Symbolische Exploration von Planungsproblemen

Enrico Kaden
mai97jmu@studserv.uni-leipzig.de

Problemseminar: Wissensbasiertes Planen

Zusammenfassung

In den Artikeln *Directed Symbolic Exploration in AI-Planning* [1] und *Exhibiting Knowledge in Planning Problems to Minimize State Encoding Length* [2] beschreiben S. Edelkamp et al. die Zustandskodierung von Planungsproblemen mit binären Entscheidungsdiagrammen. Sie entwickeln zudem die Exploration der so repräsentierten Planungsräume von der Breiten- zur heuristischen Suche fort. Diese Arbeiten der Autoren sollen hier vorgestellt werden.

Beispiel

```
(define (domain logistics-strips)
   (:predicates (OBJ ?obj) (TRUCK ?truck) (LOC ?loc)
                (at ?obj ?loc) (in ?obj1 ?obj2))
   (:action LOAD-TRUCK
      :parameters (?obj ?truck ?loc)
      :precondition (and (OBJ ?obj) (TRUCK ?truck) (LOC ?loc)
                    (at ?truck ?loc) (at ?obj ?loc))
      :effect (and (not (at ?obj ?loc)) (in ?obj ?truck)))
   (:action UNLOAD-TRUCK
      :parameters (?obj ?truck ?loc)
      :precondition (and (OBJ ?obj) (TRUCK ?truck) (LOC ?loc)
                    (at ?truck ?loc) (in ?obj ?truck))
      :effect (and (not (in ?obj ?truck)) (at ?obj ?loc)))
   (:action DRIVE-TRUCK
      :parameters (?truck ?loc-from ?loc-to)
      :precondition (and (TRUCK ?truck) (LOC ?loc-from) (LOC ?loc-to)
                    (at ?truck ?loc-from))
      :effect (and (not (at ?truck ?loc-from)) (at ?truck ?loc-to))))
```

```
(define (problem strips-log-x-1)
   (:domain logistics-strips)
   (:objects package6 package5 package4 package3 package2 package1
             city6 city5 city4 city3 city2 city1
             truck6 truck5 truck4 truck3 truck2 truck1)
   (:init (OBJ package6) ... (OBJ package1)
          (TRUCK truck6) ... (TRUCK truck1)
          (LOC city6) ... (LOC city1)
          (at truck6 city6) (at truck5 city5)
          (at truck4 city4) (at truck3 city3)
          (at truck2 city2) (at truck1 city1)
          (at package6 city3) (at package5 city4)
          (at package4 city1) (at package3 city1)
          (at package2 city1) (at package1 city2))
    (:goal (and (at package6 city1) (at package5 city6)
                (at package4 city3) (at package3 city6)
                (at package2 city6) (at package1 city2))))
```

vereinfachtes Planungsproblem *Logistics* aus dem Planungswettbewerb AIPS 1998 [5]

Zustandskodierung von Planungsproblemen (I)

- Ausführung vor der eigentlichen Plansuche
- Minimierung der Zustandsbeschreibung hinsichtlich der Anzahl der Zustände und der Länge der Kodierung in drei voneinander unabhängigen Phasen
  - konstante Prädikate
  - Merging von Prädikaten
  - Faktum-basierte Exploration des Prädikatenraums

im Beispiel *Logistics*

- 6 Pakete, 6 Städte und 6 Lieferwagen
- 3 unäre Prädikate `OBJ`, `TRUCK` und `LOC` sowie 2 binäre Prädikate `at` und `in`
- naive Zustandskodierung: $3 \cdot (6 + 6 + 6) + 2 \cdot (6 + 6 + 6)^2 = 702$ Bits pro Zustand, wenn ein Bit für jedes Faktum verwendet wird

Konstante Prädikate

Konstante Prädikate

- Instantiierung konstanter Prädikate ist invariant bezüglich der Operatoren
- finden keine Berücksichtigung bei der Zustandskodierung
- im Beispiel *Logistics* sind `OBJ`, `TRUCK` und `LOC` (zur Typisierung der Objekte) konstante Prädikate, somit werden bloß $2 \cdot (6+6+6)^2 = 648$ Bits für die Kodierung eines Zustands benötigt
- treten nicht als Effekte auf

One-Way-Prädikate

- Instantiierungen von *One-Way*-Prädikaten ändern sich zeitlich nur in eine Richtung
- Beispiel aus der *Grid*-Domain [5]
  - Türen können mit Schlüsseln geöffnet und nicht wieder geschlossen werden
  - `locked` und `open` sind One-Way-Prädikate
  - diese Prädikate werden lediglich für diejenigen Objekte kodiert, die im Anfangszustand nicht offen sind
- treten entweder als *add*- oder *delete*-Effekte auf

Zusammenfassen von Prädikaten

Balancierte Prädikate

- im Beispiel *Logistics* ist es nicht erforderlich, alle Instanzen des Prädikats `at` unabhängig voneinander zu betrachten: ein Paket `p` kann sich nicht zeitgleich in zwei oder mehr verschiedenen Städten befinden
- insbesondere steigt die Anzahl der Orte von `p` nicht
- $\#\texttt{at}_2\texttt{(p)}$ Anzahl der Objekte `q`, für welche das Faktum `(at p q)` erfüllt ist
- Verallgemeinerung für n-näre Prädikate: $\#\texttt{pred}_i\texttt{(p}_1\texttt{, . . . , p}_{i-1}\texttt{, p}_{i+1}\texttt{, . . . , p}_n\texttt{)}$ Anzahl der Objekte $\texttt{p}_i$ in einem gegebenen Zustand, für die $\texttt{pred(p}_1\texttt{, . . . , p}_n\texttt{)}$ gilt
- `pred` heißt im i-ten Parameter *balanciert*, wenn es keinen Operator gibt, der $\#\texttt{pred}_i\texttt{(p}_1\texttt{, . . . , p}_{i-1}\texttt{, p}_{i+1}\texttt{, . . . , p}_n\texttt{)}$ erhöht

Merging von Prädikaten

- `at` nicht balanciert, da die Aktion `UNLOAD-TRUCK` $\#\texttt{at}_2\texttt{(p)}$ erhöht
- `at` und `in` werden zu dem neuen Prädikat `at+in` zusammengefaßt, welches wegen $\#\texttt{at+in}_2 = \texttt{at}_2 + \texttt{in}_2 =$ const. im zweiten Parameter balanciert ist

- Zustandskodierung des balancierten Prädikats `at+in`
  - $\#\texttt{at+in}_2\texttt{(p)} = 0$: Informationen über den Ort von `p` werden nicht benötigt
  - $\#\texttt{at+in}_2\texttt{(p)} = 1$: Kodierung der l möglichen Orte von `p` sowie der Prädikate `at` und `in` mit $\lceil \log l \rceil + 1$ Bits
  - $\#\texttt{at+in}_2\texttt{(p)} > 1$: naive Zustandskodierung
- im Beispiel gibt es 6 Pakete und 6 Lieferwagen, deren Orte zu kodieren sind, somit werden nur noch $(6 + 6) \cdot (\lceil \log(6 + 6 + 6) \rceil + 1) = 72$ Bits für die Kodierung eines Zustands gebraucht

Test eines Prädikats `pred` auf Balance im i-ten Parameter

- für jede Aktion `a` wird geprüft, ob ein *add*-Effekt $\texttt{e}_1$ $\#\texttt{pred}_i$ erhöht
- gibt es keinen solchen Effekt $\texttt{e}_1$, ist `pred` im i-ten Parameter balanciert
- ansonsten wird nach einem `pred`-Prädikat $\texttt{e}_2$ in den *delete*-Effekten von `a` gesucht, dessen Argumente sich bis auf den i-ten Parameter mit denen von $\texttt{e}_1$ gleichen
- existiert ein derartiger Effekt $\texttt{e}_2$, balanciert dieser den *add*-Effekt $\texttt{e}_1$ aus
- andernfalls wird nach einem beliebigen *delete*-Effekt $\texttt{e}_2$ gesehen, dessen Argumente sich bis auf den i-ten Parameter mit denen von $\texttt{e}_1$ gleichen
- wird kein solcher Effekt $\texttt{e}_2$ gefunden, schlägt der Test fehl
- sonst wird `pred` mit dem ermittelten *delete*-Effekt `other` zu dem neuen Prädikat `pred+other` zusammengefaßt, das rekursiv auf Balance im i-ten Parameter geprüft wird

Faktum-basierte Exploration des Prädikatenraums

Exploration des Prädikatenraums

- im Beispiel *Logistics* kann sich ein Paket an einem Ort oder im Lieferwagen befinden, nicht aber in einem Paket etc.
- Faktum `f` heißt *erreichbar*, wenn es im Startzustand ist oder durch eine gültige Folge von Operatoren instantiiert wird
- Abschwächung: `f` soll durch eine Aktion mit den Vorbedingungen `g` und `h` *erreichbar* sein, wenn `g` und `h` als erreichbar gekennzeichnet worden sind
- möglicher gegenseitiger Ausschluß von `g` und `h` bleibt dabei unberücksichtigt
- Breitensuche nach allen Prädikatinstanzen, die vom Anfangszustand erreichbar sind
- insbesondere findet die Exploration nicht im Planungsraum statt

Faktum-basierte Exploration

- mit den Fakten, die im Startzustand gelten, wird ein FIFO-Speicher initialisiert, die Menge der erreichbaren Fakten ist leer

- solange der FIFO-Speicher nicht leer ist, wird diesem jeweils ein Faktum `f` entnommen und in die Menge der erreichbaren Fakten eingefügt
- Instantiierung aller Operatoren, deren Vorbedingungen sich in der Menge der erreichbaren Fakten befinden und `f` enthalten
- dem FIFO-Speicher werden diejenigen Effekte der instantiierten Aktionen hinzugefügt, die weder in diesem noch in der Menge der erreichbaren Fakten vorkommen
- vernachlässigt die *delete*-Effekte
- vermeidet die mehrfache Instantiierung eines Operators mit derselben Parameterliste
- im Beispiel *Logistics*
  - wahre Fakten: (OBJ package$i$), (TRUCK truck$i$) und (LOC city$i$) für $i = 1, \ldots, 6$
  - Fluents: (at package$i$ city$j$), (at truck$i$ city$j$) und (in package$i$ truck$j$) für $i, j = 1, \ldots, 6$

Zustandskodierung von Planungsproblemen (II)

- Zusammenführen der Ergebnisse aus den drei vorherigen Phasen
- Prädikate, die weder konstant noch balanciert sind, werden naiv kodiert, d. h. für jedes Faktum wird ein Bit verwendet
- letztere Prädikate sind selten (im Planungswettbewerb AIPS 1998 [5] nur in einem Fall der *Grid*-Domain)

im Beispiel *Logistics*

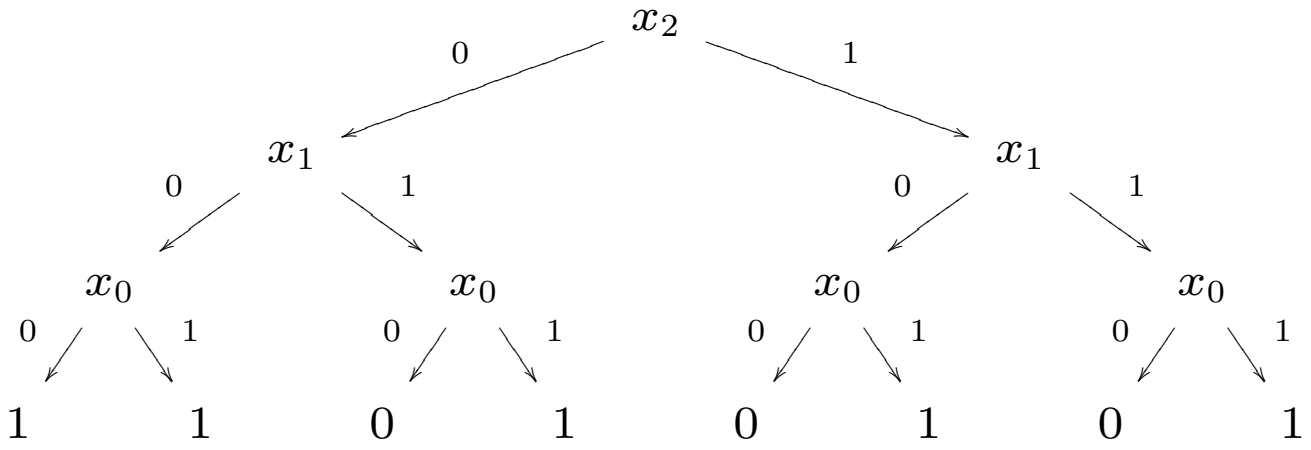
- `(OBJ package`i`)`, `(TRUCK truck`i`)` und `(LOC city`i`)` für $i = 1, \ldots, 6$ sind konstante Prädikate
- `at` und `in` werden zu dem im zweiten Parameter balancierten Prädikat `at+in` zusammengefaßt

`at+in`	`package`i	`city`j	für $i, j = 1, \ldots, 6$
		`truck`j	
	`truck`i	`city`j	
	$3 + 3$ Bits	$(4 + 0) + (3 + 0)$ Bits	

- Kodierung eines Zustands mit $(3 + 3) \cdot ((4 + 0) + (3 + 0)) = 42$ Bits

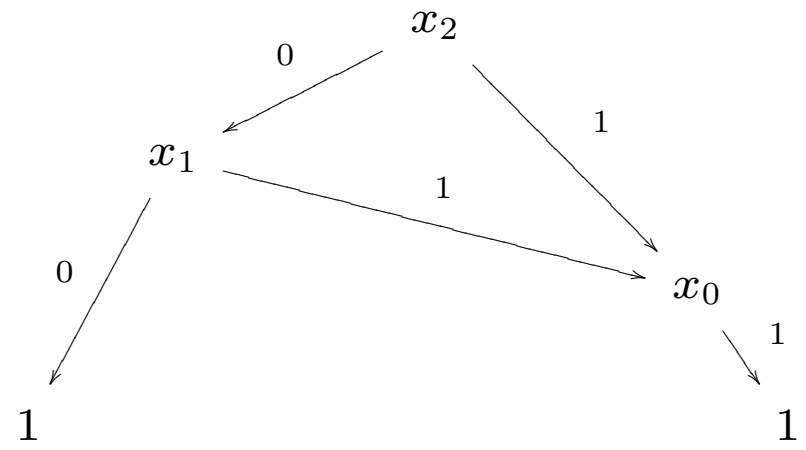
Binäre Entscheidungsdiagramme

- Abbildung einer Zustandsmenge S auf Boolesche Funktionen: die charakteristische Funktion $\phi_S(x)$ ist genau dann wahr, wenn x die Binärkodierung eines Zustands aus S ist
- *Binary decision diagrams* (*BDDs*) zur effizienten Speicherung und Bearbeitung von ϕ_S
- Binärkodierung der Zustandsmenge $\phi_{\{0,1,3,5,7\}} = (\overline{x_2} \wedge \overline{x_1}) \vee (\overline{x_2} \wedge x_1 \wedge x_0) \vee (x_2 \wedge x_0)$



- Reduktion von BDDs
  - Entfernen von Knoten mit isomorphen Nachfolgern
  - isomorphe Teilbäume werden mehrfach genutzt

- reduzierte Darstellung der Zustandsmenge $\phi_{\{0,1,3,5,7\}}$



- es existieren Boolesche Funktionen, deren Repräsentation als BDDs exponentiell viele Knoten erfordern – in praktischen Planungsproblemen selten
- Anzahl der Knoten von der Reihenfolge der Variablen abhängig
- reduzierte geordnete binäre Entscheidungsdiagramme sind eindeutig
- Übergangsrelation T über ein Zustandspaar (s', s) ist genau dann wahr, wenn s' ein Vorgänger von s ist, es also eine Aktion gibt, die s' in s überführt
- T wird als Menge von Zustandstupeln auf Boolesche Funktionen abgebildet
- Berechnung der Übergangsrelation: welche Variablen in der Binärkodierung der Zustände ändern sich durch einen Operator und welche nicht
- im Beispiel *Logistics* wird T als BDD mit $2 \cdot 42$ Bits kodiert

Symbolische Exploration

- S Startzustandsmenge, G Zielzustandsmenge
- S_i Menge der Zustände, die von S in i Schritten erreichbar sind
- Initialisierung mit $S_0 = S$
- Zustand s gehört zu ϕ_{S_i}, wenn es einen Vorgängerzustand s' in $\phi_{S_{i-1}}$ sowie einen Operator gibt, welcher s' in s transformiert

$$\phi_{S_i}(s) = \exists s'(\phi_{S_{i-1}}(s') \wedge T(s', s))$$

- Plansuche terminiert, wenn der Schnitt der aktuellen Zustandsmenge S_i und der Menge der Zielzustände G nicht leer ist
- Iterationsindex i: Länge des gefundenen Plans

Unidirektionale Breitensuche

- Algorithmus

  $Open := \phi_S$

  do

  $Succ(x) := \exists x'(Open(x') \wedge T(x', x))$

  $Open := Succ$

  while $Open \wedge \phi_G \equiv 0$

- ϕ_S, ϕ_G, $Open$, $Succ$ und T als binäre Entscheidungsdiagramme kodiert
- findet einen optimalen Plan

Bidirektionale Breitensuche

- Algorithmus

  $fOpen := \phi_S$
  $bOpen := \phi_G$
  do
    if $forward$
      $Succ(x) := \exists x'(fOpen(x') \wedge T(x', x))$
      $fOpen := Succ$
    else
      $Succ(x') := \exists x(bOpen(x) \wedge T(x', x))$
      $bOpen := Succ$
  while $fOpen \wedge bOpen \equiv 0$

- parallele Vorwärts- und Rückwärtssuche
- findet einen optimalen Plan mit der minimalen Lösungslänge $f + b$ (Anzahl der Iterationen der Vorwärts- und Rückwärtssuche)

Breitensuche mit Forward Set Simplification

- Algorithmus

  $Closed := Open := \phi_S$

  do

  $Succ(x) := \exists x'(Open(x') \wedge T(x', x))$

  $Open := Succ \wedge \neg Closed$

  $Closed := Closed \vee Succ$

  while $Open \wedge \phi_G \equiv 0$

- $Closed$ enthält alle diejenigen Zustände, die schon besucht worden sind
- vermeidet Wiederholungen in der Exploration des Planungsraums
- findet einen optimalen Plan

BDDA*-Suche

- heuristische Plansuche
- $g(s)$ Pfadlänge vom Startzustand zum Zustand s
- $h(s)$ minimaler Abstand von s zur Zielzustandsmenge
- Berechnung von $f(s)$

$$\begin{aligned} f(s) &= g(s) + h(s) \\ &= g(s') + 1 + h(s) && (s' \text{ Vorgänger von } s) \\ &= f(s') - h(s') + 1 + h(s) \end{aligned}$$

- heuristische Abschätzung von $h(s)$
- einfache Heuristik im Beispiel *Logistics*: Anzahl der Pakete, die sich noch nicht am Zielort befinden

- A*-Algorithmus

  $Open(f, x) := (h(x) = f) \wedge \phi_S(x)$

  do

  $f_{\min} := \min\{f | f \wedge (Open(f, x) \neq \emptyset)\}$

  $Min(x) := \exists f(Open(f, x) \wedge (f = f_{\min}))$

  $Rest := Open \wedge \neg Min$

  $Succ(f, x) := \exists x'(Min(x') \wedge T(x', x) \wedge (f = f_{\min} - h(x') + 1 + h(x)))$

  $Open := Rest \vee Succ$

  while $Open \wedge \phi_G \equiv 0$

- findet einen optimalen Plan, wenn $h(s') \leq h(s) + 1$, s Nachfolger des Zustands s', gilt
- je besser die heuristische Funktion dem konkreten Planungsproblem angepaßt ist, desto mehr wird der Suchraum eingeschränkt
- Anzahl der zu untersuchenden Knoten kann im Vergleich zur Breitensuche für viele praktische Planungsprobleme signifikant reduziert werden

Best First-Suche

- Algorithmus

  $Open(f, x) := (h(x) = f) \wedge \phi_S(x)$

  do

  $f_{\min} := \min\{f | f \wedge (Open(f, x) \neq \emptyset)\}$

  $Min(x) := \exists f(Open(f, x) \wedge (f = f_{\min}))$

  $Rest := Open \wedge \neg Min$

  $Succ(f, x) := \exists x'(Min(x') \wedge T(x', x) \wedge (f = h(x)))$

  $Open := Rest \vee Succ$

  while $Open \wedge \phi_G \equiv 0$

- von BDDA* abgeleitet
- nur die Pfadlänge $h(s)$ vom Zustand s zur Zielzustandsmenge wird betrachtet
- Annahme: h ist auf einem Lösungsweg $[s_0, s_1, \ldots, s_n]$ fallend, d. h. es gilt

$$h(s_0) > h(s_1) > \cdots > h(s_n)$$

- Anzahl der zu besuchenden Zustände kann im Vergleich zu BDDA* für nicht wenige praktische Planungsprobleme weiter reduziert werden
- findet im allgemeinen keinen optimalen Plan

Heuristiken

- vor der eigentlichen Plansuche wird zunächst für jedes Fluent p_i der Abstand $h(p_i)$ zum Ziel geschätzt
- vernachlässigt etwaige gegenseitige Ausschlüsse von Fluents
- HSP-Heuristik (siehe auch [3])
  - während der Faktum-basierten Exploration des Prädikatenraums wird für jedes aus dem FIFO-Speicher entnommene Fluent p_i (Wiederholungen sind ausgeschlossen) die Distanz $\hat{h}_S(p_i)$ zum Startzustand S berechnet

    $$\hat{h}_S(p_i) = 1 + \hat{h}_S(C) \qquad (C \text{ Menge der Vorbedingungen von } p_i)$$

  - additive oder Maximum-Heuristik

    $$\hat{h}_S(C) = \sum_{p'_i \in C} \hat{h}_S(p'_i) \qquad \text{bzw.} \qquad \hat{h}_S(C) = \max_{p'_i \in C} \{\hat{h}_S(p'_i)\}$$

  - Ermittlung von $h(p_i)$

    $$h(p_i) = \hat{h}_{p_i}(G) \qquad (G \text{ Zielzustand})$$

- FF-Heuristik
- aus den Paaren $(p_i, h(p_i))$, welche in einer *Heuristic Pattern Database* gespeichert sind, wird nun die minimale Pfadlänge $h(s)$ des Zustands s zur Zielzustandsmenge bestimmt
- additive Heuristik: Überschätzung des Abstands, wenn die Fluents nicht unabhängig voneinander sind

$$h(s) = \sum_{\phi_{\{s\}} \Rightarrow \phi_{\{p_i\}}} h(p_i)$$

- Maximum-Heuristik: findet eine untere Schranke für die Distanz

$$h(s) = \max_{\phi_{\{s\}} \Rightarrow \phi_{\{p_i\}}} \{h(p_i)\}$$

- geeignete heuristische Funktionen ermöglichen die gesteuerte Exploration des Planungsraums in Richtung der Zielzustände

Ergebnisse

- im Vergleich zur eigentlichen Plansuche kurze Ausführungszeiten für die Zustandskodierung und die Berechnung der heuristischen Funktion
- besonders gute Resultate bei der Best First-Suche mit einer (additiven) FF-Heuristik
- bei den Planungswettbewerben AIPS 2000 sowie 2002 für *Distinguished Performance* ausgezeichnet
- im Beispiel *Logistics* von MIPS [4] gefundener optimaler Plan

  ```
  (load-truck package3 truck1 city1), (load-truck package4 truck1 city1),
  (load-truck package6 truck3 city3), (load-truck package5 truck4 city4),
  (load-truck package2 truck1 city1), (drive-truck truck4 city4 city6),
  (drive-truck truck1 city1 city3), (drive-truck truck3 city3 city1),
  (unload-truck package5 truck4 city6), (unload-truck package4 truck1 city3),
  (unload-truck package6 truck3 city1), (drive-truck truck1 city3 city6),
  (unload-truck package3 truck1 city6), (unload-truck package2 truck1 city6)
  ```

- Rechenzeit des MIPS-Planers
  - Breitensuche: 34 Sekunden
  - A*-Suche mit einer FF-Heuristik: 0.1 Sekunden

Literatur

[1] S. Edelkamp, Directed Symbolic Exploration in AI-Planning, AAAI-Spring Symposium on Model-based Validation of Intelligence, Stanford, pp. 84-92, AAAI-Press, 2001.

[2] S. Edelkamp, M. Helmert, Exhibiting Knowledge in Planning Problems to Minimize State Encoding Length, European Conference on Planning (ECP), Durham, pp. 135-147, LNAI, Springer, 1999.

[3] B. Bonet, H. Geffner, Planning as heuristic search, Artificial Intelligence, Volume 129, Issues 1-2, pp. 5-33, 2001.

[4] `http://www.informatik.uni-freiburg.de/~edelkamp/`

[5] `http://www-2.cs.cmu.edu/~aips98/planning-competition.html`