# Artificial Intelligence (CS303)

## Lecture 7: Planning

# Hints for this lecture

- **Formulating and solving search problems to decide a set of actions for an agent to achieve its goal.**

# Outline of this lecture

- **Planning as a SAT**

- **Planning Domain Definition Language (PDDL)**

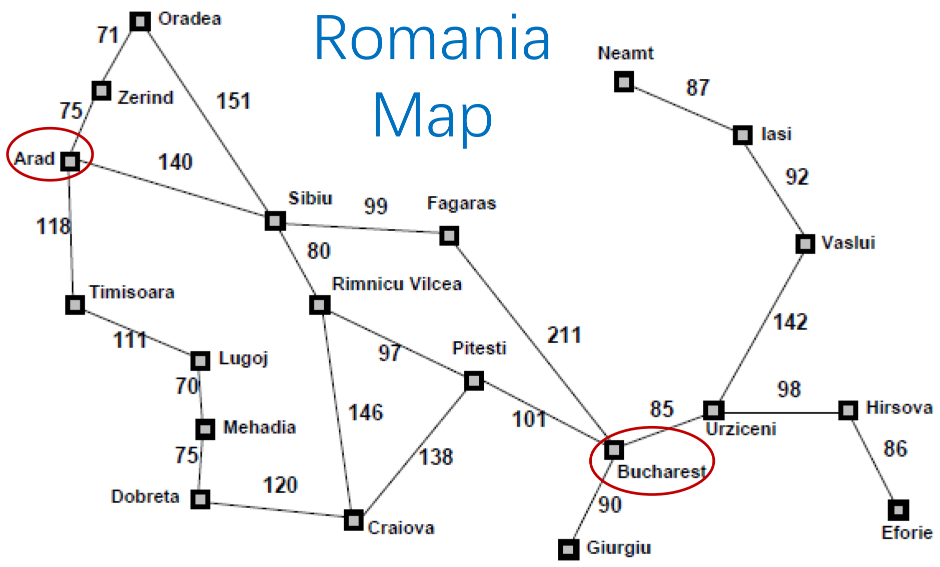
- **General Heuristics for Planning**

- **GRAPHPLAN Algorithm**

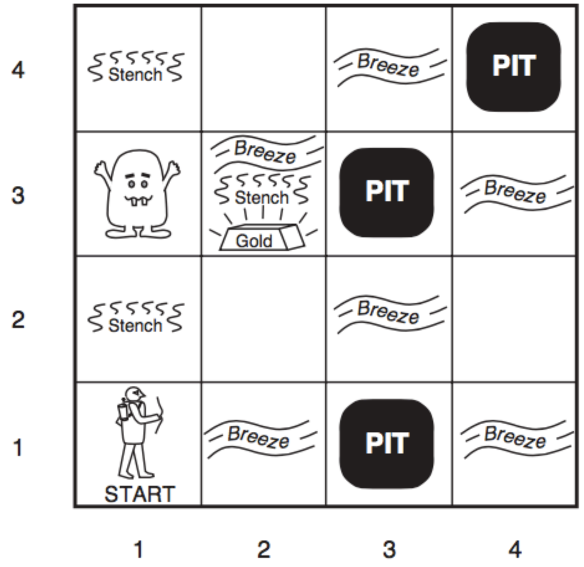
# I. Planning as a SAT

# What is Planning?

- **Decide (plan) a set of actions (solution) for the agent to achieve a goal.**




Can we have a more general approach for planning?



# Planning as a SAT

- Planning does not necessarily requires "optimality", it concerns more about whether a goal can be achieved and how.

- Planning can be formulated as Boolean SAT problem
  - Variables: Actions (to do or not)
  - Domain: to do or not
  - Sentence: the goal

Different from the inference problem?

- Might not be efficient due to the propositional representation (e.g., action space is huge). Note that the problem is PSPACE, i.e., harder than NPC problems.

# II. Planning Domain Definition Language (PDDL)

# PDDL

**Action Schema**

$$
\begin{aligned}
&Action(Fly(P_1, SFO, JFK),\\
&\quad \text{PRECOND:} At(P_1, SFO) \land Plane(P_1) \land Airport(SFO) \land Airport(JFK)\\
&\quad \text{EFFECT:} \neg At(P_1, SFO) \land At(P_1, JFK))
\end{aligned}
$$

- **Restricted Language (details removed), lead to more efficient algorithms.**
- **Defines relationship between actions and states (thus an abstract graph).**

# III. General Heuristics for Planning

# Forward State-Space Search

- **Just like any other tree-search methods we discussed before.**
- **May be quite inefficient due to exploration of irrelevant actions.**
- **Example:**
  - **10 airports**
  - **Each airport has 5 planes and 20 cargos**
  - **Goal: move all cargos at airport A to airport B**
- **Numerous irrelevant actions**

# Heuristics

- **How to reduce (or remove) the exploration of irrelevant actions?**
- **A*: need an admissible heuristic.**
- **General idea: Add edges or Remove nodes on an abstract graph.**

# Add edges

- **Ignore preconditions: Leads to a Set-Cover problem.**

$$
\begin{array}{llll}
\text{minimize} & \displaystyle\sum_{S\in\mathcal{S}} x_S & & \text{(minimize the number of sets)} \\
\text{subject to} & \displaystyle\sum_{S:e\in\mathcal{S}} x_S \geqslant 1 & \text{for all } e \in \mathcal{U} & \text{(cover every element of the universe)} \\
& x_S \in \{0,1\} & \text{for all } S \in \mathcal{S}. & \text{(every set is either in the set cover or not)}
\end{array}
$$

- **Ignore delete lists: hill climbing is applicable since the goal state can be monotonically satisfied.**

# Remove nodes

- Add more assumptions/restrictions to the problem, such that some nodes in the abstract graph can be merged.

- Example:
  - 10 airports
  - Each airport has 5 planes and 20 cargos → only 5 airports have cargos
  - Goal: move all cargos to some destination → All cargos at the same airport have the same destination.

# IV. GRAPHPLAN Algorithm

# Planning graph

- **A directed graph that can guide the selection a good heuristic**

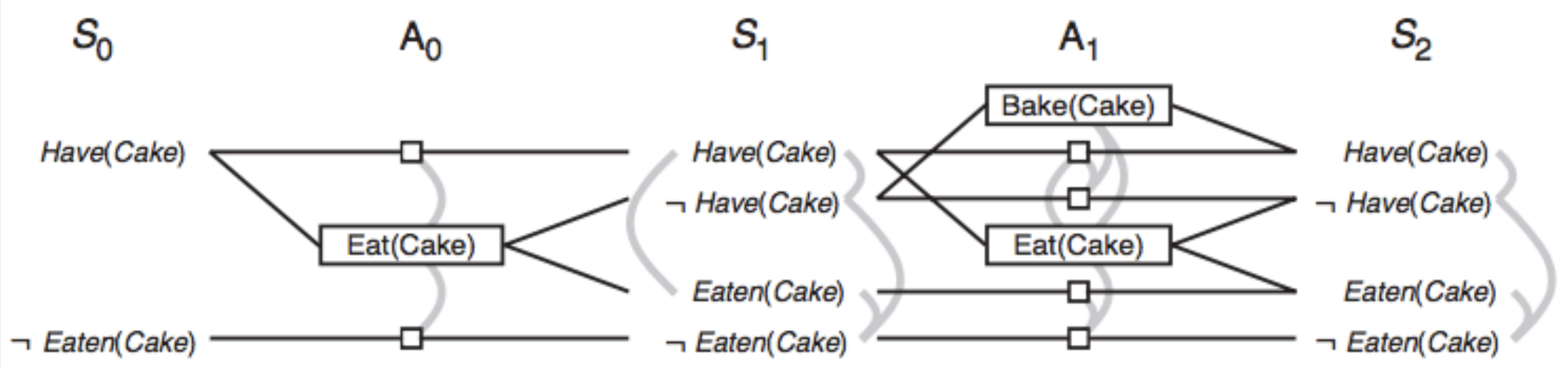



**Figure 10.8** The planning graph for the "have cake and eat cake too" problem up to level $S_2$. Rectangles indicate actions (small squares indicate persistence actions), and straight lines indicate preconditions and effects. Mutex links are shown as curved gray lines. Not all mutex links are shown, because the graph would be too cluttered. In general, if two literals are mutex at $S_i$, then the persistence actions for those literals will be mutex at $A_i$ and we need not draw that mutex link.

Levels of the graph provide rich information.

Graph size polynomial in the size of the planning problem.

# Planning graph

- A directed graph that can guide the selection a good heuristic

**function** GRAPHPLAN(*problem*) **returns** solution or failure

*graph* ← INITIAL-PLANNING-GRAPH(*problem*)
*goals* ← CONJUNCTS(*problem*.GOAL)
*nogoods* ← an empty hash table
**for** $tl = 0$ **to** $\infty$ **do**
  **if** *goals* all non-mutex in $S_t$ of *graph* **then**
    *solution* ← EXTRACT-SOLUTION(*graph*, *goals*, NUMLEVELS(*graph*), *nogoods*)
    **if** *solution* ≠ *failure* **then return** *solution*
  **if** *graph* and *nogoods* have both leveled off **then return** *failure*
  *graph* ← EXPAND-GRAPH(*graph*, *problem*)

**Figure 10.9** The GRAPHPLAN algorithm. GRAPHPLAN calls EXPAND-GRAPH to add a level until either a solution is found by EXTRACT-SOLUTION, or no solution is possible.

Solving a Boolean CSP

To be continued