2.4. HEURISTIC GRAPH-SEARCH PROCEDURES

The uninformed search methods, whether breadth-first or depth-first, are exhaustive methods for finding paths to a goal node. In principle, these methods provide a solution to the path-finding problem, but they are often infeasible to use to control AI production systems because the search expands too many nodes before a path is found. Since there are always practical limits on the amount of time and storage available to expend on the search, more efficient alternatives to uninformed search must be found.

For many tasks it is possible to use task-dependent information to help reduce search. Information of this sort is usually called *heuristic information*, and search procedures using it are called *heuristic search methods*. It is often possible to specify heuristics that reduce search effort (below that expended by, say, breadth-first search) without sacrificing the guarantee of finding a minimal length path. Some heuristics greatly reduce search effort but do not guarantee finding minimal cost paths. In most practical problems, we are interested in minimizing some *combination* of the cost of the path and the cost of the search required to obtain the path. Furthermore, we are usually interested in search methods that minimize this combination *averaged* over all problems likely to be encountered. If the averaged combination cost of search method 1 is lower than the averaged combination cost of search method 2, then search method 1 is said to have more *heuristic power* than search method 2. Note that according to our definition, it is not necessary (though it is a common misconception) that a search method with more heuristic power give up any guarantee for finding a minimal cost path.

Averaged combination costs are never actually computed, both because it is difficult to decide on the way to combine path cost and search effort cost and because it would be difficult to define a probability distribution over the set of problems to be encountered. Therefore, the matter of deciding whether one search method has more heuristic power than another is usually left to informed intuition, gained from actual experience with the methods.

2.4.1. USE OF EVALUATION FUNCTIONS

Heuristic information can be used to order the nodes on *OPEN* in step 8 of **GRAPHSEARCH** so that search expands along those sectors of the frontier thought to be most promising. In order to apply such an ordering procedure, we need a method for computing the "promise" of a node. One important method uses a real-valued function over the nodes called an *evaluation function*. Evaluation functions have been based on a variety of ideas: Attempts have been made to define the probability that a node is on the best path; distance or difference metrics between an arbitrary node and the goal set have been suggested; or in board games or puzzles, a configuration is often scored points on the basis of those features that it possesses that are thought to be related to its promise as a step toward the goal.

Suppose we denote the evaluation function by the symbol f. Then $f(n)$ gives the value of the function at node n. For the moment we let f be any arbitrary function; later, we propose that it be an estimate of the cost of a minimal cost path from the start node to a goal node constrained to go through node n.

We use the function f to order the nodes on *OPEN* in step 8 of **GRAPHSEARCH**. By convention, the nodes on *OPEN* are ordered in increasing order of their f values. Ties among f values are ordered arbitrarily, but always in favor of goal nodes. Supposedly, a node having a low evaluation is more likely to be on an optimal path.

The way in which **GRAPHSEARCH** uses an evaluation function to order nodes can be illustrated by considering again our 8-puzzle example. We use the simple evaluation function:

$$f(n) = d(n) + W(n)$$

where $d(n)$ is the depth of node n in the search tree and $W(n)$ counts the number of misplaced tiles in that database associated with node n. Thus the start node configuration

```
2 8 3
1 6 4
7   5
```

has an f value equal to $0 + 4 = 4$.

The results of applying **GRAPHSEARCH** to the 8-puzzle using this evaluation function are summarized in Figure 2.8. The value of f for each node is circled; the uncircled numbers show the order in which nodes are