Planning & Reinforcement Learning

Artificial Intelligence

Jacopo Parretti

I Semester 2025-2026

Indice

Parte I

Classical Planning

1 Introduction to Planning

1.1 Historical Context and Motivation

Planning has been a central topic in artificial intelligence since the inception of the field at the **Dartmouth Conference in 1956**, where the founding fathers of AI first gathered to define the scope and ambitions of this new discipline. Planning represents one of the fundamental capabilities required for intelligent behavior: the ability to reason about sequences of actions that achieve desired goals.

The motivations for studying automated planning are manifold:

- **Autonomous agents**: Enabling robots, software agents, and autonomous systems to make decisions and act independently in complex environments
- **Resource optimization**: Finding optimal or near-optimal strategies for resource allocation, scheduling, and logistics
- **Scientific applications**: Modeling and solving problems in domains such as space exploration, manufacturing, and healthcare
- **Theoretical foundations**: Understanding the computational complexity and formal properties of reasoning about action and change

1.2 The Planning Cycle

The complete planning process involves several interconnected phases that form a continuous cycle:

1. **Plan Generation**: Given a description of the current state, available actions, and desired goals, synthesize a sequence of actions (a plan) that transforms the initial state into a goal state. This phase involves:
   - Analyzing the initial state to understand what is currently true
   - Identifying the gap between the current state and the goal state
   - Searching through the space of possible action sequences
   - Evaluating candidate plans for correctness and optimality
   - Selecting the best plan according to specified criteria (e.g., shortest length, minimum cost, fastest execution)
2. **Plan Deployment**: Execute the generated plan in the actual environment, monitoring the execution to detect deviations from expected behavior. This phase includes:
   - Translating abstract plan actions into concrete executable commands
   - Monitoring sensors and feedback to verify that actions have the expected effects
   - Detecting discrepancies between predicted and actual states
   - Maintaining a record of executed actions for debugging and learning

3. **Replanning**: When execution monitoring detects that the current plan is no longer valid (due to unexpected events, action failures, or environmental changes), generate a new plan or repair the existing one. Replanning strategies include:
    - **Plan repair**: Modify the existing plan minimally to accommodate the new situation
    - **Replan from scratch**: Generate an entirely new plan from the current state
    - **Contingency planning**: Use pre-computed alternative plans for anticipated failures

This cycle reflects the reality that planning systems must operate in dynamic, partially unpredictable environments where initial plans may need adaptation. The cycle continues iteratively until the goal is achieved or deemed unreachable.

1.2.1 Challenges in Real-World Planning

Real-world deployment of planning systems faces several challenges:

- **Execution uncertainty**: Actions may fail or have unexpected outcomes
- **Incomplete information**: The planner may not have complete knowledge of the environment
- **Dynamic environments**: The world may change while the plan is being executed
- **Computational constraints**: Planning must often occur in real-time with limited computational resources
- **Plan quality vs. planning time**: Trade-off between finding optimal plans and responding quickly

These challenges motivate the development of robust planning algorithms that can handle uncertainty, adapt to changes, and operate efficiently under resource constraints.

2 Formalization of the Planning Problem

2.1 State Transition Systems

The mathematical foundation of planning rests on the concept of a **state transition system**, which provides a formal model of how actions transform states.

Definizione 2.1 (State Transition System)**.** A state transition system is a tuple $\Sigma = \langle S, A, \gamma \rangle$ where:

- S is a finite or countably infinite set of **states**
- A is a finite or countably infinite set of **actions**
- $\gamma : S \times A \to S$ is a **state transition function** that maps a state and an action to a resulting state

The transition function $\gamma(s, a) = s'$ specifies that executing action a in state s results in state s'. This function encodes the **dynamics** of the domain—how the world changes in response to actions.

2.1.1 Properties of State Transition Systems

State transition systems can be characterized by several important properties:

- **Determinism**: In a deterministic system, γ is a function—each state-action pair leads to exactly one successor state. In non-deterministic systems, multiple outcomes are possible.
- **Reachability**: A state s' is reachable from state s if there exists a sequence of actions that transforms s into s'. The reachable state space from an initial state is often much smaller than the total state space.
- **Reversibility**: An action is reversible if there exists another action that undoes its effects. Many real-world actions are irreversible (e.g., breaking an object).
- **State space structure**: The connectivity and topology of the state space significantly affect planning complexity. Highly connected spaces may have many solution paths, while sparse spaces may have few or no solutions.

2.2 Classical Planning: Fundamental Assumptions

Classical planning makes several simplifying assumptions that restrict the class of problems considered but enable efficient algorithmic solutions. These assumptions define the **classical planning framework**:

1. **Finitely many states**: The state space S is finite, allowing exhaustive search techniques. This assumption ensures that the planning problem is decidable and that search algorithms will terminate.

   Justification: While real-world domains may have infinite state spaces (e.g., continuous variables), finite approximations are often sufficient for practical purposes.

2. **Finitely many actions**: The action space A is finite, ensuring decidability. Each state has a finite branching factor in the state space graph.

   Justification: Even in complex domains, the number of distinct action types is typically manageable, though the number of ground actions (instantiated with specific objects) may be large.

3. **Deterministic transition function**: For every state s and action a, there is exactly one resulting state $\gamma(s, a)$. Actions have predictable, certain outcomes.

   Justification: Determinism simplifies reasoning about action effects. Non-deterministic planning requires more complex formalisms (e.g., Markov Decision Processes).

4. **Full observability**: The planner has complete knowledge of the current state at all times. There is no uncertainty about which state the system occupies.

   Justification: Full observability eliminates the need for belief state reasoning and sensing actions. Partial observability requires more sophisticated planning approaches.

5. **Instantaneous actions**: Actions have no duration; they occur instantaneously. Temporal reasoning is not required.

   Justification: Ignoring action durations simplifies the planning model. Temporal planning extends classical planning to handle durations and concurrent actions.

6. **No exogenous events**: Only the planning agent can change the world through deliberate actions. The environment remains static unless acted upon.

   Justification: Exogenous events (e.g., other agents, natural processes) introduce additional complexity. Classical planning assumes a static world between actions.

These assumptions, while restrictive, capture a significant and important class of planning problems and provide a foundation for understanding more complex, realistic planning scenarios.

2.2.1 Relaxing Classical Assumptions

Modern planning research has developed extensions that relax these assumptions:

- **Probabilistic planning**: Handles non-deterministic actions with probability distributions over outcomes
- **Conformant planning**: Plans under partial observability without sensing
- **Contingent planning**: Generates conditional plans that include sensing actions
- **Temporal planning**: Reasons about action durations and temporal constraints
- **Multi-agent planning**: Coordinates plans among multiple agents

2.3 The Planning Problem

Given a state transition system $\Sigma = \langle S, A, \gamma \rangle$, we can now formally define the planning problem.

Definizione 2.2 (Planning Problem)**.** A planning problem is a tuple $P = \langle \Sigma, s_0, G \rangle$ where:

- $\Sigma = \langle S, A, \gamma \rangle$ is a state transition system
- $s_0 \in S$ is the **initial state**
- $G \subseteq S$ is a set of **goal states**

Alternatively, goals can be specified as a **goal formula** g, a logical expression that is satisfied by exactly those states in G. This allows more compact and expressive goal specifications.

2.3.1 Goal Specification

Goals can be specified in several ways:

- **Explicit goal states**: Enumerate the set G of acceptable final states. This is impractical for large state spaces.
- **Goal formula**: A logical formula g such that $G = \{s \in S \mid s \models g\}$. For example, in propositional logic: $g = \texttt{At}(\text{Robot}, \text{RoomB}) \wedge \texttt{Clean}(\text{RoomB})$
- **Goal conditions**: A set of propositions that must be true in the goal state. This is the most common approach in classical planning.
- **Utility functions**: In optimization settings, goals may be specified as utility functions to maximize or cost functions to minimize.

Definizione 2.3 (Solution Plan)**.** A **solution** to a planning problem $P = \langle \Sigma, s_0, G \rangle$ is a sequence of actions $\pi = \langle a_1, a_2, \ldots, a_n \rangle$ such that:

$$s_n = \gamma(\gamma(\cdots \gamma(\gamma(s_0, a_1), a_2) \cdots, a_{n-1}), a_n) \in G$$

That is, executing the sequence of actions starting from the initial state s_0 results in a goal state.

We can also write this more compactly using the notation $\gamma^*(s, \pi)$ to denote the state reached by executing plan π from state s:

$$\gamma^*(s_0, \pi) \in G$$

2.3.2 Plan Quality

Not all solution plans are equally desirable. Common quality metrics include:

- **Plan length**: Number of actions in the plan. Shorter plans are often preferred.
- **Plan cost**: Sum of action costs. Each action a may have an associated cost $c(a)$.
- **Makespan**: Total execution time (relevant in temporal planning).
- **Resource consumption**: Amount of resources used during execution.

An **optimal plan** minimizes the chosen quality metric among all solution plans.

3 Representation of Planning Problems

3.1 The Challenge of Explicit Representation

While the state transition system formalism is mathematically elegant, it faces a critical practical challenge: **the curse of dimensionality**. Real-world planning domains can have exponentially large state spaces. For example:

- A domain with n boolean variables has 2^n possible states
- A domain with k objects and m binary relations has up to 2^{mk^2} states
- Enumerating all states and transitions explicitly is infeasible for even moderately-sized problems

Esempio 3.1 (Blocks World Complexity)**.** Consider a Blocks World domain with n blocks. The number of possible configurations grows super-exponentially with n. For $n = 10$ blocks, there are more than 10^{13} possible configurations. Explicitly representing the transition function would require storing information about trillions of state-action pairs.

This motivates the need for **compact, structured representations** that exploit regularities in the domain to represent large state spaces and transition functions implicitly.

3.2 Action Schemas: Structured Representation

The classical approach to compact representation uses **action schemas** (also called action templates or operators). An action schema is a parameterized description of a family of related actions.

Definizione 3.1 (Action Schema)**.** An action schema consists of:

- **Name and parameters**: A symbolic name and typed parameters, e.g., $\texttt{Move}(x, y)$
- **Preconditions**: A logical formula Pre specifying when the action can be executed
- **Effects**: A logical formula Eff specifying how the state changes when the action is executed

Actions are **ground instances** of action schemas, obtained by binding the parameters to specific objects in the domain. For example, the schema $\texttt{Move}(x, y)$ might generate ground actions $\texttt{Move}(\text{RoomA}, \text{RoomB})$, $\texttt{Move}(\text{RoomB}, \text{RoomC})$, etc.

3.2.1 Advantages of Action Schemas

Action schemas provide several benefits:

- **Compactness**: A single schema can represent exponentially many ground actions
- **Generality**: Schemas capture the general structure of actions independent of specific objects
- **Scalability**: Adding new objects to the domain automatically generates new applicable actions
- **Knowledge reuse**: Schemas learned in one domain can be transferred to similar domains

3.3 State Representation: Propositional Logic

In classical planning, states are typically represented using **propositional logic**:

- A state is a set of **atoms** (propositional variables) that are true in that state

- Atoms not in the set are assumed false (**closed-world assumption**)
- Action preconditions and effects are logical formulas over these atoms

This representation allows:

- Compact encoding of structured states
- Efficient reasoning about action applicability and effects
- Use of logical inference techniques for plan generation

3.3.1 State Update Semantics

When an action is executed, the state is updated according to the action's effects:

- **Add list**: Atoms that become true after the action
- **Delete list**: Atoms that become false after the action
- **Frame axioms**: Atoms not mentioned in the effects remain unchanged

The **STRIPS assumption** (named after the Stanford Research Institute Problem Solver) states that only atoms explicitly mentioned in the effects change; all other atoms persist unchanged. This simplifies reasoning about action effects.

3.4 Example: Blocks World

Consider the classic **Blocks World** domain, one of the most studied domains in planning research.

3.4.1 Domain Description

The Blocks World consists of:

- A set of blocks that can be stacked on top of each other
- A table with unlimited space
- A robot arm that can pick up and put down blocks

State representation:

- $\texttt{On}(x, y)$: Block x is directly on top of block y
- $\texttt{OnTable}(x)$: Block x is on the table
- $\texttt{Clear}(x)$: Block x has no blocks on top of it
- $\texttt{Holding}(x)$: The robot arm is holding block x
- `ArmEmpty`: The robot arm is not holding anything

3.4.2 Action Schemas

PickUp(x): Pick up block x from the table

Parameters: x (block)
Precondition: $\texttt{Clear}(x) \land \texttt{OnTable}(x) \land \texttt{ArmEmpty}$
Effect: $\texttt{Holding}(x) \land \neg\texttt{OnTable}(x) \land \neg\texttt{Clear}(x) \land \neg\texttt{ArmEmpty}$

PutDown(x): Put down block x on the table

Parameters: x (block)
Precondition: $\texttt{Holding}(x)$
Effect: $\texttt{OnTable}(x) \land \texttt{Clear}(x) \land \texttt{ArmEmpty} \land \neg\texttt{Holding}(x)$

Stack(x, y): Stack block x on top of block y

Parameters: x, y (blocks)
Precondition: $\texttt{Holding}(x) \land \texttt{Clear}(y)$
Effect: $\texttt{On}(x, y) \land \texttt{Clear}(x) \land \texttt{ArmEmpty} \land \neg\texttt{Holding}(x) \land \neg\texttt{Clear}(y)$

Unstack(x, y): Remove block x from on top of block y

Parameters: x, y (blocks)
Precondition: $\texttt{On}(x, y) \land \texttt{Clear}(x) \land \texttt{ArmEmpty}$
Effect: $\texttt{Holding}(x) \land \texttt{Clear}(y) \land \neg\texttt{On}(x, y) \land \neg\texttt{Clear}(x) \land \neg\texttt{ArmEmpty}$

3.4.3 Example Problem Instance

Initial state: Blocks A, B, C are all on the table

$$s_0 = \{\texttt{OnTable}(A), \texttt{OnTable}(B), \texttt{OnTable}(C), \texttt{Clear}(A), \texttt{Clear}(B), \texttt{Clear}(C), \texttt{ArmEmpty}\}$$

Goal: Stack the blocks in the order A on B on C

$$G = \{\texttt{On}(A, B), \texttt{On}(B, C), \texttt{OnTable}(C)\}$$

Solution plan:

1. $\texttt{PickUp}(B)$
2. $\texttt{Stack}(B, C)$
3. $\texttt{PickUp}(A)$
4. $\texttt{Stack}(A, B)$

This plan has length 4 and achieves the goal from the initial state.