Planning research review

Planning is the task of coming up with a sequence of actions to achieve a goal. This is not only fundamental to humans but also for intelligent systems. It has a wide variety of applications like Spacecraft planning, large scale machinery overhaul procedures, search and rescue emergency response, rocket launches, delivery truck scheduling, robotics, gaming and help desk support etc. In this paper, I will provide a brief overview of 3 historical developments in the field of AI planning.

STRIPS:

"STRIPS" stands for "Stanford Research Institute Problem Solver". It was developed by Fikes and Nilsson at Stanford research Institute in 1971. This was the first major planning system and even today, many systems use specification languages that are variants of the one used in STRIPS. In STRIPS, we restrict the kind of things we can specify compared to First Order logic or situation calculus but it helps in solving the situations better. It is mainly composed of

- States which are function-free ground literals
- Goals which conjunctions of literals with variables as needed
- Actions which have descriptive name, preconditions and effect

Plans are described as a set of plan steps with ordering constraints where each step is one of the operators for the problem.

Example: A STRIPS problem involving transportation of air cargo between airports

$$
\begin{aligned}
&Init(At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK) \\
&\quad \land Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2) \\
&\quad \land Airport(JFK) \land Airport(SFO)) \\
&Goal(At(C_1, JFK) \land At(C_2, SFO)) \\
&Action(Load(c, p, a), \\
&\quad \text{PRECOND: } At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a) \\
&\quad \text{EFFECT: } \neg At(c, a) \land In(c, p)) \\
&Action(Unload(c, p, a), \\
&\quad \text{PRECOND: } In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a) \\
&\quad \text{EFFECT: } At(c, a) \land \neg In(c, p)) \\
&Action(Fly(p, from, to), \\
&\quad \text{PRECOND: } At(p, from) \land Plane(p) \land Airport(from) \land Airport(to) \\
&\quad \text{EFFECT: } \neg At(p, from) \land At(p, to))
\end{aligned}
$$

ADL, PDDL:

ADL or Action Description Language was developed by Pednault in 1986 to encode more realistic problems by relaxing some of the STRIPS restrictions. It allows both +ve and -ve literals in States, conjunctions & disjunctions in goals and has an open world assumption (unmentioned literals are treated as unknown).

STRIPS and ADL led to the development of another representational language called PDDL which stands for "Planning Domain Definition Language". It aimed to be computer-parsable and become the standard

for representing planning problems. This standardization allowed researchers to exchange benchmark problems and compare results and also for conducting the International Planning Competition fairly. PDDL includes sublanguages for STRIPS, ADL, and the hierarchical task networks

Components of a PDDL planning problem are

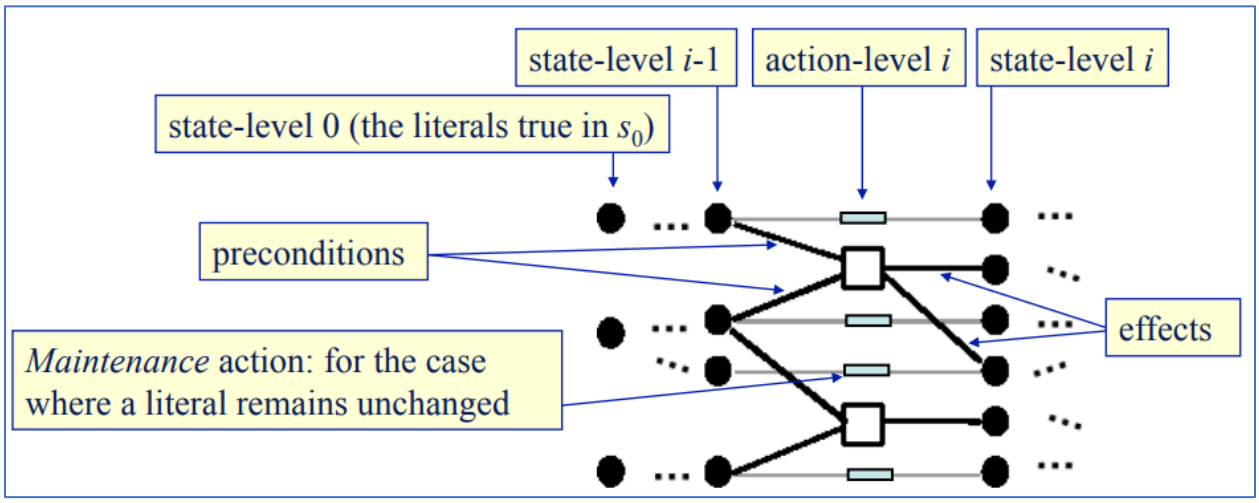
- Objects: Things in the world that interest us.
- Predicates: Properties of objects that we are interested in; can be true or false.
- Initial state: The state of the world that we start in.
- Goal specification: Things that we want to be true.
- Actions/Operators: Ways of changing the state of the world.

Planning Graph:

A planning graph is a special data structure that gives very good heuristic estimates. It consists of sequence of levels that correspond to time steps in a plan. Each level contains a set of actions and a set of literals that could be true at that time step depending on the actions taken in previous steps. For every +ve and -ve literal C, we add a persistence action with precondition C and effect C.

Illustration:

- Nodes at action-level i: actions that might be possible to execute at time i
- Nodes at state-level i: literals that might possibly be true at time i
- Edges: preconditions and effects

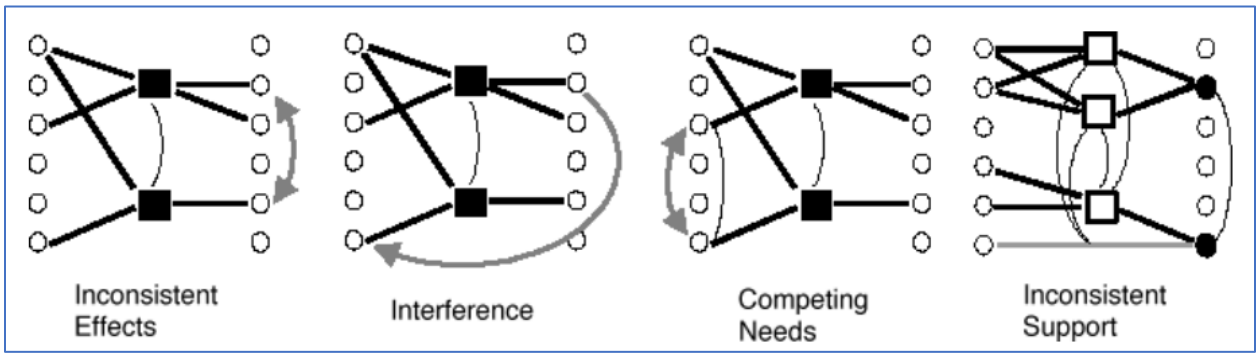


We also define the conflicts between actions that would prevent them from occurring together. They are called mutexes and apply to literals too if one literal is the negation of the other(inconsistent support) or if each possible pair of actions that could achieve the two literals is mutually exclusive. Two actions at the same action-level are mutex if

- Inconsistent effects: an effect of one negates an effect of the other
- Interference: one deletes a precondition of the other

- Competing needs: they have mutually exclusive preconditions

These are shown diagrammatically below -



A planning graph, once constructed, is a rich source of information about the problem. For example, a goal that does not appear in the final level of the graph cannot be achieved by any plan. To estimate the cost of a conjunction of goals, there are three simple approaches –

- max-level heuristic which takes the maximum level cost of any of the goals
- level sum heuristic which assumes the subgoals as independent and returns the sum of the level costs of the goals
- set-level heuristic finds the level at which all the literals in the conjunctive goal appear in the planning graph without any pair of them being mutually exclusive

Example of a planning graph : Have Cake & Eat Cake problem

$Init(Have(Cake))$
$Goal(Have(Cake) \wedge Eaten(Cake))$
$Action(Eat(Cake)$
PRECOND: $Have(Cake)$
EFFECT: $\neg Have(Cake) \wedge Eaten(Cake))$
$Action(Bake(Cake)$
PRECOND: $\neg Have(Cake)$
EFFECT: $Have(Cake)$

Figure 11.11 The "have cake and eat cake too" problem.

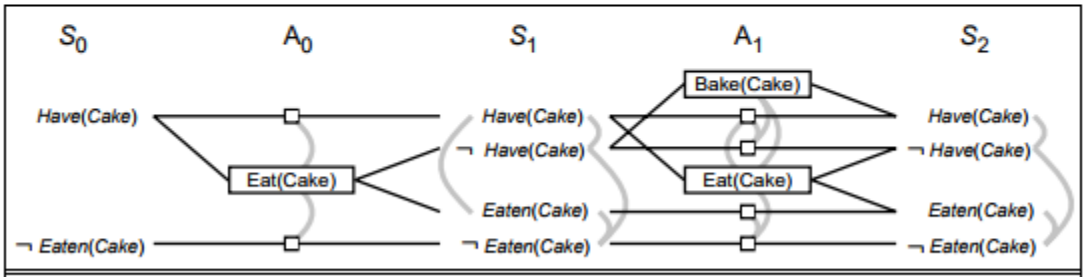


Figure 11.12 The planning graph for the "have cake and eat cake too" problem up to level S_2. Rectangles indicate actions (small squares indicate persistence actions) and straight lines indicate preconditions and effects. Mutex links are shown as curved gray lines.

References:

- https://www.youtube.com/watch?v=VmgMlP1Nevk&list=PL6EE0CD02910E57B8&index=19
- https://www.cs.umd.edu/~nau/planning/slides/chapter06.pdf
- http://aima.cs.berkeley.edu/2nd-ed/newchap11.pdf