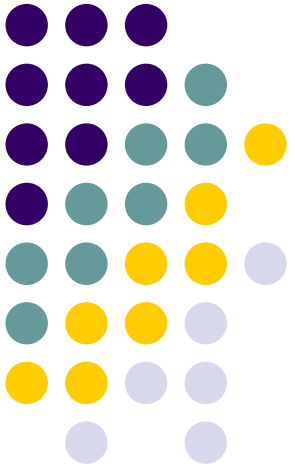
Policy Learning
Teil 2

von Susanne Schilling



Einleitung

Ziele des Ansatzes, den ich vorstellen möchte (von Huang, Selman und Kautz):

- domänenunabhängige Planung
- Lernen von allgemeinen Regeln zur Suche im Zustandsraum
- Regeln sind auch auf andere Planer übertragbar
- Geschwindigkeit der Planer soll verbessert werden
- Skalierbarkeit der Planer soll erhöht werden
- lieber wenige allgemeine statt viele spezielle Regeln
- einschränkungsbasierte Formulierung der Regeln
- kein Hintergrundwissen wird verwendet

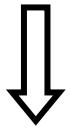
Die Vorgehensweise

Folgende Regeln werden vorgegeben:

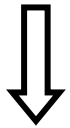
- Pakete dürfen zwischen Standorten innerhalb von Städten mit dem LKW transportiert werden
- Pakete können mit dem Flugzeug zwischen Flughäfen verschiedener Städte transportiert werden
- Pakete können be- und entladen werden
- Flugzeuge werden geflogen und LKWs gefahren

Die Vorgehensweise

Beschreibung des Problems



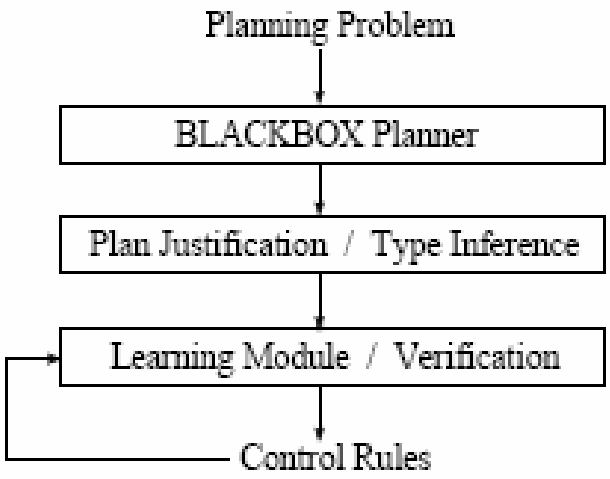
Alle Aktionen brauchen eine Zeiteinheit für die Ausführung



Zu einem Zeitpunkt können mehrere nichtkonfliktäre Aktionen ausgeführt werden

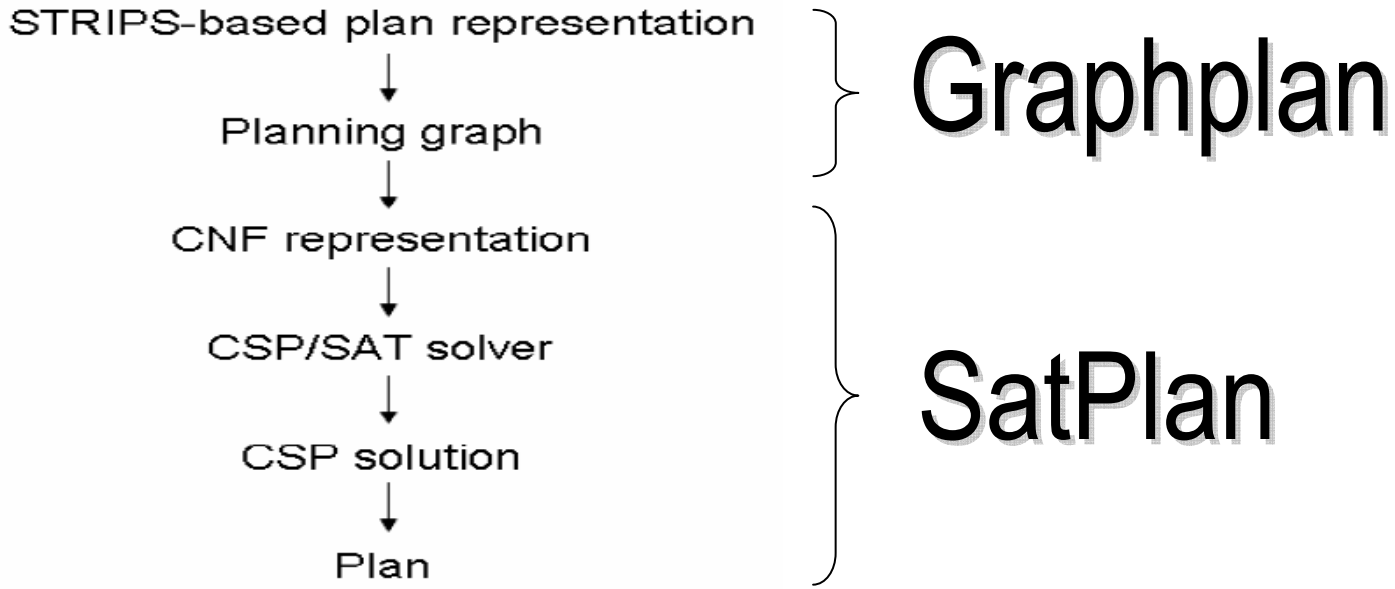
Anzahl der **Zeitschritte**	Anzahl der **Aktionen** pro Zeitschritt
Parallele Länge eines Plans	**sequentielle Länge** eines Plans

Learning Framework



- Zuerst wird dem Blackbox-Planer ein Planungsproblem übergeben
- Dieses Problem wird vom Blackbox-Planer ausgewertet
- Schließlich wird der Plan noch einmal überprüft und Typinformationen generiert
- Dann werden Kontrollregeln für die Planungsdomäne gelernt

Blackbox-Planer



- Zuerst wird dem Blackbox-Planer ein Planungsproblem in STRIPS-Notation übergeben
- Zu dem Problem wird schließlich ein Planungs-Graph erstellt
- Dieser Plan wird schließlich in eine SAT-Kodierung bzw. CNF-Repräsentation umgewandelt
- Anschließend wird das Problem mit einem der Sat-Algorithmen (z.B. WalkSat) oder einem CSP-Solver gelöst

Plan Justification/Type Inference

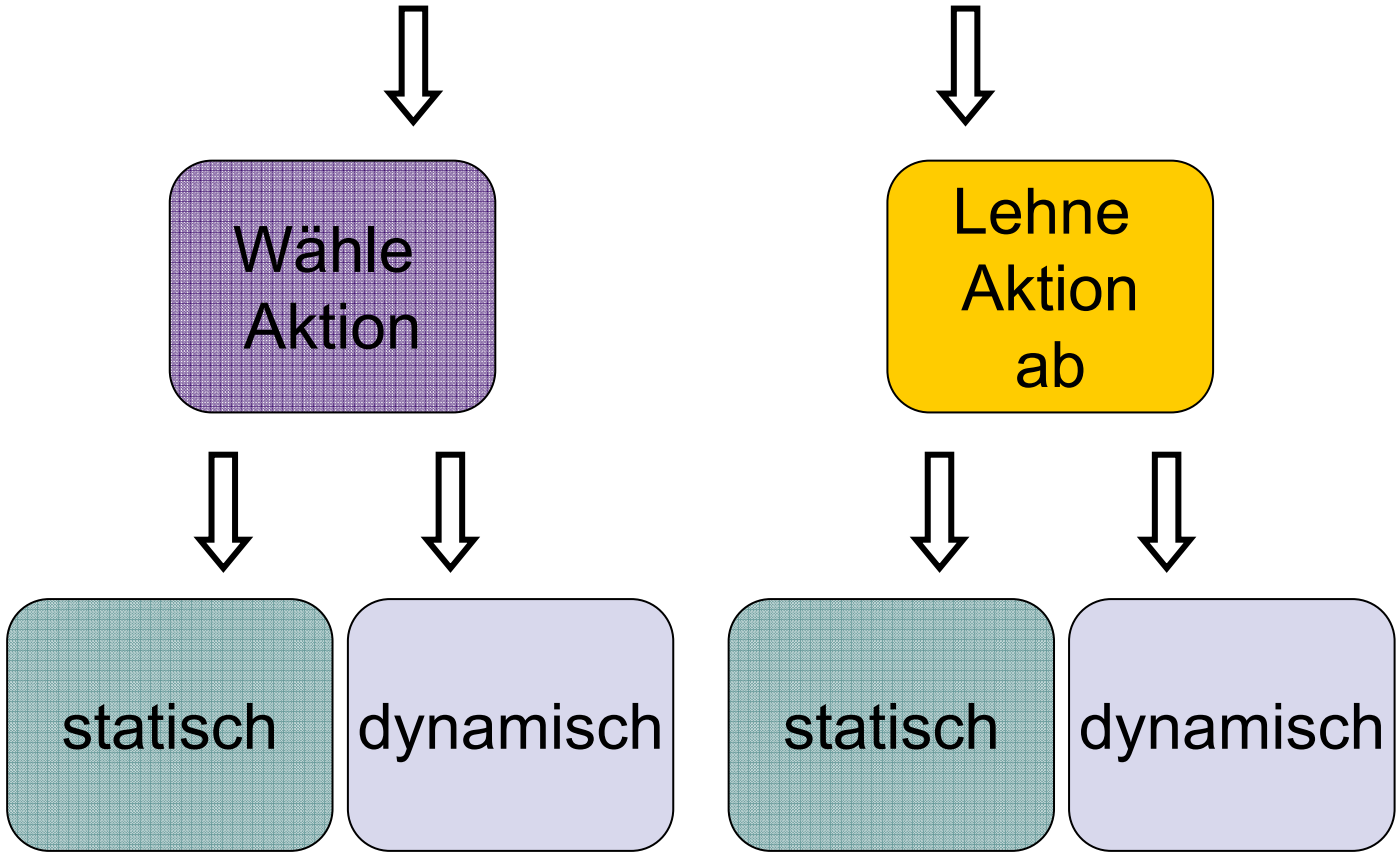
- Nachdem der Blackbox-Planer einen Plan erstellt hat, werden unnötige Aktionen gelöscht
  ⇨ optimale sequentielle Länge
- Der fertige Plan enthält schließlich eine Zustands-beschreibung nach jedem Zeitschritt, welche ausgehend vom Ausgangszustand durch Ausführen der Aktionen berechnet wird

Das Lernmodul

Nutzung des Plans	1. Inkonsistente Regeln verwerfen	2. Menge von pos. und neg. Beispielen für Zielkonzepte werden extrahiert

Zielkonzepte für Aktionen

Für jede Aktion gibt es 2 unterschiedliche Arten von Konzepten



Zielkonzepte für Aktionen

Jedes Konzept wird durch eine Menge von Regeln in einfacher temporaler Logik beschrieben:

- Select-Regeln: sagen etwas darüber aus, unter welchen Bedingungen Aktionen ausgeführt werden
- Reject-Regeln: sagen etwas darüber aus, unter welchen Bedingungen Aktionen nicht ausgeführt werden
- statische Regeln: Der Regelkörper hängt nur von den Anfangs – und Zielzuständen ab, die durch das Planungsproblem gegeben sind (gilt entweder für alle Zeitschritte im Plan, oder für keinen)
- dynamische Regeln:
  Regelkörper hängt davon ab, was in einem bestimmten Zeitschritt gerade wahr ist

Heuristiken für Trainingsbeisp.

Gegeben: Plan P (wurde durch den Planer gefunden)

Aktionen:

- ***real*** im Zeitpunkt i, wenn die Aktion zum Zeitpunkt i im Plan auftaucht
- ***virtuell*** im Zeitpunkt i, wenn dort alle Vorbe-dingungen erfüllt sind, die Aktion aber nicht im Plan auftaucht
- ***mutex virtuell*** im Zeitpunkt i, wenn die Aktion virtuell ist im Zeitpunkt i und es mindestens eine reale Aktion gibt, die sich gegenseitig mit ihr ausschließt

Jede Aktion im Plan wird nach dem oben aufgeführten Schema kategorisiert.

Heuristiken für Trainingsbeisp.

Schließlich werden die positiven und negativen Beispiele für den Lernprozess nach folgendem Schema ausgewählt:

	Select Regel		Reject Regel	
	pos. Beispiel	neg. Beispiel	pos. Beispiel	neg. Beispiel
Statisch	real	virtuell	virtuell	real
dynamisch	real	mutex virtuell	mutex virtuell	real

FOIL

- FOIL(Zielprädikat, Prädikate, Beispiele)
  - Die Beispiele, für die das Zielprädikat wahr ist, werden in Pos gespeichert
  - Die Beispiele, für die das Zielprädikat falsch ist, werden in Neg gespeichert
  - Learned Rules:= wird mit der leeren Menge initialisiert
  - Solange Pos nicht leer ist, Lerne eine neue Regel:
    - NeueRegel:= wird mit der Regel initialisiert, die das Zielprädikat ohne Vorbedingungen darstellt
    - NeueRegelneg:= dieser Menge werden die Beispiele in Neg zugewiesen
    - Solange NeueRegelneg nicht leer ist, füge der neuen Regel ein Literal hinzu, um sie zu spezialisieren:
      - KandidatenLiterale:= generiere neue Literale für die neue Regel, die auf den Prädikaten basieren
      - BestesLiteral:= argmax(für alle L Element von KandidatenLiterale FOILGain(L, NeueRegel)
      - füge BestesLiteral den Vorbedingungen von NeueRegel hinzu
      - NeueRegelneg:= Teilmenge von NeueRegelNeg, die die Vorbedingungen von NeueRegel erfüllt
    - GelernteRegeln := GelernteRegeln + NeueRegel
    - Pos:= Pos - Mitglieder von Pos, die von NeueRegel abgedeckt werden
  - Gib GelernteRegeln zurück

Wie funktioniert FOILGain?

- FOIL fügt probeweise alle Literale nacheinander der Regel hinzu
- In jedem Schritt werden die Bindungen von Werten an Variablen ausgewertet

Beispiel:

Trainingsdaten bestehen aus:

- Enkelin(Sabine,Viktor)
- Vater(Bob,Sabine), Vater(Bob,Tom), Vater(Viktor,Bob)
- weiblich(Sabine)

Start:

Enkelin(x,y) ←

⇨ Diese Regel gilt für alle x und für alle y!

Wie funktioniert FOILGain?

- Die Bindung {y/Viktor, x/Sabine} resultiert in einem pos. Beispiel
- die restlichen 15 Bindungen resultieren in negativen Beispielen
- In jedem Schritt wird die Regel aufgrund der Anzahl der positiven und negativen Bindungen beurteilt

⇨ Die Literale mit den **meisten positiven Bindungen** werden bevorzugt

- Literale mit **neuen Variablen** führen zur Erweiterung der Bindungen

⇨ {y/Viktor, x/Sabine} wird zu {y/Viktor, x/Sabine, z/Bob}

FOILGain

$$FOILGain(L, R) = t * \left(\ln \frac{p_1}{p_1 + n_1} - \ln \frac{p_0}{p_0 + n_0} \right)$$

p_0 = Anzahl von positiven Bindungen der Regel R

n_0 = Anzahl von negativen Bindungen der Regel R

p_1 = Anzahl von positiven Bindungen der Regel R‘
(R‘ = R mit dem neuen Literal)

n_1 = Anzahl von negativen Bindungen der Regel R‘

t = Anzahl der positiven Bindung der Regel R, die nach Hinzufügen des Literals L immer noch abgedeckt werden

FOIL: Generierung der Literale

Wie werden die Kandidatenliterale generiert?

Sei $P(x_1,x_2,\ldots,x_k) \leftarrow L_1,L_2,..,L_n$ die gerade betrachtete Regel.

- $P(x_1,x_2,\ldots,x_k)$ = Regelkopf
- $L_1,L_2,..,L_n$ = Regelkörper

Neue Literale können die folgende Form haben:

- $Q(v_1,v_2,\ldots,v_n)$, wobei Q ein Prädikatenname ist, der in der Prädikatenmenge vorkommt. Die v_i sind Variablen, die schon in der Regel vorhanden sind oder neu eingeführt werden
- $Equal(x_j,x_k)$, wobei x_j und x_k Variablen sind, die bereits in der Regel vorhanden sind
- Die Negation der beiden ersten Formen

FOIL: Generierung der Literale

Beispiel

- Es geht wieder um das Prädikat Enkelin(x,y)
- Regeln für dieses Zielprädikat sollen gelernt werden

Die Suche beginnt mit der allgemeinsten Regel:

Enkelin(x,y) ←

Folgende Literale werden nun generiert:

- Equal(y,x)
- weiblich(x), weiblich(y)
- Vater(x,y), Vater(y,x), Vater(**z**,y), Vater(y,**z**), Vater(**z**,x) (**z wird neu eingeführt**)

Hier wird Vater(z,x) als vielversprechendstes Literal ausgewählt

FOIL: Generierung der Literale

Die Regel sieht nun folgendermaßen aus:

Enkelin(x,y) ← Vater(z,x)

Im nächsten Schritt werden folgende Literale generiert:

- weiblich(z)
- Equal(x,z), Equal(y,z)
- Vater(z,**w**), Vater(**w**,z) (**w wird neu eingeführt**)

Einführung von z ⟹ neue Literale und Einführung der Variable w.

Nun ist das Vater(y,z) am vielversprechendsten und schließlich das Literal weiblich(x). Die Regel wird zu:

Enkelin(x,y) ← Vater(z,x) ∩ Vater(y,z) ∩ weiblich(x)

Der Algorithmus ähnlich FOIL

Die Suche erfolgt hier auf Programmen, die mit einer einfachen, beschränkten, temporalen Logik beschrieben werden. Diese Programme beinhalten folgende Bestandteile:

- **statische Prädikate:** Fakten, deren Wahrheitswert durch keine Aktion geändert werden kann
- **fließende Prädikate:** Aussagen, deren Wahrheitswert sich im Zeitablauf ändern kann
- **Aktionsprädikate:** parametrisierte Aktionen
- **Modaloperator goal:** stellt sicher, dass sein Argument eines der Ziele des Planungsproblems ist
- **Literal:** $x_i = x_j$ | $P(x_1 \ldots x_n)$ | $goal(F(x_1 \ldots x_n))$ | oder die Negation davon
- **Regeln:** Literal(Kopf) + mehrere Literale(Körper)

Der Algorithmus ähnlich FOIL

Wenn für alle Instanzen einer Regel gilt, dass der Kopf den Wert wahr annimmt und dies zu allen Zeitpunkten, zu denen alle Literale im Körper der Regel wahr werden ⇨ dann ist die Regel konsistent bezüglich eines Plans

Instanz: wird gebildet durch Ersetzung der Variablen durch Konstanten

Kopf einer Select – Regel: positives Aktionsprädikat-Literal

Kopf einer Reject – Regel: negatives Aktionsprädikat-Literal

Der Algorithmus ähnlich FOIL

Statische Regeln:

- Positive oder negative Gleichheitsliterale *oder*
- statische Prädikatenliterale *oder*
- Zielliterale

Dynamische Regeln:

- dürfen bereits oben genannte Bestandteile *und*
- müssen mindest. ein negatives oder positives fließendes Literal enthalten

Die Wahl der Literale

Wie in FOIL werden die Literale folgender-maßen ausgewählt:

- Das Literal mit dem größten Nutzen, wenn dieser nah am Maximum liegt, *sonst*
- Alle bestimmten Literale die gefunden wurden, *sonst*
- das Literal mit dem größten positiven Nutzen, *sonst*
- Das erste betrachtete Literal, dass eine neue Variable einführt

Die Wahl der Literale

Die verwendete Gleichung für die Berechnung des Nutzens ist die Gleichung für den Laplace – Schätz-Wert:

$$Gain(r) = \frac{p+1}{p+n+2}$$

- r = die gerade betrachtete Regel
- p = die Anzahl von positiven Beispielen, welche von r abgedeckt werden
- n = die Anzahl von negativen Beispielen, welche von r abgedeckt werden

Ein Beispiel aus der Logistik

Folgender Plan sei vom Blackbox-Planer entwickelt worden:

Initial: (at o1 apt-A), (at o2 apt-B),
(at pln apt-A), (at trk-C apt-C),
(in-city apt-A A), (in-city po-A A), ...
Goal: (at o1 po-C), (at o2 po-C)
Plan:
1 LOAD-AIRPLANE (o1 pln apt-A)
2 FLY-AIRPLANE (pln apt-A apt-B)
3 LOAD-AIRPLANE (o2 pln apt-B)
4 FLY-AIRPLANE (pln apt-B apt-C)
5 UNLOAD-AIRPLANE (o1 pln apt-C)
5 UNLOAD-AIRPLANE (o2 pln apt-C)
6 LOAD-TRUCK (trk-C o1 apt-C)
6 LOAD-TRUCK (trk-C o2 apt-C)
7 DRIVE-TRUCK (trk-C apt-C po-C)
8 UNLOAD-TRUCK (trk-C o1 po-C)
8 UNLOAD-TRUCK (trk-C o2 po-C)

Ein Beispiel aus der Logistik

Die Typangaben sind in folgender Tabelle verzeichnet:

	time	o	p	a	c	l
+	2	o1	pln	apt-A	A	po-C
+	3	o1	pln	apt-B	B	po-C
+	4	o1	pln	apt-B	B	po-C
+	4	o2	pln	apt-B	B	po-C
-	5	o1	pln	apt-C	C	po-C
-	5	o2	pln	apt-C	C	po-C

+ = positive Beispiele
- = negative Beispiele

Ein Beispiel aus der Logistik

Es sollen nun statische Reject-Regeln für die Aktion Flugzeug entladen(o p a) gelernt werden:

- Flugzeug entladen(o1, pln, apt-A) = positives Beispiel zum Zeitpunkt 2, da alle seine Vorbedingungen im Zeitpunkt 2 wahr sind, aber es taucht dort nicht im Plan auf

- Flugzeug entladen(o1, pln, apt-C) = negatives Beispiel zum Zeitpunkt 5, da es im Plan zum Zeitpunkt 5 auftaucht

Die Lernprozedur

Wie läuft nun die Lernprozedur ab?

- 2 bestimmte Literale incity(a c), goal(at(o l)) werden zur Regel hinzugefügt
- Dadurch werden 2 neue Variablen eingeführt: c und l
- Schließlich wird ein neues Literal nicht(incity(l c)) mit dem höchsten Nutzen gefunden und der Regel hinzugefügt
- Jetzt deckt die Regel nur noch positive Beispiele ab und das Lernen kann beendet werden

Future Work

Bei Huang, Selman und Kautz:

- detaillierte, empirische Beurteilung des Ansatzes
- ausdrucksstärkere Sprachen für die Kontrollregeln
- Kreieren hilfsreicher Trainingsbeispiele
- Erforschen aktiven Lernens

Bei Geffner:

- Der Beweis steht noch aus, dass der Ansatz auch auf andere Domänen als die Blocks-World anwendbar ist
- Problem: Anzahl der Konzepte steigt exponentiell mit n an, und nur wenige können ausgeschlossen werden