Artificial Intelligence

Classical Planning: Planning via Search

Jiamou Liu
The University of Auckland

Recap: Classical Planning

Classical planning seeks a path from the initial state to a goal through a finite, deterministic, fully-observable search space. We describe a classical planning task using PDDL (STRIPS) syntax:

PDDL domain description:

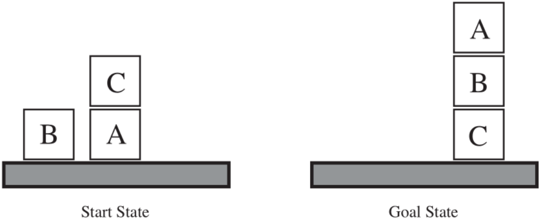
- States (predicates)
- Action scheme a
  - Parameters
  - Preconditions $Precond(a)$
  - Effects $Effect(a)$: $Add(a)$, $Del(a)$

PDDL problem description:

- Initial state I
- Goal g

Example. [Blocks world domain]

- Predicates: $onTable(x)$, $on(x, y)$, $clear(x)$
- Action $moveToTable(x, y)$
  - Preconditions: $clear(x) \wedge on(x, y)$
  - Effects: $clear(y) \wedge onTable(x) \wedge \neg on(x, y)$
- Action $moveToBlock1(x, y, z)$
  - Preconditions: $clear(x) \wedge clear(z) \wedge on(x, y)$
  - Effects: $clear(y) \wedge on(x, z) \wedge \neg clear(z) \wedge \neg on(x, y)$
- Action $moveToBlock2(x, y)$
  - Preconditions: $clear(x) \wedge clear(y) \wedge onTable(x)$
  - Effects: $on(x, y) \wedge \neg clear(y) \wedge \neg onTable(x)$



Planning Algorithms: An Overview

Question. How to build a planner?

- Planning is a problem that is closely linked to topics that we have studied.
- Search is a prevalent problem solving paradigm:
  - State space
  - Plans are paths from initial state to goals
- Logic is a description language to specify planning tasks:
  - Proposition logic
  - First-order logic

Planning Algorithms: An Overview

Question. How to build a planner?

- Planning is a problem that is closely linked to topics that we have studied.
- Search is a prevalent problem solving paradigm:
  - State space
  - Plans are paths from initial state to goals
- Logic is a description language to specify planning tasks:
  - Proposition logic
  - First-order logic

Building a planner relies on our understanding of the task's relations with search and logic. There are therefore two main paradigms:

- Search-based (to be covered in this lecture)
  - Forward (Progression) planning
  - Backward (Regression) planning
- Logic-based (to be covered in the next lecture)
  - Propositional logic-based planning: SATPlan
  - Logic programming-based planning: Prolog planner

Search-based Planning Algorithm

Planning v.s. Search: The main difference is on state representation.

- Search: A state (typically the atomic representation) is used as a single entity by the search algorithm.
- Planning: A state (structured representation) has structural information which is used by the planning algorithm.

In general, planning can be an application of search algorithms.

	Search	**Planning**
States	data structures	Logic sentences
Actions	transitions	Preconditions/effects
Goals	data structures	Logical sentence
Plan	sequence of states	Sequence of actions

Search-based Planning Algorithm

Planning v.s. Search: The main difference is on state representation.

- Search: A state (typically the atomic representation) is used as a single entity by the search algorithm.
- Planning: A state (structured representation) has structural information which is used by the planning algorithm.

In general, planning can be an application of search algorithms.

	Search	**Planning**
States	data structures	Logic sentences
Actions	transitions	Preconditions/effects
Goals	data structures	Logical sentence
Plan	sequence of states	Sequence of actions

A planning system does the following:

1. Action selection based on the internal structures of action and goal representations.
2. Subgoaling through regression.
3. Heuristics through relaxing preconditions.

In this lecture we will discuss all the above.

Algorithm 1: Forward (Progression) Search

Forward (progression) planning starts at the initial state, iteratively applying actions in the forward direction, hoping to reach a goal.

Definition

An action scheme a implicitly defines the following functions:

- For state s, the applicable set at state s is:

$$Applicable(s) = \{a \mid a \text{ is an action}, s \models Precond(a)\}$$

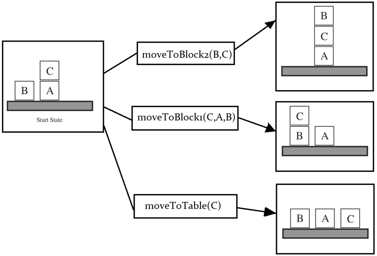
- For state s and action a, the progressed state is the resulting state after applying a at state s:

$$Progress(s, a) = (s \setminus Del(a)) \cup Add(a).$$

Note. Here we use set notation ($\setminus$, $\cup$) on conjunctions of literals.

E.g.[blocks world] Suppose s is $onTable(B) \wedge on(C, A) \wedge onTable(A) \wedge clear(C) \wedge clear(B)$.

- Action $moveToBlock1(C, A, B)$
  - $Precond(moveToBlock1(C, A, B)) = clear(C) \wedge clear(B) \wedge on(C, A)$
  - $moveToBlock1(C, A, B) \in Applicable(s)$
- Action $moveToBlock2(B, C)$
  - $Precnod(moveToBlock2(B, C)) = clear(B) \wedge clear(C) \wedge onTable(B)$
  - $moveToBlock2(B, C) \in Applicable(s)$.
- Action $moveToTable(C, A)$
  - $Precond(moveToTable(C, A)) = clear(C) \wedge on(C, A)$.
  - $moveToTable(C) \in Applicable(s)$.
- Action $moveToBlock2(A, B)$
  - $Precond(moveToBlock2(A, B)) = clear(A) \wedge clear(B) \wedge onTable(A)$.
  - $s \not\models moveToBlock2(A, B)$.



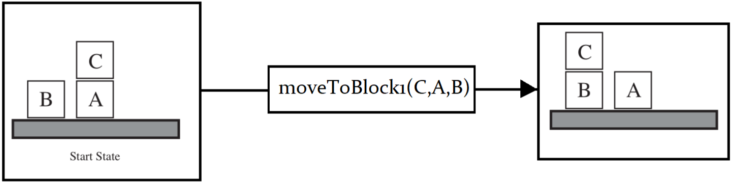
Example. [blocks world]
Suppose the state s is

$$onTable(B) \wedge on(C, A) \wedge onTable(A) \wedge clear(C) \wedge clear(B).$$

For action $a = moveToBlock1(C, A, B)$:

- $Del(moveToBlock1(C, A, B)) = \{clear(B), on(C, A)\}$,
- $Add(moveToBlock1(C, A, B)) = \{clear(A), on(C, B)\}$. So

$$\begin{aligned} & Progress(s, moveToBlock1(C, A, B)) \\ =& (s \setminus Del(moveToBlock1(C, A, B))) \cup Add(moveToBlock1(C, A, B)) \\ =& (onTable(B) \wedge onTable(A) \wedge clear(C) \wedge clear(A) \wedge on(C, B)) \end{aligned}$$



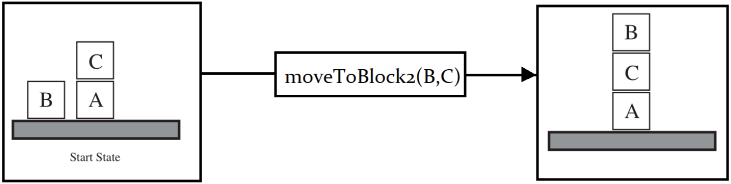
Example. [blocks world]
Suppose s is $onTable(B) \wedge on(C, A) \wedge onTable(A) \wedge clear(C) \wedge clear(B)$.

For action $a = moveToBlock2(B, C)$:

- $Del(moveToBlock2(B, C)) = \{clear(C), onTable(B)\}$,
- $Add(moveToBlock2(B, C)) = \{on(B, C)\}$.

  So

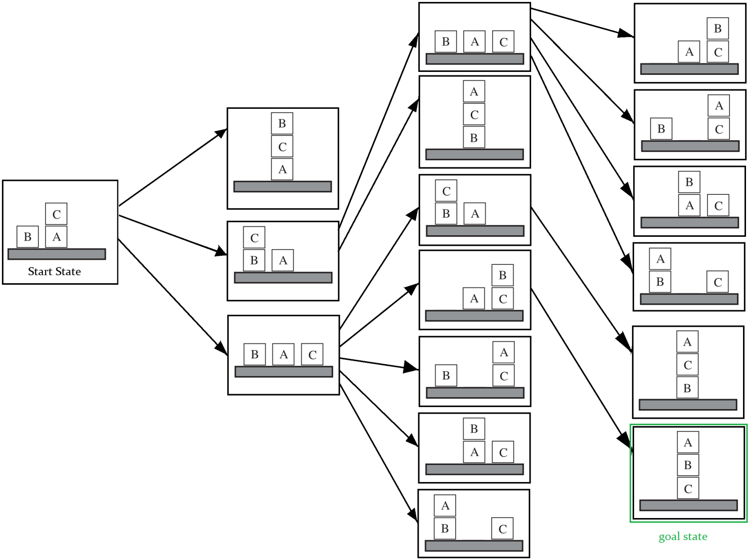
$$
\begin{aligned}
& Progress(s, moveToBlock2(B, C)) \\
=& (s \setminus Del(moveToBlock2(B, C))) \cup Add(moveToBlock2(B, C)) \\
=& (on(C, A) \wedge onTable(A) \wedge clear(B) \wedge on(B, C))
\end{aligned}
$$



Progression planning:

1. **Initialisation:** Set s_0 as the initial state.
2. **Repeat:** At iteration $t > 0$, suppose we are at state s_{t-1}.
   1. Choose an action a_t such that $a_t \in Applicable(s_{t-1})$
   2. Update state: $s_t = Progress(s_{t-1}, a_t)$.
   3. If $s_t \models Goal$, declare that a plan is found and stop.
3. **Termination:** If goal if found, return the plan. Otherwise, return $failure$.

Example. [blocks world]



Complexity issue with progression planning:

- Irrelevant actions: Forward search could explore states and actions that are not relevant to the goal.

  E.g. [buying a book] Action schemas: $Buy(isbn)$ with effect $Own(isbn)$. Goal: $Own(0136042597)$.

  - "Blind" forward planning would search over all possible isbn numbers.
  - There are potentially 10^{10} isbn numbers.

- Large search space: State space becomes too large for uninformed search.

  E.g. [air cargo] Suppose there are 10 airports, each airport has 5 planes, and 20 pieces of cargo.

  Goal: Move all the cargo at AKL to WLG.

  Forward search may search over all possible actions:

  - Flying 50 planes, each to 9 possible destinations.
  - 200 packages may be loaded/unloaded to any plane.

Algorithm 2: Backward (Regression) Search

Backward (regression) planning starts at the goal, iteratively applying actions in the backward direction, until we find a sequence of steps that reaches the initial state.

Definition

Let g be a goal.

- An action a is relevant to g if
  - at least one of a's effects (either positive or negative) unifies with an element of g; and
  - none of a's effects (positive or negative) negates an element of g

  In this case, we write $a \in \mathit{Relevant}(g)$.

- The regressed subgoal from g over action a is

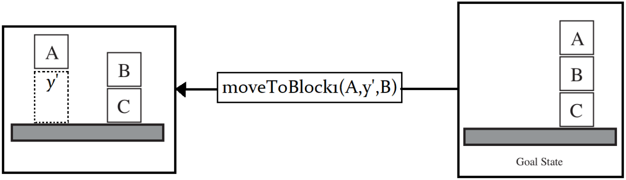
$$\mathit{Regress}(g, a) = (g \setminus \mathit{Add}(a)) \cup \mathit{Precond}(a)$$

E.g. [blocks world] Suppose the goal g is $on(A, B) \wedge on(B, C)$

- An action with effect $\neg on(A, B) \wedge on(B, C)$ is irrelevant.
- An action with effect $clear(y) \wedge onTable(x) \wedge \neg on(x, y)$ is irrelevant.

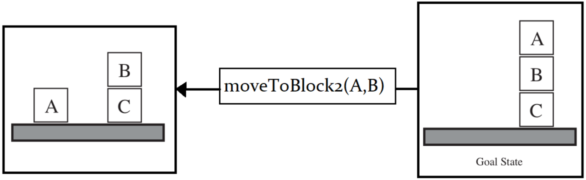
Example. [blocks world] Suppose the goal g is $on(A,B) \wedge on(B,C)$. Action a is $moveToBlock1(x,y,z)$:

- **Effect:** $clear(y) \wedge on(x,z) \wedge \neg clear(z) \wedge \neg on(x,y)$.
- **Unification:** $on(A,B)$ unifies with $on(x,z)$. So $a \in Relevant(g)$.
- **After unification:** $moveToBlock1(A,y',B)$
  - Preconditions: $clear(A) \wedge clear(B) \wedge on(A,y')$
  - Effects: $clear(y') \wedge on(A,B) \wedge \neg clear(B) \wedge \neg on(A,y')$
- **Regression:** $Regress(g,a) = on(B,C) \wedge clear(A) \wedge clear(B) \wedge on(A,y')$



Example. [blocks world] Suppose the goal g is $on(A, B) \wedge on(B, C)$. Action a is $moveToBlock2(x, y)$:

- **Effect:** $on(x, y) \wedge \neg clear(y) \wedge \neg onTable(x)$.
- **Unification:** $on(A, B)$ unifies with $on(x, y)$. So $a \in Relevant(g)$.
- **After unification:** $moveToBlock2(A, B)$
  - Preconditions: $clear(A) \wedge clear(B) \wedge onTable(A)$
  - Effects: $on(A, B) \wedge \neg clear(B) \wedge \neg onTable(A)$
- **Regression:**
  $Regress(g, a) = on(B, C) \wedge clear(A) \wedge clear(B) \wedge OnTable(A)$



Regression planning:

1. **Initialisation:** Set g_0 as the goal.
2. **Repeat:** At iteration $t > 0$, suppose we are at subgoal g_{t-1}.
   - Choose an action: $a_t \in Relevant(g_{t-1})$
   - Standardise action: Unify elements of $Effect(a_t)$ with elements of g_{t-1}. Substitute the unbounded variables with new names.
   - Update goal: $g_t = Regress(g_{t-1}, a_t)$.
   - If $I \models g_t$ (where I is the initial state), declare that a plan is found.
3. **Termination:** If a plan is found, return it; otherwise, return *failure*.

Advantage of regression planning: Generally reduced branching factor. Suitable for cases when there is a large number of ground actions.

Example. [buying a book]

- Action:

$$Action(Buy(i), PRECOND : ISBN(i), EFFECT : Own(i))$$

  Note. There are potentially 10^{10} ways to instantiate $Buy(i)$.

- Problem:
  - $Goal(Own(0136042597))$,
  - $Init(ISBN(0000000000), ISBN(0000000001), \ldots, ISBN(9999999999))$.

Applying regression planning:

1. Standardise action:

$$Action(Buy(0136042597), PRECOND : ISBN(0136042597), \\ EFFECT : Own(0136042597))$$

2. Update goal: $Goal(ISBN(0136042597))$
3. Plan found: $Buy(0136042597)$.

Disadvantages of regression planning:

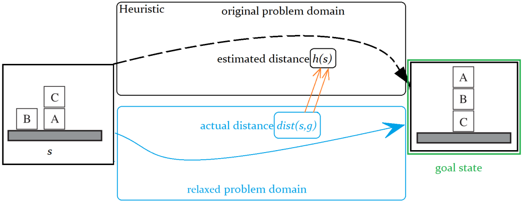
- Plans may still contain redundant steps.
- Plan may not achieve the goal.
- Planning may not terminate for some problems (when DFS is adopted).
- Issues with handling subgoals that have unbounded variables.

Heuristics for Planning

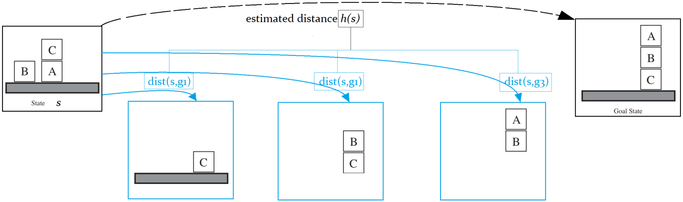
Task: Define a heuristic function to estimate the distance from a state s to the goal g during the search.

Two approaches:

1. Relaxed problem:



2. Subgoal independence:



1. Relaxed problem:

1. Define a relaxed problem with the same current state s and goal g, but with different definitions of the actions.

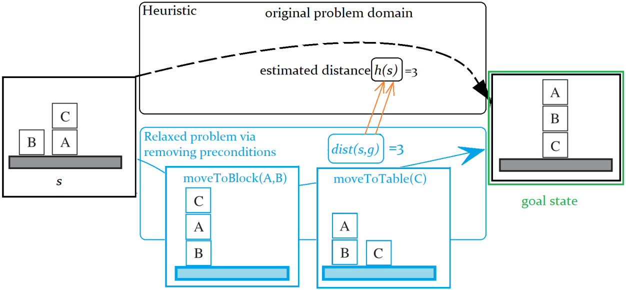
   Note. The relaxed problem should in principle be much easier to solve.

2. Solve the relaxed problem and compute the distance $dist(s, g)$ in the relaxed problem.

3. Set $h(s) = dist(s, g)$. Use this heuristic in the original planning problem.

Possible relaxation: Removing preconditions. This relaxed problem is obtained by removing all preconditions of actions in the original problem.

Note. In the relaxed problem any action can be applied at any step.



Other possible relaxations.

- Removing preconditions from some (but not all) actions.
- Emptying delete list of actions.

  E.g.

$$
\begin{aligned}
&Goal(A \land B \land C)\\
&Action(X, EFFECT : A \land P)\\
&Action(Y, EFFECT : B \land C \land Q \land \neg A)\\
&Action(Z, EFFECT : B \land \neg C)
\end{aligned}
$$

  Distance between initial state (empty) to goal is 2 (X, Y) in the relaxed problem.

- Ignoring some literals (thus simplifying the state space).

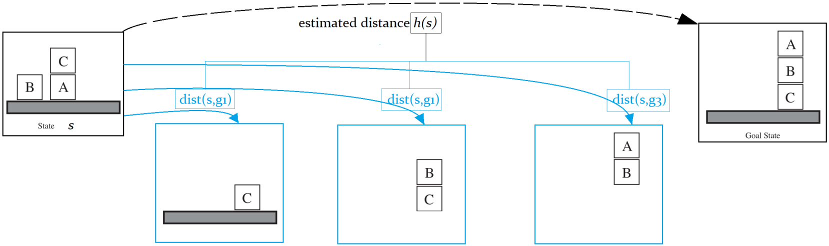
  E.g. [blocks world] Removing *onTable* predicate.

2. Subgoal independence:

1. Divide the goal g into a number of disjoint subgoals $g_1 \wedge g_2 \wedge \cdots \wedge g_k$.
2. Find plans $P_1, \ldots, P_n$ from s to $g_1, g_2, \ldots, g_k$ separately.
3. Set $h(s)$ as $|P_1| + |P_2| + \cdots + |P_n|$.

Note.

- The idea behind this heuristic is divide-and-conquer.
- Independence condition: If the effects of any P_i leave all the preconditions and goals of other P_js unchanged, then h is admissible.



Summary of The Topic

The following are the main knowledge points covered:

- There is a close resemblance between search and planning.
- The main difference is on representations (of states, actions, goals).
- Search-based planning algorithm 1: Forward (progression) search.
  - Starting from the initial state.
  - Choose actions among the applicable set.
  - Generate progressed state.
- Search-based planning algorithm 2: Backward (regression) search.
  - Starting from the goal.
  - Choose actions among the relevant set.
  - Generate regressed subgoal.
- Heuristics for planning:
  - Relaxed problem: e.g. Removing preconditions
  - Subgoal independence