11.3 Reasoning

Let us now use the puzzle-solving machine introduced in Section 11.1 to explore techniques for developing agents with elementary reasoning abilities.

Production Systems

Once our puzzle-solving machine has deciphered the positions of the tiles from the visual image, its task becomes that of figuring out what moves are required to solve the puzzle. An approach to this problem that might come to mind is to preprogram the machine with solutions to all possible arrangements of the tiles. Then the machine's task would merely be to select and execute the proper program. However, the eight-puzzle has over 100,000 configurations, so the idea of providing an explicit solution for each is not inviting. Thus, our goal is to program the machine so that it can construct solutions to the eight-puzzle on its own. That is, the machine must be programmed to perform elementary reasoning activities.

The development of reasoning abilities within a machine has been a topic of research for many years. One of the results of this research is the recognition that there is a large class of reasoning problems with common characteristics. These common characteristics are isolated in an abstract entity known as a **production system**, which consists of three main components:

1. A *collection of states*. Each **state** is a situation that might occur in the application environment. The beginning state is called the **start** (or initial) **state**; the desired state (or states) is called the **goal state**. (In our case, a state is a configuration of the eight-puzzle; the start state is the configuration of the puzzle when it is handed to the machine; the goal state is the configuration of the solved puzzle as shown in Figure 11.1.)
2. A *collection of productions* (*rules or moves*). A **production** is an operation that can be performed in the application environment to move from one state to another. Each production might be associated with preconditions; that is, conditions might exist that must be present in the environment before a production can be applied. (Productions in our case are the movements of tiles. Each movement of a tile has the precondition that the vacancy must be next to the tile in question.)
3. A *control system*. The **control system** consists of the logic that solves the problem of moving from the start state to the goal state. At each step in the process, the control system must decide which of those productions whose preconditions are satisfied should be applied next. (Given a particular state in our eight-puzzle example, there would be several tiles next to the vacancy and therefore several applicable productions. The control system must decide which tile to move.)

Note that the task assigned to our puzzle-solving machine can be formulated in the context of a production system. In this setting, the control system takes the form of a program. This program inspects the current state of the eight-puzzle, identifies a sequence of productions that leads to the goal state, and executes this sequence. It is therefore our task to design a control system for solving the eight-puzzle.

An important concept in the development of a control system is that of a **problem space**, which is the collection of all the states, productions, and preconditions in a production system. A problem space is often conceptualized in the form of a **state graph**. Here the term graph refers to a structure that mathematicians would call a **directed graph**, meaning a collection of locations called **nodes** connected by arrows. A state graph consists of a collection of nodes representing the states in the system connected by arrows representing the productions that shift the system from one state to another. Two nodes are connected by an arrow in the state graph if and only if there is a production that transforms the system from the state at the origin of the arrow to the state at the destination of the arrow.

We should emphasize that just as the number of possible states prevented us from explicitly providing preprogrammed solutions to the eight-puzzle, the problem of magnitude prevents us from explicitly representing the entire state graph. A state graph is therefore a way of conceptualizing the problem at hand but not something that we would consider drawing in its entirety. Nonetheless, you might find it helpful to consider (and possibly extend) the portion of the state graph for the eight-puzzle displayed in Figure 11.4.

When viewed in terms of the state graph, the problem faced by the control system becomes that of finding a sequence of arrows that leads from the start state to the goal state. After all, this sequence of arrows represents a sequence of productions that solves the original problem. Thus, regardless of the application, the task of the control system can be viewed as that of finding a path through a state graph. This universal view of control systems is the prize that we obtain by analyzing problems requiring reasoning in terms of production systems. If a problem can be characterized in terms of a production system, then its solution can be formulated in terms of searching for a path.

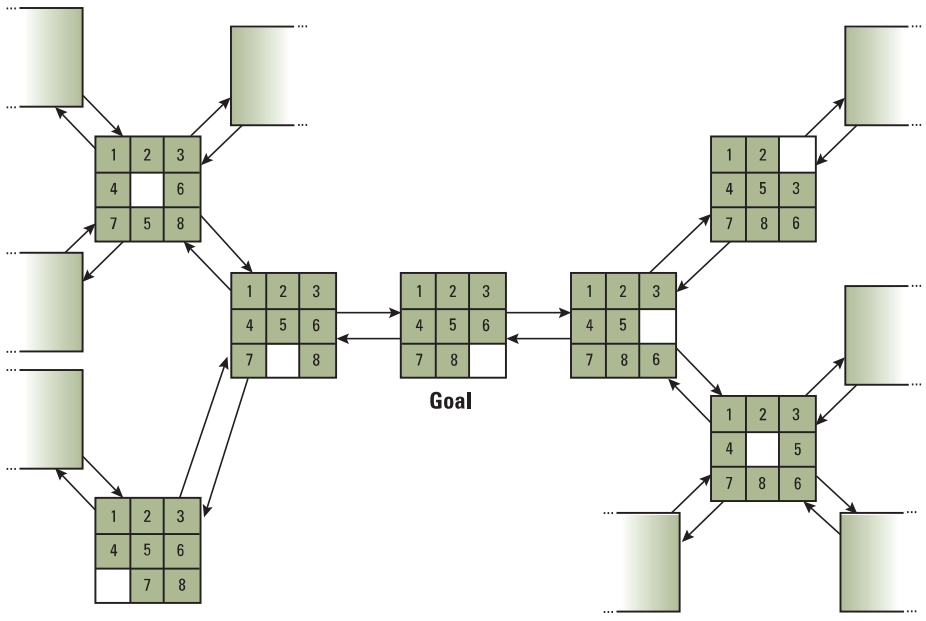


Figure 11.4 A small portion of the eight-puzzle's state graph

To emphasize this point, let us consider how other tasks can be framed in terms of production systems and thus performed in the context of control systems finding paths through state graphs. One of the classic problems in artificial intelligence is playing games such as chess. These games involve moderate complexity in a well-defined context and hence provide an ideal environment for testing theories. In chess, the states of the underlying production system are the possible board configurations, the productions are the moves of the pieces, and the control system is embodied in the players (human or otherwise). The start node of the state graph represents the board with the pieces in their initial positions. Branching from this node are arrows leading to those board configurations that can be reached after the first move in a game; branching from each of those configurations are arrows to those configurations that are reachable by the next move; and so on. With this formulation, we can imagine a game of chess as consisting of two players, each trying to find a path through a large state graph to a goal node of his or her own choosing.

Perhaps a less obvious example of a production system is the problem of drawing logical conclusions from given facts. The productions in this context are the rules of logic, called **inference rules**, that allow new statements to be formed from old ones. For example, the statements "All super heroes are noble" and "Superman is a super hero" can be combined to produce "Superman is noble." States in such a system consist of collections of statements known to be true at particular points in the deduction process: The start state is the collection of basic statements (often called axioms) from which conclusions are to be drawn, and a goal state is any collection of statements that contain the proposed conclusion.

As an example, Figure 11.5 shows the portion of a state graph that might be traversed when the conclusion "Socrates is mortal" is drawn from the collection of statements "Socrates is a man," "All men are humans," and "All humans are mortal."

There we see the body of knowledge shifting from one state to another as the reasoning process applies appropriate productions to generate additional statements. Today, such reasoning systems, often implemented in logic programming languages (Section 6.7), are the backbone of most **expert systems**, which are software packages designed to simulate the cause-and-effect reasoning that human experts would follow if confronted with the same situations. Medical expert systems, for example, are used to assist in diagnosing ailments or developing treatments.

Search Trees

We have seen that, in the context of a production system, a control system's job involves searching the state graph to find a path from the start node to a goal. A simple method of performing this search is to traverse each of the arrows leading from the start state, and in each case, record the destination

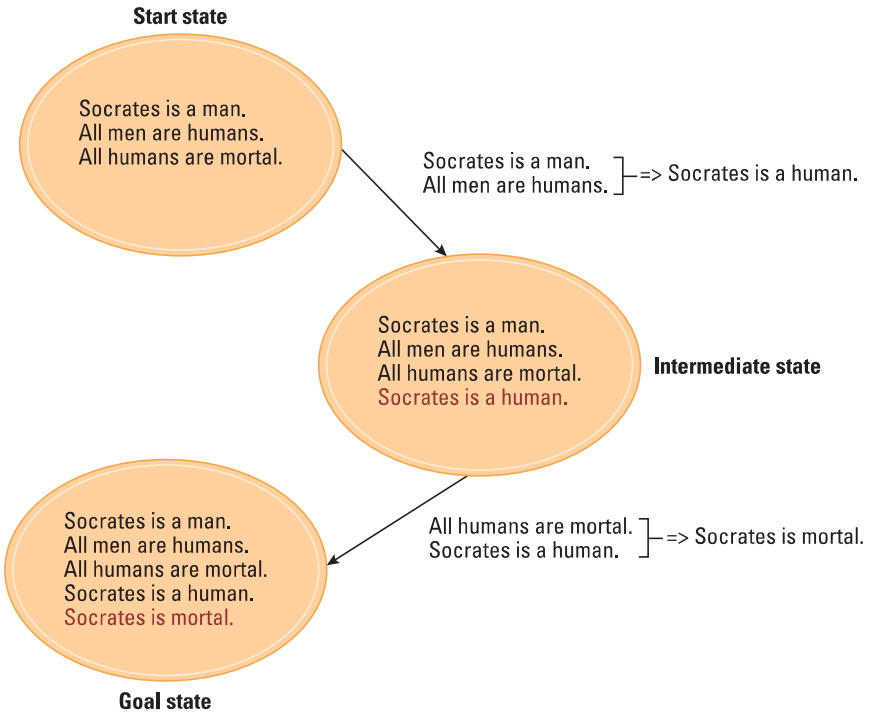


Figure 11.5 Deductive reasoning in the context of a production system

state, then traverse the arrows leaving these new states and again record the results, and so on. The search for a goal spreads out from the start state like a drop of dye in water. This process continues until one of the new states is a goal, at which point a solution has been found, and the control system needs merely to apply the productions along the discovered path from the start state to the goal.

The effect of this strategy is to build a tree, called a **search tree**, that consists of the part of the state graph that has been investigated by the control system. The root node of the search tree is the start state, and the children of each node are those states reachable from the parent by applying one production. Each arc between nodes in a search tree represents the application of a single production, and each path from the root to a leaf represents a path between the corresponding states in the state graph.

The search tree that would be produced in solving the eight-puzzle from the configuration shown in Figure 11.6 is illustrated in Figure 11.7. The leftmost branch of this tree represents an attempt to solve the problem by first moving the 6 tile up, the center branch represents the approach of moving the 2 tile to the right, and the rightmost branch represents moving the 5 tile down. Furthermore, the search tree shows that if we do begin by moving the 6 tile up, then the only production allowable next is to move the 8 tile to the right. (Actually, at that point, we could also move the 6 tile down but that would merely reverse the previous production and thus be an extraneous move.)

Figure 11.6 An unsolved eight-puzzle

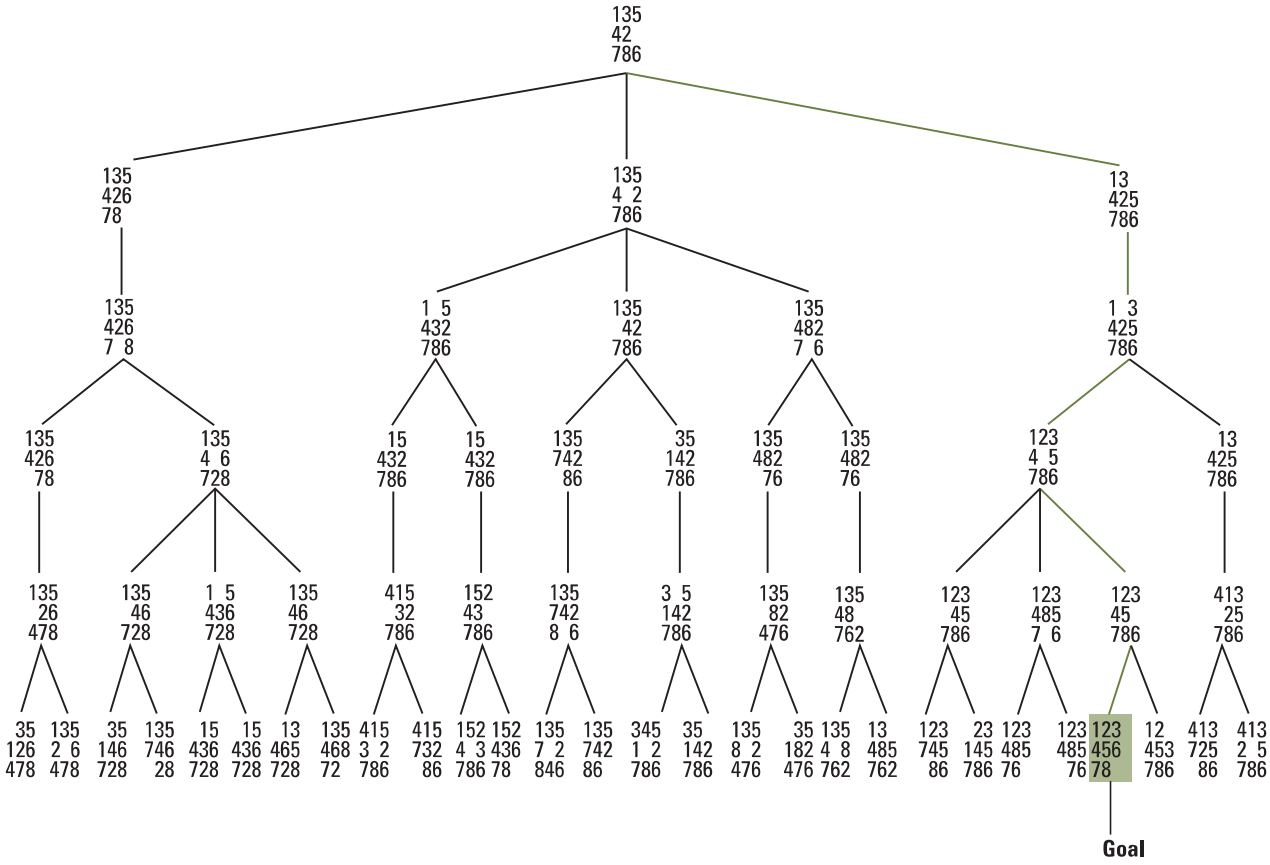


Figure 11.7 A sample search tree

The goal state occurs in the last level of the search tree of Figure 11.7. Since this indicates that a solution has been found, the control system can terminate its search procedure and begin constructing the instruction sequence that will be used to solve the puzzle in the external environment. This turns out to be the simple process of walking up the search tree from the location of the goal node while pushing the productions represented by the tree arcs on a stack as they are encountered. Applying this technique to the search tree in Figure 11.7 produces the stack of productions in Figure 11.8. The control system can now solve the puzzle in the outside world by executing the instructions as they are popped from this stack.

There is one more observation that we should make. Recall that the trees we discussed in Chapter 8 use a pointer system that points *down* the tree, thereby allowing us to move from a parent node to its children. In the case of

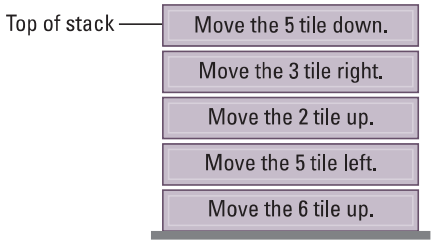


Figure 11.8 Productions stacked for later execution

a search tree, however, the control system must be able to move from a child to its parent as it moves *up* the tree from the goal state to the start state. Such trees are constructed with their pointer systems pointing up rather than down. That is, each child node contains a pointer to its parent rather than the parent nodes containing pointers to their children. (In some applications, both sets of pointers are used to allow movement within the tree in both directions).

Heuristics

For our example in Figure 11.7, we chose a starting configuration that produces a manageable search tree. In contrast, the search tree generated in an attempt to solve a more complex problem could grow much larger. In a game of chess, there are twenty possible first moves so the root node of the search tree in such a case would have twenty children rather than the three in the case of our example. Moreover, a game of chess can easily consist of thirty to thirty-five pairs of moves. Even in the case of the eight-puzzle, the search tree can become quite large if the goal is not quickly reached. As a result, developing a full search tree can become as impractical as representing the entire state graph.

One strategy for countering this problem is to change the order in which the search tree is constructed. Rather than building it in a **breadth-first** manner (meaning that the tree is constructed layer by layer), we can pursue the more promising paths to greater depths and consider the other options only if these original choices turn out to be false leads. This results in a **depth-first** construction of the search tree, meaning that the tree is constructed by building vertical paths rather than horizontal layers. More precisely, this approach is often called a **best-first** construction in recognition of the fact that the vertical path chosen for pursuit is the one that appears to offer the best potential.

Essential Knowledge Statements

- Some optimization problems such as “find the best” or “find the smallest” cannot be solved in a reasonable time, but approximations to the optimal solution can.

The best-first approach is similar to the strategy that we as humans would apply when faced with the eight-puzzle. We would rarely pursue several options at the same time, as modeled by the breadth-first approach. Instead,

we probably would select the option that appeared most promising and follow it first. Note that we said *appeared* most promising. We rarely know for sure which option is best at a particular point. We merely follow our intuition, which may, of course, lead us astray. Nonetheless, the use of such intuitive information seems to give humans an advantage over the brute-force methods in which each option was given equal attention, and it would therefore seem prudent to apply intuitive methods in automated control systems.

To this end, we need a way of identifying which of several states appears to be the most promising. Our approach is to use a **heuristic**, which in our case is a quantitative value associated with each state that attempts to measure the "distance" from that state to the nearest goal. In a sense, our heuristic is a measure of projected cost. Given a choice between two states, the one with the smaller heuristic value is the one from which a goal can apparently be reached with the least cost. This state, therefore, would represent the direction we should pursue.

A heuristic should have two characteristics. First, it should constitute a reasonable estimate of the amount of work remaining in the solution if the associated state were reached. This means that it can provide meaningful information when selecting among options—the better the estimate provided by the heuristic, the better will be the decisions that are based on the information. Second, the heuristic should be easy to compute. This means that its use has a chance of benefiting the search process rather than of becoming a burden. If computing the heuristic is extremely complicated, then we might as well spend our time conducting a breadth-first search.

A simple heuristic in the case of the eight-puzzle would be to estimate the "distance" to the goal by counting the number of tiles that are out of place—the conjecture being that a state in which four tiles are out of place is farther from the goal (and therefore less appealing) than a state in which only two tiles are out of place. However, this heuristic does not take into account how far

Behavior-Based Intelligence

Early work in artificial intelligence approached the subject in the context of explicitly writing programs to simulate intelligence. However, many argue today that human intelligence is not based on the execution of complex programs but instead on simple stimulus-response functions that have evolved over generations. This theory of "intelligence" is known as *behavior-based intelligence* because "intelligent" stimulus response functions appear to be the result of behaviors that caused certain individuals to survive and reproduce while others did not.

Behavior-based intelligence seems to answer several questions in the artificial intelligence community such as why machines based on the von Neumann architecture easily outperform humans in computational skills but struggle to exhibit common sense. Thus, behavior-based intelligence promises to be a major influence in artificial intelligence research. As described in the text, behavior-based techniques have been applied in the field of artificial neural networks to teach neurons to behave in desired ways, in the field of genetic algorithms to provide an alternative to the more traditional programming process, and in robotics to improve the performance of machines through reactive strategies.

out of position the tiles are. If the two tiles are far from their proper positions, many productions could be required to move them across the puzzle.

A slightly better heuristic, then, is to measure the distance each tile is from its destination and add these values to obtain a single quantity. A tile immediately adjacent to its final destination would be associated with a distance of one, whereas a tile whose corner touches the square of its final destination would be associated with a distance of two (because it must move at least one position vertically and another position horizontally). This heuristic is easy to compute and produces a rough estimate of the number of moves required to transform the puzzle from its current state to the goal. For instance, the heuristic value associated with the configuration in Figure 11.9 is seven (because tiles 2, 5, and 8 are each a distance of one from their final destinations while tiles 3 and 6 are each a distance of two from home). In fact, it actually takes seven moves to return this puzzle configuration to the solved configuration.

Now that we have a heuristic for the eight-puzzle, the next step is to incorporate it into our decision-making process. Recall that a human faced with a decision tends to select the option that appears closest to the goal. Thus, our search procedure should consider the heuristic of each leaf node in the search tree and pursue the search from a leaf node associated with the smallest value. This is the strategy adopted in Figure 11.10, which presents an algorithm for developing a search tree and executing the solution obtained.

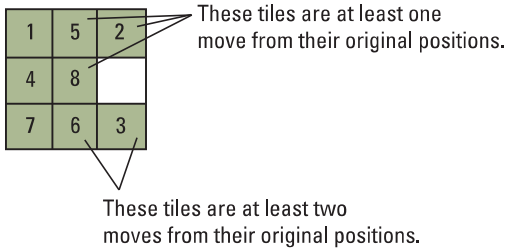


Figure 11.9 An unsolved eight-puzzle

```
Establish the start node of the state graph as the root of the
  search tree and record its heuristic value.
while (the goal node has not been reached):
  Select the leftmost leaf node with the smallest heuristic
    value of all leaf nodes.
  To this selected node attach as children those nodes that
    can be reached by a single production.
  Record the heuristic of each of these new nodes next to the
    node in the search tree.
Traverse the search tree from the goal node up to the root, pushing
  the production associated with each arc traversed onto a stack.
Solve the original problem by executing the productions as they
  are popped off the stack.
```

Figure 11.10 An algorithm for a control system using heuristics

Let us apply this algorithm to the eight-puzzle, starting from the initial configuration in Figure 11.6. First, we establish this initial state as the root node and record its heuristic value, which is five. Then, the first pass through the body of the while statement instructs us to add the three nodes that can be reached from the initial state, as shown in Figure 11.11. Note that we have recorded the heuristic value of each leaf node in parentheses beneath the node.

The goal node has not been reached, so we again pass through the body of the while statement, this time extending our search from the leftmost node ("the leftmost leaf node with the smallest heuristic value"). After this, the search tree has the form displayed in Figure 11.12.

The heuristic value of the leftmost leaf node is now five, indicating that this branch is perhaps not a good choice to pursue after all. The algorithm picks up on this and in the next pass through the while statement instructs us to expand the tree from the rightmost node (which now is the "leftmost leaf node with the smallest heuristic value"). Having been expanded in this fashion, the search tree appears as in Figure 11.13.

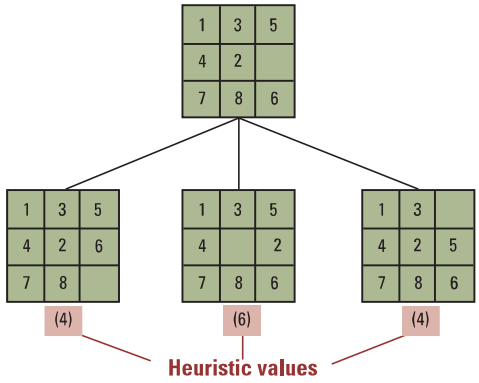


Figure 11.11 The beginnings of our heuristic search

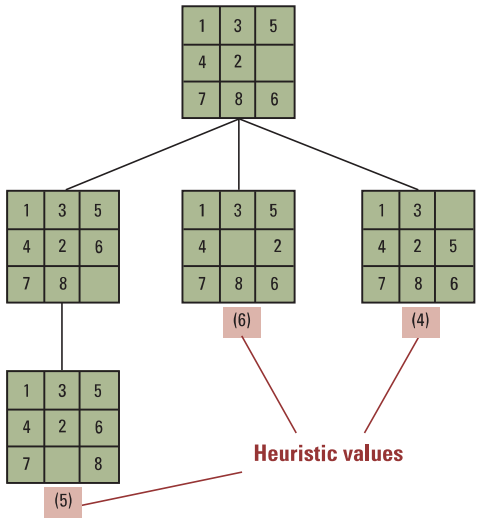


Figure 11.12 The search tree after two passes

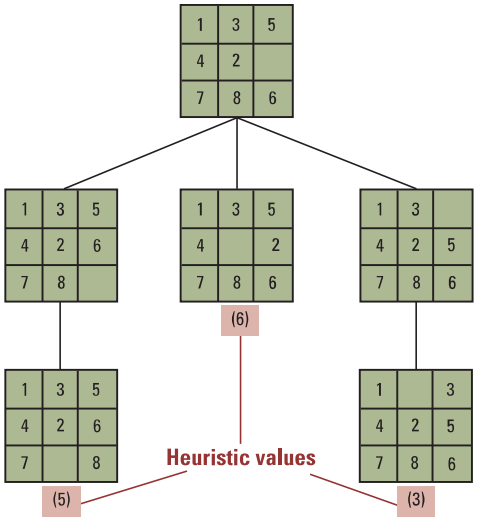


Figure 11.13 The search tree after three passes

At this point, the algorithm seems to be on the right track. Because the heuristic value of this last node is only three, the while statement instructs us to continue pursuing this path, and the search focuses toward the goal, producing the search tree appearing in Figure 11.14. Comparing this with the tree in Figure 11.7 shows that, even with the temporary wrong turn taken early on by the new algorithm, the use of heuristic information has greatly decreased the size of the search tree and produced a much more efficient process.

After reaching the goal state, the while statement terminates, and we move on to traverse the tree from the goal node up to the root, pushing the productions encountered onto a stack as we go. The resultant stack appears as depicted earlier, in Figure 11.8.

Finally, we are instructed to execute these productions as they are popped from the stack. At this point, we would observe the puzzle-solving machine lower its finger and begin to move the tiles.

One final comment regarding heuristic searching is in order. The algorithm we have proposed in this section, which is often called the best-fit algorithm, is not guaranteed to find be the best solution in all applications. For example, when searching for a path to a city using a Global Positioning System (GPS) in an automobile, one would like to find the shortest path rather than just any path. The **A* algorithm** (pronounced "A star algorithm") is a modified version of our best-fit algorithm that finds an optimal solution. The major difference between the two algorithms is that, in addition to a heuristic value, the A* algorithm takes into account the "accumulated cost" incurred to reach each leaf node when selecting the next node to expand. (In the case of an automobile's GPS, this cost is the distance traveled that the GPS obtains from its internal database.) Thus, the A* algorithm bases its decisions on estimates of the cost of complete potential paths rather than merely on projections of remaining costs.

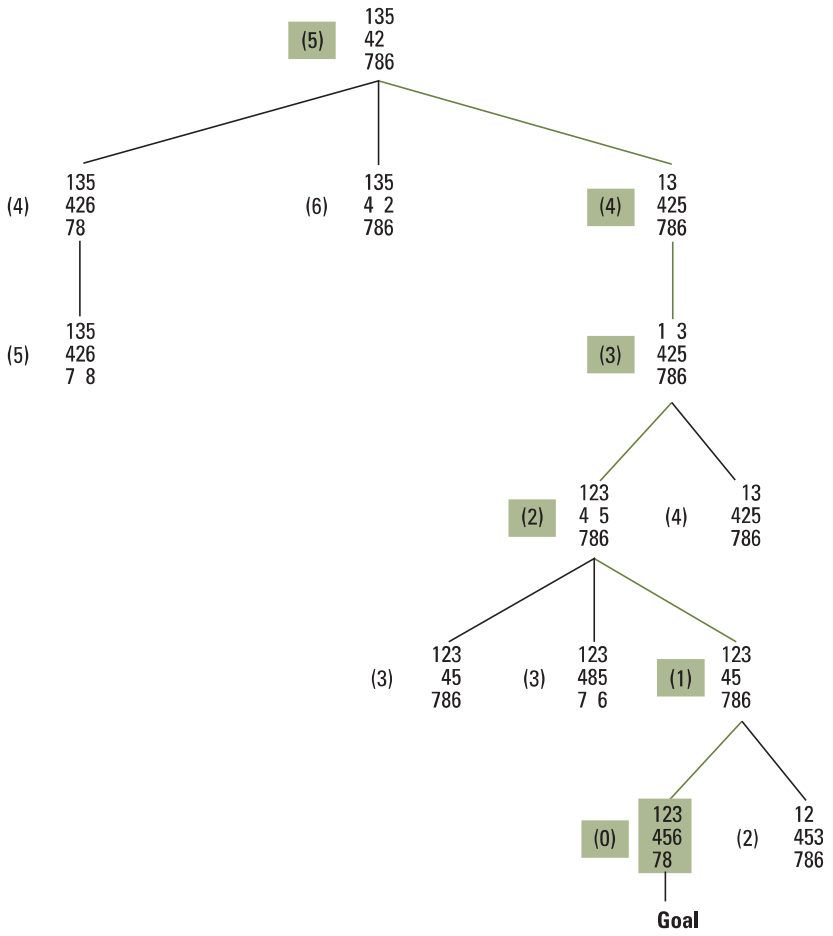


Figure 11.14 The complete search tree formed by our heuristic system

11.3 Questions & Exercises

1. What is the significance of production systems in artificial intelligence?
2. Draw a portion of the state graph for the eight-puzzle surrounding the node representing the following state:

4	1	3
	2	6
7	5	8

3. Using a breadth-first approach, draw the search tree that is constructed by a control system when solving the eight-puzzle from the following start state:

1	2	3
4	8	5
7	6	

4. Use pencil, paper, and the breadth-first approach to try to construct the search tree that is produced in solving the eight-puzzle from the following start state. (You do not have to finish.) What problems do you encounter?

4	3	
2	1	8
7	6	5

5. What analogy can be drawn between our heuristic system for solving the eight-puzzle and a mountain climber who attempts to reach the peak by considering only the local terrain and always proceeding in the direction of steepest ascent?

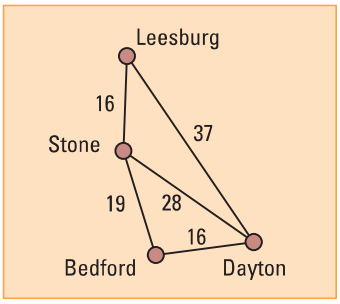
6. Using the heuristic presented in this section, apply the best-fit algorithm of Figure 11.10 to the problem of solving the following eight-puzzle:

1	2	3
4		8
7	6	5

7. Refine our method of computing the heuristic value for a state of the eight-puzzle so that the search algorithm of Figure 11.10 does not make the wrong choice, as it did in the example in this section. Can you find an example in which your heuristic still causes the search to go astray?

8. Draw the search tree produced by the best-fit algorithm (Figure 11.10) in finding the route from Leesburg to Bedford.

Each node in the search tree will be a city on the map. Begin with a node for Leesburg. When expanding a node, add only the cities that are directly connected to the city being expanded. Record in each node the straight-line distance to Bedford and use this as the heuristic value. What is the solution found by the best-fit algorithm? Is the found solution the shortest route?



Straight-line distance to Bedford from

Dayton	16
Leesburg	34
Stone	19

9. The A* algorithm modifies the best-fist algorithm in two significant ways. First, it records the actual cost to reach a state. In the case of a route on a map, the actual cost is the distance traveled. Second, when selecting a node to expand, it chooses the node whose sum of the actual cost plus heuristic value is the smallest. Draw the search tree of Question 8 that would result from these two modifications. Record in each node the distance traveled to the city, the heuristic value to reach the goal, and their sum. What is the found solution? Is the found solution the shortest route?

11.4 Additional Areas of Research

In this section, we explore issues of handling knowledge, learning, and dealing with very complex problems, which continue to challenge researchers in the field of artificial intelligence. These activities involve capabilities that appear to be easy for human minds but apparently tax the capabilities of machines. For now, much of the progress in developing "intelligent" agents has been achieved essentially by avoiding direct confrontation with these issues—perhaps by applying clever shortcuts or limiting the scope in which a problem arises.

Representing and Manipulating Knowledge

In our discussion of perception, we saw that understanding images requires a significant amount of knowledge about the items in the image and that the meaning of a sentence might depend on its context. These are examples