Artificial Intelligence Foundation – JC3001

Lecture 25: Introduction to Automated Planning - 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 (11.1)
Ghallab, Nau, and Traverso (Automated Planning: Theory and Practice)

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 4: 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: Algos 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

UNIVERSITY OF ABERDEEN

Objectives

- Search versus Planning
- Planning Problem Representation
  - STRIPS
  - PDDL
- Forward and Backward Search

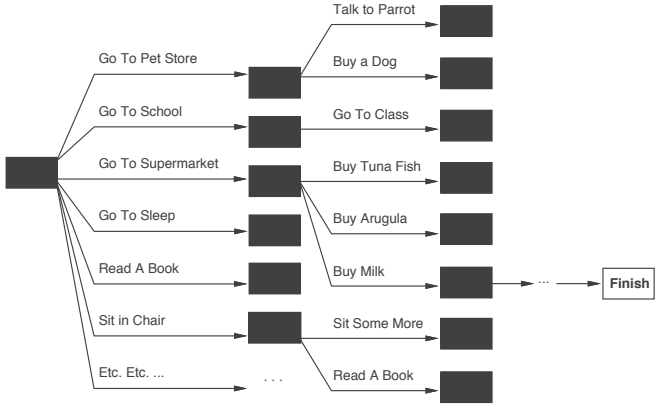
Outline

Search vs. Planning

- What happens when we try to search in the real world?
- Consider the task of buying a book with ISBN 1234567890.
- The agent has actions of the form $Buy(X)$ where X is a ISBN number.
- Using search, all 10 billion states have to be examined to see if one contains $Have(1234567890)$, which is the goal test.
- A sensible agent should reason as follows:
  I have the goal $Have(X)$ where $X = 1234567890$. Now $Buy(Y) \rightarrow Have(Y)$.
  Therefore
  $Buy(1234567890) \rightarrow Have(1234567890)$.
  Therefore, I have to execute $Buy(1234567890)$

Search vs. Planning

- Consider the task: get milk, bananas, and a cordless drill
- Standard search algorithms seem to fail miserably



Problem 1: overwhelming number of irrelevant actions

- What if we are trying to buy 4 rather than 1 book?
  There are 10^{40} plans of 4 steps, we need a heuristic function.
- But each domain needs a new new heuristic function
- We need to use a domain-independent heuristic to allow the agent to function in any situation.
- If we have goals of the form
  $Have(A) \wedge Have(B) \wedge Have(C) \wedge Have(D)$,
  a state containing $Have(A) \wedge Have(C)$ would have cost 2.
  Representing our domain in this way automatically gives us a useful heuristic.

Problem 2: Identifying a good heuristic function

- Problem: delivering packages to various locations
- It makes sense to decompose the problem, finding the nearest airport for each destination and dividing the overall problem into one subproblem for each airport.
- If it takes $O(n!)$ time to find the best plan in the original case, it will only take $(O(n/k)! \times k)$ time if we can decompose the problem into k equal parts.

Problem 3: Problem decomposition

So what can we do?

- A standard representation of the planning problem – states, actions and goals – make it possible for algorithms to take advantage of the structure of the problem.
  - We need to find a representation that is expressive enough to describe many problems.
  - But this representation must be restrictive enough to allow efficient algorithms to exist.

Outline

- Planning systems do the following:
  - open up action and goal representation to allow selection
  - divide-and-conquer by subgoaling
  - relax requirement for sequential construction of solutions

	Search	**Planning**
States	Lisp data structures	Logical sentences
Actions	Lisp code	Preconditions/outcomes
Goal	Lisp code	Logical sentence (conjunction)
Plan	Sequence from S_0	Constraints from actions

- One basic language is STRIPS
  (Stanford Research Institute Problem Solver).
- There have been many extensions and variations of this basic language.
- PDDL (version 3.1) is a more modern planning language, but at its heart, there are still many similarities to STRIPS.

- Tidily arranged action descriptions, restricted language

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

- Restricted language $\Rightarrow$ efficient algorithms

States

- A state is represented as a conjunction of positive literals
- We can consider propositional literals (e.g. $Rich \wedge Happy$)
- Typically, we will use first order literals, e.g. $At(Plane_1, Melbourne) \wedge At(Plane_2, Sydney)$
- Within a state description, such first order literals must be **ground** and **function free**
- The closed world assumption is used; any condition not mentioned is false

$$At(x, y), At(Father(Fred), Sydney))$$

$$Father(Fred, Bob) \wedge At(Bob, Sydney)$$

Goal Representation

- Goals are partially specified states, represented as conjunctions of **positive ground** literals

$$Rich \wedge Famous$$

$$At(Plane_2, Johannesburg)$$

- A state s satisfies a goal if s contains all the atoms in the goal (and possibly others).

$$Rich \wedge Famous \wedge Miserable$$

$$Rich \wedge Famous \wedge Happy$$

- An action is specified in terms of
  - The preconditions that must hold before it can be executed. The effects that occur when it is executed

    `Action(Fly(p, from, to),`
    `PRECOND: At(p, from) ∧ Plane(p) ∧ Airport(from) ∧ Airport(to)`
    `EFFECTS: ¬At(p, from) ∧ At(p, to))`
- Positive literals are added to the state, negative literals are removed
- Some planning systems use an **add** and **delete** lists

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

- We have defined the **syntax** of STRIPS.
- What are its **semantics**?
- We specify these by describing how actions affect states.

- An action is applicable in any state that satisfies the precondition.
- For the first order action schema, determining applicability requires determining whether a legal substitution exists for variables in the precondition.
  - $At(P_1, JFK) \land At(P_2, SFO) \land Plane(P_1) \land Plane(P_2) \land Airport(JFK) \land Airport(SFO)$
  - $At(p, from) \land Plane(p) \land Airport(from) \land Airport(to)$
    $\{p/P_1, from/JFK, to/SFO\}$
- The concrete action $Fly(P_1, JFK, SFO)$ is applicable.

- Starting in a state s, the result of executing an applicable action a is a new state s', which is identical except
  - Any positive literal P in the effects of a is added to s'
  - Any negative literal $\neg P$ in the effects of a is removed from s'
- States are sets, if a positive effect is already there, we don't re-add it; if a negative effect is not present, then that part of the effect is ignored.
- The STRIPS assumption: **any literal not mentioned in the effect remains unchanged**.
  - STRIPS thus overcomes the **frame problem**.

- Action
  $Action(Fly(p, from, to),$
  $PRECOND : At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$
  $EFFECTS : \neg At(p, from) \wedge At(p, to))$
- State
  $At(C1, SFO) \wedge At(C_2, JFK) \wedge At(P_1, SFO) \wedge At(P_2, JFK) \wedge Cargo(C_1) \wedge$
  $Cargo(C_2) \wedge Plane(P_1) \wedge Plane(P_2) \wedge Airport(SFO) \wedge Airport(JFK)$
- Result of applying $Fly(P_1, SFO, JFK)$ to previous state

- Action **after applying** $\{p/P_1, from/JFK, to/SFO\}$
  $Action(Fly(P_1, SFO, JFK),$
  $PRECOND : At(P_1, SFO) \wedge Plane(P_1) \wedge Airport(SFO) \wedge Airport(JFK)$
  $EFFECTS : \neg At(P_1, SFO) \wedge At(P_1, JFK))$
- State
  $At(C1, SFO) \wedge At(C_2, JFK) \wedge At(P_1, SFO) \wedge At(P_2, JFK) \wedge Cargo(C_1) \wedge$
  $Cargo(C_2) \wedge Plane(P_1) \wedge Plane(P_2) \wedge Airport(SFO) \wedge Airport(JFK)$
- Result of applying $Fly(P_1, SFO, JFK)$ to previous state
  $At(C_1, SFO) \wedge At(C_2, JFK) \wedge At(P_1, JFK) \wedge At(P_2, JFK) \wedge Cargo(C_1) \wedge$
  $Cargo(C_2) \wedge Plane(P_1) \wedge Plane(P_2) \wedge Airport(SFO) \wedge Airport(JFK)$

- A solution for a planning problem is an action sequence that, when executed in the initial state, results in a state that satisfies the goal.
- This is a simple form of the solution concept, we will encounter more complex ones later.

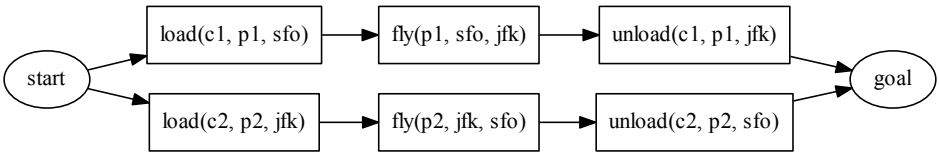
Example

$$
\begin{aligned}
&\mathit{Init}(\mathit{At}(C_1, \mathit{SFO}) \land \mathit{At}(C_2, \mathit{JFK}) \land \mathit{At}(P_1, \mathit{SFO}) \land \mathit{At}(P_2, \mathit{JFK}) \\
&\quad \land \mathit{Cargo}(C_1) \land \mathit{Cargo}(C_2) \land \mathit{Plane}(P_1) \land \mathit{Plane}(P_2) \\
&\quad \land \mathit{Airport}(\mathit{JFK}) \land \mathit{Airport}(\mathit{SFO})) \\
&\mathit{Goal}(\mathit{At}(C_1, \mathit{JFK}) \land \mathit{At}(C_2, \mathit{SFO})) \\
&\mathit{Action}(\mathit{Load}(c, p, a), \\
&\quad \text{PRECOND: } \mathit{At}(c, a) \land \mathit{At}(p, a) \land \mathit{Cargo}(c) \land \mathit{Plane}(p) \land \mathit{Airport}(a) \\
&\quad \text{EFFECT: } \neg\, \mathit{At}(c, a) \land \mathit{In}(c, p)) \\
&\mathit{Action}(\mathit{Unload}(c, p, a), \\
&\quad \text{PRECOND: } \mathit{In}(c, p) \land \mathit{At}(p, a) \land \mathit{Cargo}(c) \land \mathit{Plane}(p) \land \mathit{Airport}(a) \\
&\quad \text{EFFECT: } \mathit{At}(c, a) \land \neg\, \mathit{In}(c, p)) \\
&\mathit{Action}(\mathit{Fly}(p, \mathit{from}, \mathit{to}), \\
&\quad \text{PRECOND: } \mathit{At}(p, \mathit{from}) \land \mathit{Plane}(p) \land \mathit{Airport}(\mathit{from}) \land \mathit{Airport}(\mathit{to}) \\
&\quad \text{EFFECT: } \neg\, \mathit{At}(p, \mathit{from}) \land \mathit{At}(p, \mathit{to}))
\end{aligned}
$$

Find a plan for

$$
\begin{aligned}
&\mathit{Init}(\mathit{At}(C_1, \mathit{SFO}) \land \mathit{At}(C_2, \mathit{JFK}) \land \mathit{At}(P_1, \mathit{SFO}) \land \mathit{At}(P_2, \mathit{JFK}) \\
&\quad \land \mathit{Cargo}(C_1) \land \mathit{Cargo}(C_2) \land \mathit{Plane}(P_1) \land \mathit{Plane}(P_2) \\
&\quad \land \mathit{Airport}(\mathit{JFK}) \land \mathit{Airport}(\mathit{SFO})) \\
&\mathit{Goal}(\mathit{At}(C_1, \mathit{JFK}) \land \mathit{At}(C_2, \mathit{SFO})) \\
&\mathit{Action}(\mathit{Load}(c, p, a), \\
&\quad \text{PRECOND: } \mathit{At}(c, a) \land \mathit{At}(p, a) \land \mathit{Cargo}(c) \land \mathit{Plane}(p) \land \mathit{Airport}(a) \\
&\quad \text{EFFECT: } \neg\, \mathit{At}(c, a) \land \mathit{In}(c, p)) \\
&\mathit{Action}(\mathit{Unload}(c, p, a), \\
&\quad \text{PRECOND: } \mathit{In}(c, p) \land \mathit{At}(p, a) \land \mathit{Cargo}(c) \land \mathit{Plane}(p) \land \mathit{Airport}(a) \\
&\quad \text{EFFECT: } \mathit{At}(c, a) \land \neg\, \mathit{In}(c, p)) \\
&\mathit{Action}(\mathit{Fly}(p, \mathit{from}, \mathit{to}), \\
&\quad \text{PRECOND: } \mathit{At}(p, \mathit{from}) \land \mathit{Plane}(p) \land \mathit{Airport}(\mathit{from}) \land \mathit{Airport}(\mathit{to}) \\
&\quad \text{EFFECT: } \neg\, \mathit{At}(p, \mathit{from}) \land \mathit{At}(p, \mathit{to}))
\end{aligned}
$$

Resulting Plan



To continue in the next session.