeitig auf drei verschiedenen Tischen sein. Jede der drei Teilaussagen schließt die jeweils anderen aus. Die Variablen bilden eine Mutexgruppe. Das folgt aus der Domänenbeschreibung und den Problemrichtlinien. Eine Invariante des Ameisen-Problems mit drei Tischen ist, dass sich eine Ameise nur auf genau einem Tisch gleichzeitig befinden darf. Wenn wir dieses Problem nun unter Kenntnis der Invariante umformen, könnten wir die drei Aussagenvariablen (*at ...*) durch eine neue Variable mit endlichem Wertebereich ersetzen.

$$dom(position\ of\ a_{Willy}) = \{t_1, t_2, t_3\}$$

Diese Variable kann nur genau einen Wert annehmen und bildet so die oben genannte Invariante besser ab.

Definition FDR-Planungsaufgabe Eine Planungsaufgabe in Finite-Domain-Repräsentation ist ein 4-Tupel $\Pi = \langle V, I, O, G \rangle$. Dabei ist V eine endliche Menge von FDR-Variablen. Das sind Variablen v mit endlichem Wertebereich $dom(v)$. Atomare Formeln haben die Form $(v = d)$ und geben an, dass v den Wert d hat. Ähnlich den propositionalen Planungsaufgaben ist I der Startzustand, G die Menge von Zielzuständen und O die Menge von Operatoren, nur werden diese jetzt über die FDR-Variablen V definiert. Der Startzustand muss eine vollständige Belegung über V sein. Für einen Finite-Domain-Operator $o = \langle c, e \rangle$ sind alle atomaren Effekte in der Form $v := d$ wobei $d \in dom(v)$ ist. Diese Formel bedeutet, dass durch Operator o die Variable v den Wert d annimmt.

Eine Planungsaufgabe in aussagenlogischer Darstellung ist ein Spezialfall einer FDR-Aufgabe, bei der alle Variablen Wertebereiche der Größe zwei haben. Darüber hinaus lässt sich jede FDR-Aufgabe in eine klassische Planungsaufgabe umformen. Sei $\Pi' = \langle A', I', O', G' \rangle$ mit

- $A' = \{(v, d) | v \in V, d \in \text{dom}(v)\}$
- $I'((v, d)) = 1$ genau dann, wenn $I(v) = d$
- O' und G' gehen aus O und G hervor: Jede atomare Formel $(v = d)$ wird durch eine Aussagenvariable (v, d) und jeder atomare Effekt $(v := d)$ wird durch einen Effekt $(v, d) \wedge \bigwedge_{d' \in \text{dom}(v) \setminus \{d\}} \neg(v, d')$ ersetzt.

Π' wird als die von Π induzierte aussagenlogische Planungsaufgabe bezeichnet.

1.2.2 Vorteile der Finite-Domain-Repräsentation für Planungssysteme

In der aussagenlogischen Form existieren sehr viele syntaktisch mögliche Zustände, die in Wirklichkeit nie erreicht werden können. Ein Beispiel dafür haben wir zu Beginn dieses Kapitels gezeigt. Der offensichtlichste Vorteil einer kompakten Darstellung von Variablen unter Berücksichtigung der Beschränkungen des Planungsproblems ist die reduzierte Anzahl syntaktischer Zustände. Das kommt vor allem Planungssystemen zugute, die keine interne Erreichbarkeitsprüfung durchführen. Erst kürzlich wurde von Helmert ein Artikel über die FDR verfasst [17]. Darin werden verschiedene positive Effekte für unterschiedliche Anwendungsgebiete in der Handlungsplanung genannt. Wir möchten uns hier auf positive qualitative Auswirkungen der FDR auf Planungssysteme beschränken, die Teil unserer Experimente sind:

- Für Suchverfahren, die Abstraktionsheuristiken verwenden, ist die kompakte Darstellung von Vorteil. Der Informationsgewinn von Abstraktionen, die ein Planungssystem auf eine FDR-Variable projizieren, ist besser, als wenn eine binäre Variable verwendet werden würde. Diese relativ speicherkritischen Heuristiken profitieren von der kleineren Anzahl syntaktisch möglicher Zustände.
- Suchheuristiken, die eine Zerlegung der Probleme verwenden (z.B. die Causal-Graph-Heuristik oder die Kontext-erweiterte additive Heuristik) gewinnen durch die einfachere Struktur der kausalen Graphen: Weniger variablen-darstellende Knoten führen zu weniger komplexen Graphen, wodurch die Berechnung der Heuristiken vereinfacht wird.
- BDD-basierte Suche: Durch Zusammenfassen von mutexen Planungsvariablen kann die Anzahl der Entscheidungsvariablen exponentiell sinken. Zwar müssen im BDD mehrwertige Knoten mit einem erhöhten Speicherverbrauch verwaltet werden, dies ist jedoch verschwindend gering im Vergleich zum Rückgang der BDD-Größe. Der größte Vorteil ist eine Verbesserung der Variablenordnung. Diese spielt bei BDDs eine kritische Rolle, was den Speicherbedarf und die Komplexität der BDD-Operationen angeht. Ohne eine kompakte Darstellung ist der Einsatz von BDDs zur Suche selten sinnvoll. Darauf werden wir im Abschnitt 3.1 eingehen.

Um dem Leser ein Verständnis für die Zusammenhänge zwischen Darstellung und Performanz der Planer zu vermitteln, skizzieren wir nun den Algorithmus, um eine normale

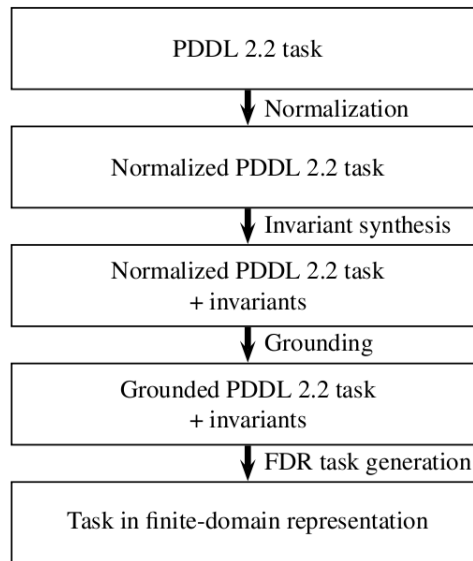


Abbildung 1: Übersetzung von PDDL in FDR nach [17].

PDDL Planungsaufgabe in Finite-Domain-Form umzuwandeln. Der Übersetzungsvorgang aus diesem Artikel kommt im Fast Downward Planer [16] zum Einsatz. Wir haben für unsere Experimente die Vorverarbeitung dieses Planers verwendet und halten uns deshalb an die Beschreibung von Helmert (Abbildung 1). Ein PDDL 2.2 Problem wird in vier Schritten in FDR überführt:

1. Zunächst wird eine *Normalisierung* durchgeführt. Alle Typen aus der Domänenbeschreibung werden entfernt und durch einstellige Prädikate des gleichen Namens ersetzt. Formeln werden durch logische Äquivalenzumformungen in eine konjunktive Form gebracht. Das Ergebnis ist eine syntaktisch eingeschränkte Version von PDDL.
2. Nun werden Mutexgruppen mit Hilfe von Invarianten gesucht. Die *Invariantensynthese* folgt einem induktiven Beweisschema: Für Formeln wird zunächst getestet, ob diese im Startzustand gelten. Man zeigt dann, dass wenn eine potentielle Invariante in einem beliebigen Zustand gilt, sie durch die Anwendung aller Operatoren nicht verletzt wird. Der Algorithmus von Helmert berechnet in diesem Schritt Kandidaten für Invarianten, die den Induktionsschritt erfüllen. Erst später wird getestet, ob für diese Kandidaten der Induktionsanfang gilt.
3. Im sogenannten *Grounding*-Schritt wird eine variablenfreie Darstellung der Planungsaufgabe erzeugt. Dazu werden mit Hilfe eines relaxierten Erreichbarkeitstests für das System irrelevante Atome, Operatoren und Axiome entfernt. Anschließend werden die Variablen instanziiert. Das heißt, die parametrisierten Teile werden durch Einsetzen aller passenden Objekte ersetzt. Hier kann es zu exponentiellem Wach-

tum der Darstellung der Planungsaufgaben kommen.

4. Im letzten Schritt wird aus den Ergebnisse der Schritte 2 und 3 die FDR-Planungsaufgabe zusammengesetzt. Die Mutexgruppen werden zu Variablen endlichen Wertebereichs zusammengefasst. Bemerkenswert ist die Tatsache, dass eine Mutexgruppe nicht unbedingt einer FDR-Variable entspricht, denn Mutexgruppen können sich durchaus überschneiden. Die Operatoren sowie Start- und Zielbeschreibung werden an die neuen Variablen angepasst.

Im vorhergehenden Abschnitt haben wir die Grundlagen der Handlungsplanung zusammen gefasst und zuletzt die Finite-Domain-Repräsentation eingeführt. Wir haben versucht, die wesentlichen Konzepte anhand von Beispielen zu verdeutlichen. Nicht ohne Grund haben wir dafür Situationen aus einer Ameisenwelt gewählt. Diese entsprechen nämlich Problemen aus ANTS – einer Planungsdomäne, die wir für diese Arbeit entwickelt haben und im nächsten Abschnitt genauer besprechen möchten.

2 Die Testdomäne ANTS

Um die Performance der FDR zu verdeutlichen, haben wir eine Planungsdomäne konstruiert. Folgende Grundidee wurde dabei verfolgt: Der Vorteil der FDR ist, dass sie Gruppen von sich gegenseitig ausschließenden Variablen besonders knapp darstellen kann. Wenn die Anzahl von aussagenlogischen Variablen größer wird, die sich zu einer FDR Variablen zusammen fassen lassen, sollte auch der Performance-Unterschied von Planern zunehmen. Wichtig war uns außerdem, dass die Domäne einfach verstanden werden kann.



Abbildung 2: Die Domäne ANTS

Das Resultat ist die Domäne ANTS. Deren PDDL Definition und ein kleines Problem können in den Listings 1 und 2 am Ende dieses Abschnitts eingesehen werden. Um sich typische ANTS-Probleme zu veranschaulichen, stelle man sich ein Volk von Ameisen vor, das bereits ein Netzwerk von Ameisenstraßen angelegt hat. Diese Straßen verlaufen zwischen n Tischen. Jede Straße beginnt und endet jeweils an einem unterschiedlichen Tisch. Für eine bestimmte Anzahl m unterscheidbarer Ameisen muss ein Plan erstellt werden, wie diese von einem Starttisch zu einem Zieltisch gelangen können. In der PDDL-Formulierung hat ANTS die in Tabelle 1 aufgeführten Prädikate.

Tabelle 1: PDDL Prädikate der Domäne ANTS.

Prädikat	Beschreibung
(table ? <i>t</i>)	Kennzeichnet einen Tisch ? <i>t</i> als solchen.
(ant ? <i>a</i>)	Charakterisiert eine Ameise ? <i>a</i> .
(path ? <i>t</i> ₁ ? <i>t</i> ₂)	Zwischen den Tischen ? <i>t</i> ₁ und ? <i>t</i> ₂ besteht ein Pfad.
(at ? <i>ant</i> ? <i>table</i>)	Die Ameise ? <i>ant</i> befindet sich auf dem Tisch ? <i>table</i> .

Die einzige Aktion in dieser Domäne ist

$$(\text{move } ant_A t_1 t_2).$$

Diese bringt eine Ameise ant_A von Tisch t_1 zu Tisch t_2 . Diese Aktion hat lediglich zwei Bedingungen: Die Ameise ant_A muss sich vor der Ausführung auf Tisch t_1 befinden und zwischen den beiden Tischen muss eine Ameisenstraße existieren.

Das Ameisenreich kann man sich als Graphen vorstellen: Die Straßen und Tische entsprechen Kanten und Knoten. Wir nehmen im Folgenden an, dass alle Instanzen einen ungerichteten Graphen beschreiben. Das heißt (path $t_1 t_2$) gilt genau dann, wenn auch (path $t_2 t_1$) gilt. Weiterhin handelt es sich um einen zusammenhängenden Graphen. Es gibt also keinen Tisch, von dem aus eine Ameise über eine beliebige Anzahl von Aktionen nicht jeden anderen Tisch erreichen könnte. Für die Struktur des Graphen ergeben sich verschiedene Auswirkungen auf unser Untersuchungsobjekt. Wir haben für unsere Experimente nur die Form eines linearen Netzwerks untersucht. Hier hat jeder Knoten mindestens einen und höchstens zwei Nachbarn. Darin sind die äußeren Enden dieser Kette von Tischen der Start- und Zielzustand.

Besonderheiten von ANTS Nach dieser ersten Vorstellung der Domäne wird sich mancher fragen, was an diesem Problem besonders schwierig sein soll. Ein menschlicher Planer erkennt sofort, dass die Ameisen einfach nur Tisch für Tisch zum Ziel gebracht werden müssen. Ein Computer kennt aber weder die Struktur des Graphen, noch kann er ausschließen, dass ein eingeschlagener Weg nicht plötzlich abseits des Ziels endet. Wir sind uns bewusst, dass Probleme dieser Domäne für keinen modernen suboptimalen Planer ein Hindernis darstellen. Für unser Untersuchungsobjekt, die Auswirkung der FDR auf die Performance von Planungssystemen bietet diese Domäne aber einige interessante Aspekte:

Die Position einer einzelnen Ameise ant_A wird durch eine PDDL Variable (at $ant_A t$) beschrieben. Es sollte dem Leser sprichwörtlich ins Auge gesprungen sein, dass sich jede Ameise zu einem Zeitpunkt jeweils nur auf einem Tisch aufhalten kann. Die Aussagenvariablen über alle Tische und jede Ameise bilden eine Mutexgruppe. Sie können also durch die Übersetzung zu einer einzigen Variable mit endlichem Wertebereich $dom((\text{position of } ant_A)) = \{table_1, table_2, \dots, table_n\}$ zusammengefasst werden. Für jeden Tisch, den wir in ein Problem der Domäne aufnehmen, erhöht sich die Größe des

Wertebereichs der Position jeder Ameise lediglich um eins. Der Zustandsraum hat insgesamt eine Größe von n^m .

Für den aussagenlogischen Fall hingegen bedeutet die Erweiterung des Problems um einen Tisch, dass das Planungssystem für jede Ameise eine Aussagenvariable mehr verwalten muss. Die Anzahl möglicher Belegungen für eine Ameise wächst um den Faktor zwei. Für n Tische im Problem haben wir also 2^n mögliche Belegungen der Aussagenvariablen für eine Ameise. Von den 2^n Zuständen im entsprechenden Transitionssystem sind aber nur n tatsächlich erreichbar, denn die Tisch-Variablen schließen sich gegenseitig aus. Für m Ameisen hat der Zustandsraum dann die Größe $(2^n)^m = 2^{n \cdot m}$. Wir können also festhalten:

In der Domäne ANTS wächst der Unterschied zwischen der Anzahl aussagenlogischer Variablen und der Größe der FDR Variablen mit der Anzahl von Tischen n .

Werfen wir nun einen Blick auf das potentielle Wachstum des Suchbaums. Die Aktion (*move ant_i t_a t_b*) hat nur zwei Bedingungen: Die gerade betrachtete Ameise muss sich auf dem Ausgangstisch befinden und es muss ein Pfad zum nächsten Tisch existieren. Aus unserer Annahme über den Graphen folgt, dass von jedem Tisch mindestens ein Pfad weg führt. Weil sich jede der m Ameisen *immer* auf irgendeinem Tisch befindet, kann man leicht nachvollziehen, dass wir mit jeder Ameise zu jedem Zeitpunkt mindestens eine Aktion ausführen können. Sowohl auf dem Starttisch als auf dem Zieltisch kann mit jeder Ameise nur eine Aktion ausgeführt werden. Somit haben wir einen initialen Expansionsfaktor von m . Befinden sich alle Ameisen weder auf dem Start- noch auf dem Zieltisch, können jeweils zwei Aktionen ausgeführt werden. Dann würde sich unser Suchbaum durch eine Expansion um $2m$ mögliche Zustände erweitern. Wir erkennen, dass der Suchbaum ein Wachstum in $O(2^m)$ hat. Durch die dauerhafte Möglichkeit, mindestens m Aktionen anzuwenden zu können, ist unser Problem also doch nicht ganz so trivial. Halten wir fest:

Durch Veränderung der Anzahl von Ameisen m lässt sich auf den Verzweigungsgrad des Suchbaums Einfluss nehmen.

Wie wir erläutert haben, bietet unsere Domäne durch Veränderung der Parameter n und m die Möglichkeit, direkt auf die Größe der Probleme und des Unterschieds zwischen FDR und normaler Darstellung Einfluss zu nehmen. Die Länge aller optimalen Pläne ist $(n-1) \cdot m$. Durch die Anzahl der Ameisen m wird der Wachstumsfaktor des Suchbaums reguliert. Mit der Anzahl der Tische n skaliert der Nutzen der FDR. Wir werden versuchen, dies später im Experiment zu zeigen.

ANTS und suboptimales Planen Eine gierige Bestensuche hat keine Schwierigkeiten mit Problemen aus ANTS, muss nur bei großen Problemen sehr viele Zustände expandieren und die Heuristik auswerten. Die Länge der erzeugten Pläne der suboptimalen Planer sollten relativ nah an $|\pi_{opt}|$ liegen.

ANTS und optimales Planen Optimale Planer werden mit der ANTS-Domäne Probleme haben. Das liegt in erster Linie daran, dass alle möglichen Verteilungen der Ameisen über die Tische Teil eines optimalen Plans sind. Heuristische A^* -Suchen verwenden einen Knotenwert f , der sich aus dem Abstand vom Startzustand $g(x)$ und einem Heuristikwert für den Knoten des Zustandes im Transitionssystem $h(x)$ berechnet.

$$f(x) = g(x) + h(x)$$

Jeder mögliche Knoten x für das zu einem ANTS-Problem gehörende Transitionssystem hat bei einer zulässigen Heuristik einen Wert $h(x) \leq h^*(x)$. Dabei sind $h^*(x)$ die tatsächliche Anzahl Operationen, um den zu x gehörenden Zustand in einen Zielzustand zu überführen. Also hat jeder Knoten im Transitionsgraph einen f -Wert, der kleiner oder gleich der optimalen Planlänge $|\pi_{opt}|$ ist. Es wurde bereits erwähnt, dass sich zu jedem Zeitpunkt zwischen m und $2m$ Aktionen ausführen lassen. Aufgrund der linearen Anordnung der Tische wird jede Bewegung einer Ameise "zurück" in Richtung Starttisch als "schlecht" erkannt. Es werden mit jeder Expansion m neue optimale Knoten ("in Richtung Zieltisch") erzeugt und an die Schlange noch auszuwertender Knoten angeheftet. Weil es für jeden Zustand einen Plan gibt, der Teil eines optimalen Planes für das gesamte Problem ist, überprüfen die Planungssysteme die meisten Knoten des Transitionssystem. Wie wir bereits erläuterten, hat dieses aber m^n erreichbare Knoten.

In ANTS könnten Planer einen Vorteil haben, die eine Abstraktionsheuristik auf die einzelnen Ameisen berechnet und für das Teilproblem jeder Ameise einen optimalen Plan bestimmt.

Anmerkung Übrigens lassen wir in ANTS die Intelligenz des Ameisenvolks außer Acht. Diese führte sicherlich dazu, dass der Stamm in kürzester Zeit einen direkten Weg zwischen Start und Zieltisch anlegt. Damit würden wir allerdings die Ergebnisse unserer Experimente sehr verfälschen, und schließlich liegt unser Interesse eher bei der *künstlichen* Intelligenz.

Listing 1: Die Domäne ANTS in PDDL

```
(define (domain ants)
  (:predicates (table ?t)
               (path ?t1 ?t2)
               (ant ?a)
               (at ?ant ?table))
  (:action move
   :parameters (?ant ?from ?to)
   :precondition (and (table ?from)
                      (table ?to)
                      (ant ?ant)
                      (at ?ant ?from)
                      (path ?from ?to))
   :effect (and (not (at ?ant ?from))
                (at ?ant ?to)))
)
```

Listing 2: Ein typisches ANTS Problem in PDDL

```
(define (problem problem-ants-2tables-01ants)
  (:domain ants)
  (:objects ant0 ant1 table0 table1)
  (:init (ant ant0)
         (at ant0 table0)
         (ant ant1)
         (at ant1 table0)
         (table table0)
         (table table1)
         (path table0 table1)
         (path table1 table0))
  (:goal (and (at ant0 table1)
              (at ant1 table1)))
)
```

3 Symbolische Exploration mit Binary Decision Diagrams

Unsere Experimente, deren Ergebnisse wir in Abschnitt 4 vorstellen werden, wurden mit Planungssystemen durchgeführt, die in den Fast Downward Planer eingebettet sind. Da alle dieser bereits implementierten Verfahren auf Heuristiken aufbauen, haben wir einen alternativen Ansatz zur Lösung von Planungsproblemen in einem Programm umgesetzt. Dieser Ansatz zur symbolischen Exploration des Zustandsraumes kam dann neben den heuristischen Suchen zum Einsatz. Im nächsten Abschnitt stellen wir die symbolische Exploration und die zu ihrer Umsetzung benutzten Binary Decision Diagrams vor. Anschließend besprechen wir Details unserer Implementierung.

Die im Folgenden beschriebene Suchmethode kommt ohne eine Heuristik aus. Wir testen nicht mehr in einem bestimmten Zustand die Anwendbarkeit eines Operators und bauen nach und nach einen Suchbaum auf, sondern verzichten auf die explizite Darstellung von Operatoren und Zuständen und führen eine sogenannte *symbolische Exploration* des Zustandsraums durch. Dafür verwenden wir Binary Decision Diagrams (BDDs). Wir werden in diesem Abschnitt zunächst das Prinzip der symbolischen Exploration vorstellen und anschließend auf die Datenstruktur der BDDs eingehen.

3.1 Planerzeugung durch symbolische Exploration

Eine Menge von Zuständen lässt sich durch eine logische Formel darstellen, wie wir es für die Zielformeln kennen gelernt haben (siehe 1.1.1). Wir suchen nun nach einer Methode zur Erzeugung der Menge all jener Zustände, die von dieser Menge in einem Schritt erreicht werden können. Diese in der Literatur [23] als *Image* bezeichnete Zustandsmenge erhalten wir durch Bildung des relationalen Produkts aus der Ausgangsmenge und einer geeigneten Darstellung der Operatoren. Die Gesamtheit aller Operatoren wird dabei als Transitionsrelation formuliert. Wir möchten kurz auf deren Berechnung eingehen, bevor wir mit der Suche fortfahren.

3.1.1 Die Transitionsrelation

Die Transitionsrelation $T_A(O)$ beschreibt die durch Operatoren in O verursachten Zustandsübergänge im Transitionssystem. Wir bilden sie als Disjunktion über einzelne Formeln für Operatoren

$$T_A(O) = \bigvee_{o \in O} \tau_A(o)$$

Die Formel $\tau_A(o)$ beschreibt die Anwendbarkeit und den Effekt der Anwendung eines Operators $o = \langle c, e \rangle$ bezüglich der Variablen A und ist wie folgt definiert:

$$\tau_A(o) = c \tag{1}$$

$$\wedge \bigwedge_{a \in A} (\text{EPC}_a(e) \vee (a \wedge \neg \text{EPC}_{\neg a}(e))) \Leftrightarrow a' \tag{2}$$

$$\wedge \bigwedge_{a \in A} \neg(\text{EPC}_a(e) \wedge \text{EPC}_{\neg a}(e)) \tag{3}$$

Teil (1) der Formel beschreibt die Vorbedingung des Operators und somit seine Anwendbarkeit. Teil (2) betrifft die Veränderung der Variablen a : Der neue Wert a' ist genau dann wahr, wenn die Variable a durch e wahr gemacht wird oder sie schon wahr gewesen ist und nicht durch e falsch gemacht wird. Der Ausdruck $\text{EPC}_\psi(e)$ beschreibt die Bedingung, unter der Effekt e dazu führt, dass ψ wahr wird. Die Effektvorbedingung¹ $\text{EPC}_\psi(e)$ eines Literals ψ in einem Effekt e ist wie folgt definiert:

$$\begin{aligned}\text{EPC}_\psi(\psi) &= \top \\ \text{EPC}_\psi(\phi) &= \perp, \text{ if } \phi \neq \psi \\ \text{EPC}_\psi(e_1 \wedge \dots \wedge e_k) &= \text{EPC}_\psi(e_1) \vee \dots \vee \text{EPC}_\psi(e_k) \\ \text{EPC}_\psi(\phi \triangleright e) &= \text{EPC}_\psi(e) \wedge \phi\end{aligned}$$

Der dritte Teil der Formel für die Transitionsrelation (3) stimmt genau dann, wenn der Effekt e keine Variable gleichzeitig wahr und falsch macht. Diese Bedingung stellt die Konsistenz des Transitionssystems sicher. Die hier definierte Relation gilt für Aussagenvariablen. Da wir später aber Variablen verwenden, die endlich viele Werte annehmen können, werden wir noch die entsprechende Erweiterung der Formeln besprechen. Die Formel (1) wird dann aus atomaren Formeln ($v = d$) gebildet. In Formel (2) müssen wir alle möglichen Werte der Variablen a berücksichtigen:

$$\bigwedge_{a \in A} \bigwedge_{d \in \text{dom}(a)} \left(\left(\text{EPC}_{(a=d)}(e) \vee \left((a = d) \wedge \bigwedge_{d' \in \text{dom}(a) \setminus \{d\}} \neg \text{EPC}_{(a=d')}(e) \right) \right) \Leftrightarrow (a' = d) \right) \quad (4)$$

Mit der Formel (3) muss für den erweiterten Fall sicher gestellt werden, dass der Effekt e der Variablen a nur einen der Werte gibt:

$$\bigwedge_{a \in A} \bigwedge_{d \in \text{dom}(a)} \bigwedge_{d' \in \text{dom}(a) \setminus \{d\}} \neg (\text{EPC}_{(a=d)}(e) \wedge \text{EPC}_{(a=d')}(e)) \quad (5)$$

Einige Leser werden bereits vermuten, dass die Formel für die Transitionsrelation sehr groß werden kann, weil wir eine Disjunktion über alle Operatoren bilden. Tatsächlich ist dieser Teil für die Suche kritisch. Wir werden in den Implementationsdetails in Abschnitt 3.3.1 noch darauf zurück kommen. Zunächst gehen wir auf die symbolische Exploration des Zustandsraums ein.

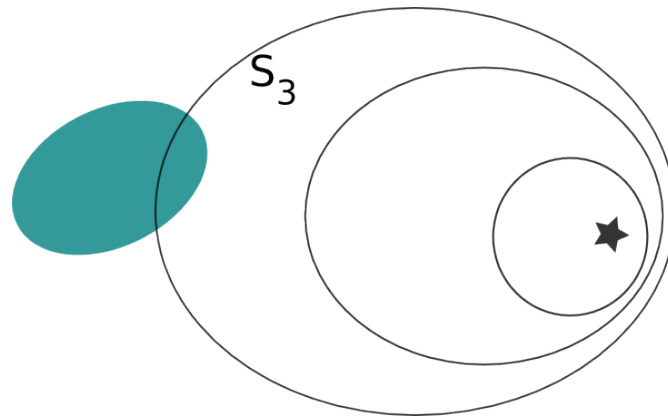
3.1.2 Symbolische Exploration des Zustandsraumes

Wollen wir eine symbolische Exploration in der Handlungsplanung durchführen, stellen wir den Startzustand als Formel dar und wenden darauf die Transitionsrelation wiederholt an. Das Verfahren wird durch Abbildung 3a veranschaulicht.

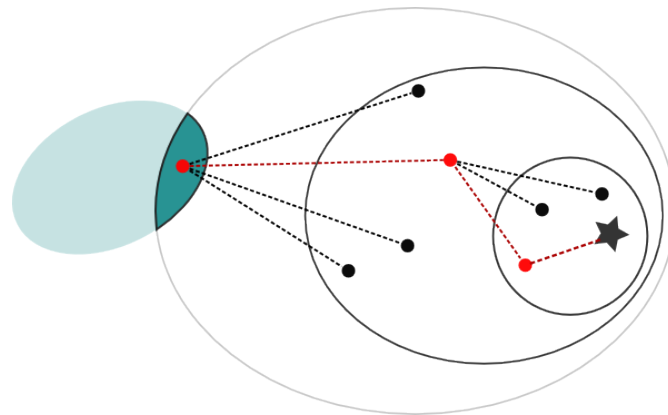
Wenn wir nach n Wiederholungen ein Modell für die Zielformel erzeugt haben, wissen wir: Es existiert eine Sequenz von n Aktionen, mit der man den Startzustand in einen

¹Effect Pre-Condition

Zielzustand überführen kann. Falls sich die Menge der erreichten Zustände nach n Iterationen nicht mehr ändert, sind alle erreichbaren Zustände erkundet, aber kein Zielzustand gefunden. Das bedeutet: Es gibt für dieses Planungsproblem keine Lösung.



(a) Exploration: Vom Startzustand \star werden wiederholt die relationalen Produkte mit der Transitionsrelation gebildet (schwarze Kreise), bis eine Schnittmenge mit den Zielzuständen (türkise Fläche) besteht.



(b) Plan Extraktion: Aus dem Schnitt mit der Zielmenge wird ein Zustand ausgewählt, auf diesen die inverse Transitionsrelation angewandt. Aus dem Schnitt mit der Menge vorher erreichbarer Zustände wird erneut ein Zustand ausgewählt. Dies wird wiederholt, bis der Startzustand erreicht ist.

Abbildung 3: Erzeugen eines Plans durch symbolische Exploration des Zustandsraums

3.1.3 Planextraktion

Sollte die Exploration ergeben haben, dass es einen Plan der Länge n gibt, können wir jetzt, beginnend mit dem Modell des Ziels, aus den durchlaufenen Mengen einen Plan bestimmen. Dazu wählen wir einen beliebigen Zustand x_n aus der Schnittmenge des n -

ten relationalen Produkts mit der Zielmenge. Auf diesen Zustand wenden wir nun die inverse Transitionsrelation an. Dadurch erhalten wir die Menge $O^{-1}(x_n)$ aller Zustände, von denen x_i in einem Schritt erreichbar ist. Nun schneiden wir diese mit dem $(n - 1)$ -ten Image. Für diese Zustände gilt neben der ersten Eigenschaft auch, dass sie in $n - 1$ Schritten vom Startzustand erreicht werden können.

Wir fahren damit fort, einen Zustand x_i zu wählen und die oben beschriebene Prozedur auf diesen anzuwenden, bis der Startzustand $x_0 = I$ erreicht ist. Wir haben nun eine Sequenz x_0, x_1, \dots, x_n , die einen Plan durch eine Folge von Zuständen beschreibt. Abschließend durchlaufen wir diese Zustandssequenz und suchen jeweils für x_i und x_{i+1} einen Operator o_i , der diesen Zustandsübergang durchführt (siehe Abbildung 3b). Die Abfolge von Operatoren $o_1 \dots o_n$ ist der gesuchte Plan. Dieser ist in jedem Fall optimal, da sonst die Exploration des Zustandsraums nach weniger als n Schritten terminiert wäre. Die Extraktion des Plans ist für die Suche insgesamt zeitlich unkritisch. Die Komplexität der Extraktion ist $O(n \cdot m)$, wobei n die Länge des Plans und m die Anzahl der Operatoren ist.

3.2 Binary Decision Diagrams

Für die Darstellung der Formeln für Zustände, der Mengen erreichter Zustände und für die Transitionsrelation während der symbolischen Exploration wählen wir Binary Decision Diagrams (BDDs). Wir fassen diese Datenstruktur kurz zusammen:

- Ein BDD ist ein gerichteter Graph mit genau einem Wurzelknoten ohne Eingangskanten.
- Jeder Endknoten ohne ausgehende Kanten ist mit **0** oder **1** beschriftet. Jeder Pfad durch den Graph endet also in den Knoten **0** oder **1**.
- Jeder innere Knoten mit dem Label X_i hat zwei Ausgangskanten *low* und *high*. Wir bezeichnen diese Knoten als Entscheidungsknoten. Die Kinder des Knotens nennt man *high*- bzw. *low*-Nachfolger von X_i .

Doch wie stellt nun ein BDD eine aussagenlogische Formel dar? Betrachten wir eine Belegung $\psi : x_i \rightarrow \{0; 1\}$ über Variablen $x_1 x_2 \dots x_k$. Wir traversieren den Graphen beginnend an der Wurzel und folgen am Knoten X_i der *high*-Kante, falls $\psi(x_i) = 1$. Sonst gehen wir zum *low*-Nachfolger des Knotens.

Jeder BDD B repräsentiert eindeutig eine aussagenlogische Formel F [1]. Für jede F erfüllende Variablenbelegung gibt es einen in **1** endenden Pfad durch B . Alle anderen Belegungen, für die der BDD durchlaufen wird, enden im Endknoten **0**. Abbildung 4 zeigt einige Beispiele durch BDDs dargestellter Formeln.

Für jeden BDD gibt es eine kanonische Darstellung. Um in diese gebracht zu werden, wird der BDD zunächst in eine geordnete Form überführt. Durch Isomorphiereduktionen werden dann Kanten verschmolzen, die auf den gleichen Teil-BDD gerichtet sind. Abschließend werden jene Knoten aus dem BDD entfernt, deren *low*- und *high*- Kanten auf denselben Knoten zeigen. Bei einer solchen Shannon-Reduktion werden die Eingangskanten des entfernten Knotens auf den Nachfolger gesetzt. Das Resultat dieser Umformungen

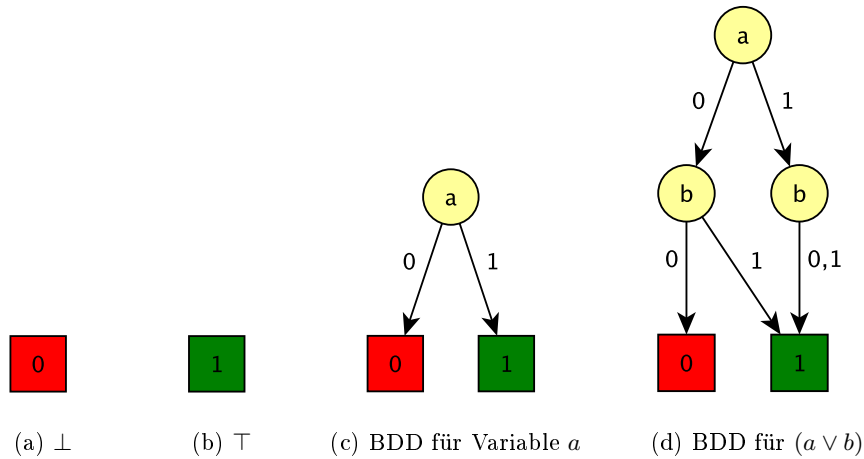


Abbildung 4: Beispiele für BDDs

ist ein geordneter, reduzierter BDD. Diese kanonische Form hat den Vorteil, dass jeder BDD einzigartig ist und wichtige Operationen in effizienter Zeit durchgeführt werden können. Zum Beispiel kann man zwei BDDs in konstanter Zeit auf Äquivalenz prüfen. Außerdem wird die Anzahl der Knoten minimiert, ohne dass der BDD an Aussagekraft verliert.

Um die Bedeutung der Reduktionen zu verdeutlichen, betrachten wir ein ANTS-Problem. Nehmen wir einen Zustand mit zwei Tischen und zwei Ameisen Willy und Garry an, die sich beide auf Tisch t_1 befinden. Die zugehörige aussagenlogische Formel wäre

$$(at\ ant_{Willy}\ t_1) \wedge \neg(at\ ant_{Willy}\ t_2) \wedge (at\ ant_{Garry}\ t_1) \wedge \neg(at\ ant_{Garry}\ t_2).$$

Diese Formel wird durch die BDDs in Abbildung 5 ausgedrückt. Die Knotenbezeichnungen entsprechen den einzelnen Aussagen. Sie wurden lediglich gekürzt, um die Lesbarkeit zu verbessern:

$$\begin{aligned} (at\ ant_{Willy}\ t_1) &\hat{=} var_1 \\ (at\ ant_{Willy}\ t_2) &\hat{=} var_2 \\ (at\ ant_{Garry}\ t_1) &\hat{=} var_3 \\ (at\ ant_{Garry}\ t_2) &\hat{=} var_4 \end{aligned}$$

Die BDDs sind bereits in die Variablenordnung $var_1 \prec var_2 \prec var_3 \prec var_4$ gebracht. Wie der Leser erkennt, hat der reduzierte, geordnete BDD wesentlich weniger Knoten und Kanten.

Binary Decision Diagrams sind zur Darstellung von Zustandsmengen geeignet, weil eine große Anzahl möglicher Zustände durch einen BDD mit geringer Anzahl von Knoten ausgedrückt werden kann. Mit zunehmender Größe der Formeln können BDDs zwar exponentielle Größe in der Zahl der Variablen annehmen, tatsächlich lässt sich das aber

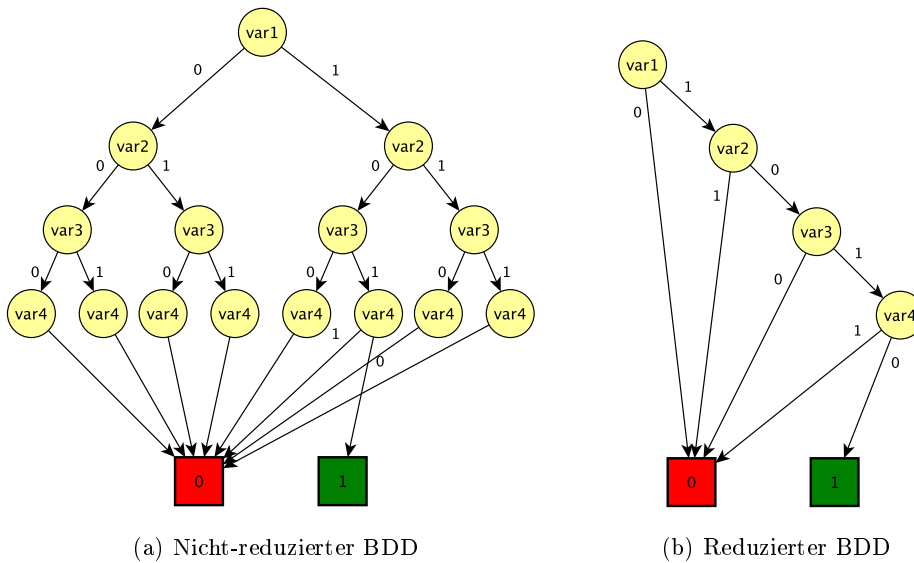


Abbildung 5: Unterschiedliche Darstellungen von $(var_1 \wedge \neg var_2 \wedge var_3 \wedge \neg var_4)$ durch BDDs (ungekennzeichnete Ausgänge entsprechen *low*- und *high*-Kanten)

oft umgehen [4]. Außerdem lassen sich logische Verknüpfungen (\wedge , \vee , \exists , ...) von BDDs mit Hilfe von Graph-basierten Algorithmen effizient durchführen. Sie eignen sich durch diese Eigenschaften hervorragend für die Realisierung einer symbolischen Suche im Zustandsraum eines Planungsproblems.

In der Literatur [10, 17, 23] wird die Ansicht geteilt, dass erst eine kompakte Darstellung der Variablen den Einsatz von BDD-Planern sinnvoll macht. Andernfalls sind die Zustandsmenge und Transitionsrelation repräsentierenden BDDs zu groß, um die Verknüpfungsoperationen schnell durchführen zu können. Wie oben erwähnt wurde, hat ein BDD eine Größe in $O(2^n)$ bei n Entscheidungsvariablen. Anhand einer Datenstruktur mit Entscheidungsknoten endlichen Wertebereiches könnte man den Nutzen von BDDs für eine symbolische Exploration in der Handlungsplanung also erhöhen. Die als *Integer Valued Decision Diagrams* bezeichnete Datenstruktur verwendet für jeden Entscheidungsknoten einen BDD, der die möglichen Werte der zugehörigen Variablen als Binärzahlen darstellt und dessen Ausgänge auf andere endliche Entscheidungsknoten gerichtet sind.

Wir möchten den Vorteil der Finite-Domain-Repräsentation für BDDs verdeutlichen, in dem wir im letzten Beispiel FDR-Variablen einführen. Abbildung 6 vergleicht den reduzierten BDD für den Zustand aus ANTS mit dem entsprechenden BDD für die Finite-Domain-Repräsentation. Die sich gegenseitig ausschließenden Aussagenvariablen über den Aufenthalt einer Ameise auf einem bestimmten Tisch wurden zu FDR-Variablen zusammengefasst (Abb. 6b). Jede dieser Variablen nimmt Werte entsprechend den möglichen Positionen einer Ameise an. Deshalb ändern sich auch die Ausgangskanten der

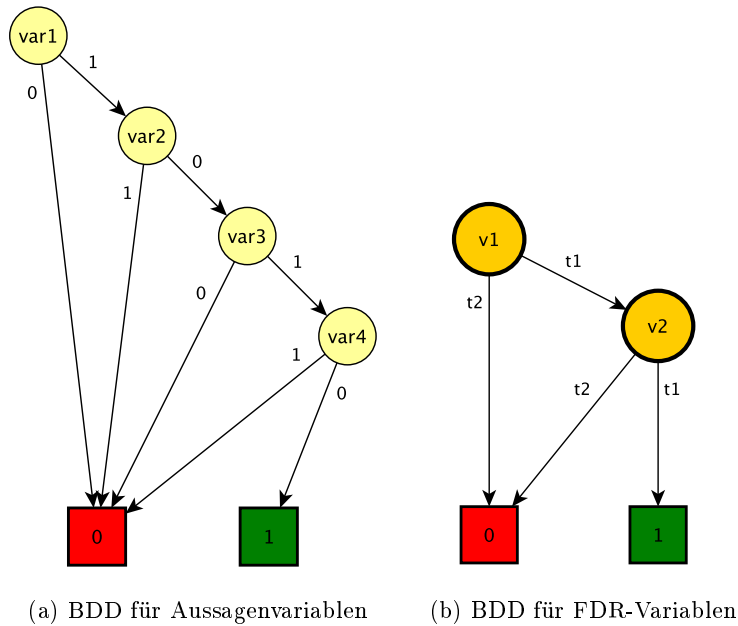


Abbildung 6: Unterschiedliche Größe von BDDs im Normalfall und bei Verwendung von FDR-Variablen. Die größeren Knoten bei der FDR sollen andeuten, dass die Wertebereiche innerhalb der Knoten durch BDDs realisiert werden.

Entscheidungsvariablen im BDD.

$$v_1 \hat{=} \text{position-of-ant}_{Willly} \in \{t_1, t_2\}$$

$$v_2 \hat{=} \text{position-of-ant}_{Garry} \in \{t_1, t_2\}$$

3.3 Implementierung eines BDD-basierten Planers

Unsere Implementierung folgt den Ansätzen aus der Vorlesung “Handlungsplanung” von Malte Helmert an der Universität Freiburg aus dem Wintersemester 2008/2009. Diese orientiert sich wiederum an frühen Versionen des Model Checking Integrated Planning System (MIPS) von Edelkamp und Helmert [11]. Im Gegensatz zu diesem Planer unterstützt unser Programm keine Routinen zum Übersetzen und Vorverarbeiten der Planungsprobleme, sondern verwendet die Ausgabe der im Fast Downward Planer [16] enthaltenen Übersetzung von PDDL in FDR. Dadurch und durch die Implementierung in C++ möchten wir eine bessere Vergleichbarkeit zu anderen Planungsalgorithmen sicherstellen. Als BDD-Library haben wir BuDDy von J. Lind-Nielsen [26] verwendet. Diese ist zum einen gut dokumentiert und findet Anwendung in anderen Planern [22], zum anderen bietet sie in Form von FDD-Blöcken ein hervorragendes Werkzeug zur Realisierung von Entscheidungsbäumen mit FDR-Variablen.

3.3.1 Implementationsdetails

Im folgenden Abschnitt gehen wir auf Eckpunkte der Implementierung ein. Anhand dieser soll der Leser in die Lage versetzt werden, den Planer nachzuvollziehen. Anschließend möchten wir mögliche Optimierungen diskutieren.

Unser Planer fügt sich nahtlos in das Fast Downward System ein. Dieses initialisiert zunächst die eingebundenen Planungssysteme und führt dann eine schrittweise Suche durch Aufruf einer Methode *step* durch.

Die Initialisierung Wird ein Objekt von unserem Planer erstellt, initialisieren wir die BDD Library und diverse Hilfsstrukturen. Wir erzeugen aus der von Übersetzungs- und Vorverarbeitungsprogrammen ausgegebenen FDR-Beschreibung des Problems den Startzustand, die Zielformel und die Transitionsrelation in Form von BDDs.

Erstellung der Transitionsrelation Rufen wir uns noch einmal die Definition der Transitionsrelation für einzelne Operatoren (Abschnitt 3.1.1) ins Gedächtnis.

$$\tau_A(o) = c \tag{1}$$

$$\wedge \bigwedge_{a \in A} \bigwedge_{d \in \text{dom}(a)} \left(\left(\text{EPC}_{(a=d)}(e) \vee \left((a = d) \wedge \bigwedge_{d' \in \text{dom}(a) \setminus \{d\}} \neg \text{EPC}_{(a=d')}(e) \right) \right) \right) \Leftrightarrow (a' = d) \tag{2}$$

$$\wedge \bigwedge_{a \in A} \bigwedge_{d \in \text{dom}(a)} \bigwedge_{d' \in \text{dom}(a) \setminus \{d\}} \neg (\text{EPC}_{(a=d)}(e) \wedge \text{EPC}_{(a=d')}(e)) \tag{3}$$

Unsere Implementierung unterstützt keine bedingten Effekte. Dies scheint uns ausreichend, weil unser Untersuchungsobjekt der Unterschied zwischen FDR und normaler Darstellung bei der Suche im Zustandsraum ist und sich dies auch ohne bedingte Effekte untersuchen lässt. Die Formel (3) drückt aus, dass eine Variable durch einen Operator nicht mehrere unterschiedliche Werte gleichzeitig annehmen darf. Wir können die Formel missachten, weil wir keine bedingten Effekte umsetzen müssen und deshalb die Operatoren des Planungsproblems nach Übersetzung in FDR konsistent sind. Die Formel (2) lässt sich für alle Variablen, die nicht vom Operator o betroffen sind, zu $\bigwedge_{d \in \text{dom}(a)} (a = d) \Leftrightarrow (a' = d)$ vereinfachen. Für alle Variablen, deren Werte durch o verändert werden, behalten wir sie bei. Der resultierende Code dafür kann in Listing 5 eingesehen werden. Im Folgenden bezeichnen wir den BDD für die Transitionsrelation als Operator-BDD.

Variablenordnung Ein ausgesprochen wichtiger Punkt für eine annehmbare Laufzeit des Planers ist ein Detail im Umgang mit der BuDDy-Library. Bei der Erzeugung der Variablenblöcke ist eine gute Variablenordnung von enormer Auswirkung auf die Größe der später erstellten Komponenten. Die durch die Vorverarbeitung von Fast Downward generierte Ordnung berücksichtigt den Informationsgehalt der FDR-Variablen bereits und

wir konnten sie einfach übernehmen. Für eine annehmbare Laufzeit müssen Entscheidungsvariablen, die alte und neue Werte der Variablen darstellen, paarweise angeordnet werden:

$$a_1 \prec a'_1 \prec a_2 \prec a'_2 \prec \dots \prec a_n \prec a'_n$$

Die Variablen a_i entsprechen im Programm geraden und a'_i ungeraden Indizes. Durch Ordnungen wie $a_1 \prec a_2 \prec \dots \prec a_n \prec a'_1 \prec a'_2 \prec \dots \prec a'_n$ wurden die Operator-BDDs schon für kleine Probleme riesig und die Suche dauerte unannehmbar lange. Der entsprechende C++-Code ist in Listing 3 abgebildet. Mit einer inkrementellen Erstellung der Wertebereiche mittels $fdd_extdomain(a_i, dom(a_i))$ stellen wir die Umsetzung dieser Ordnung sicher. Ein einzelner Aufruf mit dem Array aller Variablen und Wertebereiche hat sich hier kontraproduktiv ausgewirkt.

Die Methoden *apply* und *step* – Symbolische Exploration Der Downward Planer, in den unser Programm eingebettet ist, führt die Suche schrittweise durch Aufruf einer Methode *step* aus (siehe Listing 4). In unserer BDD-basierten Suche entspricht das einem Schritt der symbolischen Exploration. Die Menge *reached* enthält die nach i Schritten erreichten Zustände S_n . Wir testen zunächst, ob die Schnittmenge des aktuellen Images mit den Zielzuständen Elemente enthält. In diesem Fall haben wir einen Zustand erreicht, der die Zielformel erfüllt. Demnach können wir mit der Extraktion eines Plans fortfahren und die Suche danach terminieren. Wenn wir aber noch kein Ziel erreicht haben, bilden wir mit der Funktion *apply* das nächste Image und fügen die Menge der neu erreichten Zustände *current* den bisher erreichten *reached* hinzu. Jetzt prüfen wir, ob durch das letzte gebildete Image Zustände erreicht wurden, die bisher noch nicht in *reached* waren. Falls nicht, bricht die Suche ab, weil es keinen Plan gibt. Sonst fahren wir mit der Exploration fort.

Bleibt zu klären, wie genau die *apply*-Funktion funktioniert. Diese bildet zunächst die Konjunktion aus Operator-BDD und dem BDD für die Menge aktuell erreichter Zustände *current*. Aus diesem BDD werden im gleichen Zug alle Entscheidungsknoten entfernt, die alte Werte der Variablen darstellen (entspricht a_i). Abschließend werden alle Variablen $a'_i \in A'$ in $a_i \in A$ umbenannt. Das Resultat ist das *Image*, die Menge der von *current* in einem Schritt erreichbaren Zustände.

Die Planextraktion und inverse Transitionsrelation Hat die symbolische Exploration ergeben, dass in n Schritten ein Zielzustand erreicht werden kann, startet die Planextraktion. Der Code dazu kann in Listing 6 nachgelesen werden. Zunächst wählen wir einen Zustand x_n aus der Teilmenge der Zielzustände, die nach n Schritten erreicht ist (*intersect*). Nun wenden wir auf x_n die inverse Transitionsrelation an und erhalten $O^{-1}(x_n)$ in Form eines BDD. Dazu können wir einfach die Transitionsrelation $T_A(O)$ wie in *apply* benutzen. Lediglich bei der Umbenennung und Quantifizierung von Variablen vertauschen wir jeweils a_i und a'_i . Wir bilden nun die Schnittmenge $O^{-1}(x_n) \cap S_{n-1}$ mit den vom Startzustand in $n - 1$ Schritten erreichbaren Zuständen S_{n-1} . Daraus extrahieren wir den nächsten Zustand x_{n-1} . Diese Prozedur wiederholen wir insgesamt n -mal. Für die Zustandsfolge $x_0 \dots x_n$ testen wir im letzten Arbeitsschritt für jeden Zustand x_i die

Anwendbarkeit aller Operatoren $o \in O$ und fügen dann einen Operator mit $o(x_i) = x_{i+1}$ unserem Plan hinzu.

3.3.2 Diskussion von Optimierungen

Partitionierung der Transitionsrelation Weil die Transitionsrelation in jedem Suchschritt an den Operationen beteiligt ist, würde sich hier eine Optimierung besonders lohnen. Burch, Clarke und Long [5, 24] haben ursprünglich für Model-Checking-Probleme Überlegungen angestellt, wie man durch eine Aufteilung einer Formel die Komplexität von Existenz-Quantifizierungen verringern kann. Eine solche Quantifizierung führen wir in jedem Schritt durch, wenn wir nach Bildung des relationalen Produkts und vor der Umbenennung die “alten” Variablenwerte a_i vergessen. Auf unser Problem angewendet lautet die Zerlegung wie folgt:

$$\exists x' (S_i \wedge T_A(O)) = \bigvee_{o \in O} \exists x' (S_n \wedge \tau_A(o))$$

Von Edelkamp und Helmert [12] wird dieses Verfahren für den Einsatz in BDD-basierten Planern empfohlen. Die Idee ist also, die Transitionsrelation nicht über alle Operatoren zu bilden, sondern für jeden Operator einzeln zu bilden, um so die Berechnung des Images zu beschleunigen. Allerdings konnten wir im Gegensatz zu UMOP von Jensen&Veloso [23] keine Verbesserung der Laufzeit beobachten. Im Gegenteil: Wir verzeichneten durch die aufeinander folgende Anwendung von einzelnen Operatoren einen leichten Overhead.

Beschränken des Operator BDDs auf Wertebereiche Die Entscheidungsvariablen endlichen Wertebereichs werden intern durch BDDs realisiert, die den aktuellen Wert der Variable als Binärzahl darstellen. Dabei tritt folgende Situation auf, wenn die Domäne einer Variable eine Größe verschieden von einer Summe aus Zweierpotenzen hat. Nehmen wir eine Variable a mit Wertebereich $dom(a) = \{0, 1, 2, \dots, 5\}$ an. Sechs verschiedene Zahlen lassen sich durch einen BDD mit drei Entscheidungsknoten entsprechend den Potenzen $2^0, 2^1, 2^2$ darstellen. Allerdings könnte dieser BDD auch noch zwei weitere Werte, nämlich $6 = 2^2 + 2^1$ und $7 = 2^2 + 2^1 + 2^0$ darstellen. Diese Werte können bei einer korrekten Implementierung nicht auftreten. Trotzdem können diese explizit verboten werden, indem man den Operator-BDD mit der Disjunktion $(a = 0 \vee a = 1 \vee \dots \vee a = 5)$ über die zulässigen Werte aller Variablen $a \in A$ schneidet. Laut Malte Helmert (persönliche Kommunikation) kann dies zu Performanzvorteilen bei einer rückwärts gewandten Suche führen – für unsere Progressionssuche konnten wir jedoch keine Verbesserung feststellen.

Forward Set Simplification Wir haben nun bereits einige Methoden kennen gelernt, die den Operator-BDD verkleinern können. Andere Überlegungen zur Optimierung von BDD-basierten Planern betreffen die gespeicherten Zustandsmengen.

Die Berechnung des $i+1$ -ten Image S_{i+1} erfolgt mit dem BDD für die Transitionsrelation und dem BDD für die Menge S_i .

$$S_{i+1} = S_i \cup apply(S_i)$$

Wir führen diese Operation nur durch, wenn das Ziel nach i Schritten noch nicht erreicht ist. Somit gibt es keinen bisher erreichten Zustand $s \in S_i$ mit $s \models G$. Also können wir S_{i+1} zu einer Menge S_{cut} vereinfachen, indem wir aus S_{i+1} die Zustände von S_i entfernen.

$$S_{cut} = S_{i+1} \setminus S_i = S_i \setminus S_i \cup \text{apply}(S_i) \setminus S_i = \text{apply}(S_i) \setminus S_i$$

In jedem Aufruf von *step* würden wir dann diese Vereinfachung durchführen. Die zusätzliche Berechnung lohnt sich natürlich nur, wenn der BDD für S_{cut} eine kleinere Darstellung als S_{i+1} hat.

Constrain, Restrict und generalisierter Cofaktor Aus der letzten Überlegung können wir schlussfolgern, dass es eine Anzahl von Zustandsmengen S_i gibt, so dass gilt:

$$S_{cut} = S_1 \subset S_2 \subset \dots \subset S_l = S_{i+1}$$

Jede dieser Mengen repräsentiert die Menge der aktuell erreichten Zustände. Das Ziel ist nun, die Menge $S_{min} \in \{S_1, \dots, S_l\}$ mit einer möglichst kleinen Repräsentation zu finden. In den Arbeiten von Coudert et al. [8] wird dieser Vorgang als Vereinfachung unter einem Constraint und S_{min} als generalisierter Cofaktor (GCF) bezeichnet. In [13] wird dieser wie folgt beschrieben: Für zwei Funktionen f und c ist $g = \text{gcf}(f, c)$ so definiert, dass $f \wedge c$ und $g \wedge c$ gleich sind. In der BuDDy-Library sind mit *bdd_constrain*, *bdd_restrict* und *bdd_simplify* unterschiedliche Implementierungen dieser Ideen vorhanden. Mit ihrer Hilfe könnte die Darstellung der Zustandsmengen in jedem Schritt der Exploration verkleinert werden.

3.3.3 Über den entwickelten Planer

Wir haben letztendlich weder eine Verkleinerung des Operator-BDDs noch der Zustandsmengen in den Planer integriert. Ein Geschwindigkeitsvorteil fiel selten und dann nur minimal aus und wir haben mehr Wert auf einen übersichtlichen Code gelegt. Am wichtigsten ist unserer Ansicht nach eine sinnvolle Variablenordnung. Das Problem, eine optimale Variablenordnung zu finden, ist aber NP-schwierig. Eventuell würde sich bei großen Problemen anbieten, die Transitionsrelation für unterschiedliche Variablenordnungen zu berechnen und die kompakteste Variante zu verwenden.

Die Erstellung der Transitionsrelation ist der kritische Part in unserem Planer. Die von uns gewählte Formel für die Transitionsrelation führt dabei oft zum Volllaufen des Hauptspeichers. Ist die Relation zu komplex, scheitert der Planer stets schon an ihrer Berechnung. Über die Geschwindigkeit der eigentlichen Exploration mit einem großen Operator-BDD können wir mangels derartiger Beispiele keine Aussage treffen.

Obwohl sie mit anderen Implementierungen [23, 11] sicherlich nicht mithalten kann, erfüllt unsere Implementation ihren Zweck: Sie löst sowohl propositionale als auch in FDR übersetzte Versionen von gängigen IPC Planungsproblemen. Damit können wir die Auswirkung der Finite-Domain-Repräsentation auf die allgemeine Performanz von BDD-basierten Planern untersuchen.

Listing 3: Erzeugen der Wertebereiche für die BDD-Knoten

```

1 void FDDProgressionSearchEngine::create_domains(){
2   for (int i = 0 ; i < num_domains; i++){
3     domains [2*i] = g_variable_domain.at(i);
4     domains [2*i+1] = g_variable_domain.at(i);
5   }
6   for(int i = 0; i < 2*num_domains; i++){
7     int domain [1] = {domains[i]};
8     fdd_extdomain(domain, 1);
9   }
10 }

```

Listing 4: Die Methoden *step* und *apply* - Symbolische Exploration mit BDDs

```

1 FDDProgressionSearchEngine::step(){
2   bdd intersect = reached & goal;
3   if(intersect != bdd_false()){
4     extract_plan(intersect);
5     return SOLVED;
6   }
7   current = apply(current);
8   new_reached = reached | current;
9   state_space.push_back(new_reached);
10  if(new_reached == reached){
11    return FAILED;
12  }
13  reached = new_reached;
14  return IN_PROGRESS;
15 }
16
17 bdd FDDProgressionSearchEngine::apply(bdd &current){
18   //intersect and forget values a
19   bdd b = bdd_appex(operator_bdd, current, bddop_and, forget_evens);
20   b = bdd_replace(b, rename_pairs);
21   operator_counter++;
22   return b;
23 }

```

Listing 5: Erstellen eines BDDs aus einem FDR-Operator

```

1 bdd FDDProgressionSearchEngine::operator_to_bdd(Operator &o){
2   //build operator precondition bdd c
3   vector<Prevail> prevails = o.get_prevail();
4   bdd c = build_prevails_bdd(prevails);
5
6   //build the bdds for all variables effected by o
7   bdd new_values = bdd_true();
8   vector<PrePost> preposts = o.get_pre_post();
9   set<int> affected;
10  for(int j = 0; j < preposts.size(); j++){
11    PrePost effect = preposts.at(j);
12    //old value of a
13    if(effect.pre != -1)

```

```

14     c &= fdd_ithvar(2*effect.var, effect.pre);
15     //new value of a
16     new_values &= fdd_ithvar(2*effect.var+1, effect.post);
17     affected.insert(effect.var);
18 }
19
20 //build bdds for all other variables of the problem
21 for(int j = 0; j < num_domains; j++){
22     if(affected.count(j) > 0)
23         continue;
24     for(int i = 0; i < fdd_domainsize(2*j); i++){
25         bdd bdd_a = fdd_ithvar(2*j, i);
26         bdd bdd_a_new = fdd_ithvar(2*j + 1, i);
27         new_values &= bdd_apply(bdd_a, bdd_a_new, bddop_biimp);
28     }
29 }
30 return c & new_values;
31 }

```

Listing 6: Planextraktion im C++ Code

```

1 void FDDProgressionSearchEngine::extract_plan(bdd &intersect){
2     //get and store a state
3     bdd satisfying_state = get_sat_bdd(intersect);
4     State st = bdd_to_state(satisfying_state);
5     state_sequence.push_back(st);
6
7     int i = operator_counter;
8     while(i > 0){
9         i--;
10        bdd b = apply_inverse(satisfying_state);
11        b = b & state_space.at(i);
12        //get and store an single state
13        satisfying_state = get_sat_bdd(b);
14        st = bdd_to_state(satisfying_state);
15        state_sequence.push_back(st);
16    }
17
18    for(int i = state_sequence.size(); i > 1; i--){
19        //find an operator and store this operator in the plan
20        State& state1 = state_sequence[i-1];
21        State& state2 = state_sequence[i-2];
22        Operator& o = find_operator(state1, state2);
23        plan.push_back(&o);
24    }
25    set_plan(plan);
26 }

```

4 Ergebnisse der Experimente

Wir haben in den vorhergehenden Abschnitten die Finite-Domain-Repräsentation in der Handlungsplanung vorgestellt, die Besonderheiten der Domäne ANTS erklärt und zuletzt den von uns programmierten BDD-Planer besprochen. Damit haben wir alle Grundlagen geschaffen, um die Experimente für optimale und suboptimale Planungssysteme auszuwerten, welche im Rahmen dieser Arbeit durchgeführt wurden. In diesen haben wir suboptimale und optimale Planungssysteme getrennt auf Problemen in den unterschiedlichen Darstellungsformen getestet.

4.1 Suboptimale Planer

4.1.1 Versuchsaufbau

Wir haben in unseren Experimenten die Finite-Domain-Repräsentation und die klassische propositionale Darstellung (im Folgenden mit PR, in den Tabellen mit tief gestelltem P abgekürzt) für eine gierige Bestensuche mit vier verschiedenen Heuristiken untersucht. Alle Kombinationen aus Repräsentationen und Heuristiken wurden auf eine Auswahl von Problemen getestet. Bei den Heuristiken handelt es sich um die Causal-Graph-Heuristik h^{CG} [15], die additive Heuristik h^{add} von Bonet und Geffner[3], die FF-Heuristik h^{FF} [21] und die Context-enhanced additive Heuristik h^{cea} [19]. Alle diese Heuristiken sind im Fast Downward Planungssystem implementiert. Die Funktionsweise dieser Heuristiken im Detail zu erklären, würde den Rahmen dieser Arbeit sprengen. Für den Augenblick müssen wir nur wissen, dass die additive und die FF-Heuristik ihren Heuristikwert über ein relaxiertes Planungsproblem berechnen. Das ist eine Umformung der ursprünglichen Planungsaufgabe, in der diverse Beschränkungen fallen gelassen werden. Die Causal-Graph-Heuristik bezieht ihren Schätzwert aus dem Abhängigkeitsgraphen eines Problems und die Context-enhanced additive Heuristik vereint die Ideen der Causal-Graph- und der additiven Heuristik.

Diese Heuristiken wurden nun auf Probleminstanzen der IPC 1 – 5 in den zwei verschiedenen Darstellungsformen getestet. Eine detaillierte Auflistung der verwendeten Domänen befindet sich in Tabelle 2. Alle Experimente liefen auf einem Rechner mit zwei Quad-Core Intel Xeon 2,66GHz Prozessoren und 32 GB Hauptspeicher. Dabei wurden immer vier Suchen gleichzeitig ausgeführt. Jedem Suchprozess standen 2048 MB Speicher zur Verfügung und die Suche wurde spätestens nach 30 Minuten abgebrochen. Als Input bekamen alle Planer das Ergebnis der Übersetzung und Vorverarbeitung des Fast Downward Planers. Für die Experimente mit aussagenlogischer Darstellung wurde dabei der letzte Schritt der Übersetzung (siehe Abbildung 1) ausgeschaltet, so dass dann eine Planungsaufgabe in FDR-Format, aber mit nur zweiwertigen Variablen vorliegt.

4.1.2 Auswertung und Interpretation der Ergebnisse

Wir werden jetzt die Ergebnisse des ersten Experiments strukturiert vorstellen und Auffälligkeiten hervorheben. Für die Anzahl gelöster Instanzen, Suchzeit und Planlänge werden wir jeweils eine Übersicht geben und dann über Ergebnisse für einzelne Domänen und

Tabelle 2: Domänen für die Experimente mit suboptimalen Planern.

AIRPORT	MICONIC-FULLADL	PSR-LARGE
ASSEMBLY	MICONIC-SIMPLEADL	PSR-MIDDLE
BLOCKS	MOVIE	PSR-SMALL
DEPOT	MPRIME	ROVERS
DRIVERLOG	MYSTERY	SATELLITE
FREECELL	OPENSTACKS	SCHEDULE
GRID	OPTICAL-TELEGRAPHS	STORAGE
GRIPPER	PATHWAYS	TPP
LOGISTICS00	PHILOSOPHERS	TRUCKS
LOGISTICS98	PIPESWORLD-NOTANKAGE	ZENOTRAVEL
MICONIC	PIPESWORLD-TANKAGE	

Heuristiken berichten. Anschließend werden wir eine Wertung der Ergebnisse vornehmen. Dabei ist unser Ziel stets der Vergleich der Ergebnisse, die mit den beiden unterschiedlichen Darstellungen der Planungsprobleme erzielt wurden. Der Vergleich der Heuristiken untereinander ist dagegen nicht Teil dieser Untersuchung.

Gelöste Probleme Insgesamt lösen die Planer mehr Probleme in der FDR als in der normaler Darstellung. Von den insgesamt 1 612 Problemen löste die Causal-Graph-Heuristik h^{CG} in FDR 1 226 und in PR 1 150 Probleme. Die Context-enhanced additive Heuristik h^{cea} löste 1 304 beziehungsweise 1 264, die additive Heuristik h^{add} 1 280 beziehungsweise 1 262 und die FF-Heuristik h^{FF} löste 1 368 beziehungsweise 1 359 Probleme. Diese Daten sind in Abbildung 7 zusammen gefasst. Tabelle 3 hebt zudem die unterschiedlich starke prozentuale Verbesserung der Anzahl gelöster Probleme hervor.

Von den 12 896 Kombinationen aus Darstellungsvarianten, Heuristiken und Problemen wurden insgesamt 2 683 nicht gelöst. Die Ursachen dafür verteilen sich in etwa gleich auf durch Speicher- und Zeitlimitüberschreitungen.

Nachdem wir uns einen Überblick verschafft haben, kommen wir nun zu einer detaillierteren Betrachtung für die einzelnen Heuristiken. Die h^{CG} -Heuristik löste unabhängig von der Darstellung die wenigsten Probleme. Deshalb fällt auf, dass sie ausgerechnet mit der propositionalen Darstellung die meisten Probleme für die Domäne OPTICAL-TELEGRAPHS löste (nämlich sechs), in der alle anderen Kombinationen von Heuristiken und Repräsentationen nur ein oder zwei Probleme lösen konnten.

Tabelle 3: Anzahl der durch die Heuristiken gelöste Probleme.

	h^{CG}	h^{add}	h^{FF}	h^{cea}
FDR	1 226	1 280	1 368	1 304
PR	1 150	1 262	1 359	1 264
Verbesserung $\left(\frac{FDR-PR}{1612}\right)$	4,7%	1,1%	0,6%	2,5%

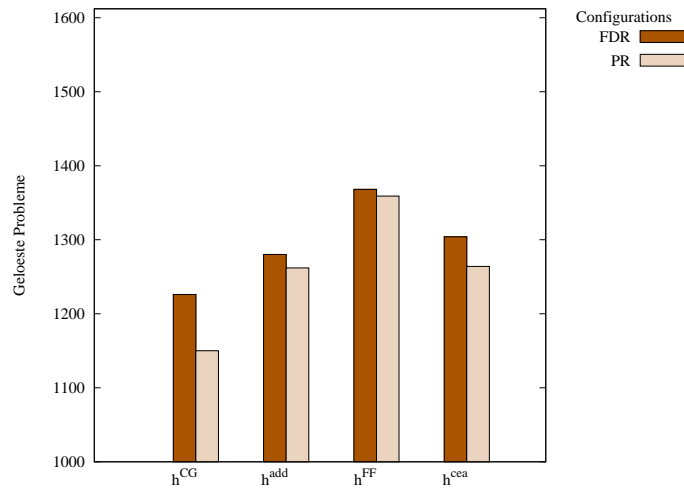


Abbildung 7: Anzahl der durch die Heuristiken gelöste Probleme (von 1612)

In einigen Domänen wurden mit der FDR von allen Heuristiken mehr Probleme gelöst. Für andere konnten mit der PR mehr Probleme gelöst werden. In wieder anderen werden mit beiden Varianten der Darstellung alle Probleme gelöst, darunter GRIPPER, PSR-MIDDLE und ZENOTRAVEL. Tabelle 4 auf der nächsten Seite zeigt die Anzahl gelöster Probleme für alle Domänen, Heuristiken und Darstellungsformen. Die Anzahl von Problemen, in denen die Planer unter Verwendung von FDR erfolgreicher waren, überwiegt und der Vorteil ist in diesen Domänen auch deutlicher ausgeprägt, als in den Domänen, in denen mit der PR mehr Probleme gelöst werden. Viele Domänen müssen differenzierter betrachtet werden, denn bei diesen wirkt sich die Darstellungsform der Planungsprobleme unterschiedlich auf die Heuristiken aus. Für die Domänen MICONIC-FULLADL und TRUCKS kann man im Ergebnis des Übersetzers feststellen, dass es keine Mutexgruppen gibt oder die gebildeten FDR-Variablen wenig Aussagekraft besitzen, weil hier der Vorteil der normalen Darstellung sehr gering ist.

Tabelle 4: Gelöste Probleme nach Domäne, Heuristik und Darstellungsform.

Domäne (# Probleme)	h^{CG}	h_P^{CG}	h^{add}	h_P^{add}	h^{FF}	h_P^{FF}	h^{cea}	h_P^{cea}
AIRPORT (50)	23	22	34	35	34	33	43	34
ASSEMBLY (30)	6	6	22	20	29	29	22	22
BLOCKS (35)	35	35	35	35	35	35	35	35
DEPOT (22)	12	10	14	14	17	17	13	13
DRIVERLOG (20)	20	18	20	20	19	19	18	19
FREECELL (80)	68	78	76	73	77	74	77	74
GRID (5)	4	4	3	3	5	4	5	3
GRIPPER (20)	20	20	20	20	20	20	20	20
LOGISTICS00 (28)	28	28	28	28	28	28	28	28
LOGISTICS98 (35)	35	6	29	27	34	32	35	28
MICONIC (150)	150	150	150	150	150	150	150	150
MICONIC-FULL. (150)	135	136	139	139	134	136	139	139
MICONIC-SIMPLE. (150)	150	150	150	150	150	150	150	150
MOVIE (30)	30	30	30	30	30	30	30	30
MPRIME (35)	35	21	34	34	29	31	35	34
MYSTERY (30)	17	14	17	18	17	18	19	18
OPENSTACKS (30)	24	24	25	25	30	30	24	24
OPTICAL-TELE. (48)	1	6	1	1	2	2	2	1
PATHWAYS (30)	9	9	16	13	19	17	15	13
PHILOSOPHERS (48)	48	19	48	48	48	48	48	48
PIPESWORLD-N. (50)	26	17	32	32	38	41	38	31
PIPESWORLD-T. (50)	14	20	19	14	22	25	20	18
PSR-LARGE (50)	32	32	32	32	35	35	31	31
PSR-MIDDLE (50)	50	50	50	50	50	50	50	50
PSR-SMALL (50)	50	50	50	50	50	50	50	50
ROVERS (40)	32	29	33	32	32	31	34	31
SATELLITE (36)	36	32	35	35	31	32	35	34
SCHEDULE (150)	61	61	60	62	124	115	61	60
STORAGE (30)	18	20	18	17	20	20	17	17
TPP (30)	25	21	23	17	24	19	24	22
TRUCKS (30)	12	12	17	18	15	18	16	17
ZENOTRAVEL (20)	20	20	20	20	20	20	20	20
Summe (1612)	1226	1150	1280	1262	1368	1359	1304	1264

Generell finden die Heuristiken mit der Finite-Domain-Repräsentation häufiger einen Plan. Es lässt sich aber nicht sagen, dass wenn ein Problem von einer Heuristik in FDR nicht gelöst wurde, die gleiche Heuristik mit dem äquivalenten Problem in aussagenlogischer Form auch keine Lösung gefunden hat. So wurde zum Beispiel das 18. Problem aus TRUCKS mit h^{cea} für die propositionale Form gelöst, während die entsprechende Darstellung in FDR zu keiner Lösung kam.

Suchzeit Als nächstes betrachten wir die Suchzeit. Wir vergleichen die Suchzeiten der beiden Kodierungs-Varianten miteinander und zählen, in welcher Darstellung in kürzerer Zeit ein Plan gefunden wurde. Es wurden nur solche Probleme verglichen, die mit beiden Varianten gelöst wurden. Wir werten einmal nach einer Differenz im Bereich einer hundertstel Sekunde und ein andermal berücksichtigen wir nur Zeitunterschiede ab einer Sekunde. Die erste Methode stellt die vielen Probleme besser dar, welche in kurzer Zeit gelöst werden können. Da hier aber Schwankungen in der Auslastung der Versuchshardware großen Einfluss haben können und für die FDR eigentlich noch ein gewisser Zeitaufwand für die Übersetzung des Problems aufgeschlagen werden müsste, halten wir die zweite Variante für aussagekräftiger. Die Tabelle 5 fasst die Bewertungen zusammen. Die Werte darin geben an, wie oft mit der entsprechenden Repräsentation ein Plan in kürzerer Zeit gelöst wurde. Darunter sind die Anteile dieser Wertung an der Gesamtzahl von beiden Darstellungen gelöster Probleme aufgeführt. Die Anzahl und der Anteil der Probleme, die in gleicher Zeit gelöst wurden, haben wir zur Wahrung der Lesbarkeit nicht in die Tabelle aufgenommen.

Tabelle 5: Bewertung des Zeitunterschieds Δt . Wie oft eine Darstellungsform zu einer kürzeren Suchzeit als die andere führte.

	h^{CG}		h^{add}		h^{FF}		h^{cea}	
	FDR	PR	FDR	PR	FDR	PR	FDR	PR
$\Delta t \geq 0.01s$	527	242	713	169	700	197	578	272
Anteil	47,0%	21,6%	57,0%	13,5%	52,9%	14,9%	46,1%	21,7%
$\Delta t \geq 1s$	178	116	290	58	277	68	280	75
Anteil	15,9%	10,3%	23,2%	4,6%	20,9%	5,1%	22,3%	6%

Durch die Beschränkung auf einen Zeitunterschied von mindestens einer Sekunde wird bis auf h^{CG} der Anteil von besseren Suchzeiten für die FDR im Vergleich zur propositionalen Darstellung deutlicher. Anscheinend ist der Performance-Unterschied bei Problemen mit längerer Suchzeit stärker. Die Werte lassen vermuten, dass der Vorteil der kompakten Darstellung bei kleinen Problemen nicht so stark zur Geltung kommt. Im Gegensatz zur Anzahl der gelösten Probleme scheinen die Relaxierungsheuristiken h^{add} und h^{FF} in der Suchzeit mehr zu profitieren: Hier ist der Anteil der Probleme, für die in der FDR schneller ein Plan gefunden wurde, jeweils fünf- beziehungsweise viermal so hoch.

Es gibt eine ganze Reihe von Domänen, in denen jede Heuristik mit der FDR bei mehr Problemen schneller einen Plan gefunden hat. Darunter sind DRIVERLOG, GRID, LOGISTICS98, OPENSTACKS, PHILOSOPHERS oder auch ZENOTRAVEL. In den Domänen OPTICAL-TELEGRAPHS und PSR-MIDDLE hat dagegen die propositionale Darstellung in kürzerer Zeit zu einem Plan geführt.

Qualitativ zeichnet sich also eine vorteilhafte Auswirkung der FDR ab. Doch wie groß ist nun dieser Vorteil? Oder kann es sogar sein, dass die bisher verwendete Statistik verschleiert, dass die FDR nur bei kleinen Problemen einen Vorteil bringt und die propo-

sitionale Repräsentation bei längeren Problemen im Vorteil ist? Um eine bessere Aussage treffen zu können, betrachten wir für jedes Problem das Verhältnis der Suchzeiten von FDR und PR und berechnen den geometrischen Mittelwert \bar{x}_t^{geom} für jede Domäne. Dabei betrachten wir nur Probleme, die in beiden Darstellungen gelöst wurden.

$$\bar{x}_t^{geom}(domain) = \sqrt[|domain|]{\prod_{i \in domain} \frac{t_{FDR}(i)}{t_{PR}(i)}}$$

Ein Wert kleiner $\bar{x}_t^{geom} < 1$ bedeutet dann, dass im Durchschnitt die FDR Repräsentation besser war, während ein Wert über 1 für eine kürzere mittlere Suchzeit mit der PR steht. Betrachten wir die Werte für die IPC-Domänen, fällt auf (Tabelle 6 auf der nächsten Seite), dass die durchschnittliche Suchzeit in drei Viertel der Fälle für die FDR besser ist. Eher selten sind Kombinationen aus Problemen und Heuristiken, bei denen die Mittelwerte für beide Repräsentationen annähernd gleich sind (vor allem die triviale Domäne MOVIE). Beispiele für Domänen, in denen die PR mit allen Heuristiken besser ist, sind PSR-LARGE, PSR-MIDDLE und OPTICAL-TELEGRAPHS. In letzter Domäne ist das durchschnittliche Verhältnis der Suchzeiten bis auf h^{cea} deutlich zugunsten der propositionalen Darstellung ausgeprägt.

Wir bilden jetzt den geometrischen Mittelwert über alle Domänen. Dieser Gesamtwert zeigt, dass die h^{CG} im Durchschnitt für Probleme in FDR dreimal so schnell einen Plan findet, wie für Probleme in der klassischen Darstellung. Für die anderen Heuristiken sind diese Werte nicht ganz so stark ausgeprägt: Die Context-enhanced additive Heuristik ist für FDR-Probleme etwa doppelt so schnell wie für Probleme in PR. Für die additive und FF-Heuristik bringt die FDR ein Verkürzung auf 63% beziehungsweise 68% der Zeiten der PR.

Für h^{CG} fällt auf, dass relativ viele Domänen mit der aussagenlogischen Darstellung kleinere Durchschnittszeiten haben. So zum Beispiel FREECELL oder PIPESWORLD-TANKAGE. Für OPTICAL-TELEGRAPHS gibt es den größten Unterschied, allerdings hatten wir bereits bemerkt, dass für diese Domäne in der FDR nur ein Problem gelöst wurde, daher Schwankungen in dem Wert recht plausibel sind und in der PR denkbar viele Probleme gelöst wurden.

Für die h^{FF} -Heuristik gibt es folgende Auffälligkeiten: In den meisten Domänen, in denen die anderen Heuristiken klar von der FDR profitieren, ist der Unterschied für h^{FF} schwächer ausgeprägt. In größeren Problemen einiger Domänen wie ROVERS und TRUCKS dauert die Suche mit FDR sogar mehr als doppelt so lang.

Die additive Heuristik und die Context-enhanced additive Heuristik sind mit der FDR in der Regel schneller zu einem Plan gekommen. Bei der letzten Heuristik ist der Unterschied der Zeiten zwischen den Varianten sehr deutlich zugunsten von FDR.

Anzahl expandierter Knoten Zwischen der Suchzeit und der Anzahl expandierter Zustände kann man einen direkten Zusammenhang vermuten: Je mehr Knoten während der Bestensuche nach einem Zielzustand expandiert werden müssen, desto länger dauert die Suche. Deshalb werden wir als nächstes diesen Teil des Ergebnis betrachten. Wenn wir

Tabelle 6: Geometrisches Mittel des Verhältnis der Suchzeiten.

Domäne	h^{CG}	h^{add}	h^{FF}	h^{cea}
AIRPORT	0.8	0.75	0.57	0.27
ASSEMBLY	1.03	0.8	0.95	0.91
BLOCKS	1.54	0.67	0.67	0.73
DEPOT	1.15	0.55	0.44	0.21
DRIVERLOG	0.15	0.7	0.42	0.36
FREECELL	5.52	0.45	0.52	0.85
GRID	0.09	0.31	0.07	0.09
GRIPPER	0.17	0.53	0.74	0.68
LOGISTICS00	0.01	0.52	0.66	0.54
LOGISTICS98	0.0	0.22	0.27	0.1
MICONIC	0.39	0.76	0.8	1.14
MICONIC-FULLADL	0.86	0.83	0.88	0.78
MICONIC-SIMPLEADL	0.35	0.57	0.81	1.11
MOVIE	1.0	1.0	1.0	1.0
MPRIME	0.0	0.45	0.46	0.07
MYSTERY	0.01	0.42	0.45	0.17
OPENSTACKS	0.53	0.58	0.4	0.42
OPTICAL-TELE.	19.06	3.93	4.4	1.0
PATHWAYS	0.67	0.92	0.68	0.89
PHILOSOPHERS	0.04	0.4	0.69	0.54
PIPESWORLD-NOT.	0.73	0.37	0.52	1.98
PIPESWORLD-T.	2.09	0.45	0.25	1.71
PSR-LARGE	1.13	1.06	1.12	1.07
PSR-MIDDLE	1.15	1.08	1.04	1.06
PSR-SMALL	0.95	0.9	0.77	0.82
ROVERS	0.28	1.13	0.95	0.63
SATELLITE	0.05	0.99	0.6	0.39
SCHEDULE	1.28	1.11	0.54	1.28
STORAGE	0.71	0.97	0.94	0.85
TPP	0.43	0.56	0.52	0.32
TRUCKS	1.7	1.37	1.91	0.78
ZENOTRAVEL	0.15	0.4	0.4	0.31
Gesamt	0.324	0.683	0.631	0.557

von Expansion sprechen, so ist das der Vorgang, wenn für einen Zustand im Transitionssystem durch Anwendung aller möglicher Operatoren die Nachfolgezustände berechnet werden. Wird bei der heuristischen Suche ein Zustand expandiert, hat der korrespondierende Knoten im Suchbaum den aktuell niedrigsten h -Wert aller noch nicht expandierten Knoten.

Für einige ausgewählte, auffällige Wertepaare durchschnittlicher Suchzeiten möchten wir die dazugehörigen durchschnittlichen Anzahlen von Expansionen prüfen. Für die h^{FF} Heuristik sind uns vergleichsweise lange durchschnittliche Suchzeit in der Domäne TRUCKS aufgefallen. Für diese war der Mittelwert des Verhältnis der Suchzeiten von FDR zu PR 1,19 : 1. Für die Expansionen ist das Verhältnis 3,58 : 1. Für das ausgeprägt gute Abschneiden der h^{cea} -Heuristik mit FDR in Domänen wie LOGISTICS98 oder MPRIME sowie für die anderen Heuristiken lassen sich in den zugehörigen Expansionszahlen ähnliche Zusammenhänge ausmachen: Ist die Anzahl expandierter Knoten für FDR erhöht, dann ist auch die Suchzeit im Vergleich zur PR verlängert. Die Werte für alle Domänen und Heuristiken in Tabelle 7 auf der nächsten Seite bestätigen den vermuteten Zusammenhang. Allerdings ist dabei der Anteil der Anzahl von Zuständen, die bei einer Suche mit FDR expandiert werden, insgesamt erhöht. Für h^{add} und h^{FF} werden mit der FDR sogar etwas mehr Zustände expandiert, als mit der propositionalen Darstellung PR. Wir werden diese Besonderheit gleich noch besprechen, zunächst werden wir aber die Ergebnisse für die Planlänge auswerten.

Planlänge Die Planlänge ist zwar bei suboptimalen Planern nicht so wichtig wie die Suchzeit, aber die erzeugten Pläne sollten trotzdem nicht zu weit von einem optimalen Plan abweichen. Vor allem lässt die Länge der erzeugten Pläne einen Rückschluss darüber zu, wie informativ eine Heuristik in Kombination mit den unterschiedlichen Repräsentationen ist. Deshalb haben wir das geometrische Mittel des Verhältnis der berechneten Planlängen ermittelt (Tabelle 8 auf Seite 39). Es zeichnet sich für alle Heuristiken ein leichter Vorteil für die FDR ab. Nur die h^{FF} -Heuristik erzeugt mit der propositionalen Darstellung im Mittel leicht längere Pläne. Für die Domänen FREECELL und GRIPPER sind die in der PR gefundenen Pläne mit allen Heuristiken im Durchschnitt kürzer. Für die Domänen PHILOSOPHERS und ZENOTRAVEL finden alle Heuristiken bessere Pläne, wenn deren Probleme in der FDR dargestellt werden. Dies ist vor allem interessant, weil für diese Domänen bereits die durchschnittliche Suchzeit wesentlich geringer als für die PR war. Nach den Übersetzungsvorgängen sind in diesen Problemen viele Aussagenvariablen zu Variablen endlichen Wertebereichs zusammengefasst und die FDR wirkt sich sehr positiv auf den Informationsgehalt des Heuristikwertes aus, der für die Knoten im Suchbaum berechnet wird.

Vor allem die Güte der von h^{CG} erzeugten Pläne nimmt durch die Darstellung mit Finite-Domain-Variablen zu. Nur für fünf Domänen ist sie schlechter als die mit der propositionalen Darstellung erzeugten Pläne.

Die Context-enhanced additive Heuristik h^{cea} erzeugt mit der FDR in 19 von 33 Domänen durchschnittlich kürzere Pläne. Nur in sieben Domänen sind die Pläne im Schnitt länger als mit der propositionalen Darstellung. Die h^{add} -Heuristik verhält sich ähnlich.

Tabelle 7: Geometrisches Mittel der Verhältnis der Anzahl von Expansionen.

Domäne	h^{CG}	h^{add}	h^{FF}	h^{cea}
AIRPORT	0.55	1.04	0.86	0.28
ASSEMBLY	1.0	0.79	0.98	1.13
BLOCKS	1.19	0.93	1.06	0.95
DEPOT	2.03	1.22	0.77	0.25
DRIVERLOG	0.57	1.04	0.94	0.77
FREECELL	8.65	1.09	1.26	1.6
GRID	0.19	0.94	0.24	0.47
GRIPPER	0.22	0.76	1.03	0.7
LOGISTICS00	0.03	0.93	0.97	0.51
LOGISTICS98	0.0	0.65	0.66	0.2
MICONIC	0.61	0.8	0.86	1.32
MICONIC-FULLADL	1.04	1.01	1.1	1.02
MICONIC-SIMPLEADL	0.44	0.54	0.93	1.25
MOVIE	1.18	1.18	1.18	1.18
MPRIME	0.02	0.86	0.93	0.15
MYSTERY	0.07	0.88	0.96	0.23
OPENSTACKS	1.3	1.27	1.11	1.33
OPTICAL-TELE.	34.54	7.07	6.73	1.48
PATHWAYS	0.81	1.33	0.77	1.4
PHILOSOPHERS	0.04	0.7	1.23	0.62
PIPESWORLD-NOT.	0.66	1.06	1.46	0.41
PIPESWORLD-T.	2.23	1.09	0.95	1.43
PSR-LARGE	0.98	0.98	1.06	1.01
PSR-MIDDLE	1.04	1.0	1.0	0.99
PSR-SMALL	1.32	0.99	1.01	0.99
ROVERS	0.58	1.36	1.2	1.28
SATELLITE	0.12	1.17	0.78	0.99
SCHEDULE	1.14	1.15	0.49	1.01
STORAGE	0.81	1.05	1.01	0.79
TPP	0.81	1.09	1.0	0.72
TRUCKS	1.45	2.1	3.58	1.18
ZENOTRAVEL	0.38	0.78	0.87	0.44
Gesamt	0.507	1.057	1.02	0.759

Tabelle 8: Geometrisches Mittel der Planlängenverhältnisse.

Domäne	h^{CG}	h^{add}	h^{FF}	h^{cea}
AIRPORT	1.01	1.01	1.01	0.98
ASSEMBLY	0.98	1.01	1.0	0.97
BLOCKS	0.95	0.96	1.07	0.96
DEPOT	1.08	1.01	1.05	0.9
DRIVERLOG	0.93	0.99	1.04	0.99
FREECELL	1.17	1.0	1.0	1.08
GRID	0.96	1.07	0.85	0.79
GRIPPER	1.01	1.01	1.02	1.0
LOGISTICS00	0.77	0.95	1.0	0.87
LOGISTICS98	0.85	0.98	1.0	0.93
MICONIC	0.99	0.95	0.95	1.05
MICONIC-FULLADL	0.96	1.03	1.01	1.06
MICONIC-SIMPLEADL	1.0	0.9	0.97	1.06
MOVIE	1.0	1.0	1.0	1.0
MPRIME	0.93	0.97	1.0	0.88
MYSTERY	0.98	0.97	1.04	0.87
OPENSTACKS	1.0	1.0	1.0	1.0
OPTICAL-TELE.	1.0	1.0	1.0	1.0
PATHWAYS	0.99	0.99	1.0	1.0
PHILOSOPHERS	0.62	0.96	0.98	0.97
PIPESWORLD-NOT.	0.89	1.01	1.05	0.96
PIPESWORLD-T.	0.77	0.95	1.09	0.95
PSR-LARGE	0.99	0.99	1.02	0.99
PSR-MIDDLE	0.99	0.97	1.01	0.97
PSR-SMALL	1.0	1.0	1.0	1.0
ROVERS	1.0	1.01	1.01	1.02
SATELLITE	0.85	1.02	0.98	1.02
SCHEDULE	1.0	0.99	0.99	0.99
STORAGE	0.98	1.01	1.0	1.01
TPP	1.03	1.02	1.0	0.9
TRUCKS	1.0	1.02	1.0	0.98
ZENOTRavel	0.98	0.96	0.98	0.92
Gesamt	0.953	0.991	1.004	0.97

Für h^{FF} sind die durchschnittlichen Planlängen in 17 Fällen für beide Formen gleich. Diese Heuristik scheint für die Planlänge einen geringeren Vorteil aus der FDR zu ziehen.

Interpretation Wie man für die Anzahl gelöster Probleme in Abbildung 7 und Tabelle 3 sieht, ist der Unterschied in der Anzahl gelöster Probleme zwischen den Varianten bei den Heuristiken, die einen Abhängigkeitsgraphen berechnen (h^{CG} , h^{cea}) größer als für die anderen Heuristiken. Dies könnte man als Indiz dafür sehen, dass die FDR für solche Heuristiken tatsächlich einen erhöhten Informationsgehalt bringt, wie von Helmert vermutet wurde [17]. Dafür spricht auch, dass die qualitativen Vorteile dieser beiden Heuristiken in gleichen Domänen auftreten: Wenn h^{CG} in einer Domäne eine deutlich bessere Suchzeit mit der FDR hat, dann auch h^{cea} und umgekehrt. Dieser Zusammenhang gilt auch für Domänen wie SCHEDULE, in denen beide Heuristiken mit der FDR nicht so gut abschneiden.

Die FF-Heuristik scheint in einer anderen Weise oder allein durch die verringerte Anzahl an syntaktisch möglichen Zuständen von der FDR zu profitieren. Ihr Informationsgehalt ist schon für die propositionale Darstellung sehr gut und kann deswegen nicht so deutlich verbessert werden, wie bei den anderen Heuristiken. Darauf weisen die Ergebnisse hin, in denen der Vorteil in Anzahl gelöster Probleme, Suchzeit und Planlänge für h^{FF} nicht so stark ausfällt.

Auffällig war die im Vergleich zur Suchzeit weniger stark zugunsten der FDR verschobene Anzahl der expandierten Zustände (vergleiche Gesamtwerte der geometrischen Mittelwerte der Tabellen 6 auf Seite 36 und 7 auf Seite 38). Dieser Sachverhalt ist dadurch begründet, dass sich die Heuristiken, welche ihre Werte ausschließlich aus dem relaxierten Planungsgraphen berechnen (h^{add} , h^{FF}), für beide Darstellungen ähnlich verhalten, weil die FDR während der Berechnung wieder in eine aussagenlogische Form zerlegt wird.

Wie Helmert und Geffner in [19] zeigen, verhält sich die Kontext-erweiterte Heuristik h^{cea} auf Probleminstanzen in propositionaler Darstellung wie die additive Heuristik h^{add} . Dies geht aus den Daten ebenfalls hervor.

4.1.3 Verhalten von suboptimalen Planern in der Domäne ANTS

Wir hatten in der Vorstellung von ANTS die Vermutung geäußert, dass Probleme dieser Domäne für suboptimalen Planern keine Schwierigkeiten bereiten. Deshalb haben wir für dieses Experiment etwas schwierigere Probleme mit vielen Tischen und Ameisen getestet. Es wurden für die selben Heuristiken auf Kombinationen von $n \in \{20, 40, \dots, 120\}$ Tischen und $m \in \{20, 40, \dots, 100\}$ Ameisen untersucht.

Erwartungsgemäß sollten alle Planer diese 30 Probleme sowohl in FDR als auch in PR lösen. Die Suche mit h^{CG} konnte jedoch kein einziges Problem lösen. Die Suche scheitert bereits an dem Problem mit je 20 Ameisen und Tischen. Der Startzustand wird expandiert, aber innerhalb des Zeitlimits wird kein anderer Zustand mit einem niedrigeren Heuristikwert gefunden. Ein Problem mit je 5 Ameisen und Tischen kann mit der h^{CG} -Heuristik jedoch gelöst werden. Vermutlich ist für die propositionale Darstellung von ANTS-Problemen der Abhängigkeitsgraph zu komplex, so dass die Heuristik nicht in ausreichend kleiner Zeit berechnet werden kann. Alle anderen Kombinationen von

Heuristiken und Darstellungen lösen alle Probleme. Diese anderen Heuristiken lassen sich für beide Darstellungen schnell berechnen und schätzen die tatsächlichen Kosten h^* wahrscheinlich sehr gut ab. Ein Indiz dafür ist die Planlänge: Alle gefundenen Pläne haben die optimale Länge $(n - 1) \cdot m$.

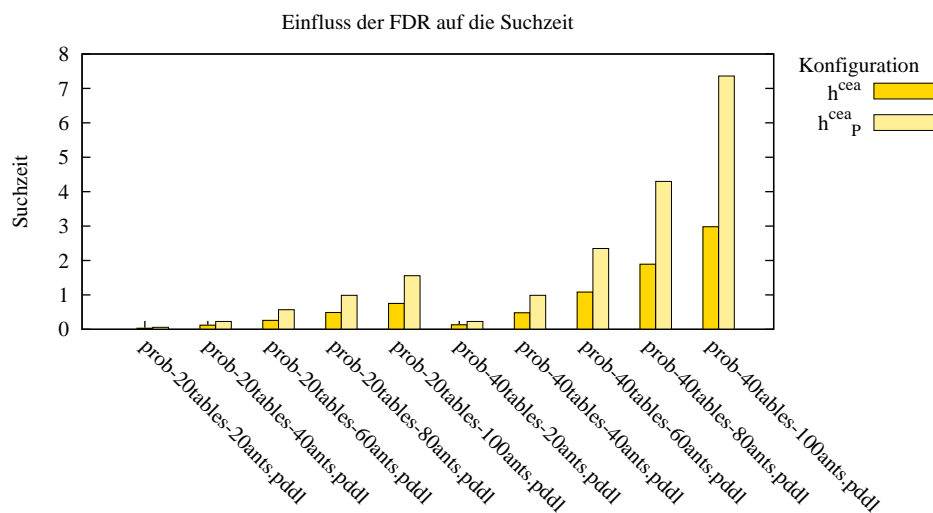
Alle Heuristiken, bei denen ein Vergleich der Suchzeiten möglich war, haben die FDR Variante der Probleme immer schneller gelöst als die PR Variante. Dementsprechend sind die durchschnittlichen Suchzeiten für FDR kürzer (Tabelle 9). Bei allen Problemen ist die Anzahl expandierter Knoten für die verschiedenen Heuristiken gleich. Allerdings werden während der Suche mit der Finite-Domain-Repräsentation immer 2% mehr Knoten expandiert.

Im Abschnitt über die Domäne ANTS äußerten wir die Vermutung, dass die Anzahl der Tische die Ausprägung des Unterschieds zwischen der Finite-Domain- und der propositionalen Repräsentation beeinflusst. Das Histogramm 8 stellt diesen Sachverhalt für die Context-enhanced additive Heuristik mit den Tischzahlen 20 und 40 sehr anschaulich dar.

Tabelle 9: Geometrischer Mittelwert der Suchzeiten und Anzahl expandierter Zustände von suboptimalen Planern in ANTS.

ANTS-SATISFYING	h^{CG}	h^{add}	h^{FF}	h^{cea}
Suchzeit	-	0.62	0.58	0.34
Expansionen	-	1.02	1.02	1.02

Abbildung 8: Entwicklung der Suchzeit bei veränderter Anzahl Tische und Ameisen.



4.1.4 Zusammenfassung

Fassen wir die Ergebnisse der Experimente noch einmal zusammen: Wir haben gesehen, dass mit der Finite-Domain-Repräsentation alle Heuristiken in den meisten Domänen mehr Probleme lösen und die Suchzeit, sofern sie sich vergleichen ließ, auf 68 bis 32% verkürzt ist. Auch die Güte der gefunden Pläne verbessert sich in der Regel. Die Anzahl der expandierter Zustände kann sich in der FDR verringern. Die kürzere mittlere Suchzeit ist wahrscheinlich auf die schneller zu berechnenden Heuristikwerte durch die verringerte Anzahl von Variablen zurückzuführen, aber auch ein höherer Informationsgehalt der Heuristiken kann dafür verantwortlich sein.

Trotzdem haben wir festgestellt, dass die FDR nicht immer zu einer besseren Performance der Planer führt. Es gibt einige Domänen, für die sich eine Darstellung in FDR anscheinend nachteilig auswirkt. Auch unterscheiden sich Art und Stärke des Vorteils zwischen den Heuristiken. Die h^{CG} - und h^{cea} -Heuristiken profitieren stärker, die h^{FF} - und h^{add} -Heuristiken weniger stark von einer Finite-Domain-Darstellung der Planungsprobleme.

Anmerkungen zur Aussagekraft der Statistiken Über den Wert der aufgeführten Statistiken kann gestritten werden. Insbesondere kann es sein, dass die untersuchten Domänen, die einen Vorteil für die FDR bringen, die Gesamtheit *aller* Planungsaufgaben

Tabelle 10: Domänen für die Experimente mit optimalen Planern

AIRPORT	GRIPPER	MYSTERY
BLOCKS	LOGISTICS00	PSR-SMALL
DEPOT	MICONIC	ROVERS
DRIVERLOG	MPRIME	SATELLITE
FREECELL	PIPESWORLD-NOTANKAGE	TPP
GRID	PIPESWORLD-TANKAGE	ZENOTRAVEL

schlecht repräsentieren. Leider ist bisher nicht bekannt, wie diese Gesamtheit aussieht. Deshalb müssen wir mit einem Querschnitt über die Domänen der IPC vorlieb nehmen. Unsere Ergebnisse lassen sich also mindestens auf diese und die daraus abgeleiteten Domänen und Probleme verallgemeinern.

4.2 Optimale Planer

4.2.1 Versuchsaufbau

Für die Untersuchung des Einflusses der Finite-Domain-Repräsentation auf Planungssysteme, die einen optimalen Plan erzeugen, haben wir wieder vier unterschiedliche Suchverfahren evaluiert. Neben dem von uns implementierten BDD-Planer zur symbolischen Exploration haben wir drei heuristische Suchen verwendet. Wird optimales Planen mit einer Single-State-Suche unter Verwendung einer Heuristik durchgeführt, so muss diese Heuristik zulässig sein und der Zustandsraum muss mit einer A^* -Suche durchlaufen werden. Eine Heuristik wird als zulässig bezeichnet, wenn sie für alle Zustände x die tatsächlichen Kosten höchstens unterschätzt: $h(x) \leq h^*(x)$. Die im Experiment zum Einsatz gekommenen Heuristiken möchten wieder nur nennen. Die genauen Algorithmen zur Berechnung der Heuristiken können in den entsprechenden Veröffentlichungen nachgelesen werden. Mit der Merge&Shrink-Heuristik $h^{M\&S}$ [20] haben wir eine Abstraktionsheuristik benutzt. Die h^{max} -Heuristik [2] ist eine zulässige relaxierte Heuristik. Die dritte benutzte Heuristik ist die Landmark-Cut Heuristik $h^{LM\ cut}$ [18].

Diese Heuristiken und unser BDD-Planer wurden wieder auf eine Auswahl von Domänen der IPC 1 – 5 getestet. Für den optimalen Teil der Experimente haben wir auf Domänen mit bedingten Operatoren verzichtet und auch sonst weniger Domänen untersucht. Eine detaillierte Auflistung der verwendeten Domänen befindet sich in Tabelle 10. Alle Experimente wurden wie im Experiment mit suboptimalen Planern auf einem Rechner mit zwei Quad-Core Intel Xeon 2,66GHz Prozessoren und 32 GB Hauptspeicher durchgeführt. Dabei liefen immer vier Suchen gleichzeitig und jedem Suchprozess standen 2048 MB Speicher zur Verfügung. Die Suche wurde spätestens nach 15 Minuten abgebrochen. Für die Eingabe der Planer gelten die gleichen Voraussetzungen wie für die anderen Experimente.

4.2.2 Auswertung und Interpretation der Ergebnisse

Wir möchten die Ergebnisse ähnlich wie im suboptimalen Experiment besprechen. Wir betrachten zunächst die Anzahl der gelösten Probleme, werden dann auf die qualitativen und quantitativen Unterschiede in der Suchzeit zu sprechen kommen und die Ergebnisse anschließend bewerten. Die Betrachtung der Planlänge entfällt für optimale Planer.

Von den insgesamt 6 008 Suchvorgängen konnten 3 612 nicht erfolgreich beendet werden. Davon wurden knapp 2 500 wegen Erreichen des Speicherlimits abgebrochen und die restlichen konnten die Suche nicht vor Ablauf der Zeit erfolgreich beenden. Die BDD-basierte und die $h^{M\&S}$ -Suche scheitern in erster Linie an den Speicherbeschränkungen, während die Suchen mit $h^{LM\ cut}$ und h^{max} sowohl den Speicher aufbrauchen als auch das Zeitlimit überschreiten.

Gelöste Probleme In den Ergebnissen fällt sofort auf, dass die Unterschiede viel einheitlicher ausfallen, als bei den suboptimalen Planern. Alle Planer konnten mit der Finite-Domain-Repräsentation mehr Probleme lösen. Einzige Ausnahme ist die Domäne AIR-PORT. Hier wurden von unserem BDD Planer sieben Probleme in propositionaler Darstellung und nur fünf in Finite-Domain-Repräsentation gelöst. Von den 751 Problemen hat unser BDD-Planer 306 Probleme in FDR und 219 Probleme in PR gelöst. Die $h^{M\&S}$ -Heuristik hat 286 beziehungsweise 219, h^{max} hat 271 beziehungsweise 264 und mit der Landmark-Cut-Heuristik wurden 421 beziehungsweise 401 Probleme gelöst. Der Vorteil aus der FDR ist für BDD-basierte Suche mit 40% mehr gelösten Problemen und $h^{M\&S}$ (30%) deutlicher ausgeprägt als für h^{max} und $h^{LM\ cut}$ (1% bzw. 5%). Die Domäne mit der geringsten Verbesserung der Anzahl gelöster Probleme durch die FDR ist SATELLITE. Diese Ergebnisse sind in Tabelle 11 und Abbildung 9 dargestellt.

Der BDD-Planer hat wie erwartet deutlich mehr Probleme in der FDR gelöst, als in der klassischen Darstellung. Beispielhaft dafür ist die Anzahl von Lösungen für GRIPPER. In der PR kann der Operator-BDD nur für die ersten vier Probleme innerhalb des Zeit- und Speicherlimits erstellt werden. Diese Domäne zeigt auch einen Vorteil der symbolischen Suche gegenüber den Single-State-Suchen auf: In GRIPPER können zu jedem Zeitpunkt viele Aktionen angewendet werden, von denen auch noch ein Großteil optimal ist. Dadurch hat der Suchbaum für die heuristischen Suchen einen großen Wachstumsfaktor. Wird eine A^* -Suche durchgeführt, müssen diese Suchmethoden alle anderen Knoten mit $f \leq h^*$ erkunden, bevor sie sicher sein können, einen optimalen Plan gefunden zu haben. Dadurch kommen die drei eingesetzten Heuristiken in GRIPPER zu wenigen Lösungen.

Abbildung 9: Anzahl durch die optimalen Planer gelöster Probleme (von 751)

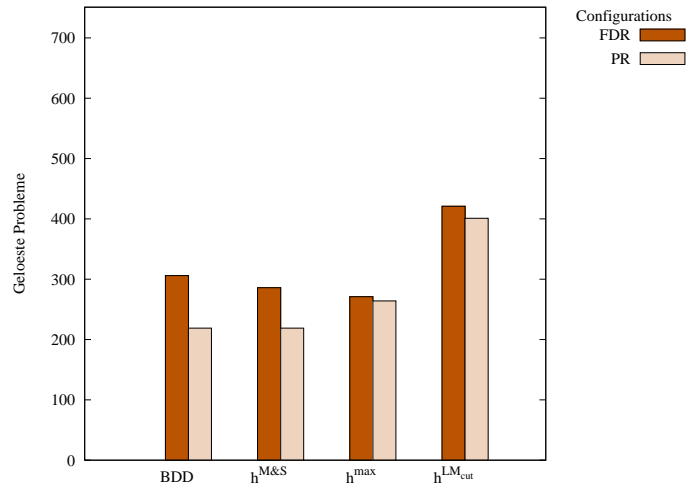


Tabelle 11: Anzahl gelöster Probleme für das Experiment mit optimalen Planern.

Domäne (# Pr.)	BDD	BDD_P	$h^{M\&S}$	$h_P^{M\&S}$	h^{max}	h_P^{max}	$h^{LM_{cut}}$	$h_P^{LM_{cut}}$
AIRPORT (50)	5	7	18	8	21	20	38	28
BLOCKS (35)	18	6	18	16	18	18	28	27
DEPOT (22)	3	1	5	2	4	4	7	7
DRIVERLOG (20)	11	8	12	10	8	7	14	13
FREECELL (80)	10	6	14	7	15	14	15	14
GRID (5)	0	0	2	0	2	2	2	2
GRIPPER (20)	20	4	7	7	7	7	6	6
LOGISTICS00 (28)	16	16	16	13	10	10	20	16
MICONIC (150)	98	76	52	52	50	50	140	140
MPRIME (35)	8	2	20	5	24	21	24	24
MYSTERY (30)	7	4	13	9	15	15	17	17
PIPES-N. (50)	13	6	18	8	16	16	17	17
PIPES-T. (50)	11	7	12	6	10	10	11	9
PSR-SMALL (50)	49	44	50	50	48	48	48	48
ROVERS (40)	11	9	6	6	5	5	7	7
SATELLITE (36)	7	7	6	6	4	4	9	8
TPP (30)	11	9	6	6	6	5	6	6
ZENOTRAVEL (20)	8	7	11	8	8	8	12	12
Summe (751)	306	219	286	219	271	264	421	401

Suchzeit Zunächst fassen wir die Suchzeiten wie in Abschnitt 4.1 zusammen. Die Suchzeiten für Probleme, die mit beiden Codierungs-Varianten gelöst wurden, werden vergli-

chen und die Variante mit der kürzeren Zeit erhält einen Wertungspunkt. Die Ergebnisse sind leicht zusammen zu fassen: Die FDR erzielt in den meisten Fällen eine höhere Wertung. Auch in der durchschnittlichen Suchzeit zeigt sich der Vorteil der FDR: Lediglich für fünf Problem-Planer-Kombinationen ist die Lösung mit FDR im Durchschnitt nicht schneller gefunden als für die PR. Auch hier haben wir für das Verhältnis der Suchzeiten den geometrischen Mittelwert berechnet (Tabelle 12 auf der nächsten Seite).

Die $h^{M\&S}$ -Heuristik hat ohne Ausnahme einen signifikant besseren Zeitwert mit der FDR erreicht. Das geometrische Mittel des Verhältnis der Suchzeiten von Finite-Domain-Repräsentation zur klassischen Darstellung ist für diese Heuristik 0,079. So stark bezog keine andere untersuchte Suchmethode einen Vorteil aus der FDR.

Die h^{max} -Heuristik kann Probleme in der FDR auch schneller lösen, allerdings ist der Unterschied weniger stark ausgeprägt und in der Domäne TPP wurden mit der klassischen Darstellung schneller Pläne gefunden.

Der *BDD*-Planer fand im Durchschnitt mit der FDR mehr als doppelt so schnell einen Plan wie mit der PR. Ausnahme ist die Domäne MICONICS. Hier ist das mittlere Verhältnis der Suchzeiten aber nur minimal unterschiedlich von 1 und der Planer löst weit mehr Probleme in FDR.

Die $h^{LM_{cut}}$ -Heuristik hat in PIPESWORLD-TANKAGE und MYSTERY mit der PR schneller einen Plan gefunden. In den Ergebnissen fällt auf, dass bei der Suche mit h^{max} und $h^{LM_{cut}}$ für beide Problemdarstellungen fast immer ähnlich viele Zustände expandiert wurden. Trotzdem ist die Suchzeit unterschiedlich, was auf eine unterschiedliche Laufzeit der Heuristik-Berechnung hindeutet. Eine detailliertere Auflistung befindet sich in Tabelle 13 auf Seite 48.

Tabelle 12: Geometrischer Mittelwert der Suchzeitverhältnisse von beiden Varianten gelöster Probleme.

Domäne	BDD	$h^{M\&S}$	h^{max}	h^{LMcut}
AIRPORT	0.92	0.0	0.46	0.93
BLOCKS	0.31	0.05	0.59	0.52
DEPOT	1.0	0.0	0.81	0.54
DRIVERLOG	0.34	0.04	0.6	0.66
FREECELL	0.25	0.07	0.62	0.59
GRID	-	-	0.67	0.87
GRIPPER	0.2	0.11	0.63	0.41
LOGISTICS00	0.54	0.05	0.39	0.54
MICONIC	1.03	0.44	0.69	0.76
MPRIME	0.05	0.02	0.54	0.86
MYSTERY	0.3	0.05	0.32	1.11
PIPES-T.	0.96	0.5	0.81	1.2
PIPES-NOT.	0.35	0.03	0.67	0.54
PSR-SMALL	0.43	0.06	0.66	0.69
ROVERS	0.57	0.85	0.58	0.6
SATELLITE	0.42	0.59	0.49	0.69
TPP	0.48	0.82	1.47	0.84
ZENOTRAVEL	0.69	0.11	0.72	0.64
Gesamt	0.426	0.079	0.619	0.696

Interpretation Dass die optimalen Planer eindeutiger von der Finite-Domain-Repräsentation profitieren wird unterschiedliche Gründe haben. Eine Eigenschaft der FDR könnte Ursache für das bessere Abschneiden der heuristischen Suchen sein: Viele syntaktisch mögliche Zustände des propositionalen Planungsproblems, die semantisch nicht erlaubt sind, sind durch Einführen der FDR-Variablen unter Berücksichtigung der Mutex-Gruppen in der Finite-Domain-Darstellung des Planungsproblems nicht mehr vorhanden.

Für die heuristischen Suchen liegt der Vorteil vor allem an dem gesteigerten Informationsgehalt, die ein Zustand in FDR für die zulässigen Heuristiken bietet. Diese Heuristiken müssen die tatsächlichen Kosten h^* immer unterschätzen. Um im Zustandsraum aber schnell ein Ziel zu finden, müssen die Heuristikwerte trotzdem möglichst nah an h^* heran kommen.

Mit den Variablen endlichen Wertebereichs kann die Heuristik $h^{M\&S}$ aussagekräftigere Abstraktionen durchführen, wie Malte Helmert in [17] vermutet hat. So können mehr Probleme gelöst werden, bevor das Speicherlimit überschritten wird. Eine informativere Heuristik sorgt während der Suche dafür, dass weniger Zustände expandiert werden müssen. Dies spiegelt sich in den Ergebnisse (siehe Tabelle 13) deutlich wieder.

Dass die h^{LMcut} -Heuristik weniger deutlich Nutzen aus der FDR zieht, könnte daran liegen, dass diese Heuristik bereits einen sehr guten Informationsgehalt hat. Ähnlich wie

Tabelle 13: Geometrischer Mittelwert des Verhältnis der Anzahl expandierter Knoten.

Domäne	$h^{M\&S}$	h^{max}	$h^{LM_{cut}}$
AIRPORT	0.52	1.0	1.74
BLOCKS	0.47	1.01	0.98
DEPOT	0.46	1.0	0.76
DRIVERLOG	0.02	1.08	1.01
FREECELL	0.28	1.0	0.83
GRID	-	0.99	0.99
GRIPPER	0.54	1.0	1.0
LOGISTICS00	0.03	1.0	1.0
MICONIC	0.78	1.0	1.0
MPRIME	0.1	0.76	0.89
MYSTERY	1.21	0.76	0.93
PIPES-NOT.	0.74	1.0	1.0
PIPES-T.	0.48	1.0	1.33
PSR-SMALL	0.54	1.0	0.99
ROVERS	1.0	1.04	1.12
SATELLITE	0.76	0.98	1.11
TPP	0.94	1.0	1.0
ZENOTRAVEL	0.26	1.0	1.06
Gesamt	0.387	0.976	1.025

die FF-Heuristik im Experiment für suboptimale Planer löst sie auch in der klassischen Darstellung schon deutlich mehr Probleme als die zweitbeste Methode.

Die h^{max} -Heuristik hat ebenfalls nur einen relativ geringen Vorteil aus der FDR. Vermutlich wird der relaxierte Planungsgraph schneller berechnet, den die Heuristik zur Berechnung ihres Schätzwertes benutzt. Daher sind die Suchzeiten leicht verbessert. Der Informationsgehalt der Heuristik nimmt aber nicht deutlich zu, sonst würde auch diese Heuristik mehr Probleme schneller lösen können. Wir haben im Abschnitt der Auswertung suboptimaler Experimente bereits angemerkt, dass Relaxierungsheuristiken die FDR vor der Erstellung des relaxierten Planungsgraphen auflösen. Dadurch kann man den geringen Performancegewinn von h^{max} begründen. Die Verbesserung der Landmark-cut Heuristik ist allerdings bemerkenswert und wurde so nicht erwartet. Sie Bedarf einer Erklärung, die wir im Rahmen dieser Arbeit aber nicht suchen werden.

Es wurde im Abschnitt 3.1 bereits besprochen, welchen Vorteil eine kompakte Variablen-darstellung für die Suche mit BDDs hat: Die Größe der verwendeten Strukturen und somit die Komplexität der Operationen zur symbolischen Exploration des Zustands-raumes kann kleiner gehalten werden.

4.2.3 Verhalten von optimalen Planern in der Domäne ANTS

Für die Untersuchung von ANTS mit optimalen Planern erwarteten wir, dass auch mit den bisher gut abschneidenden Heuristiken wie h^{LMcut} nur kleine Probleme gelöst werden können, weil von jedem Zustand mehrere unterschiedliche, aber jeweils optimale Pläne zum Ziel führen. Deshalb haben wir Kombinationen aus $n \in \{5, 10, \dots, 35\}$ Tischen und $m \in \{5, 10, \dots, 35\}$ Ameisen getestet.

Das Ergebnis aus den IPC-Domänen wird bestätigt: Alle Planer haben in ANTS mehr Instanzen gelöst, wenn diese in der Finite-Domain-Repräsentation vorlagen (Tabelle 14). Die Suche mit $h^{M\&S}$ bezieht aus der kompakten Darstellung einen deutlichen Vorteil und löst wesentlich mehr Probleme und findet viel schneller einen Plan. So ist die durchschnittliche Suchzeit für die FDR 0,1 und für die PR 74,27 Sekunden. In der FDR-Darstellung werden Abstraktionen auf einzelne Ameisen durchgeführt, diese sind additiv und daraus kann die perfekte Heuristik berechnet werden. Man kann sich leicht vorstellen, dass es für eine einzelne Ameise nur einen einzigen optimalen Plan gibt.

Für die Heuristiken h^{max} und h^{LMcut} ist der Vorteil erneut nicht so stark ausgeprägt. Die Anzahl von Expansionen ist für beide Darstellungen gleich. Trotzdem sind die Suchzeiten unter Verwendung von FDR in etwa halbiert (Tabelle 15).

Der BDD-Planer kommt auch mit der aussagenlogischen Darstellung der Probleme überraschend gut zurecht und löst relativ viele Probleme. Die Suchzeiten sind für alle gelösten Probleme im Vergleich mit den anderen Heuristiken wesentlich schneller. Die Struktur des Problems mit sehr vielen Operatoren in der variablenfreien Darstellung lässt sich mit einer symbolischen Exploration besser bewältigen. Allerdings hat unser Planer Probleme, sobald die Transitionsrelation zu groß wird und bricht wegen Erreichen des Speicherlimits bei größeren Problemen schnell ab. Für den BDD-Planer zeigen die Histogramme in Abbildung 10 (a) und (b) auf Seite 51 einmal mehr die Auswirkung einer veränderten Anzahl Tische n auf die unterschiedliche Performance des Planers.

Tabelle 14: ANTS-OPTIMAL: Anzahl gelöster Probleme.

	BDD	BDD_P	$h^{M\&S}$	$h_P^{M\&S}$	h^{max}	h_P^{max}	$h^{LM_{cut}}$	$h_P^{LM_{cut}}$
ANTS-OPT. (49)	32	27	32	5	7	5	39	33

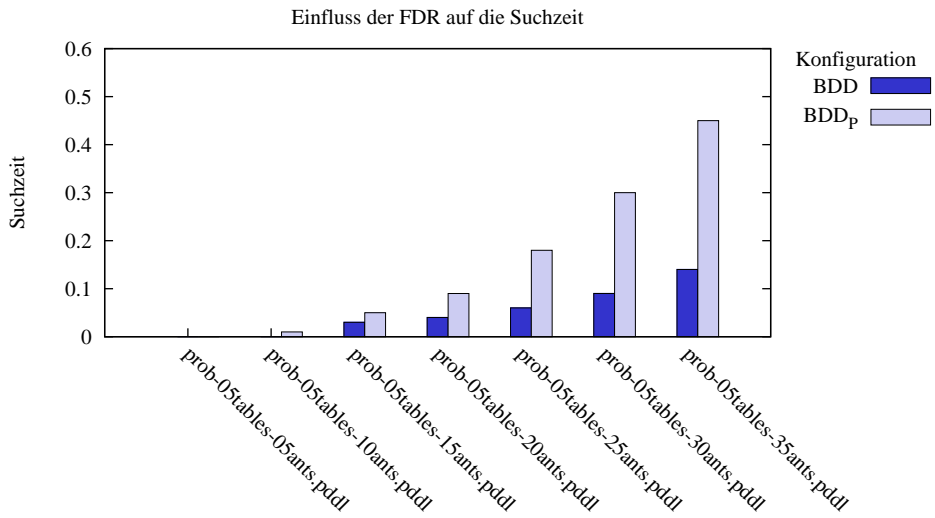
Tabelle 15: ANTS-OPTIMAL: Geometrischer Mittelwert der Suchzeit- und Expansionsverhältnisse.

ANTS-OPTIMAL	BDD	$h^{M\&S}$	h^{max}	$h^{LM_{cut}}$
Suchzeit	0.25	0.01	0.48	0.51
Expansionen	-	0.0	1.0	1.0

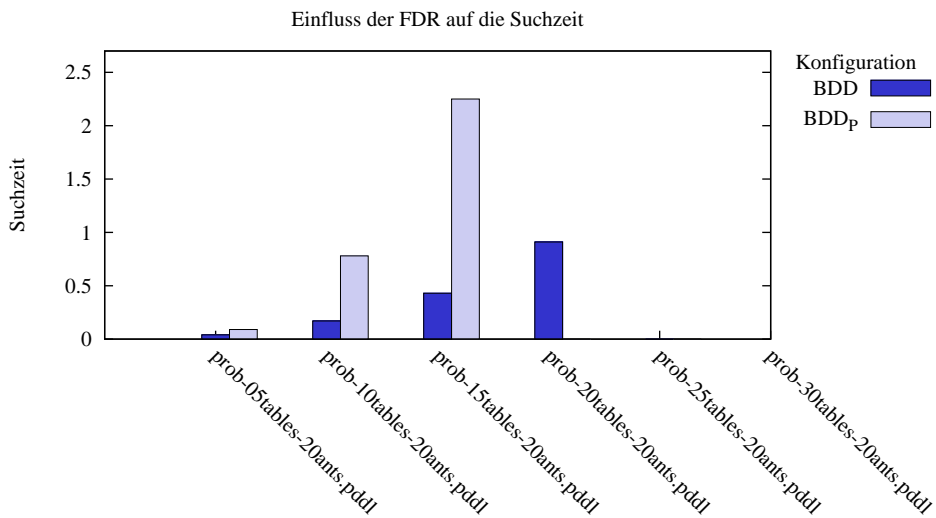
4.2.4 Zusammenfassung

Für die optimalen Planer ist die Auswirkung der Finite-Domain-Repräsentation eindeutiger als für die suboptimalen Planer: Die Suchzeit und der Speicherbedarf der heuristischen Suchen sind durch informativere Schätzwerte und die verringerte Anzahl von Variablen reduziert und deshalb werden mehr Probleme gelöst. Für die Suchen mit $h^{LM_{cut}}$ und h^{max} verkürzt sich die Suchzeit auf 70 beziehungsweise 62%. Für die Suche mit der $h^{M\&S}$ -Heuristik verbessert sich die mittlere Suchzeit auf 8% des Wertes für die aussagenlogische Darstellung. Das liegt daran, dass die Darstellung eines Planungssystems in FDR zu viel sinnvollerer Abstraktionen führt, als wenn man Abstraktionen für ein propositionales Planungssystem durchführt. Dadurch können aussagekräftigere Schätzungen berechnet werden. Die BDD-basierte Suche hat unter Verwendung der Finite-Domain-Repräsentation eine auf 42% verkürzte Suchzeit und ist in der Lage, mehr Probleme zu lösen. Den Vorteil des BDD-Planers hatten wir erwartet und bereits begründet.

Abbildung 10: Die Entwicklung der Suchzeit bei veränderten Parametern.



(a) Mehr Ameisen bei gleich vielen Tischen.



(b) Mehr Tische bei gleich vielen Ameisen.

5 Fazit

Das Ziel dieser Arbeit war, die Auswirkungen der Finite-Domain-Repräsentation auf die Leistung von Planungssystemen zu untersuchen und nach Möglichkeit zu einer konkreten Aussage über den Vorteil zu kommen. Für den Vergleich von Suchmethoden für optimale Pläne sollte ein BDD-Planer entwickelt werden. Das zweite Ziel konnten wir ohne Einschränkungen umsetzen. Der implementierte Planer läuft korrekt und kann zur Lösungssuche für alle Planungsprobleme der IPC 1–5 ohne bedingte Effekte benutzt werden. Mit ANTS haben wir uns eine Domäne ausgedacht, welche hervorragend geeignet ist, um die Performance der herkömmlichen Repräsentation von Planungsaufgaben im Vergleich mit der Finite-Domain-Repräsentation darzustellen.

Wir konnten mit den Experimenten den allgemeinen Performancevorteil der Finite-Domain-Repräsentation bestätigen. Für die suboptimalen Planer können wir sagen, dass diese für viele Domänen einen Vorteil aus der kompakten Darstellung von Variablen ziehen. Es können mehr Probleme gelöst werden, die Suchzeit ist reduziert und die unter Verwendung der FDR gefundenen Pläne sind zudem häufiger besser, als die mit der propositionalen Darstellung berechneten. Zwischen den untersuchten Heuristiken gibt es Unterschiede, wie stark sich die Performance verbessert. Außerdem gibt es einige Domänen, für die alle Heuristiken schlechter abschneiden, wenn sie die FDR benutzen. Für die optimalen Planer war der Performancevorteil einheitlicher. Die Anzahl gelöster Probleme ist für alle Suchverfahren unter Verwendung der FDR größer und die Suchzeit meistens verkürzt. Auch hier wirkt sich die Finite-Domain-Repräsentation unterschiedlich auf die Heuristiken aus. Vor allem die BDD-basierte Suche und die A^* -Suche mit Merge- & Shrink-Abstraktionen verzeichnen eine deutlich verbesserte Performance.

Für die betrachteten Suchverfahren konnten wir die Stärke des Vorteils für einige Heuristiken konkretisieren. Die FDR wirkt sich umso besser aus, je mehr durch die kompaktere Darstellung die Berechnung der Heuristik vereinfacht wird, der Informationsgehalt der Heuristik erhöht wird oder die FDR einen effizienteren Aufbau der zur Suche verwendeten Datenstrukturen ermöglicht. Leider können wir keine verallgemeinerte Aussage geben, wie viele Probleme prozentual mit der FDR mehr gelöst werden können oder um welche Größenordnung sich die Suchzeit reduziert. Die Heterogenität der IPC-Domänen lässt keine sicheren Schlüsse ausgehend von den Resultaten dieser Arbeit auf eine „durchschnittliche“ Planungsaufgabe zu.

In Zukunft wäre es interessant, die Ursachen für die Verbesserung der Suchzeiten genauer zu untersuchen. Dazu sollte man während der Experimente zusätzliche Daten erheben. Zum Beispiel müssten detaillierte Zeitmessungen für die einzelnen Arbeitsschritte der Suchvorgänge vorgenommen werden, um eine Aussage darüber treffen zu können, ob und wie stark der Vorteil der Finite-Domain-Repräsentation aus der vereinfachten Heuristikberechnung, dem erhöhtem Informationsgehalt der Heuristiken oder dem eingeschränkten Zustandsraum hervorgeht. Ebenfalls nützlich wäre eine Verallgemeinerung der hier gezeigten Ergebnisse auf allgemeine Planungsprobleme. Dazu muss man jedoch zunächst erforschen, wie die Menge *aller* Planungsprobleme aussieht und wie diese in einer geeigneten Form ausgedrückt werden kann.

Glossar

- BDD Binary Decision Diagram, Datenstruktur
- FDR Finite-Domain-Repräsentation
- GCF Generalized Cofactor
- IPC International Planning Competition
- MIPS Model Checking Integrated Planning System
- PDDL Planning Domain Description Language
- UMOP Universal Multi-agent OBDD-based Planner
- $\Pi = \langle A, I, O, G \rangle$ Planungssystem

Literatur

- [1] Bernd Becker, Rolf Drechsler, and Paul Molitor. *Technische Informatik. Eine Einführung*. Pearson Studium, München, 2005.
- [2] Blai Bonet and Hector Geffner. Planning as heuristic search: New results. In S. Biundo and M. Fox, editor, *ECP '99: Proceedings of the 5th European Conference on Planning*, pages 359–371, Durham, UK, 1999. Springer: Lecture Notes on Computer Science.
- [3] Blai Bonet and Hector Geffner. Planning as heuristic search. *Journal of Artificial Intelligence Research*, 129(1-2):5–33, 2001.
- [4] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.
- [5] J. R. Burch, E. M. Clarke, and D. E. Long. Symbolic model checking with partitioned transition relations. pages 49–58. North-Holland, 1991.
- [6] Tom Bylander. The computational complexity of propositional STRIPS planning. *Journal of Artificial Intelligence Research*, 69:165–204, 1994.
- [7] Olivier Coudert, Christian Berthet, and Jean Christophe Madre. Formal boolean manipulations for the verification of sequential machines. In *EURO-DAC '90: Proceedings of the conference on European design automation*, pages 57–61, Los Alamitos, CA, USA, 1990. IEEE Computer Society Press.
- [8] Olivier Coudert, Christian Berthet, and Jean Christophe Madre. Verification of synchronous sequential machines based on symbolic execution. In *Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems*, pages 365–373, London, UK, 1990. Springer-Verlag.

- [9] Stefan Edelkamp. Optimal symbolic PDDL3 planning with MIPS-BDD. 5th International Planning Competition Booklet (IPC-2006), Lake District, England, July 2006.
- [10] Stefan Edelkamp and Malte Helmert. Exhibiting knowledge in planning problems to minimize state encoding length. In *ECP '99: Proceedings of the 5th European Conference on Planning*, pages 135–147. Springer, 1999.
- [11] Stefan Edelkamp and Malte Helmert. The model checking integrated planning system (MIPS). *AI Magazine*, 22:200–1, 2000.
- [12] Stefan Edelkamp and Malte Helmert. On the implementation of MIPS. In *Proceedings of AIPS-00 Workshop on Model Theoretic Approaches to Planning*, pages 18–25. AAAI press, 2000.
- [13] John V. Franco, Michal Kouril, John S. Schlipf, Jeffrey Ward, Sean Weaver, Michael Dransfield, and W. Mark Vanfleet. SBSAT: a state-based, BDD-based satisfiability solver. In *SAT*, pages 398–410, 2003.
- [14] Malte Helmert. Implementation eines Planers zu symbolischen Exploration mit binären Entscheidungsdiagrammen, 1999.
- [15] Malte Helmert. A planning heuristic based on causal graph analysis. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS 2004)*, pages 161–170, 2004.
- [16] Malte Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.
- [17] Malte Helmert. Concise finite-domain representations for PDDL planning tasks. *Journal of Artificial Intelligence Research*, 173(5-6):503–535, 2009.
- [18] Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: What’s the difference anyway? In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS 2009)*, pages 162–169, 2009.
- [19] Malte Helmert and Hector Geffner. Unifying the causal graph and additive heuristics. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS 2008)*, pages 140–147, 2008.
- [20] Malte Helmert, Patrik Haslum, and Jörg Hoffmann. Flexible abstraction heuristics for optimal sequential planning. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS 2007)*, pages 176–183. AAAI Press, 2007.
- [21] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

- [22] Rune M. Jensen. A comparison study between the CUDD and BuDDy OBDD package applied to AI-planning problems. Technical report, School of Computer Science, Carnegie Mellon University, Pittsburgh, 2002.
- [23] Rune M. Jensen and Manuela M. Veloso. OBDD-based universal planning: Specifying and solving planning problems for synchronized agents in non-deterministic domains. In M. Wooldrige and M. Veloso, editors, *Artificial Intelligence Today: Recent Trends and Developments*, pages 212–248. Springer-Verlag, 1999.
- [24] Rune M. Jensen and Manuela M. Veloso. OBDD-based deterministic planning using the UMOP planning framework. In *Proceedings of the AIPS-00 Workshop on Model-Theoretic Approaches to Planning*, pages 26–31, 2000.
- [25] Martin Kreuzer and Stefan Kühling. *Logik für Informatiker*. Pearson Studium, München, 2006.
- [26] J. Lind-Nielsen. BuDDy: Binary decision diagram package. Technical report, Department of Information Technology, Technical University of Denmark, 1999.