7 Planning Algorithms

Planning: Exploiting representation structure in problem solving search

7.1 Some approaches

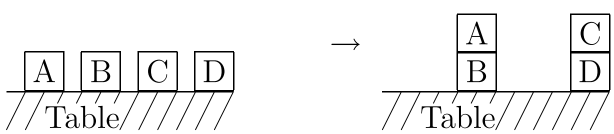
Heuristics (examine representation)

E.g., $\hat{h}(s)$ = Hamming distance from goal
$\hat{g}(s)$ = Hamming distance from initial state

Approximate divide and conquer

- Actions only affect small part of state
- Solve subgoals independently
- merge sub-plans

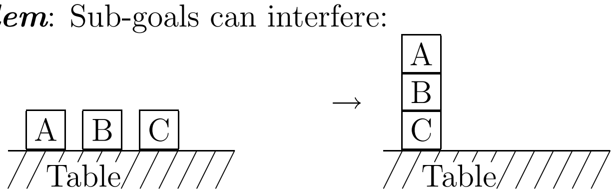
E.g.



Solve subgoals 'AonB' and 'ConD' independently, merge resulting actions.

Problem: Sub-goals can interfere:

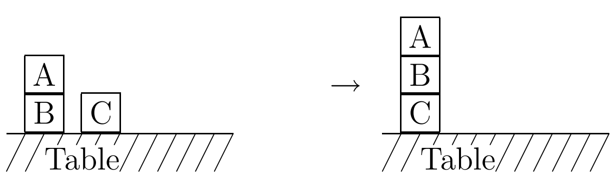
E.g.



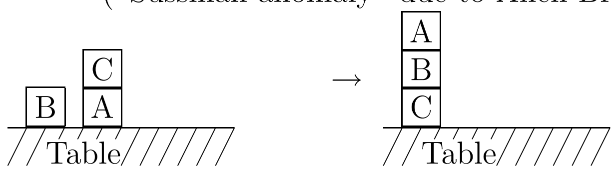
Getting A on B interferes with getting B on C.

Problem: We might even have to undo satisfied sub-goals:

E.g.

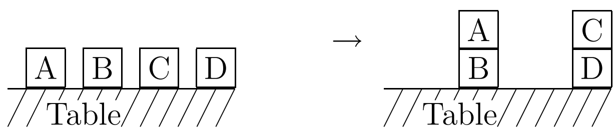


Problem: We may even have to avoid satisfying subgoals
("Sussman anomaly" due to Allen Brown):

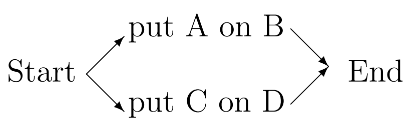


7.2 Partial order planning

For example:

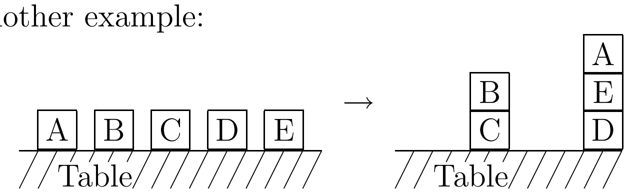


We can represent the plan as:



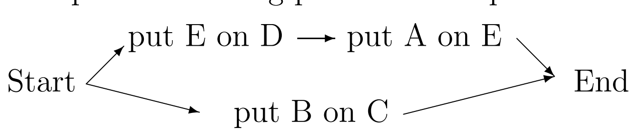
Any total ordering of the partial plan is a valid plan.

Another example:



A backtracking algorithm may waste time back-tracking the action 'putBonC'.

The partial ordering plan can be represented as



Representing a partial order plan

- set of actions: $\{a_1, \ldots, a_k\}$
- set of ordering constraints between actions: $\{a_j \prec a_i\}$
- set of reasons for actions (links, causal links): $\{a_i \xrightarrow{l} a_j\}$
  a_i establishes l for a_j:
  - l is effect of a_i
  - l is precondition for a_j

Partial order planning

- start with artificial start and goal actions a_0 and a_∞ with effect of a_0 being s_0, and precondition of a_∞ being γ
- build a plan by adding actions where effects are desired preconditions:

  $$a_i \xrightarrow{l} a_\infty \quad \text{where } l \in \gamma$$

  But add preconditions of a_i as new sub-goals
- If action a_i *threatens* a link $a_1 \xrightarrow{l} a_2$ (i.e., $\neg l$ is an effect of a_i) then a_i must be ordered before a_1 or after a_2.
- "Least commitment planning"
  Do not commit to ordering until forced
  (avoids backtracking on bad decisions)

7.3 POP algorithm

Algorithm 1 POP_main

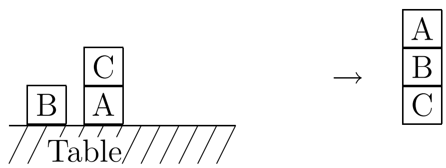
1: Create *start* and *end* actions a_0 and a_∞:
effect(a_0) $= s_0$ and precond(a_∞) $= \gamma$
2: Initialize plan (actions, ordering constraints, links):
plan $\leftarrow (\{a_0, a_\infty\}, \{a_0 \prec a_\infty\}, \{\})$
3: sub-goal list $\leftarrow \{\gamma\}$
4: **return** POP(subgoal list, plan)

Algorithm 2 POP (subgoal list, plan)

1: **if** subgoal list empty **then**
2: **return** plan
3: **end if**
4: pick next sub-goal l_{a_1} from sub-goal list
5: **for all** actions a_2 that establish l_{a_1} **do**
6: plan' $\leftarrow$ plan + $(\{a_2\}, \{a_0 \prec a_2, a_2 \prec a_1, a_2 \prec a_\infty\}, \{a_2 \xrightarrow{l_{a_1}} a_1\})$
7: subgoal list' $\leftarrow$ subgoal list $\cup$ preconditions(a_2)
8: **for all** consistent choices of order constraints in Step 9 **do**
9: for each action a threatening link $b \xrightarrow{l} c$ choose $a \prec b$ or $c \prec a$
10: plan" $\leftarrow$ plan' + additional order constraints
11: result $\leftarrow$ POP(subgoal list', plan")
12: **if** POP successful **then**
13: **return** result
14: **end if**
15: **end for**
16: **end for**
17: **return** fail

– Step 4 avoids backtracking (to some extent)
– For each sub-goal, have to keep track of the action requiring the sub-goal as precondition
– In Step 5 we can choose an action from plan, or introduce a new action
– If there are no threats in Steps 8–9, then loop 8–15 is iterated only once with an empty set of additional constraints.

7.4 Example: Sussman anomaly



Actions:

start: $\frac{}{\text{AonT } \neg\text{AonB } \neg\text{AonC BonT } \neg\text{BonA } \neg\text{BonC } \neg\text{ConT ConA } \neg\text{ConB}}$

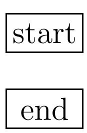
end: $\frac{\text{AonB BonC}}{}$

putConT: $\frac{\neg\text{AonC } \neg\text{BonC}}{\text{ConT } \neg\text{ConA } \neg\text{ConB}}$

putBonC: $\frac{\neg\text{AonB } \neg\text{ConB } \neg\text{AonC } \neg\text{BonC}}{\text{BonC } \neg\text{BonA } \neg\text{BonT}}$

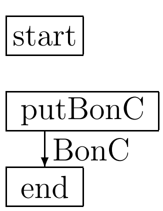
putAonB: $\frac{\neg\text{AonB } \neg\text{ConB } \neg\text{BonA } \neg\text{ConA}}{\text{AonB } \neg\text{AonC } \neg\text{AonT}}$

Algorithm trace:



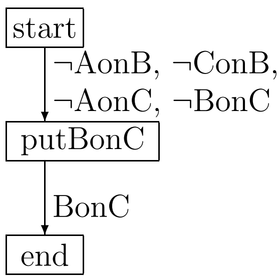
Sub-goal list: $\text{AonB}_{(\text{end})}$, $\text{BonC}_{(\text{end})}$

Pick sub-goal: $\text{BonC}_{(\text{end})}$



Sub-goal list: $\text{AonB}_{(\text{end})}$, $\neg\text{AonB}_{(\text{putBonC})}$, $\neg\text{ConB}_{(\text{putBonC})}$, $\neg\text{AonC}_{(\text{putBonC})}$, $\neg\text{BonC}_{(\text{putBonC})}$

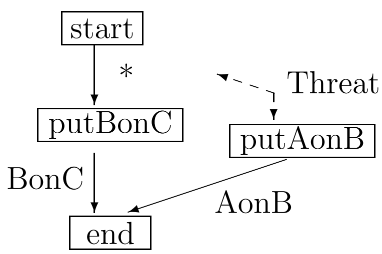
For all sub-goals that are preconditions of putBonC, we can choose action start, and obtain:



Sub-goal list: $\text{AonB}_{(\text{end})}$

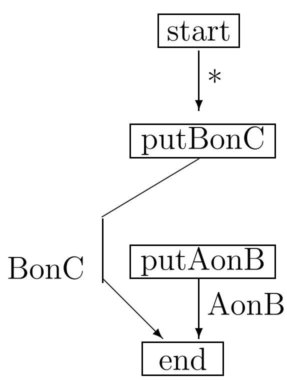
(Let * = ¬AonB, ¬ConB, ¬AonC, ¬BonC)

Sub-goal AonB is picked, and action putAonB is chosen:



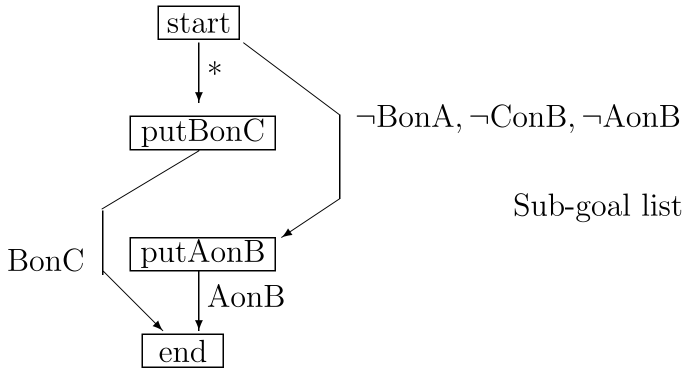
Sub-goal list: $\neg\text{BonA}_{(\text{putAonB})}$, $\neg\text{ConA}_{(\text{putAonB})}$, $\neg\text{AonB}_{(\text{putAonB})}$, $\neg\text{ConB}_{(\text{putAonB})}$

There is a threat: action putAonB threatens the link start $\xrightarrow{\neg\text{AonB}}$ putBonC. We have to put additional ordering constraints:



Sub-goal list: $\neg\text{BonA}_{(\text{putAonB})}$, $\neg\text{ConA}_{(\text{putAonB})}$, $\neg\text{AonB}_{(\text{putAonB})}$, $\neg\text{ConB}_{(\text{putAonB})}$

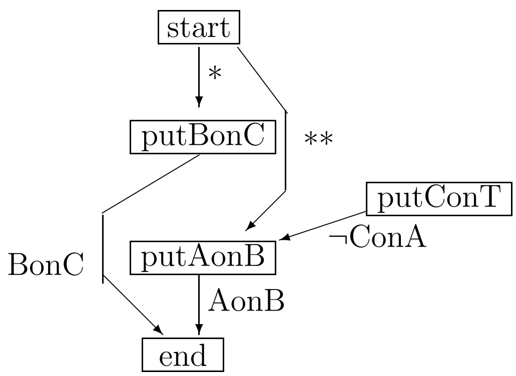
All sub-goals except ¬ConA are post-conditions of the start action:



Sub-goal list: $\neg\text{ConA}_{(\text{putAonB})}$

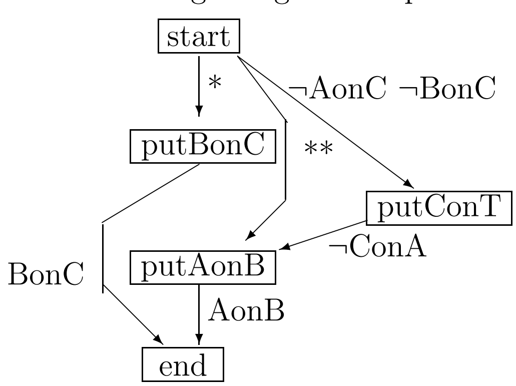
(Let ** = ¬BonA, ¬ConB, ¬AonB)

We pick the subgoal ¬ConA and choose action putConT that has this post-condition:



Sub-goal list: $\neg\text{AonC}_{(\text{putConT})}$, $\neg\text{BonC}_{(\text{putConT})}$

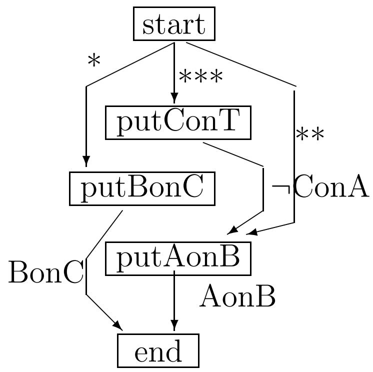
The remaining sub-goals are post-conditions of the action start:



Sub-goal list:

(Let *** = ¬AonC ¬BonC)

The action putBonC threatens the link start $\xrightarrow{\neg \text{BonC}}$ putConT, so we have to reorder:



Sub-goal list:

Done! No backtracking!

Plain goal regression (backward search)

Let us see how the same problem could be solved with backward search:

end	AonB, BonC
putBonC ↓ end	AonB, ¬AonB ¬ConB ¬AonC ¬BonC
	Stuck! (AonB and ¬AonB)
end	AonB, BonC
putAonB ↓ end	BonC, ¬BonA ¬ConA ¬ConB ¬AonB
start ↓ putAonB ↓ end	BonC ¬ConA

	Stuck!
putAonB ↓ end	<u>BonC</u>, ¬BonA ¬ConA ¬ConB ¬AonB
putBonC ↓ putAonB ↓ end	¬ConA ¬ConB ¬AonB
	Pick start, stuck, backtrack
putConT ↓ putBonC ↓ putAonB ↓ end	¬AonB
start ↓ putConT ↓ putBonC ↓ putAonB ↓ end	

Advantage of least commitment vs. plain backward search:

Smaller branching factor.

Backward search: branching factor = number of actions that can achieve some sub-goal

Least commitment: branching factor = number of actions that satisfy *next* sub-goal, does not backtrack through subgoal

7.5 Modern planning algorithms

- POP (1991)
- Graph Plan (1995)
- SAT plan (1996)
- Forward search with heuristics (2000)

Readings

Weld, *AI Magazine 15(4)*
`ftp://ftp.cs.washington.edu/pub/ai/pi.ps`

Also see: *Recent advances in AI planning* by Weld for a survey
`ftp://ftp.cs.washington.edu/pub/ai/pi2.ps`
`http://www.cs.washington.edu/homes/weld/pubs.html`