Course "Automated Planning: Theory and Practice"
Chapter 03: Planning as Search

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: Sunday 24th September, 2023

Terms of Use and Copyright

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

How to generate a plan?

- One way of defining planning:

  Using knowledge about the world, including possible actions and their results, to decide what to do and when in order to achieve an objective, before you actually start doing it

$\Rightarrow$ How?

Is planning a straight-forward process of adding actions in the right order?

while (exists unvisited position) **do**
  pos ← nearest unvisited position
  plan += flyto(pos)
  plan += aim()
  plan += take-picture()
end while

$\Rightarrow$

Usually, conditions are too complex; better to test-alternatives - search

Generate some starting point
while (not complete) **do**
  try some alternatives
  create modified alternatives
  throw away some alternatives
end while

Planning as Search

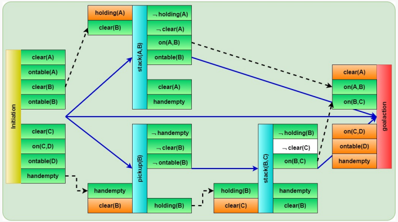
- To generate plans using search, we need:

1) A node structure defining what information is in a node

State space search: A node is a state

$s = \{fact_1, fact_2, fact_3, \ldots\}$

Partial Order Causal Link (POCL): A node is a complex plan structure

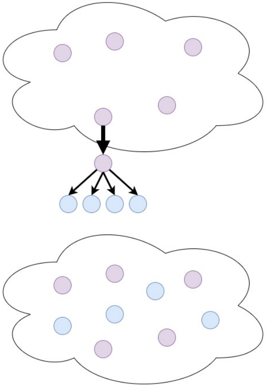


2) A way of creating an initial node from a problem instance

- Search spaces are generally too large to be represented completely
- Need to start somewhere and then expand the space incrementally

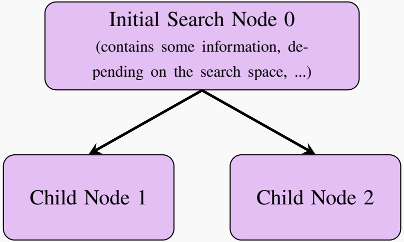
Planning as Search (cont.)

- General way of formalizing search algorithms:
  - There are some "open" nodes (at first the initial node), that we
    - Know how to reach
    - Haven't explored yet
  - Pick/Remove one of them
    - Using some *strategy* to peek "good" nodes!
  - Find neighbor nodes that can be created in a "single" step
  - Put created nodes in the set of "open" nodes
  - Repeat until a node corresponds to a solution



Planning as Search (cont.)

Search Space
Classical Planning
Finite number of search nodes



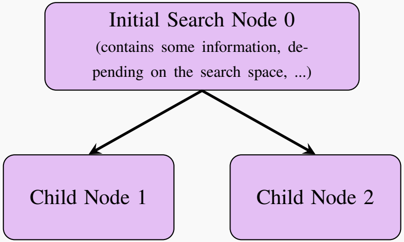
3) A successor function/branching rule

Returning all successors (neighbors) of any search node

- Edges could correspond to actions (state space search) or to something different (POCL)!
- Expand a node corresponds to generate all its successors

Planning as Search (cont.)

Search Space
Classical Planning
Finite number of search nodes



- To know when we succeeded

4) A solution criterion

To detect when a node corresponds to a solution

5) A plan extraction

To tell/extract which plan a solution node corresponds to

- To decide how to search

6) A search strategy

To choose which node to expand next

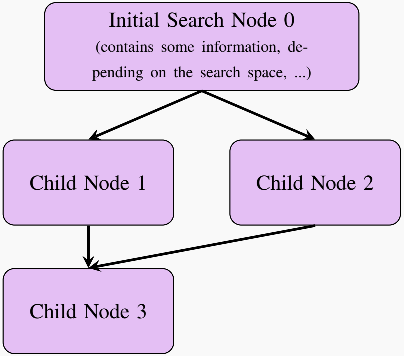
Planning as Search (cont.)

- General Search-Based Planning Algorithm

```
function SEARCH(problem)
    initial-node ← MAKE-INITIAL-NODE(problem)                    → [2]
    open ← {initial-node}
    while (open ≠ ∅) do
        node ← SEARCH-STRATEGY-REMOVE-FROM(open)                 → [6]
        if IS-SOLUTION(node) then                                → [4]
            return EXTRACT-PLAN-FROM(node)                       → [5]
        end if
        for each newnode ∈ SUCCESSORS(node) do                   → [3]
            open ← open ∪ {newnode}
        end for
    end while
    return Failure                                               → Expanded the entire search space without finding a solution
end function
```

Searching Graph

Search Space
Classical Planning
Finite number of search nodes



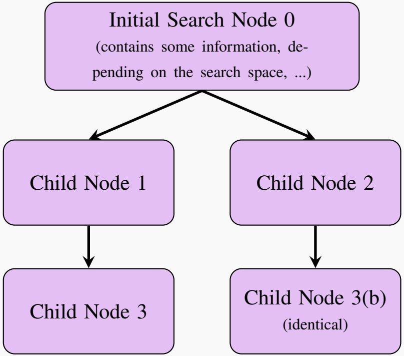
In a graph two nodes can share a successor!

Option 1

- Keep track of all visited nodes
- Detect when the same successor is generated again
  - Requires a lot of memory
  - Only investigate a given node at once
  - Second time do not expand it!

Searching Graph (cont.)

Search Space
Classical Planning
Finite number of search nodes

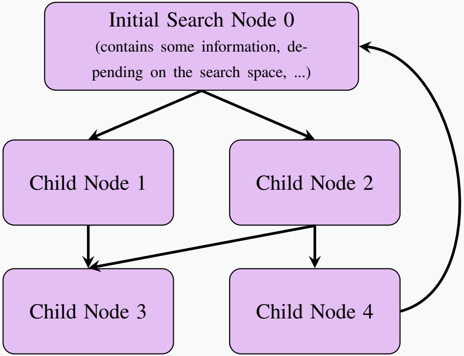


Option 2

- Do not keep track of visited nodes!
  - Saves memory
  - Investigate some subtrees multiple times
  - Search space is visited as a *tree*

Searching Graph (cont.)

Search Space Classical Planning Finite number of search nodes



Loops in the search

- An ancestor may also be a successor
  - $\Longrightarrow$ loops in the search graph
- Depending on the search strategy it may or may not be necessary to detect and handle this!

Searching Graph (cont.)

- To avoid searching sub-graphs twice (shared successors + loops):

```
function SEARCH(problem)
    initial-node ← MAKE-INITIAL-NODE(problem)              → [2]
    open ← {initial-node}
    added ← {initial-node}
    while (open ≠ ∅) do
        node ← SEARCH-STRATEGY-REMOVE-FROM(open)           → [6] // Nodes removed from open, but not from added!
        if IS-SOLUTION(node) then                          → [4]
            return EXTRACT-PLAN-FROM(node)                 → [5]
        end if
        for each newnode ∈ SUCCESSORS(node) do             → [3]
            if newnode ∉ added then
                open ← open ∪ {newnode}
                added ← added ∪ {newnode}
            end if
        end for
    end while
    return Failure                                         → Expanded the entire search space without finding a solution
end function
```

Aspects of Search

Defined by the search space

1) Node structure
2) Generating initial search node
3) Branching rule, creating successors
4) Determining if a node is a solution
5) Extracting a plan from a node

Forward State Space, Backward Goal Space, Partial Order Causal Link, ...

Defined by the search strategy

Uninformed	Informed
• Depth first (DFS)	• Greedy Best First
• Breadth first (BFS)	• A*
• Dijkstra	• Weighted A*
• Uniform cost	• Iter. Deep. A*
• Depth limited DFS	• Beam Search
• Iter. Deep. DFS	• Hill Climbing (HC)
• ...	• Enforce HC
	• Simul. Annealing
	• ...

$\Downarrow$

Heuristics!

Independence!

Planning as Search: Summary

Search Spaces

Forward State Space
Backward Goal Space
Partial Order Causal Link
Hierarchical Task Networks

Search Strategies

Hill Climbing
Enforced Hill Climbing
A*
(Repeated) Weighted A*

Heuristics

Goal count
Landmarks
Pattern Databases
Relaxation, Delete Relaxation
Relaxed Planning Graphs

Tweaking Search Space

Predicates vs State variables
Lifted Search Space

Tweaking Search Strategies

Helpful Actions / Preferred Operators
Dual Queues, Boosted Dual Queues
Lazy Search

Meta Search Strategies

Portfolio Planning

Preparation for Hands-on Session next Week

- We will create and solve planning problems
- On your laptops!
  - Experiment with command line planners
    - Planutils (several pre-compiled planners on Linux) `https://pypi.org/project/planutils/`
    - Fast Downward[1] `https://www.fast-downward.org/`
  - Experiment with online planners
    - `http://editor.planning.domains`
    - `https://web-planner.herokuapp.com`

[1] Instructions to compile it for Linux/Window/MacOS X `https://www.fast-downward.org/ObtainingAndRunningFastDownward`

Some tips for Planutils

- To install `pip` or `pip3` (in Debian like Linux distributions e.g. Ubuntu):
    - Run `apt-get install python-pip` or `apt-get install python3-pip`
- Run `planutils list` for a list of available planners
- To install a planner run `<name>` using names from previous command
- You need `python` version >= 3.6.* in order to successfully install and then run `planutils`.
- Planutils requires the `singularity` package to be installed on the machine (a kind of lightweight Docker container).
- If the `pip3 install planutils` does not install also `singularity`, you can install it following the instructions you can find at: `https://sylabs.io/guides/3.8/user-guide/quick_start.html#quick-installation-steps`
- Singularity runs on Linux natively and can also be run on Windows and Mac through virtual machines (VMs).

Some tips for Planutils (cont.)

- At this URL: `https://sylabs.io/guides/3.8/admin-guide/installation.html` you can find instructions to install singularity on a Linux machine.
- At this URL are reported instructions to install singularity on Windows and Mac Os X
  - Notice that, in Windows and MacOX you need to install then planutils from the virtual machine that is installed with the instructions at the link.
- If for some reason you have `PYTHONHOME` or `PYTHONPATH` set, this may interfere with the singularity images (that are not completely sandboxed). To avoid this, then you can edit the files in `~/.planutils/packages/<planner>/run` changing planner with one of the installed planners (e.g. `downward`) adding option `-e` to `run` i.e. `singularity run -e $(dirname $0)/downward.sif`
- Under Linux singularity do not mount all the host files by default.
  - Edit file `/usr/local/etc/singularity/singularity.conf` (or `/etc/singularity/singularity.conf`) changing `mount hostfs = no` to `mount hostfs = yes`.

References I

[1] 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`.

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

[3] 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`.