7 Planning Algorithms

Planning: Exploiting representation structure in problem solving search

7.1 Some approaches

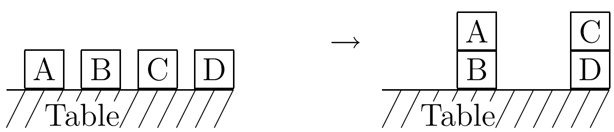
Heuristics (examine representation)

E.g., $\hat{h}(s)$ = Hamming distance from s to goal
$\hat{g}(\gamma)$ = Hamming distance from subgoal γ to initial state s_0

Approximate divide and conquer

If actions only affect small parts of state, we can solve subgoals independently and merge sub-plans.

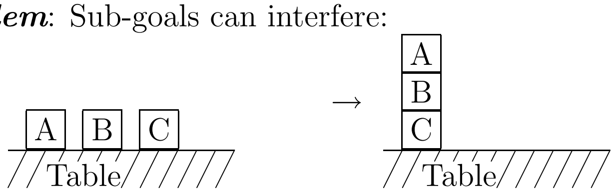
E.g.



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

Problem: Sub-goals can interfere:

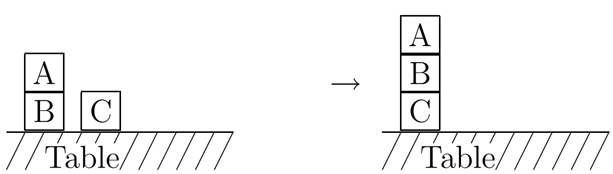
E.g.



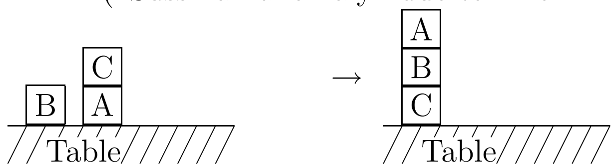
Getting A on B interferes with getting B on C.

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

E.g.

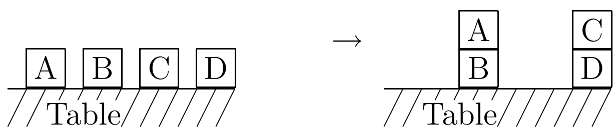


Problem: We may even have to avoid satisfying subgoals ("Sussman anomaly" due to Allen Brown):

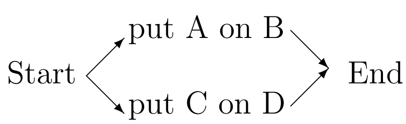


7.2 Partial order planning

For example:

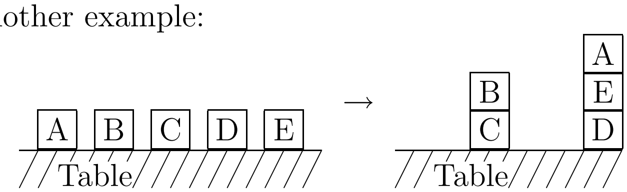


We can represent the plan as:



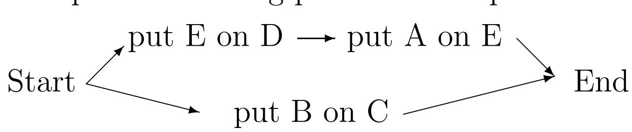
Any total ordering of the partial plan is a valid plan.

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: $a_i \xrightarrow{l} a_\infty$, 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 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_∞) = γ
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)

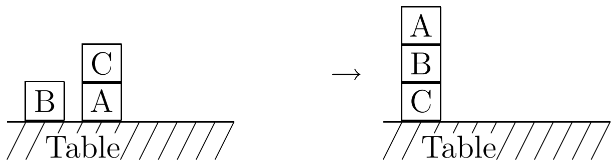
Algorithm 2 POP (subgoal list, plan)

1: **if** subgoal list is empty **then**
2:    **return** plan
3: **end if**
4: Pick 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 + $(\{a_2\}, \{a_0 \prec a_2, a_2 \prec a_1, a_2 \prec a_\infty\}, \{a_2 \stackrel{l_{a_1}}{\rightarrow} a_1\})$
7:    subgoal list' $\leftarrow$ subgoal list $\cup$ preconditions of a_2
8:    **for all** choices of additional order constraints in step 9 **do**
9:       for each action a threatening link $b \stackrel{l}{\rightarrow} c$ 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)
- with 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 8–13 is iterated only once with an empty set of additional constraints.

7.4 Example: Sussman anomaly



Actions:

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

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

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

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

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

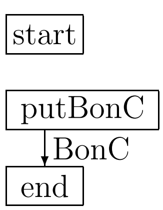
Algorithm:

start

Sub-goal list: AonB (end), BonC (end)

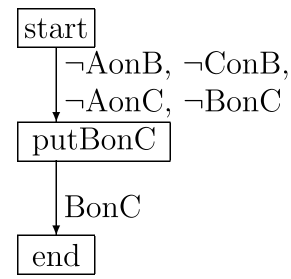
end

Pick sub-goal: BonC (end)



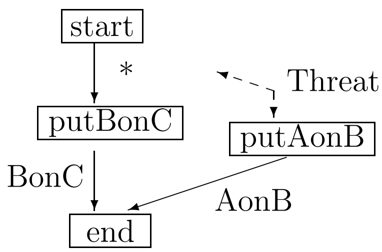
Sub-goal list: AonB (end), ¬AonB (putBonC), ¬ConB (putBonC), ¬AonC (putBonC), ¬BonC (putBonC)

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



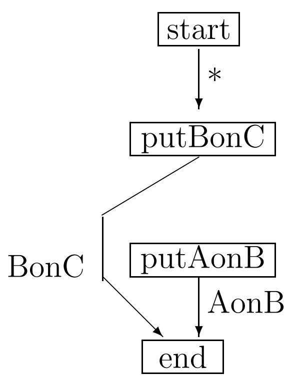
Sub-goal list: AonB (end)

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



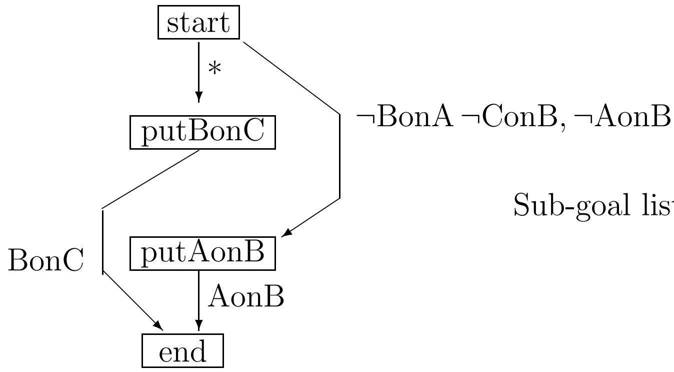
Sub-goal list: ¬BonA (putAonB), ¬ConA (putAonB), ¬AonB (putAonB), ¬ConB (putAonB),

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



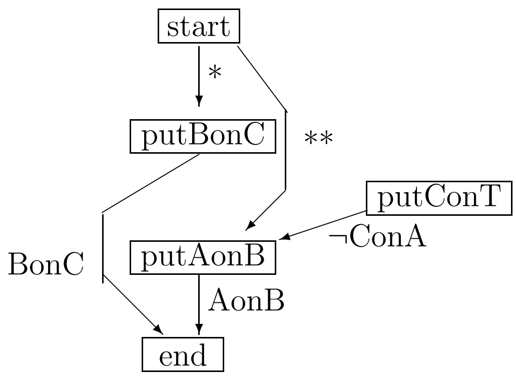
Sub-goal list: ¬BonA (putAonB), ¬ConA (putAonB), ¬AonB (putAonB), ¬ConB (putAonB),

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



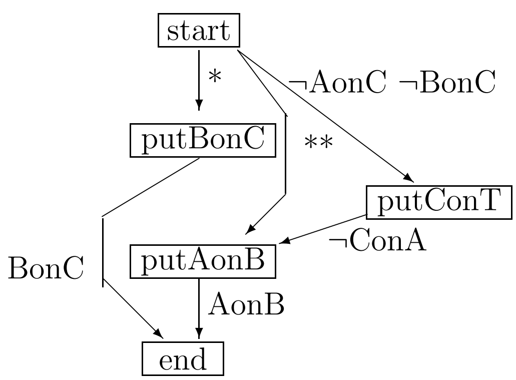
Sub-goal list: ¬ConA (putAonB)

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



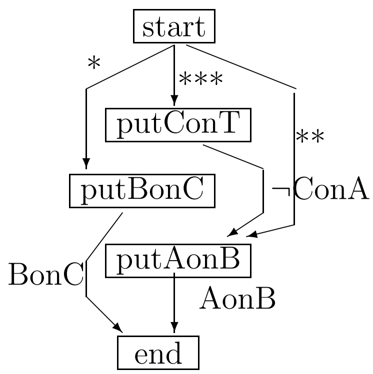
Sub-goal list: ¬AonC (putConT), ¬BonC (putConT)

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



Sub-goal list:

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



Done! No backtracking!

Sub-goal list:

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	BonC, $\neg$BonA $\neg$ConA $\neg$ConB $\neg$AonB
putBonC ↓ putAonB ↓ end	$\neg$ConA $\neg$ConB $\neg$AonB
	Pick start, stuck, backtrack
putConT ↓ putBonC ↓ putAonB ↓ end	$\neg$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 of more recent developments.
`ftp://ftp.cs.washington.edu/pub/ai/pi2.ps`
`http://www.cs.washington.edu/homes/weld/pubs.html`