

Informatik Seminar

Herr Prof. Iwanowski

Planungsstrategien intelligenter Agenten

Wedel, 29.08.2005

Eingereicht von: Sven Tollmien
Hellgrund 9
22880 Wedel
04103/85499
WI5011
Sven.Tollmien@gmx.de



1.	Einleitung.....	1
2.	Grundlagen der Planung	2
3.	Klassische Planmethoden	3
3.1.	STRIPS – Mengenbasiertes Planen.....	3
3.2.	Regression	4
4.	Planbasiertes Planen	6
4.1.	Einleitung.....	6
4.2.	Der POP-Algorithmus	7
5.	Graphenbasiertes Planen.....	10
5.1.	Einleitung.....	10
5.2.	Lösungsexpansion und Lösungsextraktion	11
5.3.	Ausführliches Beispiel	12
6.	Vergleich zwischen POP und Graphenplan	17
7.	Fazit – Graphenplan.....	17
8.	Quellen	18
9.	Abbildungsverzeichnis.....	18

Vorwort

Die Ausarbeitung dieses Seminars basiert auf dem Buch „Handbuch der KI“ von Günther Görz (März 2003). Weitere Quellen sind an den entsprechenden Textstellen angegeben.

1. Einleitung

Man nehme eine Urlaubsreise als ein einleitendes Beispiel. Man bucht das Hotel in seinem Urlaubsort für einen bestimmten Zeitraum, die Flüge, die einen zu Beginn dieses Zeitraumes dort hin bringen und am Ende wieder wegfliegen. Dann bucht man die Bahn zum Flughafen, und man stellt sich den Wecker, damit man rechtzeitig aufsteht um die Bahn nicht zu verpassen. Diese Folge von Aktionen muss nun in der richtigen Reihenfolge abgearbeitet werden. Wurde alles gut geplant, kann man seinen Urlaub entspannt antreten. Da diese Folge aber eine gewisse Komplexität aufweist und somit störanfällig ist, kann es auch zu Problemen kommen. Man hat den Wecker zu spät gestellt, verpasst erst die Bahn, dann das Flugzeug und kommt verspätet im Hotel an.

Um das Verhalten von autonomen Agenten, die aus einer Aufgabenbeschreibungen, wie hier zum Beispiel Reiseplanung, eine Folge von Aktionen ermitteln sollen, als „vernünftig“ zu bezeichnen, werden seit langem Forschungen auf diesem Gebiet durchgeführt.

Aus diesen Forschungen entwickelten sich zwei Ansätze, die Handlungsplanung, um mittel- und langfristige Ziele zu erreichen und die situierte Aktivität, um auf Unvorhersehbarkeiten zu reagieren. Grundsätzlich benötigen Agenten beide Methoden um sinnvoll zu agieren.

Dieses Seminar legt den Schwerpunkt auf die Handlungsplanung.

Handlungsplanung, oder auch klassisches Planen, wird eingesetzt, wenn Aktionen in einer bestimmten Reihenfolge angewendet werden müssen, um ein vorher festgelegtes Ziel zu erreichen oder wenn einzelne Aktionen stark voneinander abhängig sind.

2. Grundlagen der Planung

1969 entwickelten McCarthy und Hayes das Situationskalkül. Dieses Kalkül ist begründet auf Situationen, eine Momentaufnahme des interessierenden Ausschnittes der Welt zu einem Zeitpunkt, die durch eine Menge logischer Formeln beschrieben ist, welche zu diesem Zeitpunkt gelten und Aktionen, die eine Beschreibung der Handlung in dieser Welt liefern. Formal lässt sich dieses folgendermaßen beschreiben $A(S) \rightarrow S'$. Wenn eine Aktion A auf eine Ausgangssituation S angewendet wird, entsteht daraus eine neue Situation S', die Nachfolgesituation. Dieser Übergang von S nach S' unter A wird bei der klassischen Planung als Zeit betrachtet. Durch die Komponente Zeit, treten innerhalb dieser Planung aber Probleme auf, die vom System selbstständig gelöst oder umgangen werden müssen.

Der Übergang von einer Vorher-Situation in eine Nachher-Situation weist in Bezug auf das Resultat dieser Aktion Detailprobleme auf. Diese Probleme resultieren nicht nur aus der Aktion, sondern beliebige Ereignisse können das Resultat beeinflussen.

1. Qualification Problem: Die Schwierigkeit, Angaben über den Ausgang der Aktion(en) zu machen ohne die Informationen der Ausgangssituation zu kennen.
2. Prediction Problem: Die Schwierigkeit, zu bestimmen, welche Fakten der Nachher-Situation wahr sind.
3. Persistence Problem: Die Schwierigkeit, eine Aussage darüber zu machen, welche Fakten der Ausgangssituation, nach Durchführung einer Menge von Aktionen unverändert bleiben.
4. Frame Problem: Ähnelt dem Persistence Problem, es bezieht sich allerdings nur auf die Änderung der Fakten, nach Durchführung einer Aktion.
5. Ramification Problem: Die Schwierigkeit, vorherzusagen, welche weiteren Faktoren wahr werden, wenn ein Faktor durch eine Aktion wahr geworden ist.

Durch die Menge dieser Probleme, wird es für ein System unmöglich zu entscheiden, wenn es alle Probleme in Betracht zieht. Daraus folgt, dass ein solches intelligentes System ungenau arbeitet, wenn es eine halbwegs

effiziente Planung durchführen soll. Allerdings beinhaltet jede Planung einen Unsicherheitsfaktor, der die Notwendigkeit absolut fehlerfrei Planer zu entwickeln minimiert.

3. *Klassische Planmethoden*

3.1. STRIPS – Mengenbasiertes Planen

Das Ursprungsprogramm aller Planer wurde 1972 entwickelt. Es basiert auf dem in den Grundlagen beschriebenen Ansatz des Situationskalküls, die Welt durch eine Menge von Eigenschaften beziehungsweise Prädikaten zu beschreiben. In dieser Welt kann eine Menge von Aktionen ausgeführt werden, welche aus einer Vorbedingung bestehen, sowie dem Effekt oder dem Resultat der Aktion. Die Beschreibung der Welt mit Prädikaten kann wie folgt aussehen:

Situation(Object ₁ , Object ₂)

Die oben genannte Formel würde in Sprache formuliert etwa: Object₁ verhält sich zu Object₂ entsprechend der Situation. Als Beispiel nimmt man die Situation in der Fachhochschule zur Vorlesungszeit:

In(Studenten, Hörsaal)

Diese Formel sagt aus, dass sich in der aktuellen Situation, die Studenten im Hörsaal befinden. Somit lässt sich die Startsituation über eine Menge von Literalen beschreiben. STRIPS fordert über dieses hinaus, dass eine Startsituation vollständig beschrieben ist, was bedeutet, dass nicht beschriebene Zustände als falsch angenommen werden. Diese Annahme wird Closed World Assumption genannt. Ausgehend von dieser Startsituation lässt sich auch das Ziel formulieren. Geltende Einschränkungen die STRIPS vorschreibt ist die positive Eigenschaft von Zielen, sowie die Beschreibung eines endlichen, feststehenden Zieles, nicht dem Verhalten auf dem Weg zum eigentlichen Ziel.

Mit dieser Methodik sind nun die Startsituation und das Ziel beschrieben. Aktionen, die für die Transformation der Objekte von Situation 1 zu Situation 2 führen, werden auf ähnliche Weise beschrieben, wie die Startsituation und die Zielsituation. Jede Aktion benötigt eine Vor- und eine Nachbedingung. Diese Bedingungen bestehen aus Literalen, wobei nur die Nachbedingung negative Literale enthalten darf, nicht aber die Vorbedingung. Abstrakt würde diese Formulierung folgendermaßen aussehen:

Überführe-Objekt ₁ -in-Startsituation-nach-Objekt ₂ -in-Zielsituation Vorbedingung: ... Nachbedingung: ...
--

Aus dem Beispiel mit den Studenten im Hörsaal, würde ein Aktion, wie Studenten gehen in die Pause, folgendermaßen aussehen:

Studenten-in-Hörsaal-werden-zu-Studenten-in-der-Pause Vorbedingung: $\text{in}(\text{Studenten}, \text{Hörsaal}) \wedge \text{istPause}(\text{Studenten}, \text{Hörsaal})$ Nachbedingung: $\text{in}(\text{Studenten}, \text{Pausenhof})$

Wenn ein Planer nun eine endliche Menge solcher Aktionen bekommt, sowie Start- und Endsituation, dann kann daraus eine Taskabfolge generiert werden, ein sogenannter Plan.

Die in diesem Kapitel beschriebene Planungsmethode versucht ausgehend von einer Startsituation, einen Plan zu entwickeln, der zur Zielsituation führt. Dieses Verfahren bezeichnet man als Progression.

3.2. Regression

Die Regression arbeitet gegenüber der Progression in die andere Richtung. Ausgehend von einem Ziel wird eine Task generiert, deren Nachbedingung im Teil die Zielsituation erfüllt. Diese Task liefert dann ausgehend von der Vorbedingung ein neues Ziel, über das nun rekursiv weiter gesucht wird, erst wenn die die Startsituation, die selben Literale besitzt wie das neue Ziel, ist ein Plan gefunden. Grafik 1 beschreibt den rekursiven Ablauf der Regression.

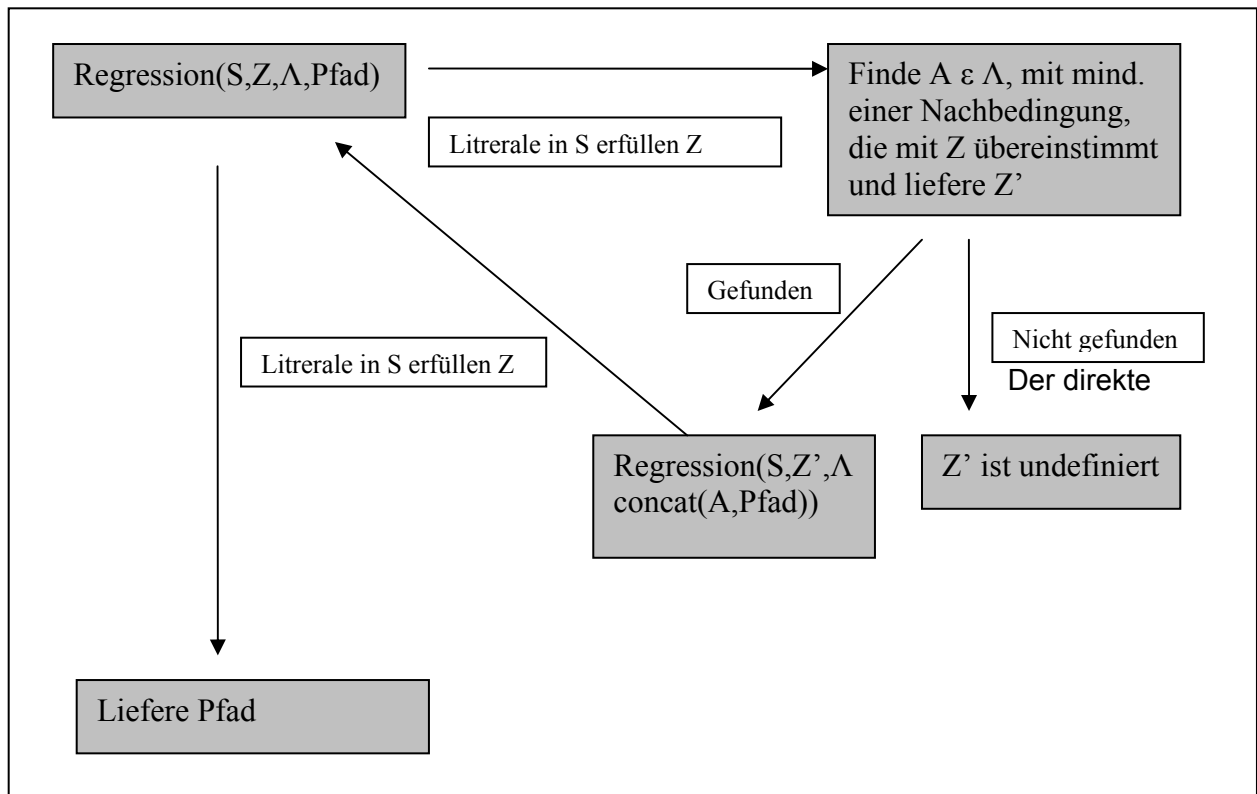


Abbildung 1 - Algorithmus der Regression

Der Vergleich zwischen den beiden Verfahren fällt so aus, dass beide einen strikt linearen Plan liefern, mit annähernd gleicher Komplexität. Allerdings arbeitet die Regression schneller, da durch das Beginnen beim Ziel, der Planer von weniger Literalen ausgeht, als die Progression, denn eine Zielbeschreibung hat meist weniger Literale als die Startsituation. Sollten allerdings wenige Literale aber viele Ziele vorhanden sein, bietet sich die Progression für eine schnellere Abarbeitung an.

4. Planbasiertes Planen

4.1. Einleitung

Die Erstellung eines Planes unter Verwendung der klassischen Planung, liefert, wie im vorigen Kapitel beschrieben, einen linearen Plan, der in vielen Fällen aus einer großen Anzahl von Literalen besteht.

Um die Problematik der Zahl der Literale zu begrenzen, wurde 1975 ein nicht-lineares Planungsmodell entworfen. Dieses Modell, in den Ursprüngen unter dem Namen NOAH bekannt, arbeitete nach dem Prinzip der Teilpläne, die in Abhängigkeit von Vorschriften vom Planer beliebig aneinander gereiht werden konnten. Die einzelnen unfertigen Pläne unterlagen einer partiellen Ordnung und durch die bedingte Festelegung, wurde diese Eigenschaft als least-commitment bezeichnet.

Eine weitere Eigenschaft, die nicht-linearen Plänen beiwohnte, war die Erweiterung der Operatoren. Neben Vor- und Nachbedingung war es nun möglich, die Aktionen weiter in Teil-Aktionen zu zergliedern. Als Beispiel werden wieder die Studenten, die in die Pause gehen, angeführt. Die Aktion „Pause“ sieht es vor, dass die Studenten in die Pause gehen, sich einen Kaffee holen, sowie am Ende der Pause wieder in den Hörsaal gehen müssen. Die Untergliederung der Aktionen wäre:

$A_1(\text{In die Pause})$ $A_2(\text{zurück in den HS})$ $A_3(\text{Pause})$ $A_4(\text{Kaffee holen})$
--

Diese drei Aktionen haben nun folgende Ordnung:

$A_2 < A_1; A_3 < A_1$

Das bedeutet, der Student muss erst den Hörsaal in die Pause verlassen, bevor er zurückgehen kann. Auch den Kaffee kann er sich erst holen, wenn er den Hörsaal verlassen hat. Wann er sich aber den Kaffee holt, ob vor der Pause oder nach der Pause, ist nicht festgelegt, der Planer hat somit die Möglichkeit zu wählen.

4.2. Der POP-Algorithmus

Ein partial geordneter Plan, wie POP (partial order planning) besteht aus einem Tripel (A, O, L). A entspricht einer Menge von Tasks, L sind Abhängigkeiten zwischen Tasks und O sind die Ordnungsconstraints, die über A definiert sind. Bilden diese Constraints eine totale Ordnung ist der Plan konsistent. Diese Bedingung muss für den fertigen Plan immer bestimmt sein.

Die Aktionen, die in L definiert sind, sehen wie folgt aus:



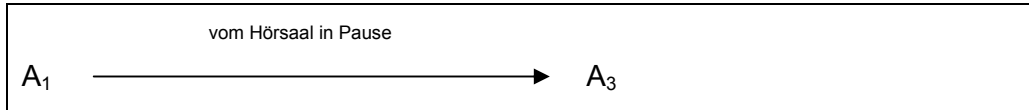
Die Aktion A_1 hat als Nachbedingung Q und A_2 besitzt Q als Vorbedingung. Q beschreibt eine Eigenschaft für eine Abhängigkeit. A_1 produziert das Literal Q, welches von A_2 verbraucht wird. Die definierten Abhängigkeiten, geben Aufschluss darüber, ob neue Tasks, die in den Plan eingeführt werden, eine Konsistenzverletzung ausüben. Diese Verletzung ist definiert, wenn zum Beispiel O mit $A_1 < A_2$ definiert ist, wobei A_2 die Vorbedingung Q besitzt und eine neue Task A_3 eingefügt werden soll, die laut Abhängigkeit $\neg Q$ als Nachbedingung besitzt, somit wäre eine Konsistenzverletzung vorhanden.

Der eigentliche POP-Algorithmus besteht aus dem oben beschriebenen Tripel, sowie einer Agenda, die die Ziele enthält, die der Plan abarbeiten soll, sowie den instanziierten Tasks aus Λ . Mit Hilfe des Studenten-Beispiels, würde es dann folgendermaßen aussehen:

<p>Hörsaal vor der Pause, Nachbedingung: es ist Pause = A_1 Hörsaal nach der Pause, Vorbedingung: Pause ist vorbei = A_2 Pause = A_3</p> <p>POP(($\{A_1, A_2\}, \{A_1 < A_2\}, \{\}$), ($\{[in(Student, Pause), A_2], [in(Student, Hörsaal)]\}$), Λ)</p>

Der Start ist der Hörsaal, das Ziel ist die Pause. Um das Ziel zu erfüllen, muss der Student in der Pause sein, dazu muss die Vorbedingung vom Ziel erfüllt sein. Als erstes wird das Ziel $in(Student, Pause)$ gewählt. Darauf folgend wird eine Task instanziiert, die besagt, Student-geht-in-Pause. Diese

Aktion hat als Vorbedingung, dass sich der Student im Hörsaal befindet und als Nachbedingung, dass er in der Pause ist. Nun können die Abhängigkeiten beschrieben werden:



Diese Abhängigkeit beeinflusst nicht das Ziel und es ist ein konsistenter Plan ermittelt worden. Die Aktion Student-geht-von-Pause-in_HS läuft nach dem gleichen Schema ab. Wenn nun noch das Ziel hat(Student, Kaffee) in der Agenda enthalten ist, wird dieses als nächstes zu bearbeitendes Ziel gewählt. Die Aktion Student-holt-sich-Kaffee hat als Vorbedingung, dass Pause ist und als Nachbedingung, dass die Pause vorbei ist. Wird diese Aktion eingebunden, kann der Planer selbst entscheiden, wo sie eingesetzt wird, da die Nachbedingung das Ziel erfüllt. Eine Aktion Student-geht-nach-Hause, wäre nicht zulässig, da sie durch die Nachbedingung, Student ist zu Hause, die Vorbedingung des Ziels, Student-ist-im-HS verletzt. Um die Konsistenz wieder herzustellen, müssten Aktionen eingeführt werden, die den Studenten vor Ende der Pause, wieder zurückkommen ließen. Wenn alle Ziele abgearbeitet sind und alle Abhängigkeiten erfüllt sind, ist der Plan vollständig.

Um die Wiederverwendbarkeit des POP-Algorithmusses zu verbessern, besteht die Möglichkeit die Operationen die ausgeführt werden können, um Variablen zu erweitern, wie zum Beispiel irgendwer-holt-irgendwas. Nun ist nicht festgelegt, ob sich ein Student oder ein Dozent sich in der Pause einen Tee oder einen Kaffee holt. Des Weiteren kann durch diese Flexibilität eine Konsistenzverletzung behoben werden, wenn zum Beispiel der Tee zu lange dauert und somit die Nachbedingung, irgendwer kommt zu spät in den Hörsaal, entsteht. Der Planer hat nun die Möglichkeit dieses Problem zu beheben.

Eine allgemeine Definition einer variablen Operation sieht folgendermaßen aus:

```
(define (operator xxx)
  :parameters (?x ?y ...)
  :precondition (...)
  :effect      (...))
```

Außerdem bietet POP bzw. UCPOP (POP mit den hier beschriebenen Erweiterungen) die Möglichkeit der Nutzung eines Allquantors, um auf mehrere Objekte zu greifen, sowie die Nutzung von Bedingungen innerhalb einer Operation, wie zum Beispiel, Student-holt-Tee-wenn-Kaffee-alle.

5. Graphenbasiertes Planen

5.1. Einleitung

Dieses Verfahren wurde 1995 von Blum und Furst, als einfachere und schnellere Variante gegenüber den bisher vorgestellten Planungsverfahren, entwickelt. Das Prinzip des Planungsgraphen, besteht wie bei den anderen Verfahren, darin, erst eine Startsituation anhand von Literalen zu erstellen, die die Ausgangssituation beschreiben. Das Gleiche erfolgt für die Zielsituation. Diese beiden Zustände definieren im Graphen die geraden Schichten 0 und 2. Die ungeraden Schichten repräsentieren die Tasks, die im Plan angewendet werden können. Propositionsknoten, Knoten die einen Zustand, wie Start- und Zielsituation, darstellen, werden als Kreise gezeichnet, Aktionsknoten als Rechtecke. Die graphische Darstellung sieht wie folgt aus:

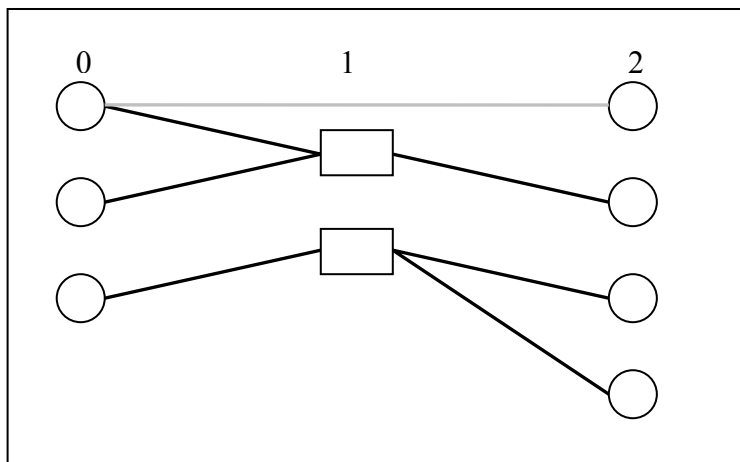


Abbildung 2 - Beispiel für expandierten Graphen

Die Verbindungslinien zwischen Präpositions-knoten und Aktionsknoten, in der Graphik als dickere Linien gezeichnet, zeigen, dass der entsprechende Präpositions-knoten, die Vorbedingung des Aktionsknoten liefert, beziehungsweise, die Nachbedingung.

Der Graphplan-Algorithmus bietet zusätzlich die Möglichkeit der Persistenzaktionen. Diese stellen eine direkte Verbindung zwischen zwei Präpositions-knoten dar und liefern einen direkten Übergang eines

Präpositionsknoten in die nächste Schicht. In der Graphik 2 als helle Verbindungslinie dargestellt.

Wie in der Graphik zu sehen ist, können Tasks innerhalb einer Schicht parallel ablaufen. Daraus lässt sich eine Problematik ableiten, die als gegenseitiger Ausschluss, kurz mutex (vom mutual exclusion), beschreiben lässt:

1. Inkonsistente Nachbedingung, das bedeutet eine Task einer Schicht liefert die negative Nachbedingung einer anderen Task innerhalb derselben Schicht.
2. Interferenz, das bedeutet eine Task liefert als Nachbedingung, die negierte Vorbedingung einer anderen Task innerhalb derselben Schicht.
3. Konkurrierende Vorbedingung, das bedeutet, dass zwei Tasks Vorbedingungen besitzen, die in der vorherigen Schicht mutex sind.
4. Inkonsistente Vorbedingung, das bedeutet, dass Tasks, die Literale zur Lösung liefern, paarweise wiederum mutex sind.

5.2. Lösungsexpansion und Lösungsextraktion

Der Algorithmus läuft zu Beginn, wie beim klassischen Planen mit Progression, vorwärts ab. Ausgehend von der Startspezifikation durch Literale, werden die Vor- und Nachbedingungen der Tasks ermittelt und das erste expandierte Modell erstellt. Auf der Schicht 0 würde dann die Startspezifikation stehen, Schicht 1 beinhaltet die Aktionsknoten, und Schicht 2, die Nachbedingungen der Aktionen. Die Verbindungen zwischen den Knoten der Schichten 0 und 1, sowie 2 und 1 würden entsprechend der Vorbedingung dargestellt werden. Das erste expandierte Modell enthält somit 3 Schichten, 2 Schichten mit Präpositionsknoten, Vor- und Nachbedingung der Tasks, sowie 1 Schicht mit Aktionsknoten.

Im folgenden wird der Graph extrahiert. Das bedeutet, dass für jedes Zielliteral eine Task oder, wenn mehrere Tasks ein Zielliteral ergeben, müssen alle unter Nutzung der Reihenfolge, selektiert werden. Sollten die Tasks gegenüber den anderen konsistent sein, also sich nicht gegenseitig ausschließen, dann wird wieder nach einer Task für das nächste Zielliteral

gesucht. Sollte keines gefunden werden, da die Tasks mutex sind, muss zurück gesprungen und eine andere Task gesucht werden.

Kann für jedes Zielliteral eine Task gefunden werden, die für das System konsistent ist, dann wurde eine Lösung gefunden. Gibt es keine Kombination von Tasks, die es erlaubt, jedes Zielliteral zu bilden, dann muss der Graph weiter expandiert werden. Dazu wird eine weitere Schicht 3 mit Aktionsknoten eingefügt, die die Schicht 2 als Vorbedingung hat und eine Präpositionsschicht, die die Nachbedingung der Aktionsknoten aus Schicht 3 enthält. Der zweite Schritt zur Lösungsfindung läuft somit regressiv ab. Der Graphplan wird wieder extrahiert. Der Planer beginnt mit der neu eingeführten Schicht, der Tasks und versucht wieder eine Kombination von Aktionsknoten zu finden, die die Zielliterale als Folge haben. Ist die Suche in dieser Schicht abgeschlossen, wird rekursiv, die vorherige Aktionsknotenschicht aufgerufen, Der Abbruch erfolgt genauso, wie vorher beschrieben.

5.3. Ausführliches Beispiel¹

Man geht von folgender Situation aus, jemand möchte für seine Freundin ein Essen vorbereiten. Dazu muss die Person kochen, einen Blumenstrauß in eine Vase stellen und das Zimmer, da es dreckig ist, wischen oder Staub saugen. Somit bekommt man 4 Tasks, die der Planer zur Verfügung hat:

Kochen, Blumen-vorbereiten, Wischen, Staubsaugen.

Diese einzelnen Tasks haben Vorbedingungen und Effekte beziehungsweise Nachbedingungen. Um zu Kochen muss die Person saubere Hände haben, aus dem Kochen resultiert ein Essen. Da die Blumen eine Überraschung sein sollen, muss vorher Ruhe herrschen, damit die Freundin nicht aufwacht. Die Blumen sollen ein Geschenk sein. Die Vorbedingung fürs Wischen und Staubsaugen ist, dass das Zimmer schmutzig ist, nach den Aktionen soll das Zimmer sauber sein.

¹ www.marcwagner.com/files/graphplan/graphplan.pdf

Nun können die Aktionen weiter definiert werden:

Kochen: Vorbedingung: saubere-Hände, Nachbedingung: Essen
Blumen-vorbereiten: Vorbedingung: Ruhe, Nachbedingung: Geschenk
Wischen: Vorbedingung: Schmutz, Nachbedingung: dreckige Hände, ¬Schmutz
Saugen: Vorbedingung: Schmutz, Nachbedingung: ¬Ruhe, ¬Schmutz

Die Aktionen sind definiert und aus der Situationsbeschreibung lassen sich nun folgende Startsituation und Zielliterale ableiten:

Start: Ruhe \wedge sauber-Hand \wedge Schmutz
Ziel: Essen \wedge ¬Schmutz \wedge Geschenk

Aus diesen einzelnen Beschreibungen lässt sich nun der erste Graphplan expandieren, wie in Graphik 3 zu sehen.

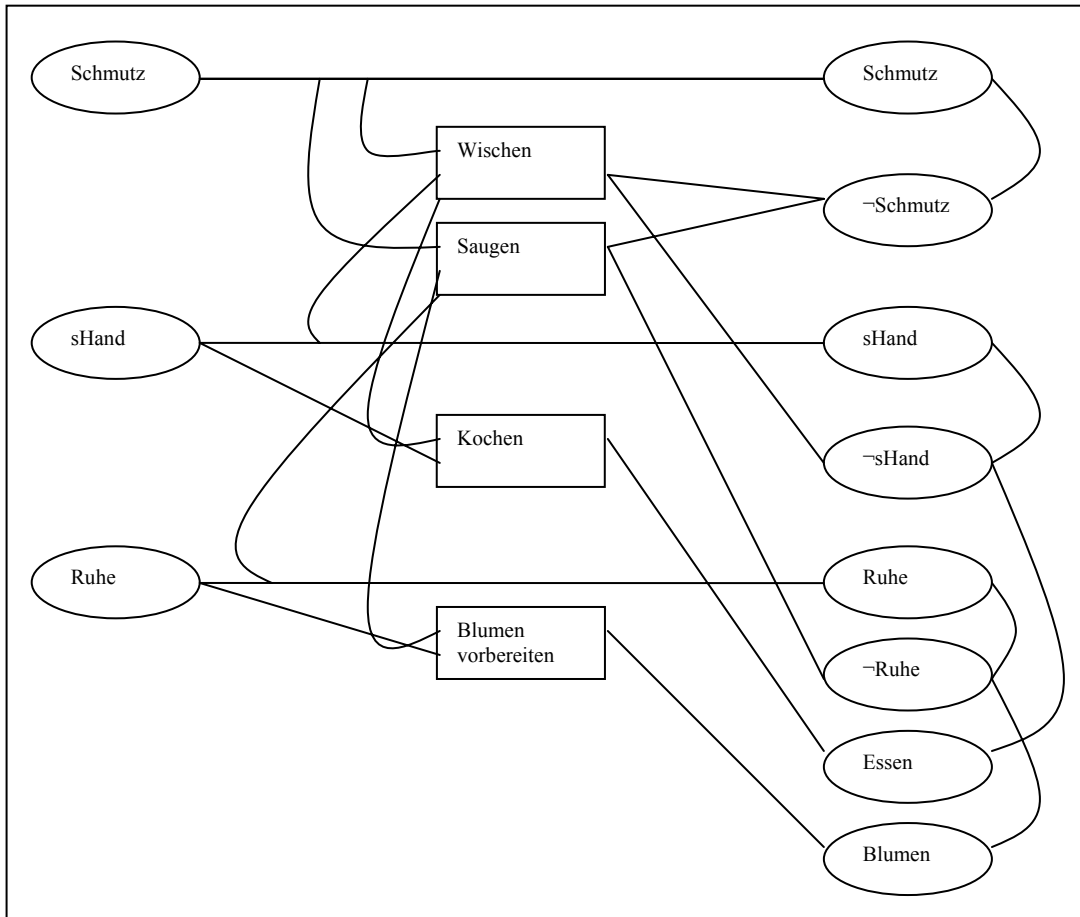


Abbildung 3 - Erste Expansion des Graphen

Anhand dieses Graphen lassen sich nun die Aktionen ableiten, die mutex sind. Wischen, besitzt zum Beispiel als Nachbedingung eine nicht saubere Hand, was mutex zum Essen ist, da Essen als Vorbedingung eine saubere Hand hat, genauso wie Saugen als Effekt nicht Ruhe hat, was sich mit Blumen ausschließt, da Blumen Ruhe voraussetzt, um die Überraschung gelingen zu lassen. Aus diesem Graphen lassen sich nun mehrere Möglichkeiten ableiten, die innerhalb der Lösungsextraktion untersucht werden. Um zu den Zielliteralen Present (Blumen), \neg Schmutz und Essen zu kommen, bieten sich zwei Lösungsmöglichkeiten an. Essen wird durch Kochen erreicht, \neg Schmutz durch Wischen und Blumen (Present) durch Blumen-vorbereiten. Die zweite Lösungsmöglichkeit ersetzt Wischen durch Saugen. Innerhalb dieser Möglichkeiten entstehen allerdings gegenseitige Ausschlüsse, was bedeutet, dass der Graph weiter expandiert werden muss.

Der um zwei Schichten erweiterte Graph ist in Abbildung 4 zu sehen. Trotz der neuen Schichten, hat sich die Zahl der gegenseitigen Ausschlüsse reduziert und die Möglichkeit der Lösungsfindung ist gestiegen. Mit Hilfe einer weiteren Lösungsextraktion lässt sich nun überprüfen, ob eine Lösung existiert.

Nimmt man sich zu erst das Zielliteral \neg Schmutz vor, ist es möglich, dieses in der zweiten Ebene durch Wischen zu erhalten. Wird Kochen, als erstes durchgeführt, verschwindet der gegenseitige Ausschluss mit Kochen, da Kochen schon abgeschlossen ist. Der Konflikt zwischen nicht saubere Hände als Nachbedingung für Wischen und saubere Hände als Vorbedingung für Kochen entfällt. Die Vorbereitung der Blumen ist beliebig einsetzbar, da dessen Nachbedingung nicht mehr mutex ist. Eine andere Möglichkeit zur Lösungsfindung wäre, die Blumen in die Ebene 1 zu setzen und anstatt Wischen, das Staubsaugen zu nutzen. Der Ausschluss zwischen Ruhe und \neg Ruhe, würde dann auch entfallen. Die beiden graphischen Lösungen sind in den Abbildungen 4 und 5 zu finden.

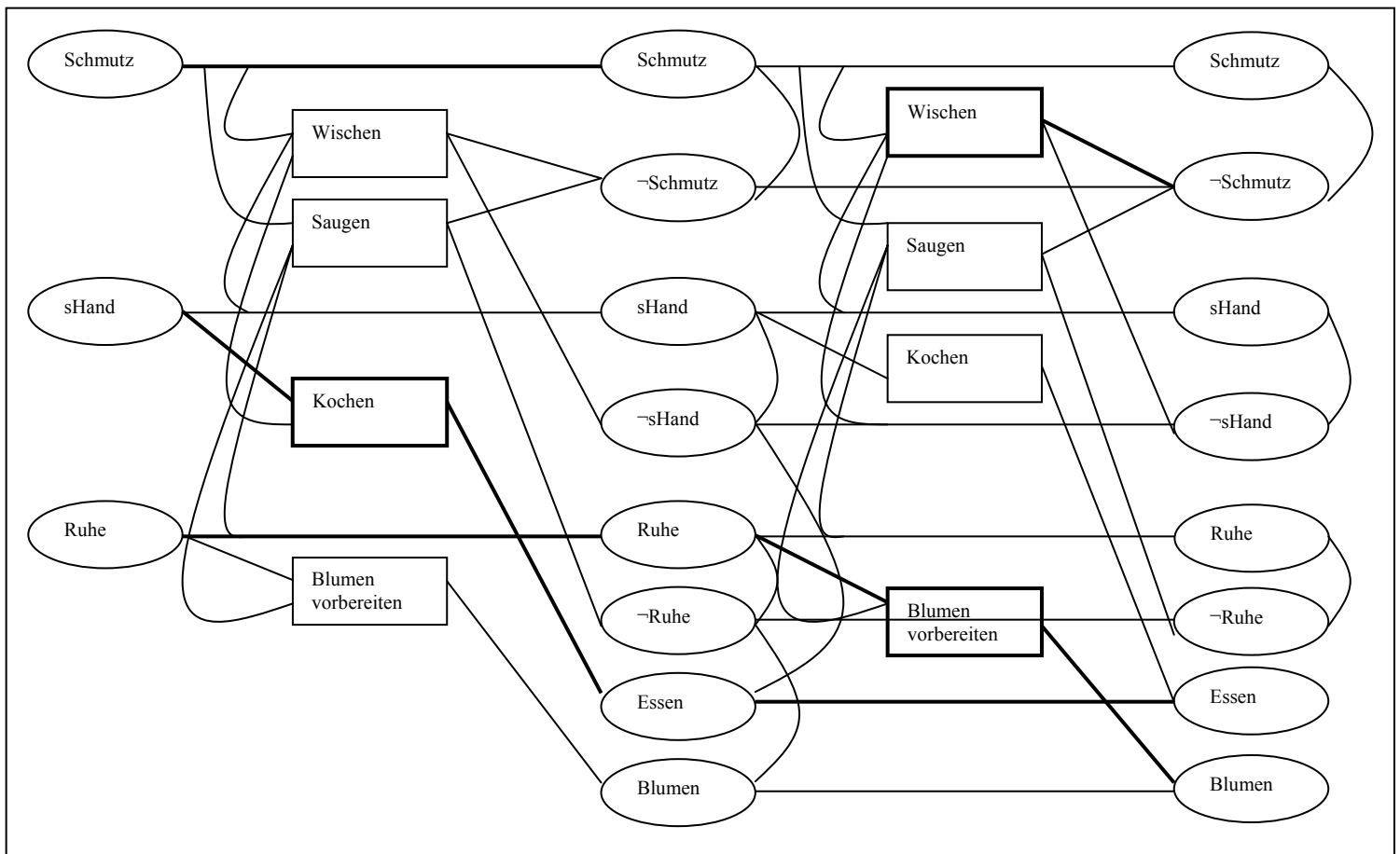


Abbildung 4 - Zweite Expansion und erster Lösungsansatz für das Graphenproblem

Wichtig ist in diesem Beispiel, dass neben der erzeugenden Aktionen auch die Persistenzaktionen genutzt werden, zum Beispiel, dass der Schmutz über die erste Ebene hinweg bestehen bleibt. So wird eine zeitliche Abfolge erkennbar, die den Plan in mehrere zeitabhängige Schichten unterteilt.

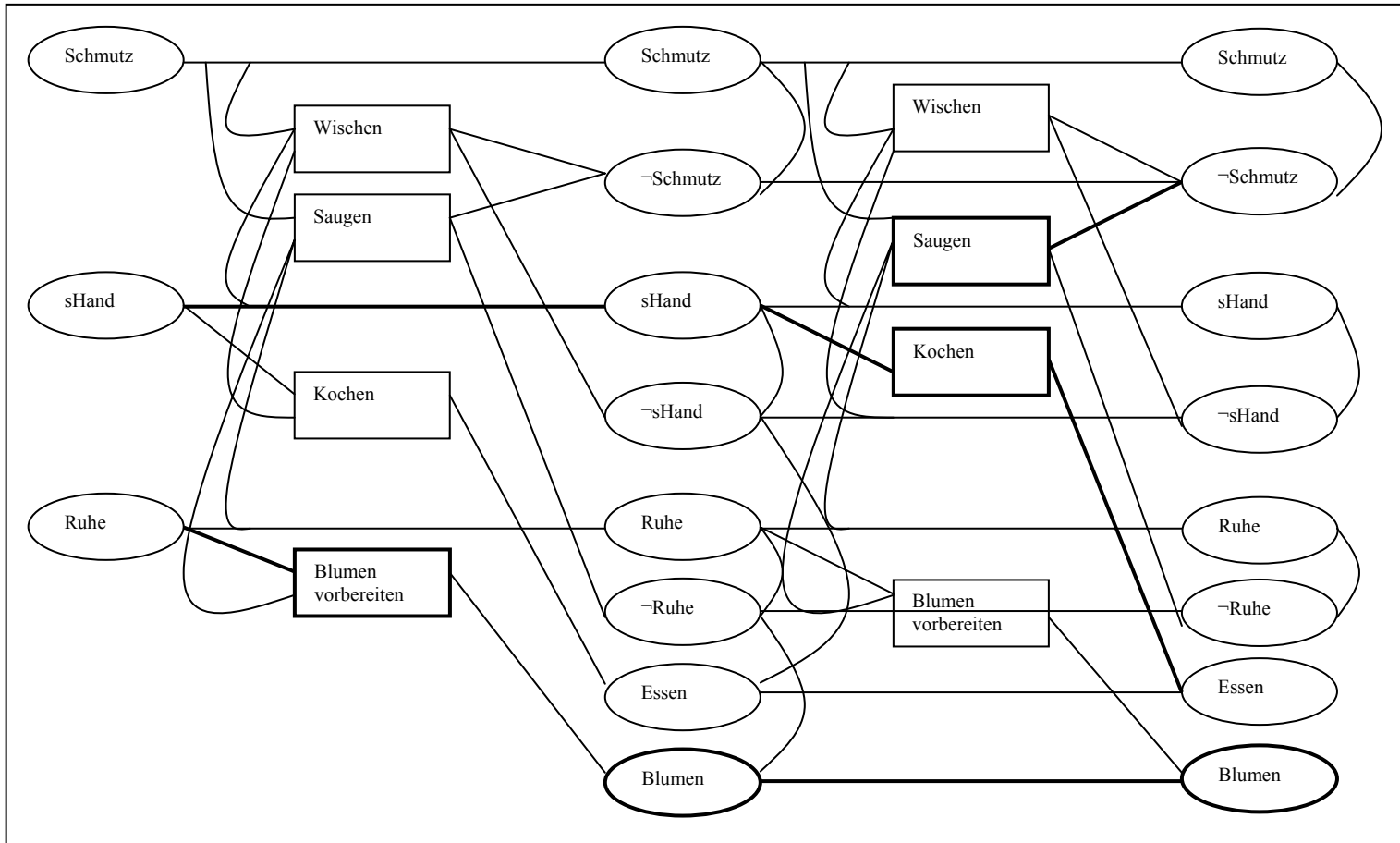


Abbildung 5 - Zweite Lösung für das Graphenproblem

6. Vergleich zwischen POP und Graphenplan²

Der wichtigste Unterschied zwischen den beiden Planungsansätzen sind die Suchräume in denen gesucht wird. Der pop-Planer zerlegt den Planungsraum in viele einzelne Teile, die nacheinander abgearbeitet werden und untereinander zu einem Plan zusammengesetzt werden. Der Graphenplan hingegen, baut erst den gesamten Suchraum auf um dann durch Selektion einen Plan zu extrahieren. Aus diesem Unterschied lässt sich weitergehend ableiten, dass der pop-Algorithmus die Lösung in einer Phase extrahiert und der Graphenplan immer zwei Phasen benötigt, die Expansion und die Extraktion.

7. Fazit – Graphenplan

Nachteilig gegenüber dem pop-Algorithmus lässt sich die fehlende Erweiterbarkeit des Graphenplan anführen. Er besitzt keine Möglichkeit auf mehrere Objekte zu referenzieren oder auf Ereignisse oder besondere Vorkommnisse einzugehen.

Der Graphenplan ist allerdings durch den trivialen Algorithmus des Expandieren und der darauf folgenden Lösungswegfindung sehr schnell. Und es wird immer ein optimaler Plan gefunden.

² <http://nakula.rvs.uni-bielefeld.de/~mirco/download/KI-Zusammenfassung>

8. Quellen

Handbuch der KI, Günther Görz (März 2003)

<http://www.marcwagner.com/files/graphplan/graphplan.pdf>

<http://nakula.rvs.uni-bielefeld.de/~mirco/download/KI-Zusammenfassung>

9. Abbildungsverzeichnis

Abbildung 1 - Algorithmus der Regression.....	5
Abbildung 2 - Beispiel für expandierten Graphen.....	10
Abbildung 3 - Erste Expansion des Graphen.....	14
Abbildung 4 - Zweite Expansion und erster Lösungsansatz für das Graphenproblem.....	15
Abbildung 5 - Zweite Lösung für das Graphenproblem.....	16