

# Artificial Intelligence Foundation – JC3001

Lecture 27: Introduction to Automated Planning - III

**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)

## 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

# Objectives

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



# Outline

## 1 Complexity of Classical Planning

► Complexity of Classical Planning

► Forward and Backward Search

# Complexity Analysis

## 1 Complexity of Classical Planning

- Complexity analyses are done on **decision problems** or **language-recognition problems**
  - Problems that have yes-or-no answers
- A language is a set  $L$  of strings over some alphabet  $A$ 
  - Recognition procedure for  $L$ 
    - A procedure  $R(x)$  that returns “yes” iff the string  $x$  is in  $L$
    - If  $x$  is not in  $L$ , then  $R(x)$  may return “no” or may fail to terminate
- Translate classical planning into a language-recognition problem
- Examine the language-recognition problem’s complexity

# Planning as a Language-Recognition Problem

## 1 Complexity of Classical Planning

- Consider the following two languages

Plan-Existence =  $\{P : P \text{ is the statement of a planning problem that has a solution}\}$

Plan-Length =  $\{(P, n) : P \text{ is the statement of a planning problem that has a solution of length } \leq n\}$

- Look at complexity of recognizing Plan-Existence and Plan-Length under different conditions

# Complexity Classes

## 1 Complexity of Classical Planning

- Complexity classes:
  - NLOGSPACE (nondeterministic procedure, logarithmic space)
  - $\subseteq$  P (deterministic procedure, polynomial time)
  - $\subseteq$  NP (nondeterministic procedure, polynomial time)
  - $\subseteq$  PSPACE (deterministic procedure, polynomial space)
  - $\subseteq$  EXPTIME (deterministic procedure, exponential time)
  - $\subseteq$  NEXPTIME (nondeterministic procedure, exponential time)
  - $\subseteq$  EXPSPACE (deterministic procedure, exponential space)
- Let  $C$  be a complexity class and  $L$  be a language
  - $L$  is  $C$ -hard if for every language  $L' \in C$ ,  $L'$  can be reduced to  $L$  in a polynomial amount of time  
NP-hard, PSPACE-hard, etc.
  - $L$  is  $C$ -complete if  $L$  is  $C$ -hard and  $L \in C$   
NP-complete, PSPACE-complete, etc.

# Possible Conditions

## 1 Complexity of Classical Planning

- Do we give the operators as input to the planning algorithm, or fix them in advance?
- Do we allow infinite initial states?
- Do we allow function symbols?
- Do we allow negative effects?
- Do we allow negative preconditions?
- Do we allow more than one precondition?
- Do we allow operators to have conditional effects?<sup>\*</sup>  
i.e., effects that only occur when additional preconditions are true

# Possible Conditions

## 1 Complexity of Classical Planning

- Do we give the operators as input to the planning algorithm, or fix them in advance?
- Do we allow infinite initial states? ← Not-classical planning
- Do we allow function symbols? ← Not-classical planning
- Do we allow negative effects?
- Do we allow negative preconditions?
- Do we allow more than one precondition?
- Do we allow operators to have conditional effects? \*  
i.e., effects that only occur when additional preconditions are true

→ These take us outside classical planning.

# Decidability of Planning

## 1 Complexity of Classical Planning

### Halting Problem

<i>Allow function symbols?</i>	<i>Decidability of PLAN-EXISTENCE</i>	<i>Decidability of PLAN-LENGTH</i>
No <sup>a</sup>	Decidable	Decidable
Yes	Semidecidable <sup>b</sup>	Decidable

<sup>a</sup> This is ordinary classical planning.

<sup>b</sup> True even if we make several restrictions (see text).

Can cut off the search at every path of length n

→ Next: analyze complexity for the decidable cases.

- In this case, can write domain-specific algorithms
  - e.g., DWR and BlocksWorld: PLAN-EXISTENCE is in P and PLAN-LENGTH is NP-complete

Kind of representation	How the operators are given	Allow negative effects?	Allow negative preconditions?	Complexity of PLAN-EXISTENCE	Complexity of PLAN-LENGTH
Classical rep.	In the input	Yes	Yes/no	EXPSpace-complete	NEXPTIME-complete
		No	Yes	NEXPTIME-complete	NEXPTIME-complete
			No	EXPTIME-complete	NEXPTIME-complete
			No <sup>a</sup>	PSPACE-complete	PSPACE-complete
	In advance	Yes	Yes/no	PSPACE <sup>b</sup>	PSPACE <sup>b</sup>
		No	Yes	NP <sup>b</sup>	NP <sup>b</sup>
			No	P	NP <sup>b</sup>
			No <sup>a</sup>	NLOGSPACE	NP

<sup>a</sup> no operator has

<sup>b</sup> PSPACE-complete or NP-complete

- PLAN-LENGTH is never worse than NEXPTIME-complete
- We can cut off every search path at depth  $n$

Kind of representation	How the operators are given	Allow negative effects?	Allow negative preconditions?	Complexity of PLAN-EXISTENCE	Complexity of PLAN-LENGTH
Classical rep.	In the input	Yes	Yes/no	EXPSpace-complete	NEXPTIME-complete
		No	Yes	NEXPTIME-complete	NEXPTIME-complete
			No	EXPTIME-complete	NEXPTIME-complete
			No <sup>a</sup>	PSPACE-complete	PSPACE-complete
	In advance	Yes	Yes/no	PSPACE <sup>b</sup>	PSPACE <sup>b</sup>
		No	Yes	NP <sup>b</sup>	NP <sup>b</sup>
			No	P	NP <sup>b</sup>
			No <sup>a</sup>	NLOGSPACE	NP

Here, PLAN-LENGTH is harder than PLAN-EXISTENCE

# Planning Complexity Summary

## 1 Complexity of Classical Planning

- We have studied the complexity of classical planning problems  
Difference between **plan-existence** and **plan-length**
- Discussed various “features” of planning languages
- Explored the effect of these features in the complexity



# Outline

## 2 Forward and Backward Search

► Complexity of Classical Planning

► Forward and Backward Search

# Forward Search

## 2 Forward and Backward Search

- Progression planning is similar to the approaches seen in search
- Start with initial state, consider sequences of actions until we find a sequence that reaches the goal state
- Specifying this type of planning problem requires
  - The initial state
  - The possible actions
  - The goal test
  - The step cost (typically 1 per action)

# Forward Search

## 2 Forward and Backward Search

- Without functions symbols  $\rightarrow$  finite state space
  - Any graph search algorithm is complete
- But the state space is huge!
  - Until late 90s it was thought that this approach would fail
  - Very accurate heuristics needed to make this type of search work

# Backward Search

## 2 Forward and Backward Search

- Regression planning moves from goals to initial states
- With STRIPS, it is easy to generate the possible predecessors of a set of goal states
  - Consider only relevant actions,  
i.e. actions that achieve one of the conjuncts of a goal
- Any existing solution can be found by backwards search allowing only relevant actions
- Air cargo transportation with 10 airports, 5 planes (per airport) and 20 pieces of cargo (per airport)
  - Thousands of actions leading out of the initial state
  - Only 20 actions working back from the goal.

## Backward Search

### 2 Forward and Backward Search

- We ask what the states could be in which applying an action leads to the goal
- Computing these states is called **regressing** the goal through the action

$$At(C_1, JFK) \wedge At(C_2, SFO)$$

- The relevant action  $Unload(C_1, p, JFK)$  achieves the first conjunct
- For this action to work, the preconditions have to be satisfied
- So any predecessor state must include  $In(C_1, p) \wedge At(p, JFK)$
- Intuitively,  $At(C_1, JFK)$  should not be true in the predecessor state (else the action would be irrelevant)
- The predecessor is thus:  $In(C_1, p) \wedge At(p, JFK) \wedge At(C_2, SFO)$

# Backward Search - Consistency

## 2 Forward and Backward Search

- An action should be **consistent**, i.e. they should not undo any desired literal
- Given a goal description  $G$ , and a relevant and consistent action  $A$ , we compute the predecessor as follows:
  - 1 Delete any positive effects of  $A$  from  $G$
  - 2 All precondition literals of  $A$  are added (unless they appear in  $G$ ).
- Terminates when a predecessor is generated that is satisfied by the initial state
- In the first order case, substitution is required, e.g. the initial state

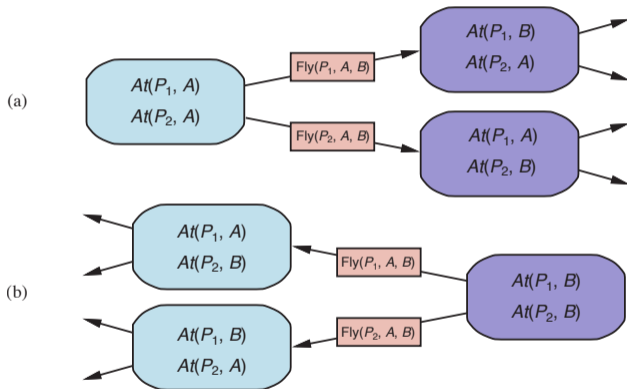
$$In(C_1, P_{12}) \wedge At(P_{12}, JFK) \wedge At(C_2, SFO)$$

is obtained via the substitution  $\{p/P_{12}\}$

This substitution must be applied to actions leading from the state to the goal.

# Search Direction

## 2 Forward and Backward Search



- Neither search is efficient without a good heuristic
- In STRIPS planning
  - actions cost 1
  - distance is plan-length (number of actions)
- Possible heuristic:
  - Look at the effects of actions and number of goals
  - Guess how many actions needed to achieve goal
  - Difficult, but reasonable estimates are possible
- With an admissible heuristic, use  $A^*$  to find the optimal solution

# Creating Heuristics

## 2 Forward and Backward Search

- Ignore specific preconditions of actions (need some domain knowledge)
- Consider the sliding blocks from search

$Action(Slide(t, s_1, s_2),$

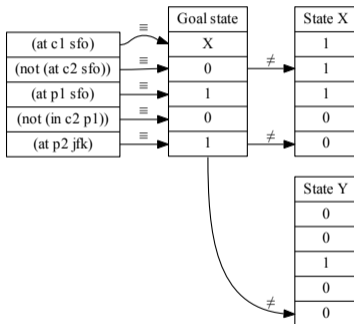
$PRECOND: On(t, s_1) \wedge Tile(t) \wedge Blank(s_2) \wedge Adjacent(s_1, s_2)$

$EFFECT: On(t, s_2) \wedge Blank(s_1) \wedge \neg On(t, s_1) \wedge \neg Blank(s_2))$

- If we remove the  $Blank(s_2)$ , we are left with Manhattan distance

# Hamming Distance

## 2 Forward and Backward Search



# Planning Graphs

## 2 Forward and Backward Search

- All heuristics we've seen so far have limitations
  - Most fast heuristics are inadmissible (but can be really fast)
- Many planning approaches use a data structure called Planning Graph from which we can derive very efficient (and accurate) heuristics
  - Next lecture

# Introduction to Planning Summary

## 2 Forward and Backward Search

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

Any Questions.