SOLVING PROBLEMS BY SEARCHING

When the correct action to take is not immediately obvious, an agent may need to ***plan ahead***: to consider a ***sequence*** of actions that form a path to a goal state. Such an agent is called a problem-solving agent, and the computational process it undertakes is called search.

Problem-Solving Agents

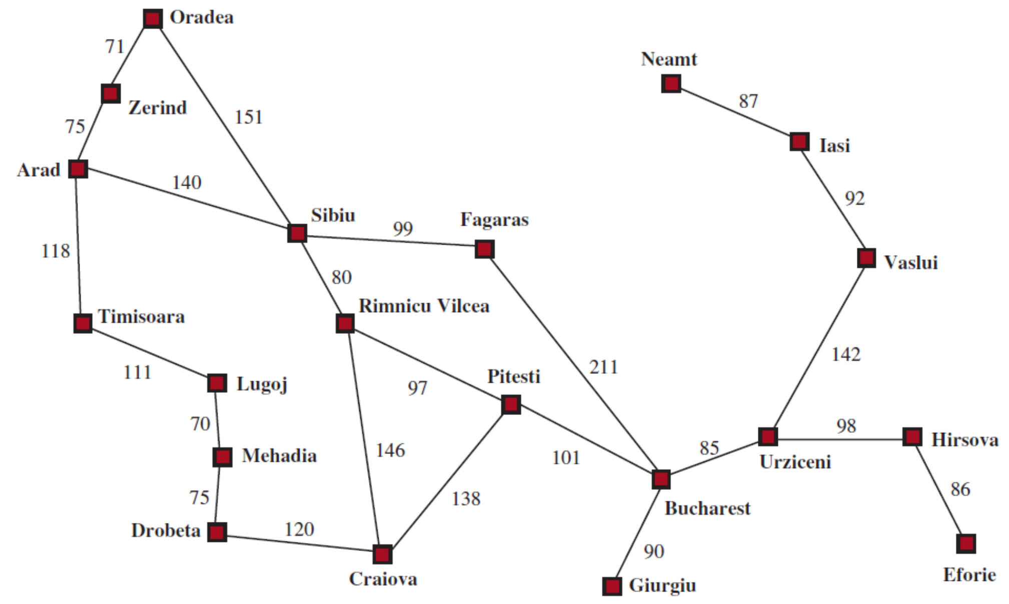

Figure 3.1 A simplified road map of part of Romania, with road distances in miles.

Imagine an agent enjoying a touring vacation in Romania. The agent wants to take in the sights, improve its Romanian, enjoy the nightlife, avoid hangovers, and so on. The decision problem is a complex one.

Now, suppose the agent is currently in the city of Arad and has a nonrefundable ticket to fly out of Bucharest the following day. The agent observes street signs and sees that there are three roads leading out of Arad: one toward Sibiu, one to Timisoara, and one to Zerind. None of these are the goal, so unless the agent is familiar with the geography of Romania, it will not know which road to follow.
If the agent has no additional information—that is, if the environment is unknown—then the agent can do no better than to execute one of the actions at random. In this chapter, we will assume our agents always have access to information about the world, such as

the map in Figure 3.1. With that information, the agent can follow this four-phase problem-solving process:

- **Goal formulation**: The agent adopts the **goal** of reaching Bucharest. Goals organize behavior by limiting the objectives and hence the actions to be considered.

- **Problem formulation**: The agent devises a description of the states and actions necessary to reach the goal—an abstract model of the relevant part of the world.

- **Search**: Before taking any action in the real world, the agent simulates sequences of actions in its model, searching until it finds a sequence of actions that reaches the goal. Such a sequence is called a **solution**. The agent might have to simulate multiple sequences that do not reach the goal, but eventually it will find a solution (such as going from Arad to Sibiu to Fagaras to Bucharest), or it will find that no solution is possible.

- **Execution**: The agent can now execute the actions in the solution, one at a time.

Search problems and solutions

A search **problem** can be defined formally as follows:

- A set of possible **states** that the environment can be in. We call this the **state space**.

- The **initial state** that the agent starts in. For example: *Arad*.

- A set of one or more **goal states**. Sometimes there is one goal state (e.g., *Bucharest*), sometimes there is a small set of alternative goal states, and sometimes the goal is defined by a property that applies to many states (potentially an infinite number). For example, in a vacuum-cleaner world, the goal might be to have no dirt in any location, regardless of any other facts about the state. We can account for all three of these possibilities by specifying an IS-GOAL method for a problem. In this chapter we will sometimes say "the goal" for simplicity, but what we say also applies to "any one of the possible goal states."

- The **actions** available to the agent. Given a state *s*, ACTIONS(*s*) returns a finite set of actions that can be executed in *s*. We say that each of these actions is **applicable** in *s*.
An example:
ACTIONS(*Arad*) = {*ToSibiu*,*ToTimisoara*,*ToZerind*}.

- A **transition model**, which describes what each action does. RESULT(*s*, *a*) returns the state that results from doing action *a* in state *s*. For example,
RESULT(*Arad*, *ToZerind*) = *Zerind* .

• An **action cost function**, denoted by ACTION-COST(s,a, s') when we are programming or $c(s,a, s')$ when we are doing math, that gives the numeric cost of applying action a in state s to reach state s'. A problem-solving agent should use a cost function that reflects its own performance measure; for example, for route-finding agents, the cost of an action might be the length in miles (as seen in Figure 3.1), or it might be the time it takes to complete the action.

A sequence of actions forms a **path**, and a **solution** is a path from the initial state to a goal state. We assume that action costs are additive; that is, the total cost of a path is the sum of the individual action costs. An **optimal solution** has the lowest path cost among all solutions.
The state space can be represented as a **graph** in which the vertices are states and the directed edges between them are actions. The map of Romania shown in Figure 3.1 is such a graph, where each road indicates two actions, one in each direction.

Example Problems

The problem-solving approach has been applied to a vast array of task environments. We list some of the best known here, distinguishing between *standardized* and *real-world* problems. A **standardized problem** is intended to illustrate or exercise various problem-solving methods. It can be given a concise, exact description and hence is suitable as a benchmark for researchers to compare the performance of algorithms. A **real-world problem**, such as robot navigation, is one whose solutions people actually use, and whose formulation is idiosyncratic, not standardized, because, for example, each robot has different sensors that produce different data.

Standardized problems

A **grid world** problem is a two-dimensional rectangular array of square cells in which agents can move from cell to cell. Typically the agent can move to any obstacle-free adjacent cell horizontally or vertically and in some problems diagonally. Cells can contain objects, which the agent can pick up, push, or otherwise act upon; a wall or other impassible obstacle in a cell prevents an agent from moving into that cell.
The **vacuum world** can be formulated as a grid world problem as follows:

• **States**: A state of the world says which objects are in which cells. For the vacuum world, the objects are the agent and any dirt. In the simple two-cell version, the agent can be in either of the two cells, and each call can either contain dirt or not, so there are $2 \cdot 2 \cdot 2 = 8$ states (see Figure 3.2). In general, a vacuum environment with n cells has $n \cdot 2_n$ states.

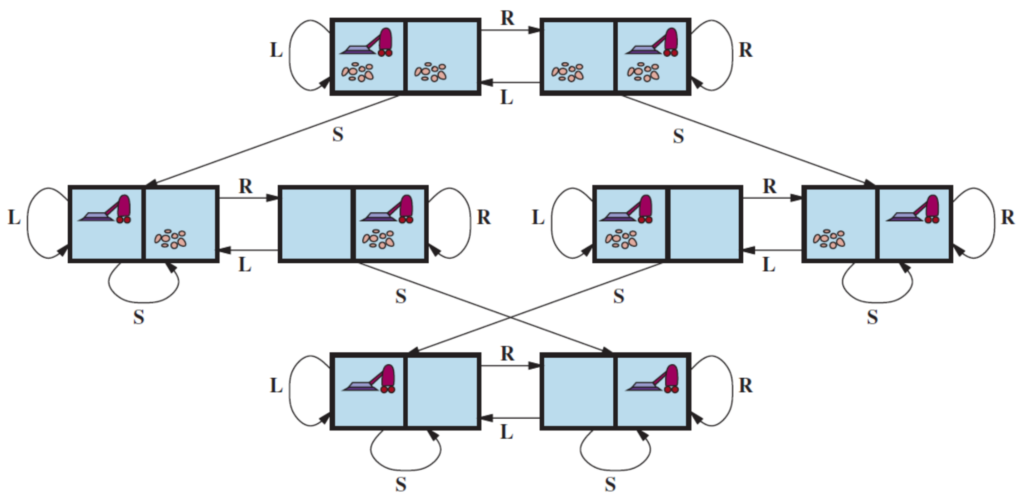

Figure 3.2 The state-space graph for the two-cell vacuum world. There are 8 states and three actions for each state: L = *Left*, R = *Right*, S = *Suck*.

- Initial state: Any state can be designated as the initial state.
- Actions: In the two-cell world we defined three actions: ***Suck***, move ***Left***, and move ***Right***. In a two-dimensional multi-cell world we need more movement actions. We could add ***Upward*** and ***Downward***, giving us four absolute movement actions.
- Transition model: *Suck* removes any dirt from the agent's cell; *Forward* moves the agent ahead one cell in the direction it is facing, unless it hits a wall, in which case the action has no effect. *Backward* moves the agent in the opposite direction, while *TurnRight* and *TurnLeft* change the direction it is facing by 90◦.
- Goal states: The states in which every cell is clean.
- Action cost: Each action costs 1.

Real-world problems

We have already seen how the route-finding problem is defined in terms of specified locations and transitions along edges between them. Route-finding algorithms are used in a variety of applications. Some, such as Web sites and in-car systems that provide driving directions, are relatively straightforward extensions of the Romania example. (The main complications are varying costs due to traffic-dependent delays, and rerouting due to road closures.) Others, such as routing video streams in computer networks, military operations planning, and airline travel-planning systems, involve much more complex specifications.

Consider the airline travel problems that must be solved by a travel-planning Web site:

- States: Each state obviously includes a location (e.g., an airport) and the current time. Furthermore, because the cost of an action (a flight segment) may depend on previous segments, their fare bases, and their status as domestic or international, the state must record extra information about these “historical” aspects.
- Initial state: The user’s home airport.
- Actions: Take any flight from the current location, in any seat class, leaving after the current time, leaving enough time for within-airport transfer if needed.
- Transition model: The state resulting from taking a flight will have the flight’s destination as the new location and the flight’s arrival time as the new time.
- Goal state: A destination city. Sometimes the goal can be more complex, such as “arrive at the destination on a nonstop flight.”
- Action cost: A combination of monetary cost, waiting time, flight time, customs and immigration procedures, seat quality, time of day, type of airplane, frequent-flyer reward points, and so on.

A problem-solving agent is an AI agent that performs reasoning, search, and problem-solving processes to find solutions to a specific problem. It analyzes the problem space, searches for possible solutions based on the available knowledge, and selects the best solution to achieve the desired goal.

Search is a fundamental problem-solving technique in AI and is used in a wide range of applications, including route planning, game playing, scheduling, and more.

Search Algorithms

A search algorithm takes a search problem as input and returns a solution, or an indication of failure.
we consider algorithms that superimpose a search tree over the state space graph, forming various paths from the initial state, trying to find a path that reaches a goal state. Each node in the search tree corresponds to a state in the state space and the edges in the search tree correspond to actions. The root of the tree corresponds to the initial state of the problem.

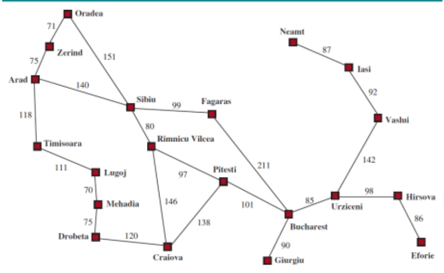

Figure 3.1 A simplified road map of part of Romania, with road distances in miles.