Seminararbeit

Planen als heuristische Suche

Planning as heuristic Search

Yan Yao

19. Januar 2007

Angewandte Informatik
insbesondere Kognitive Systeme

Universität Bamberg

Gutachterin: Prof. Dr. Ute Schmid

Inhaltsverzeichnis

1 Einführung

Planen beschäftigt sich mit der Auswahl und Organisation von Aktionen und generiert als Ergebnis eine Sequenz von Aktionen, um den aktuellen Zustand in den Zielzustand zu überführen.
Dieser Prozess beansprucht Zeit und Ressourcen mehr oder weniger stark je nach der Situation. Mit steigender Problemgröße tendiert der Aufwand für ein Planen zu einer unerträglichen Höhe, besonders wenn man die beschränkte Zeit und die knappen Ressourcen betrachtet. Die Anforderungen an die Wirtschaftlichkeit des Planens fordern Strategien und Methoden, um die Planungsprozesse zu beschleunigen und gleichzeitig die Güte des Planungsergebnisses zu erhalten.

Im klassischen Planen, das durch vollständige Information, endliches, deterministisches, statistisches System, beschränkte Ziele, implizite Zeit, sequentiellen Plan und Offline-Planen charakterisiert ist, bewegt man sich im Zustandsraum und versucht mit Heuristiken die Suche zu beschleunigen. Die erfolgreiche Performanz der heuristischen Suchplaner bei AIPS-Wettbewerben hat diese Richtung bestätigt.
Das Schlagwort „Planen als heuristische Suche“[1] stammt von den beiden Forschern Bonet und Geffner. Dies basiert auf der Grundidee, ein Planungsproblem in ein Problem der heuristischen Suche zu transformieren. Das Grundkonzept beinhaltet hauptsächlich 3 Aspekte:

1. Probleme werden in einem Zustandsraum modelliert und mit einer allgemeinen deklarativen Sprache repräsentiert.
2. Heuristiken werden direkt aus der Problemrepräsentation automatisch gewonnen.
3. Die Suche im Zustandsraum wird durch die Heuristiken gesteuert.

Folgend wird auf die Problemrepräsentation im STRIPS-Zustandsraum, Herleitung der Heuristik und die Steuerung der Heuristiken in der Suche durch den Zustandsraum eingegangen. Der besonders erfolgreiche heuristische Planer, HSP 2.0, wird in einem eigenen Kapitel vorgestellt und ein kleines Beispiel aus der Domäne Block-World wird benutzt, um das Funktionsprinzip von HSP 2.0 zu veranschaulichen. Zum Schluss wird der Nachfolger von HSP Fast-Forward-Planer erwähnt, um die Weiterentwicklung der heuristischen Suche im Zustandsraum aufzuzeigen.

2 STRIPS-Zustandsraum als Suchraum

2.1 Problemrepräsentation

Um überhaupt einen Planungsalgorithmus bzw. Planer anwenden zu können muss das Problem modelliert und in einer Sprache repräsentiert werden. Im Kontext des Zustandsraummodells sind Zustände, Aktionen und Ziele als Input für den Planungsalgorithmus in einer formalen Sprache zu repräsentieren.

An die Repräsentationssprache bzw. die so genannte „Planning Languages“ gibt es folgende Anforderungen [2]:

1. Die Sprache soll die Möglichkeit bieten, dass der Planungsalgorithmus bzw. Planer die logische Struktur des Problems nutzen kann.
2. Die Sprache soll möglichst ausdruckfähig sein, um umfangreiche verschiedene Probleme zu beschreiben. Dies unterstützt die Flexibilität eines domänenunabhängigen Planers.
3. Die Sprache soll auch beschränkt sein, damit der Planer effizient die Ausdrücke anwenden kann.

2.2 Zustandsmodell

Ein Zustandsmodell besteht aus einem Zustandsraum, ein Initialzustand und einer Menge von Zielzuständen. Im Zusammenhang mit dem konzeptuellen Zustandsübergangsmodell $\sum=\mathbf{(S,A,E,\gamma)}$[3], das dem Planen zugrunde liegt, ist ein Zustandsraum sowohl ein beschränktes Modell ohne Rücksicht auf „Event“, als auch eine Erweiterung dieses konzeptuellen Modells um die Kostenfunktion.

Die formale Definition von Zustandsmodell ist ein 6-Tupel mit $\mathbf{S =} (S, s_0, S_G, A, f, c)$

- S : eine endliche, nicht leere Menge von Zuständen
- $s_0 \in S$: der Initialzustand
- S_G: Teilmenge von S, nicht leere Menge von Zielzuständen
- A(s):Teilmenge von A, alle Aktionen anwendbar auf den Zustand s mit $s \in S$
- f(a, s): eine Zustandsübergangsfunktion für alle $s \in S$ und $a \in A(s)$
- c(a, s): Kosten für die Ausführung der Aktion a im Zustand s

Das Zustandsmodell ist ein übliches Modell für Planungsproblem als heuristische Suche. Die Lösung des Planungsproblems ergibt sich als eine Sequenz von Aktionen a_0, a_1,…a_n mit einer Zustandskurve s_0, $s_1=f(s_0)$,…$s_{n+1}=f(s_n,a_n)$, wobei a_i auf s_i anwendbar und s_{n+1} ein Zielzustand ist. Die Lösung ist dann optimal wenn die gesamten Kosten als Summe der einzelnen Aktionen minimal sind, d.h.

$\sum (s_i,a_i)$ minimal, mit i= 0,…n

2.3 Zustandsmodell in STRIPS

STRIPS ist die Abkürzung für **Stanford Research Institute Problem Solver**. Diese Repräsentationssprache wird hauptsächlich für die klassischen Planer eingesetzt. STRIPS erlaubt nur positive Literale im „Zustand“ und basiert auf „Closed World Assumption“. Die Literale in „Zielen“ (Goals) müssen keine Variablen oder Funktionen sein. „Ziele“ und „Effekte“ sind nur Literale mit Und-Verknüpfungen. Auf einer Seite aufgrund dieser strengen Beschränktheit kann ein Planungsalgorithmus effizient die Ausdrücke bearbeiten, aber auf der anderen Seite fehlt dieser Sprache die Ausdrucksfähigkeit, um ein großes Spektrum an Problemen zu repräsentieren. Demzufolge sind viele Varianten von STRIPS entstanden, welche die Beschränktheit mehr oder weniger gelockert hat. STRIPS gilt heute als Teilmenge des PDDL-Standards. PDDL steht für **Planning Domain Definition Language.**

Ein Planungsproblem in STRIPS wird durch ein 4-Tupel P=(A, O, I,G) definiert.

- A: eine Menge Atome, positive Literale
- O: eine Menge Operatoren
- I und G entsprechen jeweils den Initialsituationen und Zielsituationen, Teilmenge von A
- Ein Operator op hat eine Vorbedingung: *pre(op),* eine Add-Liste: *add(op*) und eine Delete-Liste: *del(op).* In der Add-Liste stehen die hinzuzufügenden Variablen und in der Delete-Liste die zu entfernenden Variablen. Wenn man einen Operator auf einen Zustand anwendet, werden die Variablen in der Vorbedingung, Add-Liste und Delete-Liste dieses Operators durch entsprechende Konstanten in dem Zustand instanziiert wobei die Variablen an die Konstanten gebunden werden.

Entsprechend ist die Problemdefinition in STRIPS-Zustandsmodell auch ein 6-Tupel: **$Sp=(S, s_0, S_G, A(.), f, c)$.** Der Zustand wird hier auf der Ebene von Atomen bearbeitet.

- Ein Zustand s besteht aus Atomen
- Der Initialzustand s_0 besteht aus Atomen
- Zielzustände S_G befinden sich in einem Zustand, d.h. Teilmenge von einem Zustand
- Eine Aktion, eine Anwendung von einem Operator, ist dann auf einen Zustand s anwendbar, wenn deren Vorbedingung in diesem Zustand zu finden ist.
- Die Übergangsfunktion f führt einen Zustand s in einen neuen Zustand s'über, nachdem eine auf diesen Zustand anwendbare Aktion ausgeführt worden ist. Durch die Add-Liste werden Atome dem Zustand s hinzugefügt und durch die Delete-Liste werden Atome aus dem Zustand s eliminiert. Der neue Zustand s'=s-Del(a)+Add(a)
- Die Kosten von einer Aktion sind 1 d.h. c(a,s)=1

2.4 Suche im STRIPS-Zustandsraum

Der STRIPS-Zustandsraum ist jetzt zu durchsuchen, um eine Lösung des Planungsproblems zu finden. Eine systematische Suche, durch den ganzen Suchraum z.B. Breadth-First-Search, bringt immer eine Lösung und sogar die optimale Lösung wenn solche Lösung überhaupt existiert. Neben den Kriterien „Vollständigkeit" und „ Optimalität" stehen auch die Kriterien „Zeitkomplexität" und „Speicherkomplexität" um eine Suchstrategie zu bewerten. Mit steigender Problemgröße steigt die Komplexität exponentiell. Der Suchraum wird schnell riesengroß. Eine erschöpfende Durchsuchung in so einem Raum ist eher unrealistisch, wenn die Zeit und Ressourcen unerträglich stark überfordert sind.

Die Planungseffizienz und –effektivität stellt uns die Herausforderung, eine gute Lösung schnell zu finden, indem man die Größe des Suchraums reduziert und gleichzeitig die Suchqualität sichert und sogar verbessert.

Es besteht die Notwendigkeit, eine gute Technik einzusetzen, um die Suche zu steuern. Hier wenden wir uns der Heuristik zu.

3 Heuristik steuert die Suche im Zustandsraum

3.1 Was ist Heuristik?

Das Wort „Heuristik“ stammt aus dem Altgriechischen *heuriskein* und bedeutet *finden, entdecken.* Heute werden unter „Heuristik“ im Allgemeinen Strategien verstanden, die das Auffinden von Lösungen beschleunigen können. Sie wird auch häufig als Faustregel benannt. Eine Faustregel ist meist schneller und einfacher als die Berechnung der exakten Lösung. J. Pearl definiert sie in [5] als Kriterien, Methoden und Prinzipien in der Entscheidungsauswahl für eine Alternative, welche die höchste Effektivität verspricht, um ein bestimmtes Ziel zu erreichen. Nach Pearl soll die Heuristik die Anforderung erfüllen, einfach zu handhaben und zwischen einer guten und schlechten Auswahl richtig unterscheiden zu können. H. Kaindl nennt sie in [6] „die Kunst des Entdeckens“. Auch im Sinne einer Faustregel, die im Allgemeinen, aber nicht unbedingt immer , „die Effizienz eines Systems verbessert“.

Das ultimative Ziel von Heuristik ist die Effizienzsteigerung auf der Basis der Informationen, die man über die relevanten Problemeigenschaften gewinnt. Diese Informationen dem Planungsalgorithmus implizit mitzuteilen, liegt wiederum in der Formulierung der heuristischen Funktionen.

3.2 Charakteristiken

Die Heuristik ist zwar nützlich um eine Lösung zu finden. Aber sie garantiert nicht unbedingt eine Lösung auch wenn eine Lösung existiert, weil sie in der Regel zu einer nicht erschöpfenden Suche führt. D.h. keine Vollständigkeit bei Nutzung einer Heuristik.

Die Heuristik nutzt problemspezifische Informationen und ermöglicht damit eine günstige Verknüpfung zwischen zwei Zuständen und reduziert letzten Endes den Suchraum. Das bedeutet, einen guten Ansatz zu finden, wobei eine Menge Zustände als irrelevant betrachtet werden können.

3.3 Heuristische Funktionen steuern die Suche

Eine heuristische Funktion **h(n)** schätzt die Minimalkosten bzw. die Minimaldistanz von einem Knoten (Zustand) n zum Zielknoten (Zielzustand). Eine Bewertungsfunktion **f(n)** (auch Knoten-Selektionsfunktion) bewertet die Knoten (die Zustände) und wählt den Knoten (Zustand) mit den niedrigsten Kosten bzw. der kürzesten Distanz zum Zielknoten (Zielzustand) für die Expansion aus.

In Suchalgorithmen schließt die Bewertungsfunktion eine heuristische Funktion ein, damit die heuristische Funktion die Suche steuert. In Greedy-Best-First-Search ist f(n)= h(n), da setzt man die Bewertungsfunktion der heuristischen Funktion gleich. In A*-Suche ist f(n)=g(n)+h(n). g(n) steht für die bisher bekannten Kosten bzw. Distanz vom Startknoten (Initialzustand) zum Knoten (Zustand n. Man berechnet die Summe der beiden Funktionswerte um den Zustand n zu bewerten. Die Leistung der Suchalgorithmen ist abhängig von der Qualität der eingesetzten heuristischen Funktion. Die Effektivität einer heuristischen Funktion ist daher von großer Bedeutung.

Um die Eigenschaft der Heuristik zu veranschaulichen, wird im folgenden ein 8-Puzzle-Beispiel durchgehend behandelt.

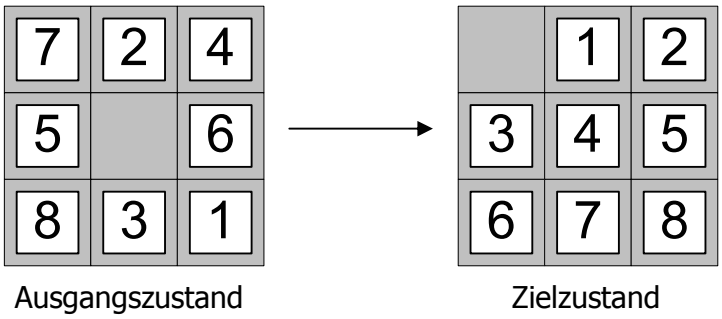

Abb. 3.1 ein 8-Puzzle-Beispiel (Quelle: [2])

Zwei heuristische Funktionen werden aus dem obigen Beispiel entwickelt. Die Heuristik basiert auf der Schätzung, wie nah ein Zustand zum Zielzustand liegt.
In der ersten heuristischen Funktion h1wird die Anzahl der deplazierten Teilchen gegenüber dem Zielzustand gezählt. h1= Anzahl der deplazierten Teilchen; In unserem Beispiel weichen 8 Teilchen von dem richtigen Platz ab. h1=8
Da die exakte Lösung 26 beträgt, ist h1 eine zulässige Funktion. Keine Überschätzung liegt vor.
Die zweite heuristische Funktion h2 berechnet die Summe der Minimaldistanz von den einzelnen Teilchen zu ihren Zielpositionen. Es wird zuerst gerechnet, wie viele Schritte ein Teilchen von dem richtigen Platz entfernt ist. Für „1“ bis „8“, die Distanz ist jeweils 3,1, 2, 2, 2, 3, 3, 2
h2=3+1+2+2+2+3+3+2=18. h2 ist somit auch zulässig.
Die Teilchen in diesem Beispiel werden voneinander unabhängig betrachtet.
Im Gegensatz zu zulässigen heuristischen Funktionen ist eine heuristische Funktion unzulässig, wenn diese die Unterschätzung der optimalen Lösung bildet.

3.4 Wie gewinnt man Heuristiken

3.4.1 „Relaxation Principle“

Das allerwichtigste Prinzip, um eine Heuristik zu gewinnen, ist das sogenannte „Relaxation Principle“. Das originale Problem wird vereinfacht, indem man dessen Bedingungen bzw. Beschränkungen lockert oder entfernt. Das vereinfachte Problem soll leichter zu handhaben sein. Wenn wir uns das vorherige 8-Puzzle-Spiel noch vor Augen halten, sieht man eindeutig, wie die beiden heuristischen Funktionen gewonnen werden.
Man gewinnt die erste Heuristik, indem man alle Bedingungen des Problems ignoriert. Jedes Teilchen könnte gleich ein anderes Teilchen aus dem Platz wegschieben und dessen Platz übernehmen. Die Spielregeln spielen hier keine Rolle mehr.
Die zweite Heuristik gewinnt man, indem man das Teilchen horizontal und vertikal schrittweise schiebt, ohne daran zu denken, dass ein Teilchen erst bewegt werden kann wenn das Nachbarsfeld leer ist.

3.4.2 Automatische Gewinnung der Heuristiken

Eine Heuristik für eine Problemlösung kann wohl vorgegeben werden. Aber für einen domänenunabhängigen Planer ist relevant, die Heuristik aus dem spezifischen Problem zu generieren. Wenn ein Problem in einer formalen Sprache definiert ist, ist es möglich, die Vereinfachung eines Problems automatisch zu konstruieren. d.h. Heuristiken sind direkt aus der Problemrepräsentation zu gewinnen.

In STRIPS erreicht man die Vereinfachung automatisch durch 3 Wege:

1. Ohne Rücksicht auf die Vorbedingung wird eine Aktion angewandt.
2. Die Delite-Listen werden ignoriert.
3. Oder man reduziert die Vorbedingung, um eine Aktion überhaupt anwenden zu können.

3.4.3 Trade-Off-Beziehung der Heuristik

Die Vereinfachung des Problems reduziert den Rechenaufwand, aber auf der anderen Seite ist die Vereinfachung immer mit Informationsverlust verbunden. Ein unangemessener Informationsverlust das originale Problem betreffend gefährdet die Effektivität der eingesetzten Heuristik. An dem 8-Puzzle-Beispiel sieht man das Trade-Off deutlich. Die Problemvereinfachung bei der ersten Heuristik ist viel stärker als bei der zweiten. Die zweite ist genauer und effektiver aber mit bedeutend größerem Rechenaufwand. Dadurch wird die Schnelligkeit beeinträchtigt.

Um bessere Runtime Performanz zu erzielen, wird in Buildtime die Rechenarbeit geleistet, aber dazu braucht man extra Speicher und die Informationen, die eventuell durch neue Berechnungen zur Runtime gewonnen werden könnten, gehen verloren. Dies bemerkt man an der Schwäche des Rückwärtssuchplaners.

3.5 Heuristische Funktionen für Zustandsraumplanen

Im Zustandsraumplanen werden die **Delete-Listen** ignoriert, um das Problem zu vereinfachen. Aber die Berechnung der optimalen Lösung des vereinfachten Problems ist immerhin NP hard. Für die Approximation wird die heuristische Funktion h(n) gebraucht. h(n)schätzt die minimalen Kosten zur Erreichung des Ziels von einem bestimmten Zustand n. Weil ein Ziel aus einer Menge von Atomen besteht, wird eine Funktion gs(p) definiert, die die minimalen Kosten zur Erreichung eines Atoms von einem bestimmten Zustand s schätzt.

$$g_s(p) = \begin{cases} 0 & \text{if } p \in s \\ \min_{op \in O(p)}[1 + g_s(Prec(op))] & \text{otherwise} \end{cases}$$

Abb. 3.2 die rekursive Definition (Quelle: [1])

Die obige Formel beschreibt die rekursive Definition. Wenn das Atom p in dem aktuellen Zustand zu finden ist, dann sind die Kosten null. Anderenfalls, betrachtet man die Minimalkosten/Minimaldistanz, zur Erreichung des Zustandes mit der Vorbedingung der Aktion, die das Atom p produziert. Da eine Aktion ausgeführt wurde, werden deren Kosten von 1 auf die noch zu berechnenden Minimalkosten addiert.

Um die Minimalkosten/-distanz zu berechnen wird ein Rechenalgorithmus benutzt. Dieser ist eine Variante des Bellman-Ford-Algorithmus, der die minimale Distanz von einem Knoten zum Startknoten findet.

- Am Anfang werden die Kosten auf 0 gesetzt wenn das Atom in dem aktuellen Zustand s zu finden ist.
- Wenn nicht, dann sind die Kosten unendlich.
- Auf den Zustand s werden alle anwendbaren Aktionen ausgeführt.
- Die entstandenen Atome aus den Add-Listen werden dem Zustand s hinzugefügt.
- Für jede Aktion, die das Atom p produziert, aktualisiert man die bisherigen Minimalkosten auf die niedrigeren Minimalkosten, die die Minimalkosten zur Erreichung der Vorbedingung der Aktion um „1“ erweitert. gs(p)=min[gs(p),1+ gs(Pre(op))]
- Der Vorgang wird wiederholt bis sich der Wert von gs(p) nicht mehr verändert.

Je nachdem wie man die geschätzten Kosten zur Erreichung der Teilziele kombiniert, entsteht eine Art heuristische Funktion.
Die Funktion gs(C) schätzt die Kosten zur Erreichung einer Menge Atome C von dem bestimmten Zustand s. Die andere Funktion gs(G) schätzt die Kosten zur Erreichung des Ziels von dem bestimmten Zustand s
Mit **h(s)= gs(G)** definiert man die heuristische Funktion als geschätzte Kosten zur Erreichung des Ziels, das aus Teilzielen besteht.
Da es komplexe Interaktionen zwischen Teilzielen gibt, entstehen dann Probleme wenn man die Teilziele unabhängig von einander betrachtet. Die Nichtbeachtung der negativen oder positiven Abhängigkeiten führt oft zur Unterschätzung oder Überschätzung der optimalen Lösung.
Da wir in unserem Fall die Delete-Listen außer Acht gelassen haben, bleiben die positiven Abhängigkeiten noch durch geeignete heuristische Funktionen zu handhaben.

Die Additive Heuristik hadd ***gs(C)=∑gs(r) (r€C)*** basiert auf der Annahme, dass die Teilziele von einander unabhängig sind. Die Kosten zur Erreichung einer Menge Atome sind die Summe der Kosten zur Erreichung einzelnen Atome von einem bestimmten Zustand aus.
Diese Heuristik ist offensichtlich unzulässig, wenn man bedenkt, dass in der Tat zwischen den Teilzielen eine positive Abhängigkeit herrscht. Eine Aktion könnte zum Beispiel auf einmal mehrere Atome produzieren. Mit **hadd** werden aber für jedes Atom mindestens einmal die Kosten von „1“ geschätzt. Obwohl diese Heuristik überschätzt, hat deren Einsatz in HSP 2.0 doch noch gute Ergebnisse gebracht.

Bei der Max-Heuristik hmax gs(C)=max gs(r) (r€C) sind die Kosten zur Erreichung einer Menge Atome gleich den Kosten zur Erreichung eines Atoms, der am teuersten ist. Diese Heuristik ist zulässig, aber wenig informativ, denn sie konzentriert sich nur auf die schwierigsten Teilziele, nämlich die teuersten Teilziele und übersieht die anderen Teilziele.

Mit $\mathbf{h^m}$ **Higher-Order-Heuristiken** wird versucht, eine generale Form von zulässigen Heuristiken darzustellen. Die komplexen Interaktionen zwischen den Teilzielen sollen
Berücksichtigt werden. Mit m=1 ist sie eine Max-Heuristik. Und mit m=2 ist sie dann die Graphplan-Heuristik oder Max-2-Heuristik in HSP 2.0 (kombiniert nur mit Rückwärtssuche). Bei Max-2-Heuristik werden die beiden Teilziele, die am meisten kosten, betrachtet. Die Heuristik schätzt die Minimalkosten zur Erreichung der beiden Teilziele. Die Kosten werden untereinander verglichen:

1. die Minimalkosten zur Erreichung der Vorbedingung einer Aktion, die die beiden Teilziele produziert.
2. die Minimalkosten zur Erreichung der Vorbedingung einer Aktion, die eines der beiden Teilziele produziert, und zur Erreichung des Zustandes, der das andere Teilziel enthält.

Der kleinere Wert wird als Minimalkosten zur Erreichung von beiden Atomen übernommen.

4 HSP 2.0 – ein heuristischer Suchplaner

4.1 Eigenschaften von HSP 2.0

HSP 2.0 ist ein domänenunabhängiger Suchplaner. Er basiert auf der Forschungsarbeit von Bonet und Geffner. [1] In AIPS'99 hat HSP 2.0 mit der Additiven Heuristik und der Best-First-Search-Suchstrategie eine hervorragende Leistung erzielt. Mit der guten Performanz hat er bewiesen, dass heuristische Suchplaner auch sehr konkurrenzfähig sein können.
Die Heuristiken werden aus der STRIPS-Kodierung automatisch gewonnen. Die Delete-Listen werden ignoriert. Der WA* Algorithmus wird eingesetzt, um einen Knoten(Zustand) zu bewerten. Der Parameter W liegt im Wertebereich zwischen 2 und 10. W ist anpassbar je nach Domänen und Problemstellungen. In WA* **f(n)=g(n)+W h(n):** Parameter W wird mit h(n) multipliziert, um die heuristische Funktion zu gewichten.

Der Suchplaner ist mittlerweile um Rückwärtssuche und die beiden zulässigen Heuristiken, Max-Heuristik und Max-2-Heuristik erweitert worden. HSP 2.0 unterstützt neben STRIPS noch andere PDDL Sprachen. z.B. „Types“, „Equality“ und „simple ADL“.
Die verschiedenen Optionen von Suchstrategien und Heuristiken können sequentiell und parallel eingesetzt werden. Mit Max-2-Heuristik, Rückwärtssuche und W=1 wird für ein optimales und sequentielles Planen gedacht.
HSP 2.0 wird in C programmiert und läuft unter Linux. Folgend wird durch ein kleines Block-World-Beispiel ein Überblick über die Funktion von HSP 2.0 geworfen.

4.2 Ein Block-World-Beispiel

Die Domänendefinition von Block-World wird direkt von der in HSP 2.0 mitgelieferten Definition übernommen
Die Domänendefinition wird in STRIPS geschrieben. 3 Prädikate: on-table(x), on(x y) und clear(x) und 3 Aktionen: MoveToTable (x), MoveToBlock1(x y z) und MoveToBlock2 (x y) sind definiert.

Block World Domain definition

```
define (domain blocks_world)
(:requirements :strips)

(:predicates (on-table ?x) (on ?x ?y) (clear ?x))

(:action MoveToTable
        :parameters (?x ?y)
        :precondition (and (clear ?x) (on ?x ?y))
        :effect (and (clear ?y) (on-table ?x) (not (on ?x ?y))))

(:action MoveToBlock1
        :parameters (?x ?y ?z)
        :precondition (and (clear ?x) (clear ?z) (on ?x ?y))
        :effect (and (clear ?y) (on ?x ?z) (not (clear ?z)) (not (on ?x ?y))))

(:action MoveToBlock2
        :parameters (?x ?y)
        :precondition (and (clear ?x) (clear ?y) (on-table ?x))
        :effect (and (on ?x ?y) (not (clear ?y) (not (on-table ?x)))))
```

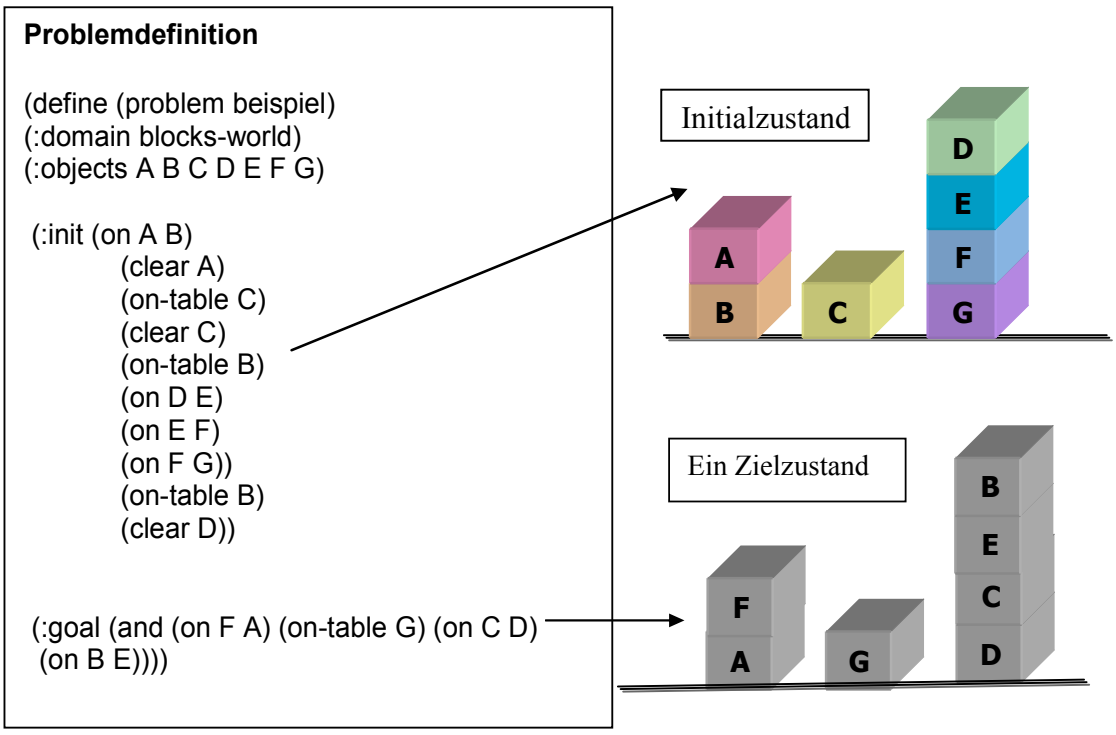

Abb. 4.1 Initialzustand und ein Zielzustand der Problemdefinition von dem Block-World-Beispiel

Das Beispiel beschäftigt sich mit einem Problem von 7 Blöcken, von „A“ bis“ G“. Die Definition des Initialzustands wird rechts oben veranschaulicht. Das Ziel wird so definiert, dass ein Zielzustand erreicht ist, sobald die Bedingungen in der Definition erfüllt sind. Ein Zielzustand wird rechts unten dargestellt.

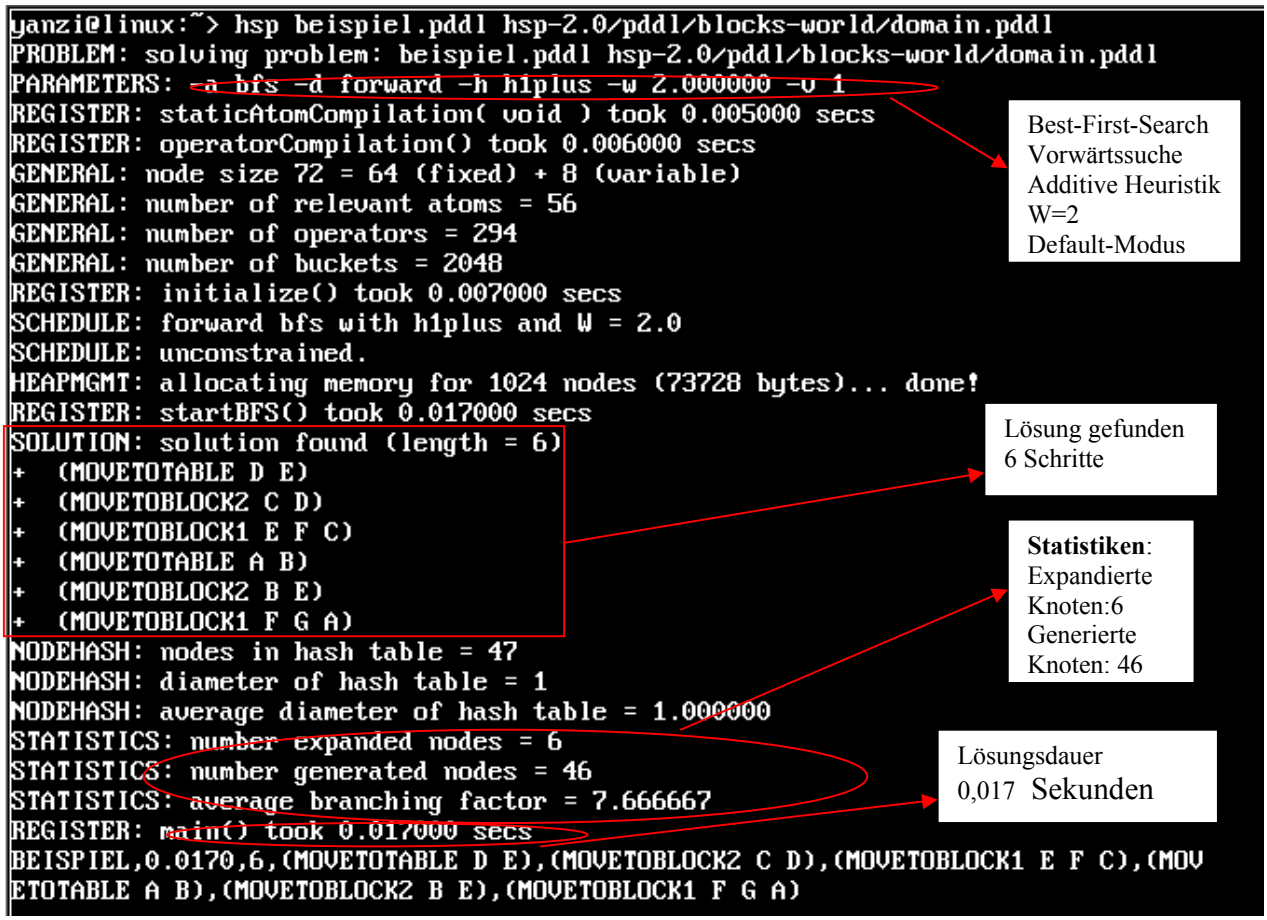

Abb. 4.2 Protokoll der Problemlösung von dem Block-World-Beispiel

Das kleine Beispielproblem ist dann von dem HSP Planer zu lösen. Der Default-Modus: Best-First-Search, Vorwärtssuche, Additive Heuristik und W=2 wird eingeschaltet. Wie das Protokoll oben zeigt, hat der Planer die Lösung gefunden: 6 Schritte, mit einer Sequenz: (MoveToTable D E), (MoveToBlock2 C D), (MoveToBlock1 E F C), (MoveToTable A B), (MoveToBlock2 B E) und (MoveToBlock1 F G A). Die Statistiken zeigen die Anzahl der bei der Lösungsfindung expandierten und generierten Knoten, jeweils 6 und 46. Die Lösungsdauer beträgt 0,017 Sekunden. Um die Lösungsschritte besser zu verfolgen sind Zustandsübergänge in Abb. 4.3 visualisiert.

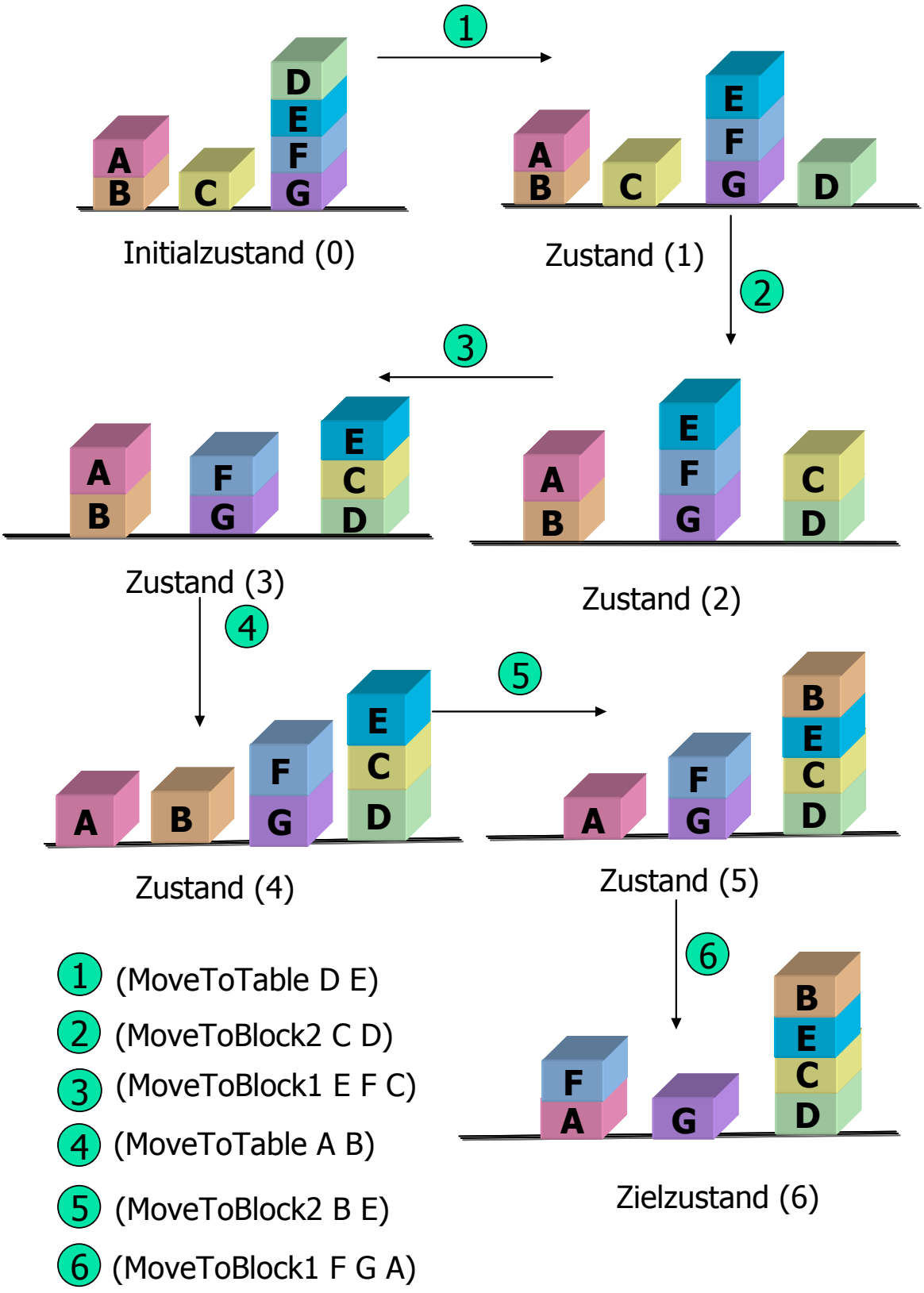

Abb. 4.3 Aktionen und Zustände des Block-World-Beispiels

5 Fazit und Ausblick

5.1 Zusammenfassung

Das Konzept und der Erfolg des HSP 2.0 haben gezeigt.

- Mit effektiven Heuristiken und Suchstrategien kann die einfache Zustandsraumplanung auch effizient sein
- Die Repräsentation des Problems in einer geeigneten formalen Sprache ist von großer Bedeutung und ermöglicht eine leichte Vereinfachung des originalen Problems, damit die automatische Gewinnung der Heuristiken
- Viele Anstrengungen zur Verbesserung der Heuristik und Repräsentation sind noch zu unternehmen, um einen kleinen Schritt weiter zum Ziel zu kommen, letzten Endes „Real-Probleme“ zu lösen,.

5.2 Fast-Forward-Planer

Die Weiterentwicklung von dem Konzept „Planen als heuristische Suche“ sieht man an dem sogenannten FF-Planer, entwickelt von J. Hoffmann von der Uni-Freiburg.
FF-Planer als erfolgreicher Nachfolger von HSP 2.0 hat in AIPS’00 ausgezeichnete Performanz erzielt und wurde „Top-Performer“ in STRIPS-Domäne in AIPS’02.
FF basiert auf der gleichen „ Relaxion Priciple“ wie HSP. d.h. Delete-Listen werden ignoriert. Er verwendet die Suchstrategie Vorwärtssuche im Zustandsraum. Er unterscheidet sich von HSP in folgenden Aspekten:
Ein expliziter **„Relaxed Plan“ (O0,…, Om-1)** wird aus dem „Planning Graph“ gewonnen. [1]
Die Heuristik basiert auf diesem „relaxed Plan“. **hFF(S):=∑|Oi| i=0,…,m-1**. Die Positiven Interaktionen zwischen den Teilzielen werden berücksichtigt.

[1] Der „Planning Graph“, bestehend aus Zustands- und Aktionsschichten, wird soweit entwickelt bis alle Ziele erreicht sind. Man fängt von der obersten Schichte m an und bearbeitet die Ziele dort. Bei jeder Schichte i wird an den anwesenden Zielen gearbeitet. Wenn das Ziel in der vorigen Schichte i-1 zu finden ist, dann gehört das Ziel zu den Zielen der vorigen Schichte i-1. Wenn nicht, nimmt man die Vorbedingung der Aktion, die das Ziel produziert hat, in die zu bearbeitende Ziele der vorigen Schichte i-1. Wenn alle Ziele einer Schichte i bearbeitet sind, setzt man mit der nächsten Schichte i-1 fort, bis man die erste Schichte erreicht hat. Als Ergebnis des ganzen Prozesses ist ein“ relaxed Plan“ (O0,…, Om-1). Oi steht für die Menge von Aktionen, ausgewählt zum i-ten Schritt.

„Enforced-Hill-Climbing“ modifiziert den Hill-Climbing- Algorithmus in HSP, wo der beste Nachfolger von einem Zustand willkürlich ausgewählt wird und ein Restart stattfindet wenn die Distanz zu lang ist. Der modifizierte Algorithmus verwendet iterative Breadth-First-Search, um Plateaus und lokale Minima zu vermeiden. Alle direkten Nachfolger eines Zustandes werden bewertet, um einen besseren heuristischen Wert als dies von dem aktuellen Zustand S zu finden. Die Nachfolger der Nachfolgern werden soweit bewertet bis man einen Nachfolger S’ mit einem besseren Wert gefunden ist. Und der Weg zu S’ wird dem aktuellen Plan hinzugefügt.
Der „Relaxed Plan“ bietet auch eine Information über die „helpful actions“, um die nützlichen Nachfolgerzustände zu identifizieren, damit die überflüssigen Nachfolger des aktuellen Zustandes auszuschneiden.
In Hinsicht auf Performanz hat FF die Runtime-Zeit und die Länge der Lösungsschritte dem HSP gegenüber verbessert. [4]

Abbildverzeichnis

Literatur

[1] Blai Bonet, Héctor Geffner, Planning as Heuristic Search, http://www.ldc.usb.ve/~hector/reports/hsp-aij.ps , 2001.

[2] Stuart J. Russell, Peter Norvig, Artificial Intelligence:A Modern Approach. Second Edition Prentice Hall, New Jersey, 2003 , Kap.3,4,11.

[3] M. Ghallab, D. Nau, and P. Traverso. Automated Planning: Theory and Practice, Morgann Kaufmann, 2004. Kap1, 2, 4, 9.

[4] Jörg Hoffmann, FF: the Fast-Forward Planning System http://www.cs.toronto.edu/~sheila/2542/w06/readings/ffplan01.pdf, 2001.

[5] Judea Pearl, Heuristics, -Intelligent Search Strategies for Computer Problem Solving- Addison-Wesley, 1984, Kap.1, 2.

[6] Hermann Kaindl, Problemlösen durch heuritische Suche in der Artificial Intelligence, Springer-Verlag, Wien, 1989.