Problems, Problem Spaces, and Search: Notes on Chapter 2

S. Sheerazuddin

SSN College of Engineering, Chennai

AI Problem Solving

In order to build a system to solve a particular problem, we need to do four things:

- Define the problem precisely (say, as a state space search).
- Analyze the problem (to find appropriate heuristics, if using state space search).
- Isolate and represent the task knowledge.
- Choose and apply the best problem-solving technique(s).

Defining AI problem as state space search

To build a program that could **Play Chess**, we have to specify

- starting positions of the chess board
- rules that define legal moves
- board positions that represent a win for either side and
- not only play a legal game of chess **but** also win the game, if possible.

Defining AI problem as state space search

For the problem **Play Chess**, it is fairly easy to provide a formal and complete problem description.

- The starting position can be described as an 8×8 array where each position contains a symbol standing for the appropriate piece in the official chess position.
- The goal is any board position in which the opponent does not have a legal move and his/her king is under attack.
- The legal moves provide the way of getting from the initial state to a goal state.

Defining AI problem as state space search

- The legal moves are described as a set of rules consisting of two parts:
  - A left side that serves as a pattern to be matched against the current board position
  - A right side that describes the changes to be made on the board to reflect the move
- An example rule:

$$\text{White pawn at } \mathit{Square}(e2) \;\&\&\; \mathit{Square}(e3) \text{ empty } \&\&$$

$$\mathit{Square}(e4) \text{ empty} \longrightarrow \text{Move pawn from } \mathit{Square}(e2) \text{ to } \mathit{Square}(e4).$$

Defining AI problem as state space search

The state space representation forms the basis of most AI methods we discuss in the course. The structure of state space representation corresponds to the structure of problem solving in two important ways:

- It allows for a formal definition of a problem as the need to convert some given situation into some desired situation using a set of permissible operations.
- It allows us to define the process of solving a particular problem as a combination of known techniques (rules) and search. Search is an important process in solution of hard problems for which no more direct techniques are available.

Water Jug Problem

Definition

You are given two jugs, a 4-gallon one and a 3-gallon one. Neither has any measuring marker on it. There is a pump that can be used to fill the jugs with water. How can you get exactly 2 gallons of water into the 4-gallon jug ?

- The state space can be described as the set of ordered pair of integers (x, y) such that $0 \leq x \leq 4$ and $0 \leq y \leq 3$, where x represents the number of gallons of water in 4-gallon jug, and y represents the number of gallons of water in 3-gallon jug.
- The start state is $(0, 0)$ and the goal state is $(0, n)$, for $0 \leq n \leq 3$.

Water Jug Problem

The operators (rules) to solve the problem can be described as follows:

1. $(x, y) \longrightarrow (4, y)$, if $x < 4$ — Fill the 4-gallon jug
2. $(x, y) \longrightarrow (x, 3)$, if $y < 3$ — Fill the 3-gallon jug
3. $(x, y) \longrightarrow (0, y)$, if $x > 0$ — Empty the 4-gallon jug
4. $(x, y) \longrightarrow (x, 0)$, if $y > 0$ — Empty the 3-gallon jug
5. $(x, y) \longrightarrow (x + y, 0)$, if $x + y \leq 4$ and $y > 0$ — Empty the 3-gallon jug into 4-gallon jug.
6. $(x, y) \longrightarrow (0, x + y)$, if $x + y \leq 3$ and $x > 0$ — Empty the 4-gallon jug into 3-gallon jug.
7. $(x, y) \longrightarrow (4, y - (x - 4))$, if $x + y \geq 4$ and $y > 0$ — Fill the 4-gallon jug with water from 3-gallon jug.
8. $(x, y) \longrightarrow (x - (y - 3), 3)$, if $x + y \geq 3$ and $x > 0$ — Fill the 3-gallon jug with water from 4-gallon jug.

Formal Description of AI Problem

To summarize, in order to provide a formal description of a problem, we must do the following:

- Define a state space that contains all the possible configurations of the relevant objects. Define this space without explicitly enumerating all of the states it contains.
- Specify one or more states within that space that describe possible situations from which the problem-solving may start. These states are called the *initial states*.
- Specify one or more states that would be acceptable as solutions to the problem. These states are *goal states*.

Formal Description of AI Problem

- Specify a set of rules that describe the actions (operators) available. Doing this will require giving thought to the following issues:
  - What unstated assumptions are present in the informal problem description ?
  - How general should the rules be ?
  - How much of the work required to solve the problem should be precomputed and represented in the rules ?

The problem can then be solved by using the rules, in combination with an appropriate control strategy (DFS, BFS, Best First Search, A* Search), to move through the problem space until a path from initial state to a goal state is found.

Production Systems

A *production system* consists of:

- A set of rules, each consisting of a left side (a pattern) that determines the applicability of the rule and a right side that describes the operation to be performed if the rule is applied.
- One or more knowledge/databases that contain whatever information is appropriate for the particular task.
- A control strategy that specifies the order in which rules will be compared to the database and a way to resolve the conflicts that arise when several rules match at once.
- A rule applier.

The process of solving an AI problem (through state space search) can be usually modeled as a production system.

Control Strategies

How to decide which rule to apply next during the process of search. Often, more than one rule will have its left side match the current state.

- The first requirement of a good control strategy is that it *causes motion*. After each step, we should be closer to the goal.
- The second requirement of a good control strategy is that it be *systemic*. That is, the strategy must have global motion. There are two standard systemic control strategies:
  - Breadth-First Search (BFS)
  - Depth-First Search (DFS)

Comparison of BFS & DFS

Advantages of DFS:

- DFS requires less memory ($O(l)$) since only nodes on the current path (of size l) are stored. This contrasts with BFS where a large part of the search tree ($O(2^l)$) has to be stored as *fringe*.
- DFS may be lucky and may find a solution without examining much of the search space. In BFS all nodes at level n have to be explored before moving to nodes at level $n + 1$.

Advantages of BFS:

- BFS never gets trapped exploring a blind alley. DFS can follow a single, unfruitful path for long.
- BFS is complete–if there is a solution, BFS is guaranteed to find it. Further, if there are multiple solutions, BFS will find the minimum one. DFS is incomplete and may not find the minimum solution.

Heuristic Search

- Heuristics improve the efficiency of search process by sacrificing completeness.
- Heuristics compromise the requirements of mobility and systematicity to construct a control structure that is not guaranteed to find the best answer but almost always finds a very good answer.
- There are some good *general-purpose* heuristics that are useful in a wide variety of problem domains.
- It is possible to construct *special-purpose* heuristics that exploit domain-specific knowledge to solve a particular problem.
- One example of a good general-purpose heuristic is the *nearest-neighbor heuristic*, which works by selecting the locally superior alternative at each step.

Heuristic Search

Applying nearest-neighbor heuristic to traveling salesman problem, we obtain the following search procedure:

1. Arbitrarily select a starting city.
2. To select the next city, look at all cities not yet visited, and select the one closest to the current city.
3. Repeat step 2 until all cities have been visited.

Heuristic Search

There are two major ways in which domain-specific, heuristic knowledge can be incorporated into a rule-based search procedure:

- In the rule themselves. For example, the rules for a chess-playing system might describe not simply the set of legal moves, but rather the set of "sensible" moves, as determined by the rule writer.
- As a heuristic function that evaluates individual problem states and determines how desirable they are.

Heuristic Function

A *heuristic function* is a function that maps from problem state description to measures of desirability, usually represented as numbers

- The value of heuristic function at a node in the search process gives an estimate of whether that node is on the desired path to a solution.
- The purpose of a heuristic function is to guide the search process in the most profitable direction by suggesting which path to follow first when more than one is available.

Problem	Heuristic Function
Chess	The material advantage of one side over them opponent.
Traveling Salesman	The sum of distances so far
Tic-Tac-Toe	1 for each row in which we could win and in which we already have one piece plus 2 for each such row in which we have two pieces.

Problem Characteristics

In order to choose the most appropriate heuristic method for a particular problem, it is necessary to analyze the problem along several key dimensions:

- Is the problem decomposable into a set of independent smaller or easier subproblems ?
- Can solution steps be ignored or undone if they prove unwise ?
- Is the problem's universe predictable ?
- Is a good solution to the problem obvious without comparison to all other possible solutions ?
- Is the desired solution a state of the world OS a path to a state ?
- Is a large amount of knowledge absolutely required to solve the problem, or is knowledge important only to constrain the search ?
- Can a program that is simply given the problem return the solution, or it will require interaction between the program and a person ?

Is the problem decomposable ?

- The problem of *symbolic integration* is decomposable.

$$\int (x^2 + 3x + sin^2 x + cos^2 x) dx$$

We can solve this problem by breaking it down into three smaller problems, each of which can be solved independently and then combined to get the solution to the original problem.

Is the problem decomposable ?

- The *blocks world problem* is not decomposable.
- Consider a simple instance, with initial configuration $On(C, A)$ and goal configuration $On(B, C)\&\&On(B, A)$. Assume the following operators are available:
  1. $Clear(x) \longrightarrow On(x, Table)$ — When x has nothing on it, pick x and put it on the table.
  2. $Clear(x)\&\&Clear(y) \longrightarrow On(x, y)$ — When x and y have nothing on them, pick x and put it on y.
- We can not solve the two subproblems $On(B, C)$ and $On(B, A)$ separately and combine them together, as they are not independent. They interact and those interactions must be considered in order to arrive at a solution for the entire problem.

Can Solution Steps be Ignored or Undone

Three important class of problems:

- Ignorable: in which solution steps can be ignored and we can always start afresh. e.g., theorem proving.
- Recoverable: in which solution steps can be undone. e.g., 8-puzzle.
- Irrecoverable: in which solution steps can not be undone. e.g., chess-playing.

Is the Universe Predictable

Two important class of problems:

- Certain-outcome: problem solving without feedback from the environment. e.g., 8-puzzle. Such problems can be solved by planning where a sequence of operators is generated that is guaranteed to lead to a solution.
- Uncertain-outcome: problem solving with feedback from the environment. e.g., bridge-playing. Such problems can be solved only through *plan revision* where there is no guarantee of an actual solution.

Is a Good Solution Absolute or Relative

Two important class of problems:

- Any-path problem: 8-puzzle problem.
- Best-path problem: Traveling salesman problem.

Is the Solution a State or a Path

- In Natural language understanding, problem of finding a consistent interpretation for a sentence has a solution that can be seen as state. In order to solve the problem, we need to produce only the interpretation and not the record of processing by which the interpretation was found.
- In the water jug problem, we know the goal state $(2, 0)$, what we are expected to find is a path to the goal state, a sequence of operations that produce the state $(2, 0)$ from any initial state.

What is the Role of Knowledge

- To constrain the search for a solution: Chess-Playing—-Knowledge about legal moves, control mechanism to implement search strategy, good search strategy and tactics to constrain and speed-up the search.
- To recognize a solution: Classifying newspapers as Democrat or Republican supporter—lot of knowledge is required even to be able to recognize a solution.

Does the Task require interaction with a person

Two types of problems:

- Solitary: in which the computer is given a problem description and produces an answer with no intermediate communication and no explanation for the reasoning process.
- Conversational: in which there is intermediate communication between a person and the computer, either to provide additional assistance to computer or to provide additional information to the user, or both.

Problem System Characteristics

Production systems are a good way to describe the operations that can be performed in a search.

- Can production systems be described by a set of characteristics ?
- Is there a correspondence between problem types and types of production systems ?

	Monotonic	Non-monotonic
Partially commutative	Theorem Proving	Robot navigation
Not Partially commutative	Chemical Synthesis	Bridge

Classes of production systems

- Monotonic production system: A production system in which the application of a rule never prevents the later application of another rule that could also have been applied at the time the first rule was selected.
- Partially commutative system: A production system with the property that if the application of a particular sequence of rules transforms state x into state y, then any permutation of those rules is allowable.
- A commutative production system is both monotonic and partially commutative.

Issues in the Design of Search Programs

- The direction in which to conduct the search: *forward* versus *backward* reasoning.
- How to select applicable rules: *matching*.
- How to represent each node of the search process: *knowledge representation problem* and *frame problem*.
- Whether to keep track of all search nodes: *tree search* versus *graph search*.

Thank You