Artificial Intelligence Foundation – JC3001

Lecture 28: Algorithms and Heuristics - I

Prof. Aladdin Ayesh (aladdin.ayesh@abdn.ac.uk)

Dr. Binod Bhattarai (binod.bhattarai@abdn.ac.uk)

Dr. Gideon Ogunniye, (g.ogunniye@abdn.ac.uk)

October 2025

Material adapted from:
Russell and Norvig (AIMA Book): Chapter 11 (11.2–11.3)
Jörg Hoffmann (University of Saarland)
Malte Helmert (University of Basel)
Dana Nau (University of Maryland)

UNIVERSITY OF ABERDEEN

Course Progression

- Part 1: Introduction
  1. Introduction to AI ✓
  2. Agents ✓
- Part 2: Problem-solving
  1. Search 1: Uninformed Search ✓
  2. Search 2: Heuristic Search ✓
  3. Search 3: Local Search ✓
  4. Search 4: Adversarial Search ✓
- Part 3: Reasoning and Uncertainty
  1. Reasoning 1: Constraint Satisfaction ✓
  2. Reasoning 2: Logic and Inference ✓
  3. Probabilistic Reasoning 1: BNs ✓
  4. Probabilistic Reasoning 2: HMMs ✓
- Part 4: Planning
  1. Planning 1: Intro and Formalism ✓
  2. **Planning 2: Algorithms and Heuristics**
  3. Planning 3: Hierarchical Planning
  4. Planning 4: Stochastic Planning
- Part 5: Learning
  1. Learning 1: Intro to ML
  2. Learning 2: Regression
  3. Learning 3: Neural Networks
  4. Learning 4: Reinforcement Learning
- Part 6: Conclusion
  1. Ethical Issues in AI
  2. Conclusions and Discussion

Objectives

- Planning Algorithms
  - Planning Graphs
  - Planning based on Heuristic Search
  - Satisfiability Testing

Outline

- Recap
- Planning Graphs and Graphplan

- Tidily arranged action descriptions, restricted language

```
Action:  Buy(x)
Precondition:  At(p), Sells(p,x)
Effect:  Have(x)
```

- Restricted language $\Rightarrow$ efficient algorithms

Actions vs. Action Schemas

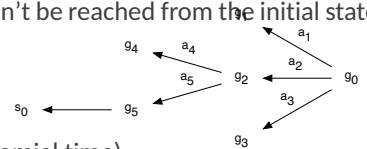
```
Action(Fly(p, from, to),
PRECOND: At(p, from) ∧ Plane(p) ∧ Airport(from) ∧ Airport(to)
EFFECTS: ¬At(p, from) ∧ At(p, to))
```

- This is an action schema; it captures a set of actions that can be obtained by instantiating the variables to different constants
  - Action name and parameter list
  - Precondition
  - Effect

- Recap
- Planning Graphs and Graphplan

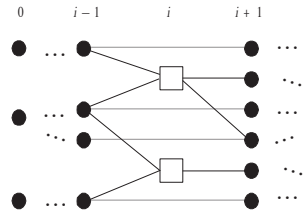
- Before Graphplan came out, most planning researchers were working on PSP-like planners
  - POP, SLNP, UCPOP, etc
- Graphplan caused a sensation because it was so much faster
- Many subsequent planning systems have used ideas from it
  - IPP, STAN, GraphHTN, SGP, Blackbox, Medic, TGP, LPG
  - Many of them are much faster than the original Graphplan

- A big source of inefficiency in search algorithms is the branching factor
  - e.g. a backward search may try lots of actions that can't be reached from the initial state
- One way to reduce branching factor:
  - First create a relaxed problem
    Remove some restrictions of the original problem
    Want the relaxed problem to be easy to solve (polynomial time)
  - The solutions to the relaxed problem will include all solutions to the original problem
- Then do a modified version of the original search
  Restrict its search space to include only actions within solutions to the relaxed problem



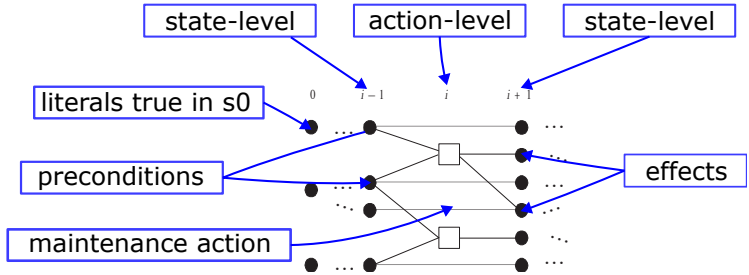
Procedure for Graphplan (very high-level)

- for $k = 0, 1, 2$
  - Graph expansion
    create a “planning graph“ that contains k “levels“
  - Check whether the planning graph satisfies a necessary (but insufficient) condition for plan existence
  - **If** it does then
    - do solution extraction:
      $\rightarrow$ backward search, modified to consider only the actions in the planning graph
      $\rightarrow$ if we find a solution, then return it



The Planning Graph

- Search space for a relaxed version of the planning problem
- Alternating layers of ground literals and actions
  - Nodes at action-level i: actions possibly to executable at time i
  - Nodes at state-level i: literals possibly true at time i
  - Edges: preconditions and effects



Example

- From Daniel Weld (UWashington)
- Suppose you want to prepare dinner as a surprise for your sweetheart (who is asleep)
  $s_0 = \{garbage, cleanHands, quiet\}$
  $g = \{dinner, present, \neg garbage\}$

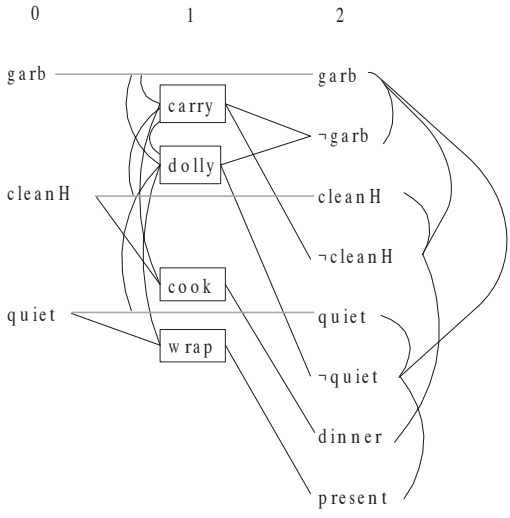
Action	Preconditions	Effects
cook()	cleanHands	dinner
wrap()	quiet	present
carry()	*none*	¬garbage, ¬cleanHands
dolly()	*none*	¬garbage, ¬quiet

Also have the maintenance actions: one for each literal

Example (continued)

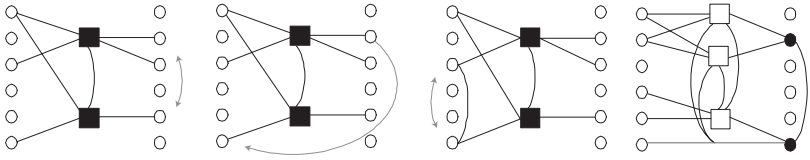
- state-level 0:
  {all atoms in $s0$} ∪
  {negations of all atoms not in s_0}
- action-level 1:
  {all actions whose preconditions are satisfied and non-mutex in s_0}
- state-level 2:
  {all effects of all of the actions in action-level 1}

Action	Preconditions	Effects
cook()	cleanHands	dinner
wrap()	quiet	present
carry()	*none*	¬garbage, ¬cleanHands
dolly()	*none*	¬garbage, ¬quiet



Mutual Exclusion

2 Planning Graphs and Graphplan

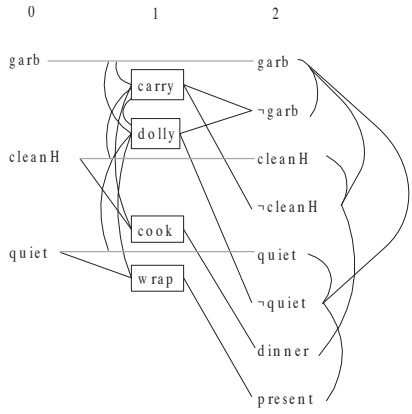


- Two actions at the same action-level are mutex if
  - Inconsistent effects: an effect of one negates an effect of the other
  - Interference: one deletes a precondition of the other
  - Competing needs: **they have mutually exclusive preconditions**
- Otherwise they don't interfere with each other
  - Both may appear in a solution plan
- Two literals at the same state-level are mutex if
  - Inconsistent support: one is the negation of the other, **or all ways of achieving them are pairwise mutex**

Example (continued)

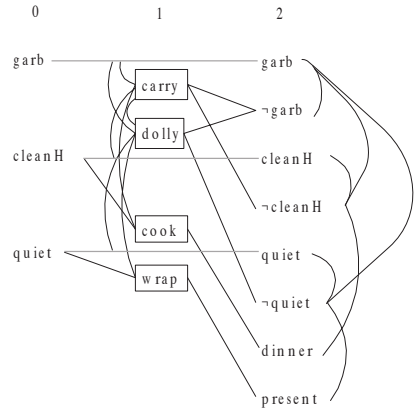
- Augment the graph to indicate mutexes
- $carry$ is mutex with the maintenance action for garbage (inconsistent effects)
- $dolly$ is mutex with wrap (interference)
- $\neg quiet$ is mutex with present (inconsistent support)
- each of $cook$ and $wrap$ is mutex with a maintenance operation

Action	Preconditions	Effects
cook()	cleanHands	dinner
wrap()	quiet	present
carry()	*none*	¬garbage, ¬cleanHands
dolly()	*none*	¬garbage, ¬quiet



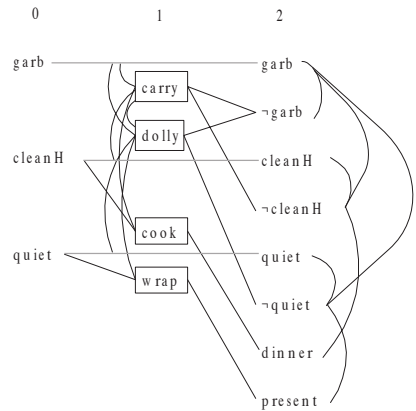
Example (continued)

- Check to see whether there's a possible solution
- Recall that the goal is $\{\neg garbage, dinner, present\}$
- Note that in (state) level 2
  - All of them are there
  - None are mutex with each other
- Thus, there is a chance that a plan exists
- Try to find it:



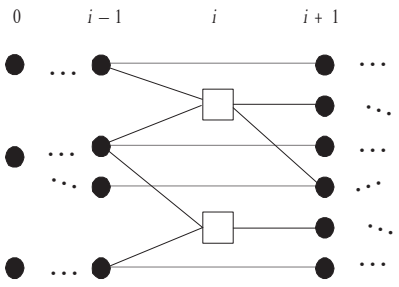
Example (continued)

- Check to see whether there's a possible solution
- Recall that the goal is $\{\neg garbage, dinner, present\}$
- Note that in (state) level 2
  - All of them are there
  - None are mutex with each other
- Thus, there is a chance that a plan exists
- Try to find it: Solution extraction



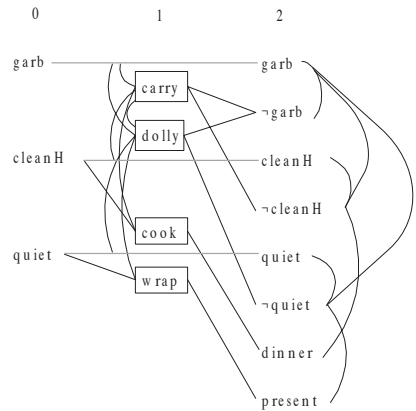
procedure Solution-extraction(g,j)

1. **if** $j = 0$ **then** return the solution
2. *for each* literal $l \in g$
   3. nondeterministically choose an action to use in state s_{j-1} to achieve l
4. **if** any pair of chosen actions are mutex **then** backtrack
5. $g' \leftarrow \{$the preconditions of the chosen actions$\}$
6. Solution-extraction(g', $j - 1$)



Example (continued)

- Two sets of actions for the goals at (state) level 2
- Neither of them works
- Both sets contain **actions** that are mutex
- This also means that $dinner$ and $present$ are **mutex**



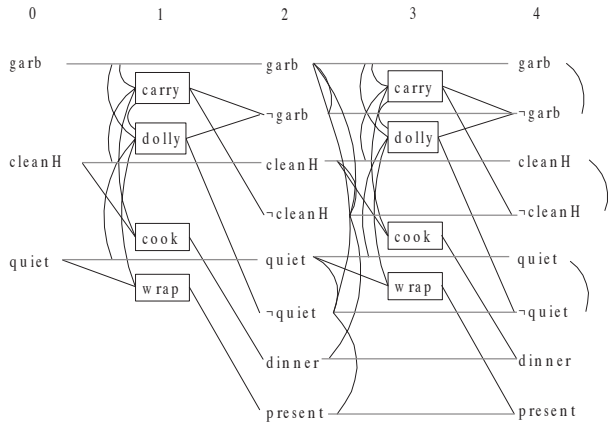
Recall what the algorithm does

Procedure for Graphplan (very high-level)

- for $k = 0, 1, 2$
  - Graph expansion
    create a “planning graph“ that contains k “levels“
  - Check whether the planning graph satisfies a necessary (but insufficient) condition for plan existence
  - **If** it does then
    - do solution extraction:
      $\rightarrow$ backward search, modified to consider only the actions in the planning graph
      $\rightarrow$ if we find a solution, then return it

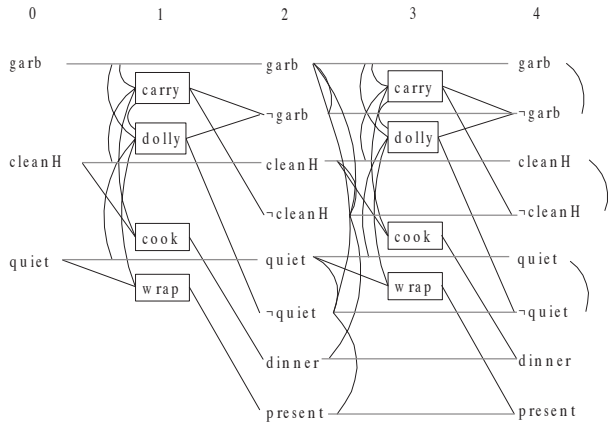
Example (continued)

- Go back and do more graph expansion
- Generate another action-level and another state-level



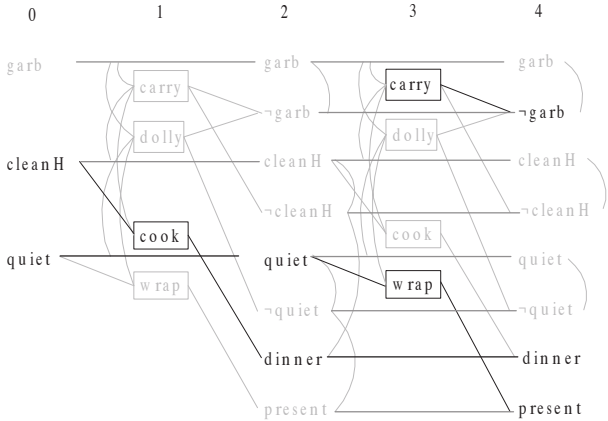
Example (continued)

- Solution extraction
- Twelve combinations at level 4
  - Three ways to achieve $\neg garb$
  - Two ways to achieve $dinner$
  - Two ways to achieve $present$



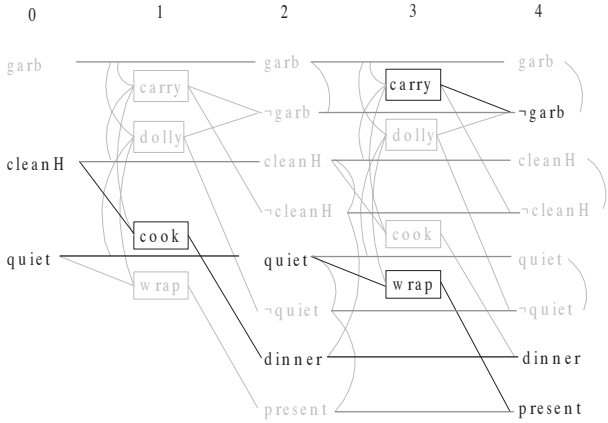
Example (continued)

- Several of the combinations look OK at level 2
- Here's one of them



Example (continued)

- Several of the combinations look OK at level 2
- Here's one of them
  - Call Solution- Extraction recursively at level 4
  - It succeeds
  - Solution whose parallel length is 2



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

To continue in the next session.