Course "Automated Planning: Theory and Practice"
Chapter 14: Relaxed Planning Graph

Teacher: Marco Roveri - marco.roveri@unitn.it

M.S. Course: Artificial Intelligence Systems (LM)

A.A.: 2023-2024

Where: DISI, University of Trento

URL: https://bit.ly/3zOkGk8

Last updated: Wednesday 8th November, 2023

Terms of Use and Copyright

The material (text, figures) in these slides is authored by Jonas Kvarnström and Marco Roveri.

Basic Idea

Apply delete relaxation

$\Downarrow$

Create a graph efficiently representing many ways of achieving the goal in the relaxed problem.

$\Downarrow$

Extract one possible solution π from the graph (not necessarily optimal!)

$\Downarrow$

$h_{FF}(n) = |\pi|$ or $h(n) = \text{cost}(\pi) \geq h^{+}(n)$ [a]

[a] FF - Fast Forward Since approach pioneered in the FF planner FF [2] as discussed in Hoffmann and Nebel [6].

Running example (by Dan Weld)

- Prepare and serve a surprise dinner,
  take out the garbage,
  and make sure the present is wrapped before waking your sweetheart!
  - $s_0 = \{\texttt{clean}, \texttt{garbage}, \texttt{asleep}\}$
  - $g = \{\texttt{clean}, \neg\texttt{garbage}, \texttt{served}, \texttt{wrapped}\}$
  -

Action	**Preconditions**	**Effects**
(cook)	clean	dinner
(serve)	dinner	served
(wrap)	asleep	wrapped
(carry)	garbage	¬ garbage, ¬ clean
(roll)	garbage	¬ garbage, ¬ asleep
(clean)	¬ clean	clean

Running example: apply delete relaxation

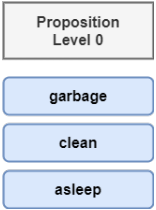
- Prepare and serve a surprise dinner,
  take out the garbage,
  and make sure the present is wrapped before waking your sweetheart!
  - $s_0 = \{\texttt{clean}, \texttt{garbage}, \texttt{asleep}\}$
  - $g = \{\texttt{clean}, \texttt{served}, \texttt{wrapped}\}$
  -

Action	Preconditions	Effects
(cook)	clean	dinner
(serve)	dinner	served
(wrap)	asleep	wrapped
(carry)	garbage	–
(roll)	garbage	–
(clean)	–	clean

Pointless actions:
No effects!

Relaxed Plan Graph: Propositions

- We want now to find a relaxed plan
  - What is true initially?
    $\Longrightarrow$ first proposition level in a relaxed planning graph

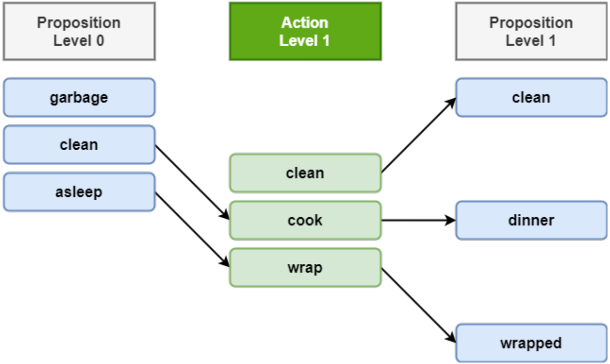


Planning Graph introduced in GraphPlan by Blum and Furst [1]

Heuristics based on Relaxed Planning Graph pioneered by FF (FastForward) FF [2] by Hoffmann and Nebel [6]

Relaxed Plan Graph: Actions and Effects

- Which actions could be executed?
- Which effects would we get?



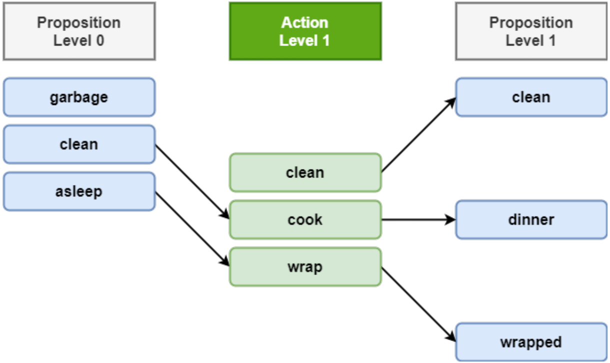
Action	Prec.	Effects
(cook)	clean	dinner
(serve)	dinner	served
(wrap)	asleep	wrapped
(clean)	–	clean

Build a graph with actions linking to preconds and effects

Assumes conjunctive preconds, effects!

Relaxed Plan Graph: Interpretation

- Which propositions can we make true in one step?
- Which actions would we need?



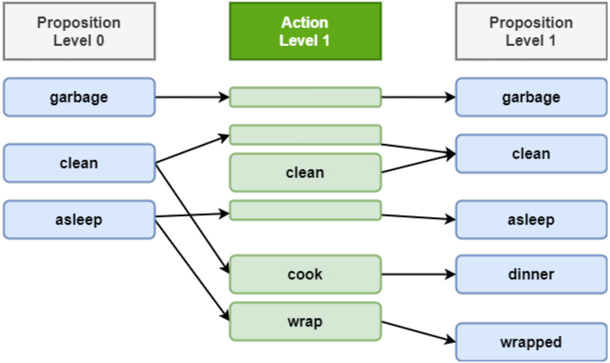
Action	**Prec.**	**Effects**
`(cook)`	`clean`	`dinner`
`(serve)`	`dinner`	`served`
`(wrap)`	`asleep`	`wrapped`
`(clean)`	–	`clean`

But wait!!

Proposition Level 1 is missing `garbage`, which could *remain* true from Proposition Level 0...

Relaxed Plan Graph: Maintenance Actions

- Solution: "No-Op" or "maintenance" actions!
  - One for each proposition (fact) that exists
  - No need to treat *persistence* separately

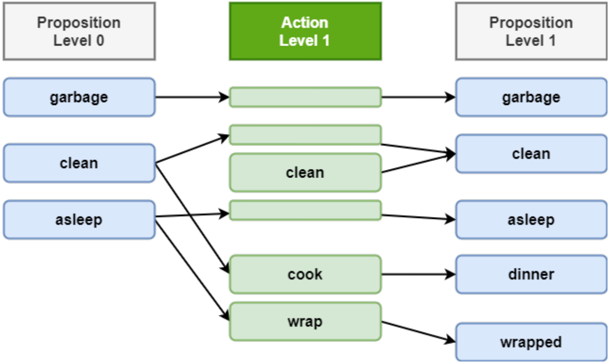


Action	Prec.	Effects
(cook)	clean	dinner
(serve)	dinner	served
(wrap)	asleep	wrapped
(clean)	–	clean

```
(noop-clean)
  preconditions:clean
  effects:      clean
(noop-garbage)
  preconditions:garbage
  effects:      garbage
(noop-asleep)
  preconditions:asleep
  effects:      asleep
```

Relaxed Plan Graph: Interpretation - actions

- What does this mean for the actions?



Action	Prec.	Effects
(cook)	clean	dinner
(serve)	dinner	served
(wrap)	asleep	wrapped
(clean)	–	clean
(noop-...)		

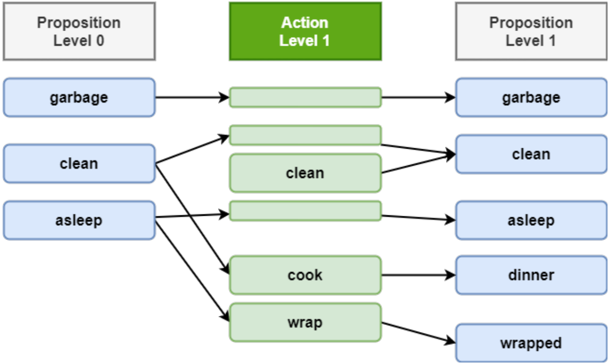
First action could be `clean`, `cook` or `wrap`

First actions could be any combination of `clean`, `cook` or `wrap`

None can invalidate the others' preconditions: No negative effects!

Relaxed Plan Graph: Interpretation - facts

- What does this mean for the facts?



Action	Prec.	Effects
`(cook)`	`clean`	`dinner`
`(serve)`	`dinner`	`served`
`(wrap)`	`asleep`	`wrapped`
`(clean)`	`-`	`clean`
`(noop-...)`		

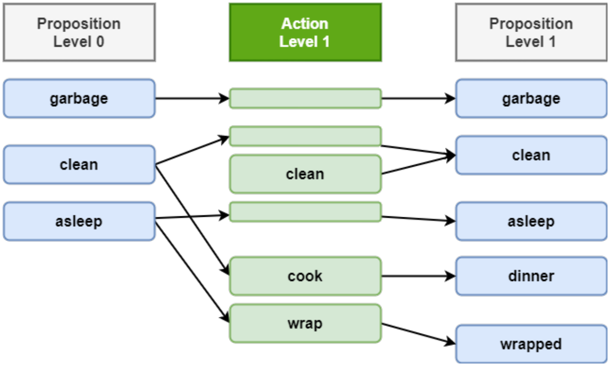
We can choose actions that achieve any subset of {`garbage`, `clean`, `asleep`, `dinner`, `wrapped`} and we don't have to care about their order!

Given delete relaxation!

In reality, negative effects interfere... but we aim for a heuristic!

RELAXED PLAN GRAPH: REACHED GOAL?

- No, can't achieve `served` yet...



Action	**Prec.**	**Effects**
(cook)	clean	dinner
(serve)	dinner	served
(wrap)	asleep	wrapped
(clean)	–	clean
(noop-...)		

We need `dinner` before `served`

Level 1 is only for actions whose pre-conds are true at the start!

Chains of dependencies

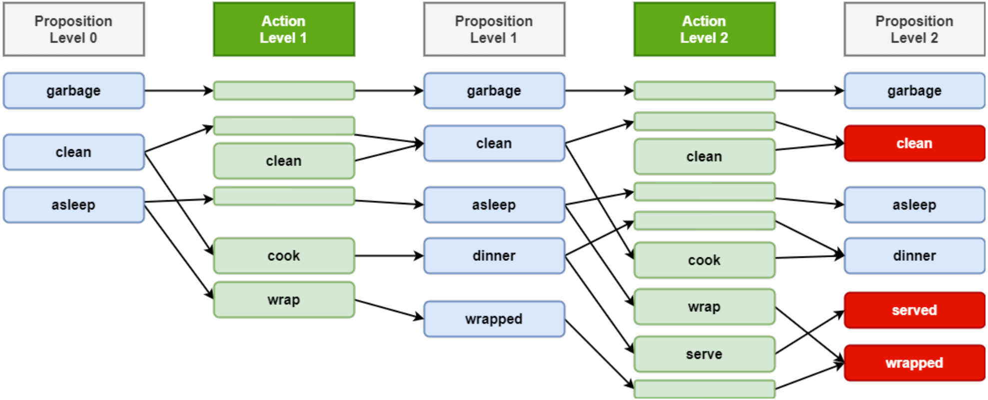
$\Longrightarrow$

Many levels in the graph

Relaxed Plan Graph: Level 2

Action	Prec.	Effects
(cook)	clean	dinner
(serve)	dinner	served
(wrap)	asleep	wrapped
(clean)	–	clean
(noop-...)		

- Achieves all goals
- Can select actions from the graph



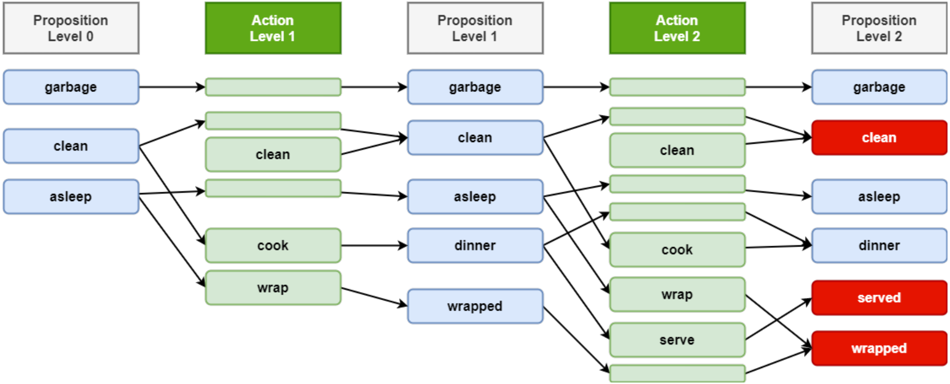
g = {clean, served, wrapped}

Relaxed Plan Graph: Solution Extraction

- For each goal fact, choose one action achieving it
  - `clean` $\Longrightarrow$ `(noop-clean)` or `(clean)`
  - `served` $\Longrightarrow$ `(serve)`
  - `wrapped` $\Longrightarrow$ `(noop-wrapped)` or `(wrap)`

2*1*2=4 alternatives!

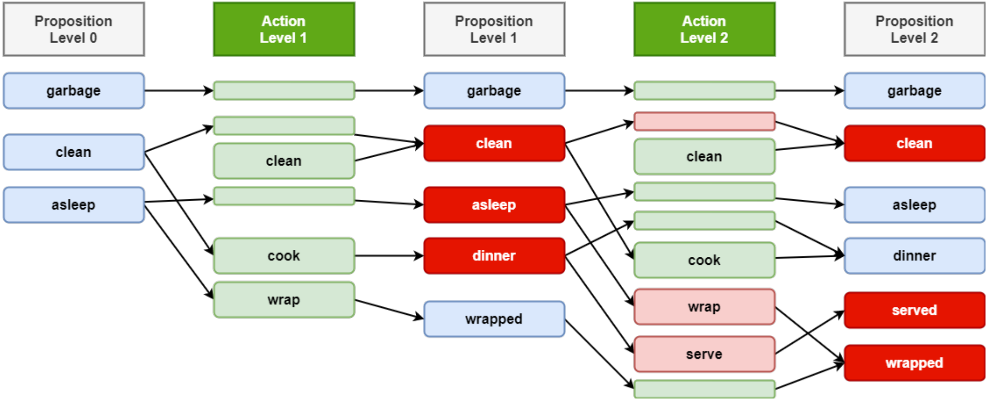
All work, but some may result in shorter plans!



Relaxed Plan Graph: Solution Extraction (cont.)

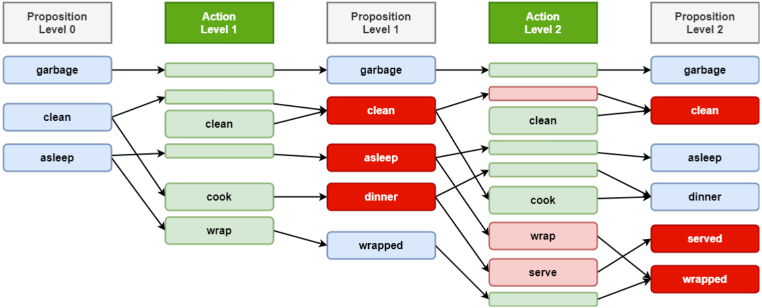
- For all selected actions in Level 2:
  - Must first achieve their preconditions!
  - The set of preconditions represents new goal to achieve by selecting actions at Level 1!

We select: `(noop-clean)`, `(wrap)`, and `(serve)`



Relaxed Plan Graph: Solution Extraction (cont.)

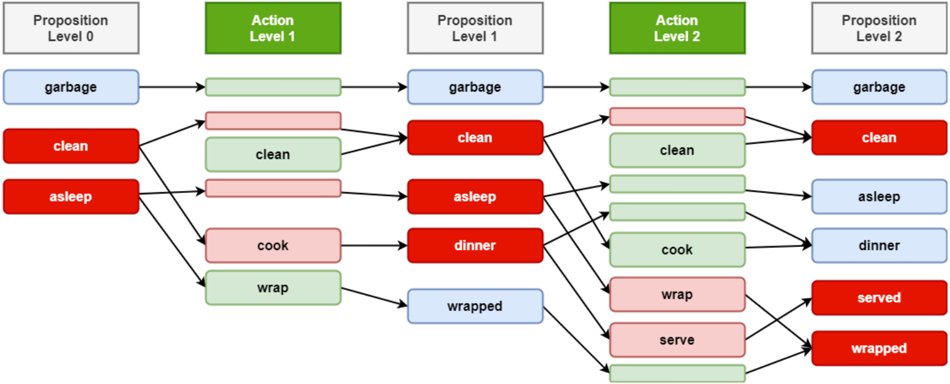
- Unlike backward search in *goal space*:
  - Simpler concept of relevance: No negative effects that interfere
  - At each level, select sets of actions, together achieving all goal facts
    - No need to consider "what the single selected action didn't achieve"
    - Simpler backward chaining: Instead of γ^{-1}, just conjoin preconds of selected actions
  - Already built a graph from the initial state
    - And no possibility of negative effect interference $\Longrightarrow$ we *can* reach the initial state



Relaxed Plan Graph: Solution Extraction (cont.)

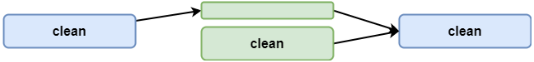
- Final relaxed plan:
  - First cook
  - Then wrap and serve, in some order
  - $h_{FF}(n) = 3$, assuming the algorithm chose this order!

Relaxed plan: Not a solution to the original problem!

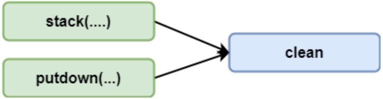


Relaxed Plan Graph: Solution Extraction (cont.)

- Does the choice of actions matter?
  - Choosing a noop action may mean fewer actual actions



  - Different actions chosen at one level:
    - May lead to different actions at previous levels
    - Which then leads to different preconds to satisfy...



  - And so on...
- Not equivalent to $h^+(n)$: would require an optimal relaxed plan
  - Would have to test different action selections
  - May require additional levels (with fewer selected actions per level)

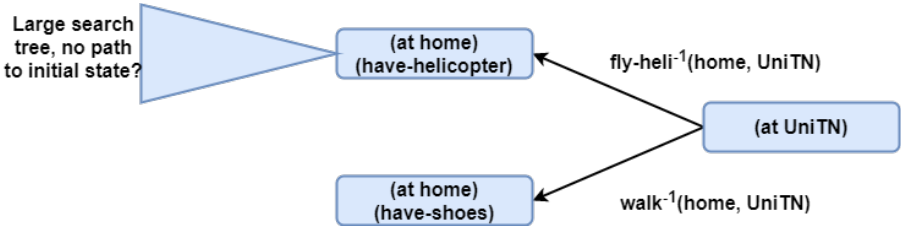
Actual solution extraction algorithm in FF uses backward search in the RPG + *heuristics* for this search!

Relaxed Plan Graph: Properties

- The relaxed planning graph considers positive interactions
    - For example, when one action achieves multiple goals
    - Ignores negative interactions
    - Can extract a Graphplan-optimal relaxed plan
      (minimal number of levels / "parallel" steps)
      in polynomial time

Backward Search - Recap

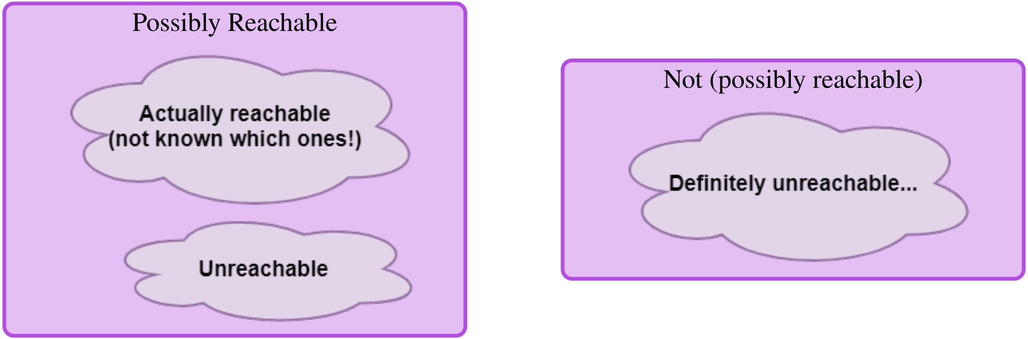
- We know if the effects of an action can contribute to the goal
- Need guidance to determine which backward paths will lead to (good) solutions



One approach: Use heuristics. But other methods exist...

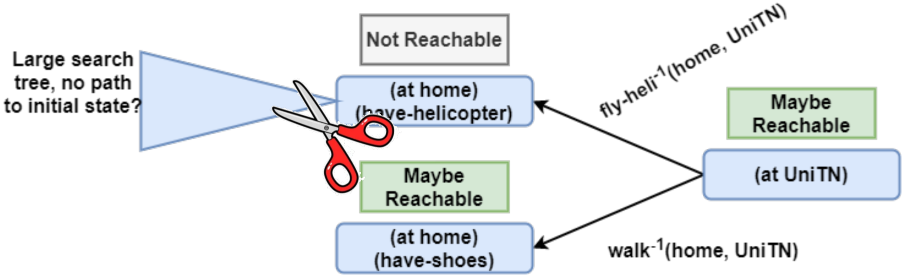
Reachable States

- Suppose that we could quickly determine
  - **possibly-reachable**(s_0,s) - may state s be reachable from s_0?



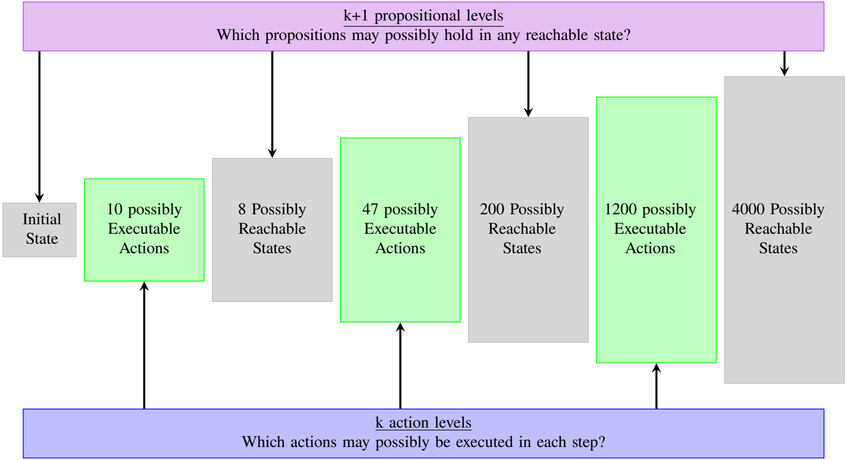
Reachable States (cont.)

- Then we could prune many "fruitless branches":

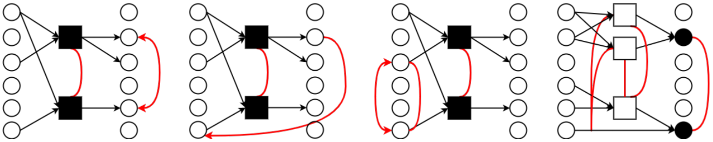


Planning Graph

- A (non-relaxed) Planning-Graph:
  - Useful to *generate states* - also useful in *backward search*!



Negative Effects $\Longrightarrow$ Mutual Exclusion

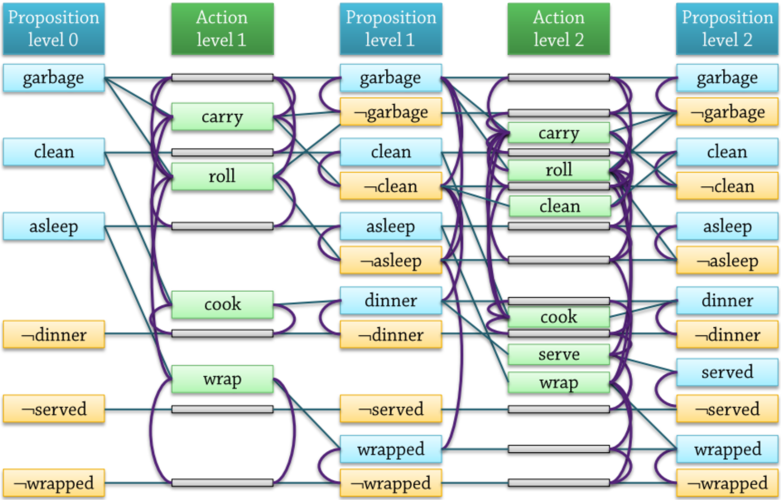


- Two actions at the same action level are mutex (can't be selected together) if
  - Inconsistent effects: an effect of one negates an effect of the other
  - Interference: one deletes a precondition of the other
  - Competing needs: they have mutually exclusive preconditions (not shown)
- Otherwise:
  - Both might appear at the same time step in a solution plan

Recursive propagation of mutexes

- Two literals at the same proposition level are mutex if
  - Inconsistent support A: one is the negation of the other,
  - Inconsistent support B: all ways of achieving them are pairwise mutex

Example



All goal literals are present in propositional level 2, and none of them are (known to be) mutex!

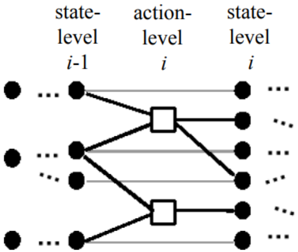
GraphPlan: Solution Extraction

The set of goals we are trying to achieve

The level of the state s_j

A real action or a maintenance action

function SOLUTION-EXTRACTION(g , j)
  if ($j = 0$) **then return** the solution
  for each $l \in g$ **do**
    nondeterministically chose an action
      to use in state s_{j-1} to achieve l
  if any pair of chosen actions are mutex **then** backtrack
  g' = {the preconditions of the chosen actions}
  SOLUTION-EXTRACTION(g', $j-1$)
end function



The GraphPlan Algorithm

function GrahPlan(solution-extraction)
  G = { }
  for each k = 0, 1, 2, ... **do**
    G = GrahExpansion(G,k) → *Create/Expand the planning graph G to contain k levels*
    if CheckSuff(G) **then** → *Check whether the planning graph satisfies necessary (but not sufficient) conditions for plan existence*
      π = BwdSearch(G) → *Backward search, modified to consider only the actions in the planning graph*
      if $\pi != \emptyset$ **then return** π
end function

Comparison with Plan-Space Planning

- Advantage:
  - The backward-search part of Graphplan – which is the hard part – will only look at the actions in the planning graph
  - Smaller search space than PSP; thus faster
- Disadvantage:
  - To generate the planning graph, Graphplan creates a huge number of ground atoms
  - Many of them may be irrelevant
- Can alleviate (but not eliminate) this problem by assigning data types to the variables and constants
  - Only instantiate variables to terms of the same data type
- For classical planning, the advantage outweighs the disadvantage
  - GraphPlan solves classical planning problems much faster than PSP

References I

[1] Avrim Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artif. Intell.*, 90(1-2):281–300, 1997. doi: 10.1016/S0004-3702(96)00047-1. URL `https://doi.org/10.1016/S0004-3702(96)00047-1`. 6

[2] FF. The Fast Forward Planner. `https://fai.cs.uni-saarland.de/hoffmann/ff.html`, 2001. 3, 6

[3] Hector Geffner and Blai Bonet. *A Concise Introduction to Models and Methods for Automated Planning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2013. ISBN 9781608459698. doi: 10.2200/S00513ED1V01Y201306AIM022. URL `https://doi.org/10.2200/S00513ED1V01Y201306AIM022`.

[4] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated planning - theory and practice*. Elsevier, 2004. ISBN 978-1-55860-856-6.

[5] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated Planning and Acting*. Cambridge University Press, 2016. ISBN 978-1-107-03727-4. URL `http://www.cambridge.org/de/academic/subjects/computer-science/artificial-intelligence-and-natural-language-processing/automated-planning-and-acting?format=HB`.

[6] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *J. Artif. Intell. Res.*, 14:253–302, 2001. doi: 10.1613/jair.855. URL `https://doi.org/10.1613/jair.855`. 3, 6