

Planning Algorithms

Murray Shanahan

Overview

- Computing the effects of actions
 - STRIPS
- Computing plans
 - Partial order planning (POP)
 - Forward planning with heuristics
 - Satisfiability planning (SAT)

Logic Programs versus Explicit Algorithms

- The logic programming approach to planning tries to follow Bob Kowalski's dictum:

$$\textit{Algorithm} = \textit{Logic} + \textit{Control}$$

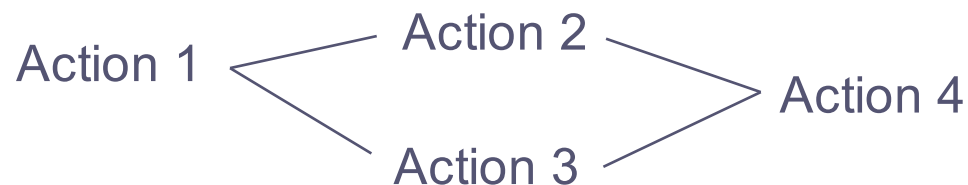
- This is a beautiful idea. But to build a practical planner it's usually easier to design a planning algorithm explicitly
- Moreover, the frame problem doesn't then arise at the implementation stage (as it does at the specification stage)
- Because in order to *compute* the effects of an action (as opposed to reasoning about them in logic), we can directly modify a data structure representing the situation (such as a list of fluents)

The STRIPS Approach

- A classic way to directly compute the effects of actions is the STRIPS approach
- According to the STRIPS approach, all we need to do to compute the effects of an action on a situation is:
 - Represent the situation as a list L of fluents
 - Represent the effects of an action with three lists: a list of fluents to be added (`Addlist`), a list of those to be deleted (`DeleteList`) and a list of `Preconditions`
 - Add fluents to and delete fluents from L according to the `Addlist` and `DeleteList`, assuming all the fluents in `Preconditions` occur in L
 - Obviously fluents that are in neither `Addlist` nor `DeleteList` will be unaffected by this

Partial Order Planning

- Partially ordered plans
 - A plan is *totally ordered* if it specifies the relative ordering of every action it contains
 - A plan that leaves the relative ordering of some of its actions unspecified is *partially ordered*



Executing actions in order 1, 2, 3, 4 or in order 1, 3, 2, 4 are both legitimate

- Partial order planning algorithms generate partially ordered plans, using the concepts:
 - Protected links
 - Threats
 - Promotion and demotion

Partial Order Planning Concepts

- Protected links
 - A protected link constrains a fluent to hold between two given time points
 - Protected links are established when an action is added to the plan to achieve a certain goal
- Threats
 - A protected link from T1 to T2 is threatened if an action is added to the plan that *could* occur between T1 and T2 which terminates the fluent being protected
- Promotion and demotion
 - Extra ordering constraints can be added to the plan to ensure either that such an action occurs before T1 or after T2

Partial Order Planning (POP) Algorithm

```
1  while goal list non-empty
2      choose a goal <F1,T1> from goal list
3      choose an action <A,T2> whose effects include F1
4      for each precondition F2 of A add <F2,T2> to goal list
5      add <A,T2> to plan (if not already a member)
6      add T2 < T1 to plan
7      add <T2,F1,T1> to protected links
8      for each <A',T3> in plan that threatens some <T4,F3,T5>
          in protected links
9          choose either
10             promotion: add T3 < T4 to plan
11             demotion: add T5 < T3 to plan
12      end for
13 end while
```

Search in the POP Algorithm

- Note that lines 2, 3, and 9 of the algorithm are non-deterministic choices
- This defines a search tree
- Some search strategy has to be used to explore this search tree – depth-first or breadth-first, for example
- Note also that the pseudo-code here doesn't mention the initial situation. But the necessary modifications are very simple

POP Example

- Consider the following situation calculus effect axioms

$Holds(At(x), Result(Go(x),s))$

$\neg Holds(At(y), Result(Go(x),s)) \leftarrow Holds(At(y),s) \wedge x \neq y$

$Holds(Have(x), Result(Buy(x),s)) \leftarrow Holds(At(y),s) \wedge Sells(y,x)$

$Sells(HWS, Drill)$

$Sells(SM, Milk)$

- and the following goal (where situation S holds at time $T1$)

$Holds(Have(Drill), S)$

$Holds(Have(Milk), S)$

- POP carries out the following computation

POP Computation (1)

Goal list	Plan	Protected links
<Have(Drill),T1> <Have(Milk),T1>		
<Have(Milk),T1> <At(HWS),T2>	<Buy(Drill),T2>, T2<T1	<T2,Have(Drill),T1>
<At(HWS),T2> <At(SM),T3>	<Buy(Drill),T2>, T2<T1 <Buy(Milk),T3>, T3<T1	<T2,Have(Drill),T1> <T3,Have(Milk),T1>
<At(SM),T3>	<Buy(Drill),T2>, T2<T1 <Buy(Milk),T3>, T3<T1 <Go(HWS),T4>, T4<T2	<T2,Have(Drill),T1> <T3,Have(Milk),T1> <T4,At(HWS),T2>

POP Computation (2)

Goal list	Plan	Protected links
	$\langle \text{Buy}(\text{Drill}), T2 \rangle, T2 \langle T1$ $\langle \text{Buy}(\text{Milk}), T3 \rangle, T3 \langle T1$ $\langle \text{Go}(\text{HWS}), T4 \rangle, T4 \langle T2$ $\langle \text{Go}(\text{SM}), T5 \rangle, T5 \langle T3$	$\langle T2, \text{Have}(\text{Drill}), T1 \rangle$ $\langle T3, \text{Have}(\text{Milk}), T1 \rangle$ $\langle T4, \text{At}(\text{HWS}), T2 \rangle$ $\langle T5, \text{At}(\text{SM}), T3 \rangle$
	$\langle \text{Buy}(\text{Drill}), T2 \rangle, T2 \langle T1$ $\langle \text{Buy}(\text{Milk}), T3 \rangle, T3 \langle T1$ $\langle \text{Go}(\text{HWS}), T4 \rangle, T4 \langle T2$ $\langle \text{Go}(\text{SM}), T5 \rangle, T5 \langle T3$ $T5 \langle T4, T3 \langle T4$	$\langle T2, \text{Have}(\text{Drill}), T1 \rangle$ $\langle T3, \text{Have}(\text{Milk}), T1 \rangle$ $\langle T4, \text{At}(\text{HWS}), T2 \rangle$ $\langle T5, \text{At}(\text{SM}), T3 \rangle$

Under threat

Promotion of Go(SM)

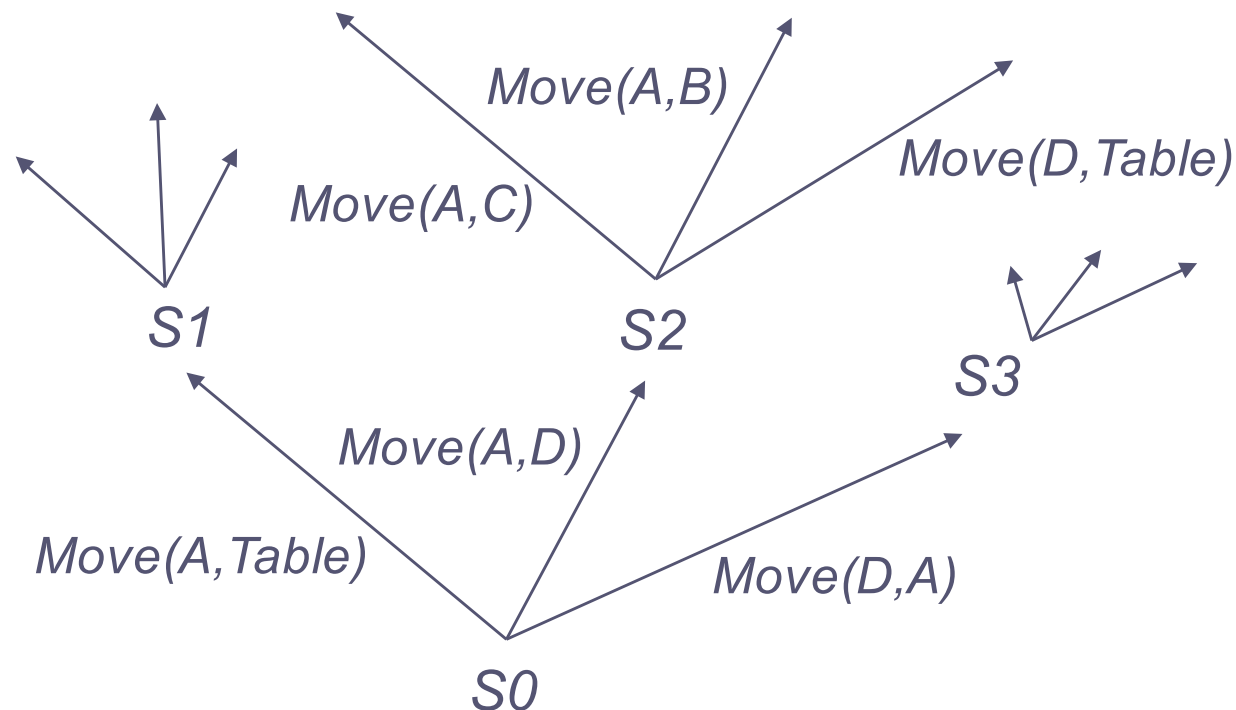
Demotion of Go(HWS)

Using Heuristics

- Most of the planners we are considering are general-purpose planners. That is to say, they assume no prior knowledge of the domain
- But with the use of a few domain-specific heuristics, we can build much faster planners
- One approach that uses heuristics is *forward planning*

Forward Planning (1)

- A forward planning algorithm starts in the initial situation, then considers all possible first actions that can be performed, then all possible second actions, and so on, until it finds a plan that leads to the goal

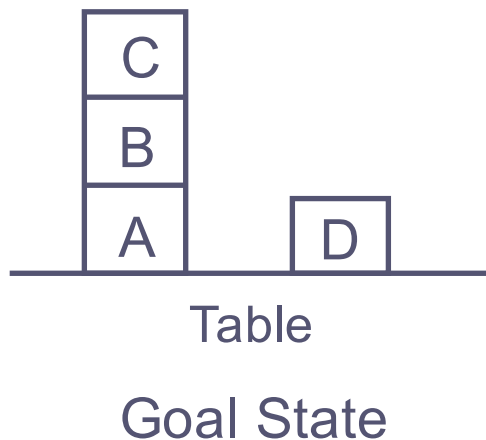


Forward Planning (2)

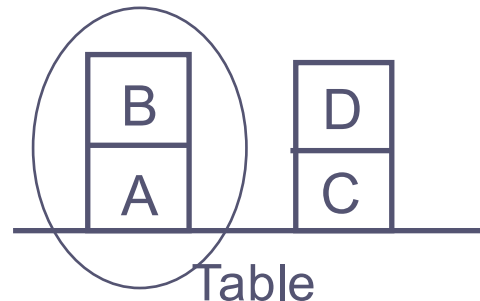
- The search tree is typically very large. And the number of nodes contained in the tree increases very rapidly with the number of actions considered
- It is impractical to explore the whole tree using, say, unmodified breadth-first search
- But a few simple domain-specific heuristics allow many branches of the tree to be pruned at each stage. This results in an effective planner
- To preserve the completeness of the planning algorithm, the heuristics must never prune branches that could lead a solution, unless they are provably redundant

Blocks World Heuristics

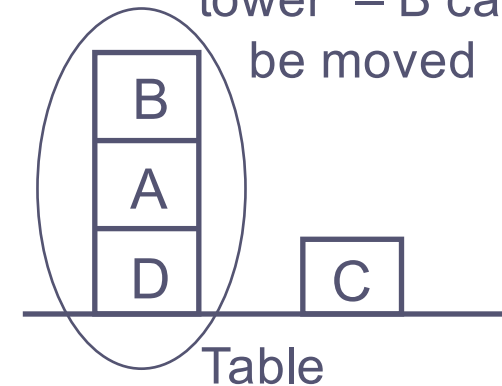
- Here are two useful Blocks World heuristics
 - Don't move x from y to z and then move x from z back to y again
 - Don't demolish a "good tower"
- A *good tower* is a pile of blocks resting on the table that is in the order required by the goal state



A "good tower" –
so don't move B



Not a "good
tower" – B can
be moved



Satisfiability Planning

- Partial order planning was the most efficient type of *general purpose* planning algorithm for many years. But its performance is still very poor for some problems, eg: the Blocks World with more than about six blocks!
- In the late 1990s, several planning techniques were developed that out-perform POP on such problems
- One of these is satisfiability planning
- The idea is to translate a given planning problem into a (typically very large) instance of propositional satisfiability
- This new problem can then be solved using an efficient off-the-shelf satisfiability algorithm

Satisfiability Problems

- Satisfiability (SAT) problems are the quintessence of combinatorial problem solving
- 3-variable SAT problems are the basis of the theory of NP-completeness
- Given a *propositional conjunctive normal form* (CNF) formula, such as

$$(A \vee \neg B \vee \neg C) \wedge (B \vee C \vee D)$$

the task is to find an assignment of true or false to each variable such that the whole formula is true

SAT Planning: the Translation

- The challenge is to translate a given planning problem into a propositional conjunctive normal form formula. A CNF formula is a conjunction of disjunctions of negated or un-negated atomic propositions. In other words, it has the form:

$$([\neg]P_{1,1} \vee \dots \vee [\neg]P_{1,n}) \wedge \dots \wedge ([\neg]P_{k,1} \vee \dots \vee [\neg]P_{k,m})$$

- We'll see how to represent effects, preconditions, and frame axioms in CNF
- Recall that:

$$\begin{aligned} P \leftarrow Q &\text{ is equivalent to } P \vee \neg Q \\ P \leftarrow Q \wedge R &\text{ is equivalent to } P \vee \neg Q \vee \neg R \end{aligned}$$

SAT Planning Effects

- Let's consider the same Blocks World example as before
- Our CNF formula must include (propositional equivalents of) *every ground instance* of:

$$\neg \text{Happens}(\text{Move}(x,y),t) \vee \text{Holds}(\text{On}(x,y),t+1)$$

- and, for $y \neq z$:

$$\neg \text{Happens}(\text{Move}(x,y),t) \vee \neg \text{Holds}(\text{On}(x,z),t+1)$$

- This will include, for example:

$$\begin{aligned} &(\neg \text{Happens-Move-A-B-0} \vee \text{Holds-On-A-B-1}) \wedge \\ &(\neg \text{Happens-Move-A-C-0} \vee \text{Holds-On-A-C-1}) \wedge \\ &\quad \vdots \\ &(\neg \text{Happens-Move-A-B-1} \vee \text{Holds-On-A-B-2}) \wedge \\ &(\neg \text{Happens-Move-A-C-1} \vee \text{Holds-On-A-C-2}) \wedge \\ &\quad \vdots \end{aligned}$$

SAT Planning Preconditions

- In addition, the CNF formula will include every ground instance of:

$$\neg \text{Happens}(\text{Move}(x,y),t) \vee \text{Holds}(\text{Clear}(x),t)$$

$$\neg \text{Happens}(\text{Move}(x,y),t) \vee \text{Holds}(\text{Clear}(y),t)$$

- (These axioms say that if the action happens, the preconditions must hold)

$$(\neg \text{Happens-Move-A-B-0} \vee \text{Holds-Clear-A-0}) \wedge$$

$$(\neg \text{Happens-Move-A-B-0} \vee \text{Holds-Clear-B-0}) \wedge$$

⋮

$$(\neg \text{Happens-Move-A-B-1} \vee \text{Holds-Clear-A-1}) \wedge$$

⋮

SAT Planning Frame Axioms

- And, furthermore, the CNF formula will include every ground instance of:

$$\text{Holds}(\text{On}(x1,y),t+1) \vee \neg \text{Holds}(\text{On}(x1,y),t) \vee \neg \text{Happens}(\text{Move}(x2,z),t)$$

such that $x1 \neq x2$

- So we have, for example:

$$\begin{aligned} &(\text{Holds-On-B-C-1} \vee \neg \text{Holds-On-B-C-0} \vee \neg \text{Happens-Move-D-A-0}) \wedge \\ &(\text{Holds-On-B-C-1} \vee \neg \text{Holds-On-B-C-0} \vee \neg \text{Happens-Move-A-D-0}) \wedge \\ &\quad \vdots \\ &(\text{Holds-On-B-C-2} \vee \neg \text{Holds-On-B-C-1} \vee \neg \text{Happens-Move-D-A-1}) \wedge \\ &\quad \vdots \end{aligned}$$

- We will also need formulae ruling out concurrent actions

SAT Planning Initial Situations

- Our CNF formula is getting very large. But it still needs to include a description of the initial state:

(Holds-On-A-B-0 \wedge Holds-On-B-C-0 \wedge Holds-On-C-Table-0 \wedge ...

- And we need a description of the goal state. But for this, we have to make a guess at the maximum number of steps n the plan will require, and we force the goal state to hold at time n . Let's suppose $n=5$.

(Holds-On-A-D-5 \wedge Holds-On-B-C-5 \wedge Holds-On-C-Table-5 \wedge ...

- (A full SAT planning system will try the same problem for several values of n)

Solving the Satisfiability Problem

- Now we have a large CNF formula Ψ , which we can submit to a satisfiability solver. This will attempt to find an assignment of *True* or *False* to every proposition in Ψ such that Ψ evaluates to *True*
- If this is successful, we can read a plan off the resulting assignment. It is the set of all *Happens* propositions that are assigned *True*
- There are many off-the-shelf solvers that can do this
- The surprise is that this planning method is efficient, even though CNF satisfiability is NP-complete *and* we are dealing with very large CNF formulae

The WalkSAT Algorithm

- WalkSAT is one algorithm (among many) for solving satisfiability problems
- It is a stochastic algorithm. It works by making random flips

```
function WalkSAT(F)
    randomly assign values to all variables in F
    while F not satisfied
        randomly pick unsatisfied clause C in F
        with probability p
            randomly pick variable V in C
        else
            pick V in C to minimise unsatisfied
                clauses in F
        flip value of V
```


Residual Topics

- These are concepts you should be aware of, although there isn't room for them in the course
- Hierarchical planning
 - This is another domain-specific planning technique that involves the decomposition of a high-level plan into lower-level sub-plans. This process is repeated until low-level executable actions are reached
- Plan execution
 - Finding a plan is one thing. Executing it is another
 - Plan execution is particularly tricky if the world changes during execution in such a way as to frustrate the plan
 - Interleaving planning (especially anytime planning) and execution is often a good idea in fast-changing domains