Classical Planning

1. Introduction

Planning is about how an agent achieves its goals. To achieve anything but the simplest goals, an agent must reason about its future. Because an agent does not usually achieve its goals in one step, what it should do at any time depends on what it will do in the future. What it will do in the future depends on the state it is in, which, in turn, depends on what it has done in the past.

Classical planning has the following assumptions:

- the agent's actions are deterministic; that is, the agent can predict the consequences of its actions.
- there are no external events beyond the control of the agent that change the state of the world.
- the world is fully observable; thus, the agent can observe the current state of the world.
- time progresses discretely from one state to the next.
- goals are predicates of states that must be achieved or maintained.

2. Representing States, Actions and Goals

To reason about what to do, an agent must have goals, some model of the world, and a model of the consequences of its actions.

A deterministic action is a partial function from states to states. It is partial because not every action can be carried out in every state.

The precondition of an action specifies when the action can be carried out.

The effect of an action specifies the resulting state.

2.1. Explicit State-Space Representation

One possible representation of the effect and precondition of actions is to explicitly enumerate the states and, for each state, specify the actions that are possible in that state and, for each state-action pair, specify the state that results from carrying out the action in that state. The resulting representation is a finite state automaton. This is not a practical approach for a domain with a large number of states and a large number of possible actions at each state.
2.2. Feature-Based Representation of Actions

A feature-based representation of actions models

- which actions are possible in a state, in terms of the values of the features of the state,
- how the feature values in the next state are affected by the feature values of the current state and the action.

For example, in the block problem, each block is characterized by two features on(x,y) and clear(x).

The feature-based representation of actions uses rules to specify the value of each variable in the state resulting from an action. The bodies of these rules can include the action carried out and the values of features in the previous state.

The rules have two forms:

- A causal rule specifies when a feature gets a new value.
- A frame rule specifies when a feature keeps its value.

It is useful to think of these as two separate cases: what makes the feature change its value, and what makes it keep its value.

In the block problem, the causal rules are represented as post-effects of the actions, while the frame rules are implicit: a feature does not change its value unless a causal rule applies that changes its value.

Causal rules and frame rules do not specify when an action is possible. What is possible is defined by the precondition of the actions.

The precondition of an action is a proposition that must be true before the action can be carried out. In terms of constraints, the robot is constrained to only be able to choose an action for which the precondition is true.

For example, in the block problem move(x,y) has preconditions clear(x) and clear(y). The action move(x,table) is always possible.

Managing feature-based representations

When the feature-based representation uses propositions, e.g. on(a, table), it is sometimes useful to model the effects by ADD and DELETE meta-actions. DELETE removes those propositions that are made false by an action, and ADD adds propositions that are made true by an action.

Example:

Previous state: on(a,b), clear(b), clear(c)
Action: move(a,c)
Meta-actions: delete(clear(c)), delete((on(a,b)), add(on(a,c)), add(clear(b)).

2.3. Initial States and Goals

In a typical planning problem, where the world is fully observable and deterministic, the initial state is defined by specifying the value for each feature for the initial time.

There are two sorts of goals:

- An **achievement goal** is a proposition that must be true in the final state.
- A **maintenance goal** is a proposition that must be true in every state through which the agent passes. These are often **safety goals** - the goal of staying away from bad states.

There may be other kinds of goals such as transient goals (that must be achieved somewhere in the plan but do not have to hold at the end) or resource goals, such as wanting to minimize energy used or time traveled.

3. Forward Planning

A deterministic **plan** is a sequence of actions to achieve a **goal** from a given starting state.

A deterministic **planner** is a problem solver that can produce a plan. The input to a planner is an initial world description, a specification of the actions available to the agent, and a goal description. The specification of the actions includes their preconditions and their effects.

One of the simplest planning strategies is to treat the planning problem as a path planning problem in the **state-space graph**. In a state-space graph, nodes are states, and arcs correspond to actions from one state to another. The arcs coming out of a state $s$ correspond to all of the legal actions that can be carried out in that state. That is, for each state $s$, there is an arc for each action $a$ whose precondition holds in state $s$, and where the resulting state does not violate a maintenance goal. A plan is a path from the initial state to a state that satisfies the achievement goal.

A **forward planner** searches the state-space graph from the initial state looking for a state that satisfies a goal description. It can use any search strategy.

Example: the state-space search tree in the jugs program.

A complete search strategy, such as $A^*$ with multiple-path pruning or iterative deepening, is guaranteed to find a solution. The complexity of the search space is defined by the forward branching factor of the graph.

The **branching factor** is the set of all possible actions at any state, which may be quite large, such as in the block problem. When the domain becomes bigger (many blocks), the branching
factor increases and the search space explodes. This complexity may be reduced by finding good
heuristics but the heuristics have to be very good to overcome the combinatorial explosion.

A state can be represented as either

a. *a complete world description*, in terms of an assignment of a value to each primitive
proposition or as a proposition that defines the state, or
b. *a path from an initial state*; that is, by the sequence of actions that were used to reach
that state from the initial state. In this case, what holds in a state can be deduced
from the axioms that specify the effects of actions.

Difficulties with (a): requires a large amount of space.

Difficulties with (b): What holds in any state is not explicitly given. It might be difficult to
determine whether two states are the same.

4. **Backward Planning**

It is also possible to search backward from the set of states that satisfy the goal. This approach is
not practical when a large number of states satisfy the goal. Example: in the block problem a
very large number of states satisfy the goal on(a,b).

5. **Regression planning**

It is often more efficient to search in a different search space - one where the nodes are not states
but rather are *goals* to be achieved.

**Regression planning** is searching in the graph defined by the following:

- The nodes are goals that must be achieved. A goal is a set of assignments to (some of)
  the features.
- The arcs correspond to actions. In particular, an arc from node g to g’, labeled with
  action act, means act is the last action that is carried out before goal g is achieved, and
  the node g’ is the goal that must be true immediately before act so that g is true
  immediately after act.
- The start node is the goal to be achieved. Here we assume it is a conjunction of
  assignments of values to features.
- The goal condition for the search, goal(g), is true if all of the elements of g are true of
  the initial state.

Given a node that represents goal g, a neighbor of g exists for every action act such that

- *act is possible*: it is possible for act to be carried out and for g to be true immediately
  after act; and
- *act is useful*: act achieves part of g.
Example in the block problem:

Goal: on(a,b).

Act move(a,b) is possible if its preconditions are true, and it is useful because the goal on(a,b) becomes true after the act.

Act move(d,c) may be possible but it might not be useful (unless it achieves the preconditions of move (a,b).

A problem with the regression planner is that a goal may not be achievable. Deciding whether a set of goals is achievable is often difficult to infer from the definitions of the actions.

6. Partial Order Planning

The forward and regression planners enforce a total ordering on actions at all stages of the planning process.

The idea of a partial-order planner is to have a partial ordering between actions and only commit to an ordering between actions when forced.

A partial ordering is a less-than relation that is transitive and asymmetric. A partial-order plan is a set of actions together with a partial ordering, representing a "before" relation on actions, such that any total ordering of the actions, consistent with the partial ordering, will solve the goal from the initial state. Write act$_0$ < act$_1$ if action act$_0$ is before action act$_1$ in the partial order. This means that action act$_0$ must occur before action act$_1$.

For uniformity, we use two pseudo actions start and finish.
- Start is an action that achieves the relations that are true in the initial state, and finish is an action whose precondition is the goal to be solved.
- The pseudo action start is before every other action, and finish is after every other action.

The use of these as actions means that the algorithm does not require special cases for the initial situation and for the goals. When the preconditions of finish hold, the goal is solved.

We must ensure that the actions achieve the conditions they were assigned to achieve.

- Each precondition $P$ of an action act$_i$ in a plan will have an action act$_0$ associated with it such that act$_0$ achieves precondition $P$ for act$_i$.
- The triple $\langle$act$_0$,P,act$_1$$\rangle$ is a causal link. The partial order specifies that action act$_0$ occurs before action act$_1$, which is written as act$_0$ < act$_1$.
- Any other action $A$ that makes $P$ false must either be before act$_0$ or after act$_1$.

Informally, a partial-order planner works as follows: Begin with the actions start and finish and the partial order start < finish. The planner maintains an agenda that is a set of $\langle P,A \rangle$ pairs, where $A$ is an action in the plan and $P$ is an atom that is a precondition of $A$ that must be achieved.
Initially the agenda contains pairs \( \langle G, \text{finish} \rangle \), where \( G \) is an atom that must be true in the goal state.

At each stage in the planning process, a pair \( \langle G, \text{act}_1 \rangle \) is selected from the agenda, where \( P \) is a precondition for action \( \text{act}_1 \). Then an action, \( \text{act}_0 \), is chosen to achieve \( P \). That action is either already in the plan - it could be the start action, for example - or it is a new action that is added to the plan.

The process continues until the agenda is empty.

7. **Summary**

- Planning is the process of choosing a sequence of actions to achieve a goal.
- An action is a function from a state to a state.
- Different planning algorithms can be used to convert a planning problem into a search problem.