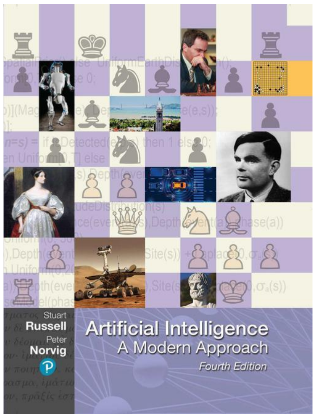
Artificial Intelligence: A Modern Approach

Fourth Edition



Chapter 11

Automated Planning

Outline

- Definition of Classical Planning
- Algorithms for Classical Planning
- Heuristics for Planning
- Hierarchical Planning
- Planning and Acting in Nondeterministic Domains
- Time, Schedules, and Resources
- Analysis of Planning Approaches

Definition of Classical Planning

- **Classical planning** is defined as the task of finding a sequence of actions to accomplish a goal in a discrete, deterministic, static, fully observable environment.
- **Planning Domain Definition Language** (Ghallab et al., 1998). PDDL is a factored representation.
  - Basic PDDL can handle classical planning domains, and extensions can handle non-classical domains that are continuous, partially observable, concurrent, and multi-agent.
  - **State**: represented as a conjunction of ground atomic fluents
  - uses **database semantics**: the closed-world assumption means that any fluents that are not mentioned are false

Definition of Classical Planning

- An **action schema** represents a family of ground actions
- The schema consists of the action name, a list of all the variables used in the schema, a precondition and an effect.
  Action(*Fly*(*p*, *from*, *to*),
  PRECOND: *At*(*p*, *from*) ∧ *Plane*(*p*) ∧ *Airport*(*from*) ∧ *Airport*(*to*)
  EFFECT: ¬*At*(*p*, *from*) ∧ *At*(*p*, *to*))
- A set of action schemas serves as a definition of a planning domain. A specific problem within the domain is defined with the addition of an initial state and a goal
- The **initial state** is a conjunction of ground fluents
- The **goal** is just like a precondition: a conjunction of literals (positive or negative) that may contain variables

Definition of Classical Planning

$Init(At(C_1, SFO) \wedge At(C_2, JFK) \wedge At(P_1, SFO) \wedge At(P_2, JFK)$
$\wedge\ Cargo(C_1) \wedge Cargo(C_2) \wedge Plane(P_1) \wedge Plane(P_2)$
$\wedge\ Airport(JFK) \wedge Airport(SFO))$
$Goal(At(C_1, JFK) \wedge At(C_2, SFO))$
$Action(Load(c, p, a),$
PRECOND: $At(c, a) \wedge At(p, a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
EFFECT: $\neg At(c, a) \wedge In(c, p))$
$Action(Unload(c, p, a),$
PRECOND: $In(c, p) \wedge At(p, a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
EFFECT: $At(c, a) \wedge \neg In(c, p))$
$Action(Fly(p, from, to),$
PRECOND: $At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$
EFFECT: $\neg At(p, from) \wedge At(p, to))$

Figure 11.1 A PDDL description of an air cargo transportation planning problem.

Definition of Classical Planning

$Init(Tire(Flat) \wedge Tire(Spare) \wedge At(Flat, Axle) \wedge At(Spare, Trunk))$
$Goal(At(Spare, Axle))$
$Action(Remove(obj, loc),$
PRECOND: $At(obj, loc)$
EFFECT: $\neg At(obj, loc) \wedge At(obj, Ground))$
$Action(PutOn(t, Axle),$
PRECOND: $Tire(t) \wedge At(t, Ground) \wedge \neg At(Flat, Axle) \wedge \neg At(Spare, Axle)$
EFFECT: $\neg At(t, Ground) \wedge At(t, Axle))$
$Action(LeaveOvernight,$
PRECOND:
EFFECT: $\neg At(Spare, Ground) \wedge \neg At(Spare, Axle) \wedge \neg At(Spare, Trunk)$
$\wedge \neg At(Flat, Ground) \wedge \neg At(Flat, Axle) \wedge \neg At(Flat, Trunk))$

Figure 11.2 The simple spare tire problem.

Definition of Classical Planning

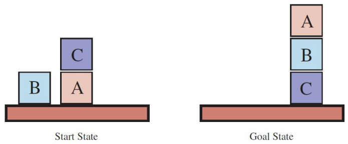


Figure 11.3 Diagram of the blocks-world problem in Figure 11.4.

$Init(On(A, Table) \land On(B, Table) \land On(C, A)$
$\quad \land Block(A) \land Block(B) \land Block(C) \land Clear(B) \land Clear(C) \land Clear(Table))$
$Goal(On(A, B) \land On(B, C))$
$Action(Move(b, x, y),$
$\quad$ PRECOND: $On(b, x) \land Clear(b) \land Clear(y) \land Block(b) \land Block(y) \land$
$\qquad (b \neq x) \land (b \neq y) \land (x \neq y),$
$\quad$ EFFECT: $On(b, y) \land Clear(x) \land \neg On(b, x) \land \neg Clear(y))$
$Action(MoveToTable(b, x),$
$\quad$ PRECOND: $On(b, x) \land Clear(b) \land Block(b) \land Block(x),$
$\quad$ EFFECT: $On(b, Table) \land Clear(x) \land \neg On(b, x))$

Figure 11.4 A planning problem in the blocks world: building a three-block tower. One solution is the sequence $[MoveToTable(C, A), Move(B, Table, C), Move(A, Table, B)]$.

Algorithms for Classical Planning

- Forward state-space search for planning
  - Start at **initial state**
  - To determine the applicable actions we **unify the current state** against the **preconditions** of each action schema
  - For each unification that successfully results in a substitution, we apply **the substitution to the action schema** to yield a **ground action** with no variables.
- Backward search for planning
  - **Start at the goal** and apply the actions backward until we find a sequence of steps that reaches the initial state
  - **Consider relevant actions at each step.**
  - **Reduces branching factor**
  - A relevant action is one with an effect that unifies with one of the goal literals, but with no effect that negates any part of the goal.

Algorithms for Classical Planning

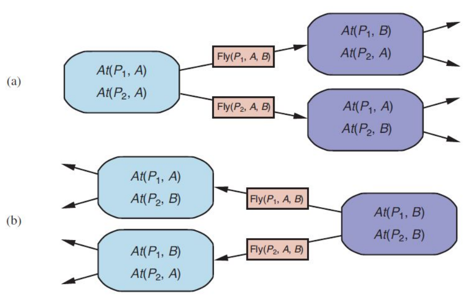


Figure 11.5 Two approaches to searching for a plan. (a) Forward (progression) search through the space of ground states, starting in the initial state and using the problem's actions to search forward for a member of the set of goal states. (b) Backward (regression) search through state descriptions, starting at the goal and using the inverse of the actions to search backward for the initial state.

Algorithms for Classical Planning

- Other classical planning approaches
  - An approach called **Graph plan** uses a specialized data structure, a **planning graph**
  - **Situation calculus** is a method of describing planning problems in first-order logic
  - An alternative called **partial-order planning** represents a plan as a graph rather than a **linear sequence**:
    - each **action** is a **node** in the graph,
    - for each **precondition** of the action there is an **edge** from another action (or from the initial state) that indicates that the predecessor action establishes the **precondition**.

Heuristics for Planning

- **Ignore preconditions heuristic**: drops all preconditions from actions
  - Every action becomes applicable
- **Ignore-delete-lists heuristic:** removing the delete lists from all actions (i.e., removing all negative literals from effects).
- Domain-independent pruning
  - **symmetry reduction:** prune out consideration all symmetric branches of the search tree except for one
  - **forward pruning**: might prune away an optimal solution
  - **relaxed plan**: solution to a relaxed problem
  - **preferred action**: step of the relaxed plan, or it achieves some precondition of the relaxed plan

Heuristics for Planning

- State abstraction in planning
  - **state abstraction**: a many-to-one mapping from states in the ground representation of the problem to the abstract representation
  - relaxations that decrease the number of states
  - **decomposition**: dividing a problem into parts, solving each part independently, and then combining the parts
  - **Subgoal independence assumption**: the cost of solving a conjunction of subgoals is approximated by the sum of the costs of solving each subgoal independently
  - The subgoal independence assumption can be optimistic or pessimistic
    - **optimistic**: negative interactions between the subplans for each subgoal
    - **pessimistic**: inadmissible, when subplans contain redundant actions

Hierarchical Planning

- Higher levels of abstraction with hierarchical decomposition to manage complexity
- Hierarchical structure reduces computational task to a small number of activities at the next lower level.
- Computational cost of finding the correct way to arrange those activities for the current problem is small.
- **High level actions (HLA):**
  - Hierarchical task networks or HTN planning
  - **Assume** full observability & determinism & **primitive actions** standard precondition-effect schemas
  - HLA offers one or more possible refinements into a sequence of actions.
  - **Implementation of HLA**: An HLA refinement that contains only primitive actions.
- **High level plan (HLP)**
  - is the concatenation of implementations of each HLA in the sequence.
  - a high-level plan achieves the goal from a given state if at least one of its implementations achieves the goal from that state

Hierarchical Planning

- Example of Refinement

$Refinement(Go(Home, SFO),$
  STEPS: $[Drive(Home, SFOLongTermParking),$
    $Shuttle(SFOLongTermParking, SFO)]\ )$
$Refinement(Go(Home, SFO),$
  STEPS: $[Taxi(Home, SFO)]\ )$

$Refinement(Navigate([a,b],[x,y]),$
  PRECOND: $a=x \land b=y$
  STEPS: $[\,]\ )$
$Refinement(Navigate([a,b],[x,y]),$
  PRECOND: $Connected([a,b],[a-1,b])$
  STEPS: $[Left, Navigate([a-1,b],[x,y])]\ )$
$Refinement(Navigate([a,b],[x,y]),$
  PRECOND: $Connected([a,b],[a+1,b])$
  STEPS: $[Right, Navigate([a+1,b],[x,y])]\ )$
...

Figure 11.7 Definitions of possible refinements for two high-level actions: going to San Francisco airport and navigating in the vacuum world. In the latter case, note the recursive nature of the refinements and the use of preconditions.

Planning and Acting in Nondeterministic Domains

- Sensorless planning
  - **open-world assumption** in which states contain both positive and negative fluents, and if a fluent does not appear, its value is unknown
  - given belief state b, the agent can consider any action whose preconditions are satisfied by b.
  - assume that the initial belief state is always a **conjunction of literals**, that is, a 1-CNF formula
  - To construct the new belief state b', we **must consider what happens to each literal *l* in each physical states** in b when action a is applied.
  - What about a literal whose truth value is **unknown** in b? There are three cases:
    - If the action adds l, then l will be true in b' regardless of its initial value.
    - If the action deletes l, then l will be false in b0 regardless of its initial value.
    - If the action does not affect , then l will retain its initial value (which is unknown) and will not appear in b'.

Planning and Acting in Nondeterministic Domains

- Sensorless planning
  - **Conditional effect:** an actions effect is dependant on a state (eg: robot's location)
  - "**when** condition: effect," where condition is a logical formula to be compared against the current state, and effect is a formula describing the resulting state.

$Action(Suck,$
  EFFECT:**when** AtL: $CleanL \land$ **when** AtR: $CleanR)$.

  - When applied to the initial belief state *True*, the resulting belief state is $(AtL \land CleanL) \lor (AtR \land CleanR)$, which **is no longer in 1-CNF**.
  - **All conditional effects** whose conditions are satisfied have their effects applied to generate the resulting belief state; if none are satisfied, then the resulting state is unchanged.

Planning and Acting in Nondeterministic Domains

- Sensorless planning
  - if a **precondition is unsatisfied**, then the action is inapplicable and the resulting state is undefined.
  - it is better to have conditional effects than an inapplicable action
  - split *Suck* into two actions with unconditional effects as follows:

$Action(SuckL,$
  PRECOND:AtL; EFFECT:$CleanL)$
$Action(SuckR,$
  PRECOND:AtR; EFFECT:$CleanR)$.

  - so the belief states all remain in **1-CNF**

Planning and Acting in Nondeterministic Domains

- Contingent planning
  - The generation of **plans with conditional branching** based on **percepts**—is appropriate for environments with **partial observability, nondeterminism, or both**
  - When executing this plan, a contingent-planning agent can maintain its **belief state as a logical formula**
  - **Evaluate** each branch condition by determining if the belief state **entails the condition formula or its negation**

Planning and Acting in Nondeterministic Domains

- Online planning
  - The need for a new plan: **execution monitoring**
  - When there are **too many contingencies** to prepare for
  - When a contingency is not prepared for **replanning** is required
  - Replanning is needed if the agent's model of the world is incorrect (missing precondition, effect or fluent)
  - For online planning an agent monitor based on three approaches.
    - **Action monitoring:** before executing an action, the agent verifies that all the preconditions still hold.
    - **Plan monitoring:** before executing an action, the agent verifies that the remaining plan will still succeed.
    - **Goal monitoring:** before executing an action, the agent checks to see if there is a better set of goals it could be trying to achieve.

Planning and Acting in Nondeterministic Domains

- Online planning (example)

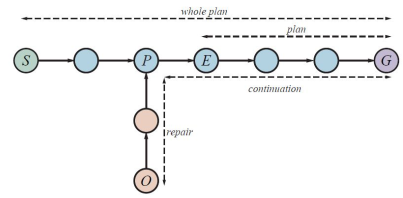


Figure 11.12 At first, the sequence "whole plan" is expected to get the agent from *S* to *G*. The agent executes steps of the plan until it expects to be in state *E*, but observes that it is actually in *O*. The agent then replans for the minimal *repair* plus *continuation* to reach *G*.

Time, Schedules, and Resources

- Classical planning talks about what to do, in what order, but does not talk about time: how long an action takes and when it occurs
- However the real world has resource constraints

- Representing temporal and resource constraints
  - Actions can have a **duration** and **constraints**
  - **Constraints:**
    - Type of resource
    - Number of resources
    - Resource is consumable or reusable
  - **Aggregation**: representation of resources as numerical quantities
  - The representation of resources as numerical quantities, such as *Inspectors*(2), rather than as named entities, such as *Inspector*(I_1) and *Inspector*(I_2),
  - This reduces complexity if multiple quantities are used

Time, Schedules, and Resources

- Solving scheduling problems
- **Graph problems**
- **Uses critical path method (CPM)** to determine the possible start and end times of each action
- A **path** through a graph representing a partial-order plan is a **linearly ordered sequence of actions beginning with Start and ending with Finish**.
- **The critical path** is that path whose total duration is longest; the path is "critical" because it determines the duration of the entire plan
- Actions that are **off the critical path** have a **window of time** in which they can be executed.
- This window of time is known as **Slack**. The window has a earliest possible start time *ES* and latest possible start time *LS*
- Together the *ES* and *LS* times for all the actions constitute a **schedule** for the problem.

Time, Schedules, and Resources

- Many approaches have been tried for **optimal scheduling** including branch-and-bound, simulated annealing, tabu search, and constraint satisfaction.
- Eg: **minimum slack heuristic**:
  - on each iteration, schedule for the earliest possible start whichever unscheduled action has all its predecessors scheduled and has the least slack; then update the *ES* and *LS* times for each affected action and repeat.
  - It is a **greedy heuristic**
  - Resembles minimum-remaining-values (MRV) heuristic in constraint satisfaction.

Analysis of Planning Approaches

- **Planning** combines the **two major areas of AI** we have covered so far: **search and logic.**
- Combination allows planners to **scale up** from toy problems where the number of actions and states as limited to around a dozen, to real-world industrial applications with millions of states and thousands of actions.
- Planning helps in **controlling combinatorial explosion** through the identification of independent subproblems.
- Unfortunately, we do **not yet have a clear understanding** of which techniques work best on which kinds of problems.
- Newer techniques will emerge that provide highly expressive first-order and hierarchal representations
  - Eg: **portfolio planning systems**, where a collection of algorithms are available to apply to any given problem

Summary

- **PDDL,** the Planning Domain Definition Language, describes the initial and goal states as conjunctions of literals, and actions in terms of their preconditions and effects

- **Hierarchical task network (HTN**) planning allows the agent to take advice from the domain designer in the form of high-level actions (HLAs) that can be implemented in various ways by lower-level action sequences.

- **Contingent plans** allow the agent to sense the world during execution to decide what branch of the plan to follow.

- **An online planning agent** uses **execution monitoring** and splices in repairs as needed to recover from unexpected situations, which can be due to nondeterministic actions, exogenous events, or incorrect models of the environment

- Many actions consume resources, such as money, gas, or raw materials