Planning - II

Manuela Veloso
CMU 15-381, Fall 2000

- Plan-space search
- Graphplan
- Satplan

These lecture slides should be complemented with the examples discussed and worked out on the blackboard during the lecture.

Outline: Selecting ACTIONS

- Planning algorithms
- Comparison of planning algorithms
- Learning in planning
- Planning, execution, and learning
- Behavior-based (reactive) planning
- Action selection in multiagent systems

Several Other Planning Algorithms

- TWEAK [Chapman 87], SNLP [McAllester & Rosenblitt 91 , UCPOP [Penberthy and Weld 92]
  - Emphasis on plan-space *search*
- NONLIN [Tate 76], O-PLAN [Tate], SIPE [Wilkins 88]
  - Emphasis on plan decomposition
- UNPOP, Planning and acting [McDermott 78]
- Reactive planning [Georgeff & Lansky 87], [Firby 87], [Hendler & Sanborn 87]

Several Other Planning Algorithms

- Action and time [Allen 84] [Dean & McDermott 87]
- Walksat, Satplan [Selman et al. 92, Kautz & Selman 92, 96]
- Flecs [Veloso & Stone 95]
- Graphplan [Blum & Furst 95]
- MBP [Cimatti, Roveri, Traverso 98]
- UMOP [Jensen & Veloso 00]

Plan-Space Partial-Order Nonlinear Planning

SNLP Planning Algorithm *McAllester & Rosenblitt 91*

1. Terminate if the goal set is empty.
2. Select a goal g from the goal set and identify the plan step that needs it, S_{need}.
3. Let S_{add} be a step (operator) that adds g, either a new step or a step that is already in the plan. Add the causal link $S_{add} \xrightarrow{g} S_{need}$, constrain S_{add} to come before S_{need}, and enforce bindings that make S_{add} add g.

Plan-Space Partial-Order Nonlinear Planning

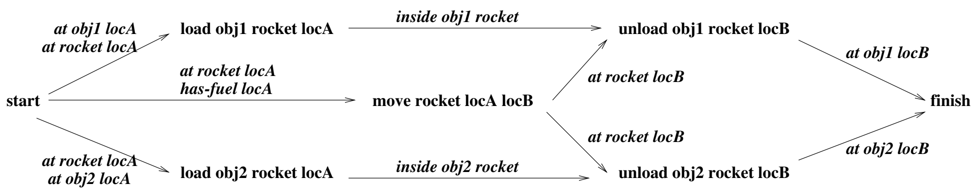
4. Update the goal set with **all** the preconditions of the step S_{add}, and delete g.
5. Identify *threats* and resolve the conflicts by adding ordering or bindings constraints.

- A step S_k threatens a causal link $S_i \xrightarrow{g} S_j$ when it occurs between S_i and S_j, and it adds or deletes p.
- Resolve threats by using *promotion*, *demotion*, or *separation*.

Plan-space Planning

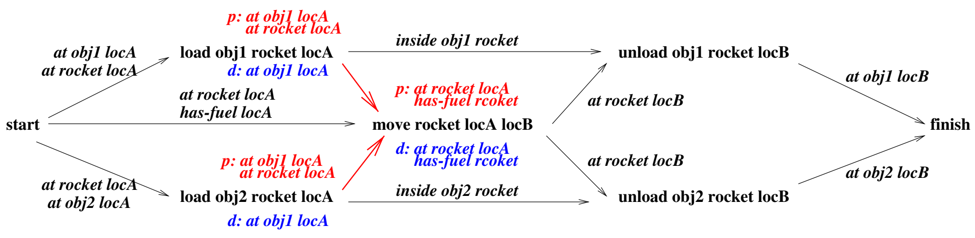
- Complete, sound, and optimal.
- Optimal handling of goal orderings.

Plan-space Planning



Example - LINKING

Plan-space Planning



Example - THREATS

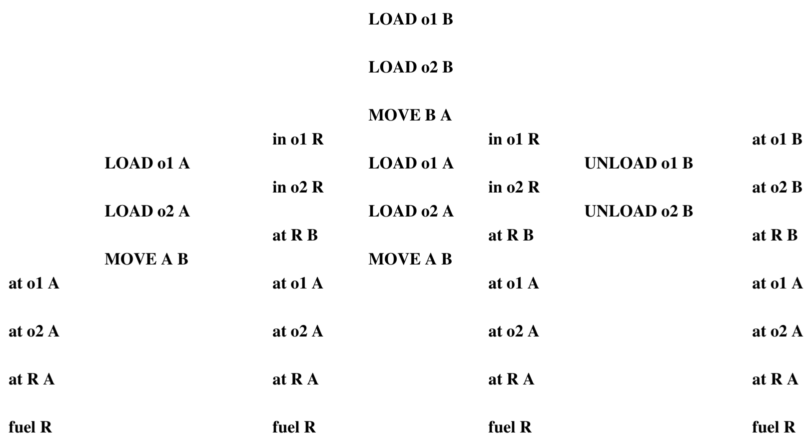
Graphplan

Blum & Furst 95

- Forward search combined with backward search.
- Uses planning graphs.
- Two stages:
  - ▷ Extend: One more time step in the planning graph.
  - ▷ Search: Is there a valid plan in the planning graph?
- Graphplan finds a plan or proves that no plan has fewer "time steps."

Planning Graphs

- A literal may exists at level $i + 1$ if it is an Add-Effect of some action in level i.
- Two propositions p and q are *exclusive* in a proposition-level if ALL actions that add p are exclusive of ALL actions that add q.
- Actions A and B are *exclusive* at action-level i, if:
  - ▷ Interference: A (or B) deletes a precondition or an Add-Effect of B (or A).
  - ▷ Competing Needs: p is a precondition of A and q is a precondition of B, and p and q are exclusive in proposition-level i.



Extending a Planning Graph

- To create an action-level i:
  - ▷ Add each instantiated operator, for which all of its preconditions are present at proposition-level i AND *no two of its preconditions are exclusive*.
  - ▷ Add all the no-op actions.
- Determine the exclusive actions.
- To create a proposition-level $i + 1$:
  - ▷ Add all the effects of the inserted actions at action-level i - distinguishing add and delete effects.
- Determine the exclusive propositions.

Exclusivity Examples

- Exclusive Actions: (Move A B) deletes a precondition of (Load o1 A). Therefore exclusive (existence of threats).
- Exclusive Propositions: (at R A) and (at R B) at time 2 are exclusive. (at R A) is added by a no-op and (at R B) is added by (Move A B) and no-op and (Move A B) are exclusive actions.
- Exclusive Actions: Then (Load o1 A) and (Load o2 B) are exclusive because (at R A) and (at R B) are exclusive.
- Propositions can be exclusive in some time step and not in others: If (at o1 A) and (at R A) at time 1, then (in o1 A) and (at R B) are exclusive at time 2, but not at time 3.

Searching a Planning Graph

- Level-by-level backward-chaining approach to use the exclusivity constraints.
- Given a set of goals at time t, identify all the sets of actions (including no-ops) at time $t - 1$ who add the those goals and are not exclusive. The preconditions of these actions are new goals for $t - 1$.

Planning as Satisfiability

- One interpretation: "first-order deductive theorem-proving does not scale well."
- One solution: "propositional satisfiability"
- Uniform clausal representation for goals and operators.
- Stochastic local search is a powerful technique for planning.

SatPlan

- Assume the plan has n (time-parallel) steps. *(strong assumption)*
- Initial state: completely specified at time 0.
  at-o1-A_0 $\wedge$ at-o2-A_0 $\wedge$ at-R-A_0
- Goal: specified at time $2n$.
  at-o1-B_6 $\wedge$ at-o2-B_6
- Actions: specified at *odd* times; An action implies its preconditions and effects.
  ($\neg$load-o1-A_1 $\vee$ at-o1-A_0) $\wedge$ ($\neg$load-o1-A_1 $\vee$ at-R-A_0) $\wedge$ ($\neg$load-o1-A_1 $\vee$ in-R-A_2) $\wedge$ ($\neg$load-o1-A_1 $\vee$ $\neg$at-o1-A_2)

Comparison of Planning Algorithms

- Complete nonlinear state-space planning
- Plan-space planning
- Graphplan
- Satplan
- And more

Is there a *universally* best planning algorithm?

State-space and Plan-space

- Planning is NP-hard.
- Two different planning approaches: state-space and plan-space planning

	State-space	*Plan-space*
Commitments in plan step orderings	Yes	No
Therefore, suffer with goal orderings	Yes	No
Therefore, handle goal interactions	Poorly	Efficiently

Step Ordering Commitments

WHY?

Use of the STATE of the world while planning

In Prodigy4.0 advantages include:

- Means-ends analysis - plan for goals that reduce the differences between current and goal states.
- Informed selection of operators - select operators that need less planning work than others.
- State useful for learning, generation and match of conditions supporting informed decisions.
- Helpful for generating anytime planning - provide *valid*, executable, plans at any time.

Parallel between Commitments - Example

Operator Polish
preconds: ()
adds: polished
deletes: ()

Operator Drill-Hole
preconds: ()
adds: has-hole
deletes: polished

Goal: *polished* and *has-hole* Initial state: empty Prodigy4.0	Goal: *polished* and *has-hole* Initial state: *polished* SNLP
- plan for goal *polished* - select *Polish* • order *Polish* as first step - plan for goal *has-hole* - select *Drill-Hole* • order *Drill-Hole* $\succ$ *Polish* • *polished* deleted, backtrack - *Polish* $\succ$ *Drill-Hole*	- plan for goal *polished* - select *Initial* state • link *Initial* to *polished* - plan for goal *has-hole* - select *Drill-Hole* • link *Drill-Hole* to *has-hole* • threat - relink *polished* - select *Polish* - link *Polish* to *polished* - *Polish* $\succ$ *Drill-Hole*

Serializability and Linkability

- A set of subgoals is *serializable* [Korf]:
  - If there exists some ordering whereby they can be solved sequentially,
  - without ever violating a previously solved subgoal.
- Easily serializable, laboriously serializable
- A set of subgoals is *easily linkable*:
  - If, independently of the order by which the planner links these subgoals to operators,
  - it never has to undo those links.
  - Otherwise it is *laboriously linkable*.

Laboriously Linkable Goals

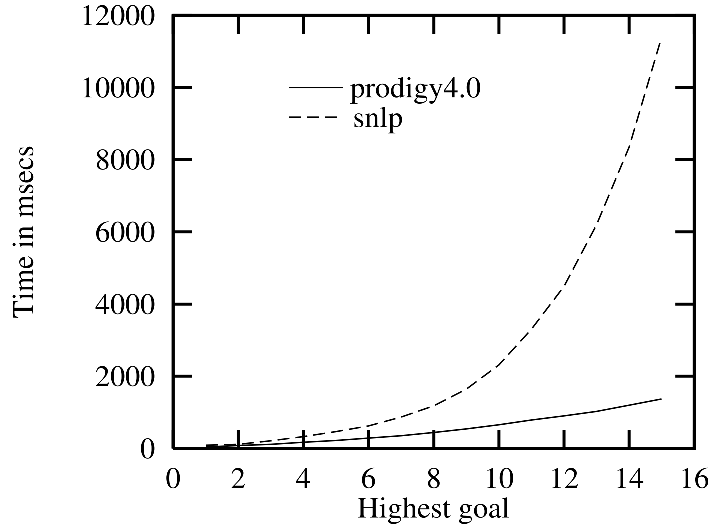
```
operator A_i                    operator A_*
    preconds   g_*, g_{i-1}         preconds   ()
    adds       g_i                  adds       g_*
    deletes    g_*                  deletes    ()
```

operator A_i
  preconds g_*, g_{i-1}
  adds g_i
  deletes g_*

operator A_*
  preconds ()
  adds g_*
  deletes ()

Initial state: g_*
Goal statement:
g_*, g_5
Plan:
$A_1, A_*, A_2, A_*, A_3, A_*, A_4, A_*, A_5$

Laboriously Linkable Goals



FLECS – Flexible Commitment Strategy

Operator:	**designate-roller** <wall> <roller> <color>	**fill-roller** <roller> <color>	**paint-wall** <wall> <roller> <color>
preconds:	(clean <roller>) (needs-painting <wall>)	(clean <roller>) (chosen <roller> <color>)	(ready <wall> <roller> <color>) (filled-with-paint <roller> <color>)
adds:	(ready <wall> <roller> <color>) (chosen <roller> <color>)	(filled-with-paint <roller> <color>)	(painted <wall> <color>)
deletes:		(clean <roller>)	(ready <wall> <roller> <color>) (needs-painting <wall>)

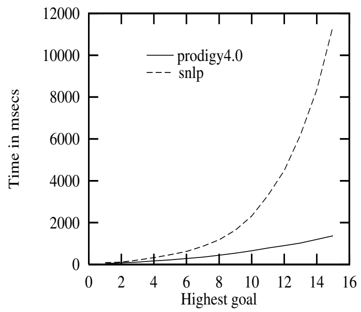
Initial State
(needs-painting wallA)
(needs-painting wallB)
(needs-painting wallC)
(needs-painting wallD)
(needs-painting wallE)
(clean roller1)
(clean roller2)

Goal Statement
(painted wallA red)
(painted wallB red)
(painted wallC red)
(painted wallD green)
(painted wallE green)

An Optimal Solution
<Designate-Roller wallA roller1 red>
<Designate-Roller wallB roller1 red>
<Designate-Roller wallC roller1 red>
<Fill-Roller roller1 red>
<Paint-Wall wallA roller1 red>
<Paint-Wall wallB roller1 red>
<Paint-Wall wallC roller1 red>
<Designate-Roller wallD roller2 green>
<Designate-Roller wallE roller2 green>
<Fill-Roller roller2 green>
<Paint-Wall wallD roller2 green>
<Paint-Wall wallE roller2 green>

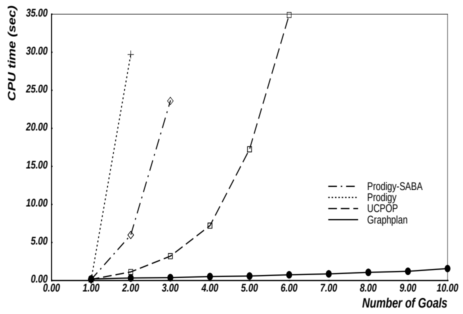
	time(sec)	solution
eager applying	500	no
eager subgoaling	500	no
variable strategy	4	yes

Performance



Nonlinear state-space planning can suffer from goal orderings, but can use state information to reduce the search space

Veloso et al.



Graph-based planning combines forward state expansion and exclusivity analysis with backwards goal search

Blum et al.

Performance (cont.)

Prob	UMOP OBDDs		Prodigy4.0 NL state		STAN graphplan		HSP A* heuristics		IPP graphplan		Blackbox satplan	
1	20	11	80	15	46	11	2007	13	50	15	113	11
2	150	17	200	23	1075	17	2150	21	380	23	7820	17
3	710	23	210	31	54693	23	2485	31	3270	31	-	-
4	1490	29	370	39	3038381	29	3060	37	26680	39	-	-
5	3600	35	430	47	-	-	3320	47	226460	47	-	-
6	7260	41	590	55	-	-	3779	53	-	-	-	-
7	13750	47	800	63	-	-	4797	63	-	-	-	-
8	23840	53	960	71	-	-	5565	71	-	-	-	-
9	36220	59	1240	79	-	-	6675	79	-	-	-	-
10	56200	65	1560	87	-	-	7583	85	-	-	-	-
11	84930	71	1820	95	-	-	9060	93	-	-	-	-
12	127870	77	2240	103	-	-	10617	101	-	-	-	-
13	197170	83	2660	111	-	-	12499	109	-	-	-	-
14	290620	89	3200	119	-	-	15050	119	-	-	-	-
15	411720	95	3740	127	-	-	16886	125	-	-	-	-
16	549610	101	4350	135	-	-	20084	135	-	-	-	-
17	746920	107	5030	143	-	-	23613	143	-	-	-	-
18	971420	113	5900	151	-	-	26973	151	-	-	-	-
19	1361580	119	6810	159	-	-	29851	157	-	-	-	-
20	1838110	125	7710	167	-	-	33210	165	-	-	-	-

planning time in ms; number of plan steps

UMOP: Jensen & Veloso (CMU); **HSP**: Geffner & Bonet (Simon Bolivar Univ, Venezuela);
STAN: Long & Fox (Durham Univ, UK); **IPP**: Köhler (Freiburg Univ, Germany);
Blackbox: Kautz & Selman (AT&T, Cornell Univ)

Performance (cont.)

Prob	UMOP OBDDs		Prodigy4.0 NL state		STAN graphplan		HSP A* heuristics		IPP graphplan		Blackbox satplan	
1	20	7	30	7	19	7	2121	7	10	7	11	7
2	20	7	70	7	18	7	2104	7	10	7	12	7
3	20	7	30	7	19	7	2144	7	10	7	14	7
4	20	7	40	7	20	7	2188	7	10	7	16	7
5	20	7	30	7	21	7	2208	7	10	7	18	7
6	20	7	40	7	22	7	2617	7	10	7	20	7
7	30	7	50	7	22	7	2316	7	20	7	22	7
8	10	7	40	7	23	7	2315	7	20	7	24	7
9	20	7	50	7	25	7	2357	7	-	-	26	7
10	30	7	80	7	26	7	2511	7	10	7	29	7
11	10	7	60	7	27	7	2427	7	30	7	30	7
12	20	7	60	7	28	7	2456	7	30	7	32	7
13	20	7	90	7	29	7	3070	7	20	7	36	7
14	20	7	60	7	31	7	2573	7	30	7	35	7
15	20	7	100	7	32	7	2577	7	30	7	38	7
16	20	7	70	7	34	7	2699	7	10	7	39	7
17	20	7	100	7	35	7	2645	7	30	7	41	7
18	20	7	90	7	37	7	2686	7	10	7	43	7
19	20	7	110	7	39	7	2727	7	30	7	45	7
20	20	7	90	7	40	7	2787	7	20	7	47	7

planning time in ms; number of plan steps

Performance (cont.)

Prob	STAN graphplan		HSP A* heuristics		IPP graphplan		Blackbox satplan	
1	767	27	79682	43	900	26	2062	27
2	4319	32	97114	44	-	-	6436	32
5	364932	29	144413	26	2400	24	-	-
7	-	-	788914	112	-	-	-	-
11	12806	34	86195	30	6940	33	6544	32

planning time in ms; number of plan steps

▷ UMOP needs tractable representation (on-going research).

▷ Prodigy needs control knowledge (planning by analogy or control rules).

Summary – Planning

- Several different planning algorithms.
- **There is not a planning strategy that is universally better than the others.**
- Even for a particular planning algorithm: **There is no single domain-independent search heuristic that performs more efficiently than others for all problems or in all domains.**

Learning is appropriate for **ANY** planner.

Comparison

- The action and problem *representation*, in addition to the planning algorithm significantly affects efficiency.
- Learning action models, control knowledge - several intermediate successes.
- Fink (CMU,99) - Learning representation.

Promising steps towards answering the question:

Which action model,
which planning algorithm,
which heuristic control
is the most efficient for a given task?