Study Notes for AI (CS 440/520)
Lecture 3: Heuristic Search

Corresponding Book Chapters: 3.5-3.6
Note: These notes provide only a short summary and some highlights of the material covered in the corresponding lecture based on notes collected from students. Make sure you check the corresponding chapters. Please report any mistakes in or any other issues with the notes to the instructor.

1 Informed/Heuristic Search

Often when solving search problems, there is additional information available beyond the successor function and the cost of actions. For example, consider the path finding problem of Figure 1. Assume that the straight-line distance between a city and the goal city, such as Bucharest, is available. Then by promoting the selection of cities closer to Bucharest according to the straight-line distance, we hope to find a solution faster.

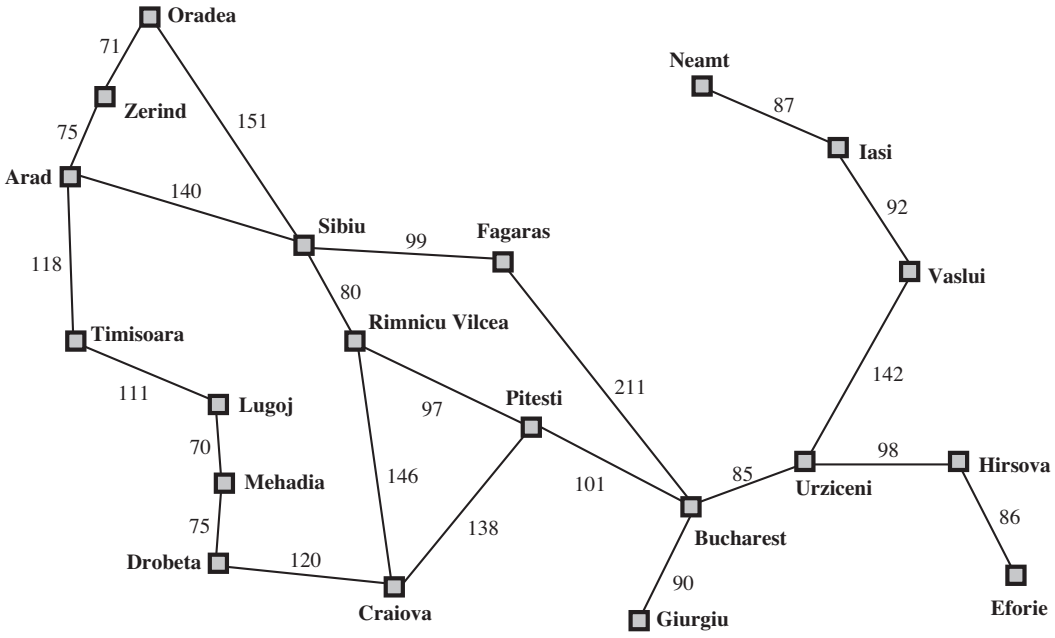

Figure 1: A roadmap for the shortest route problem.

Note that the straight-line distance between two cities does not correspond to the true cost that is involved when we follow the road network of Romania. The cities of Neamt and Fagaras are physically very close one to another but

their distance on the road network is considerably longer. Such a piece of information, like the straight-line distance in the above example, is called heuristic information. A heuristic function $h(n)$ for node n returns an estimate of the cost of the shortest path along the search tree from n to a goal node. Search techniques that attempt to make use of such heuristic information are called informed search techniques.

The basic principle in informed search is that the algorithm attempts to optimize an evaluation function $f(n) : X \to \mathbb{R}$ and is referred to as best-first search (X is the state space and $\mathbb{R}$ is the set of real numbers). Best in the sense that the algorithm does its best given the evaluation function that has available. This involves selecting the top element of a priority queue, similar to the operation of the uniform-cost search. The various informed techniques vary on how they make use of the heuristic $h(n)$ in order to define the evaluation function.

In the following discussion we will make use of the notation $g(n)$ to represent the true path cost from the initial node to node n. Thus:

- $f(n)$: overall value of node n, which is used to select the "best" node in the priority queue.
- $g(n)$: true path cost from the start node to n.
- $h(n)$: heuristic value of node n.

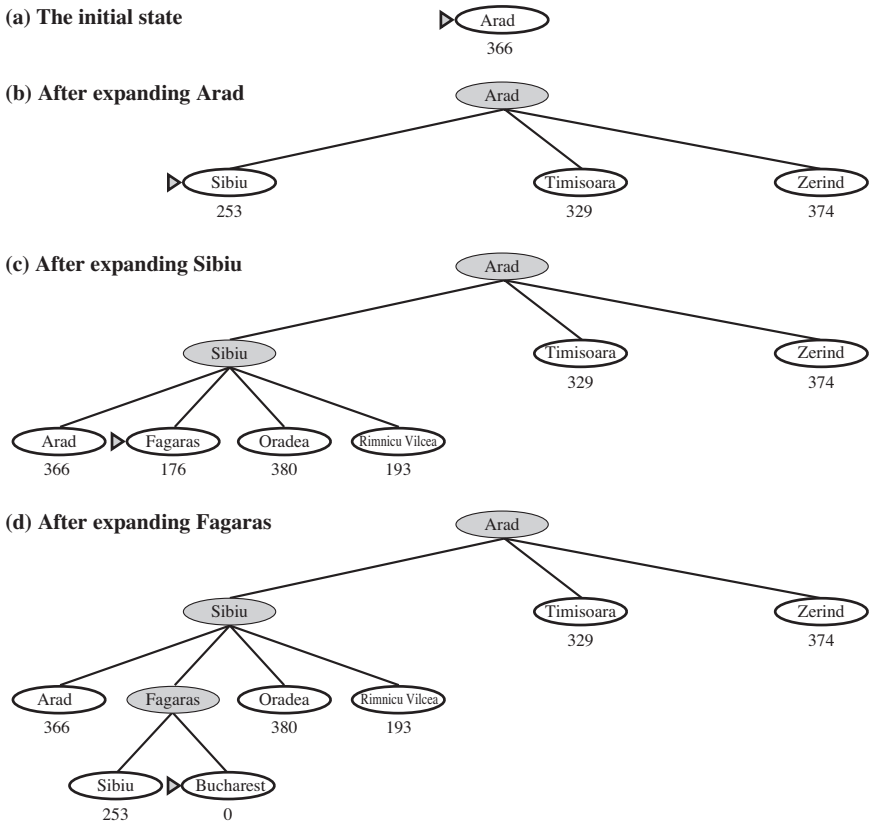

Figure 2: Progress of a Greedy Best-First Search.

1.1 Greedy Best-First Search

The most straightforward way to apply the principle of best-first search is by greedily assigning the evaluation function to the heuristic value:

$$f(n) = h(n)$$

The resulting algorithm is referred to as Greedy Best-First Search (`GBFS`). Figure 2 shows the operation of `GBFS` on a search tree. The estimated distance from each of the fringe nodes to the goal is provided below each node. In part (c) of the figure, Fagaras has the lowest estimated distance, therefore the search will select the Fagaras node to expand next. As mentioned above, however, the straight-line distance is not always a good estimate of the path cost from a node to the goal. Moreover, there is no guarantee that the heuristic estimate will not lead the search procedure to infinite paths.

Criteria:

- Complete? No, there is no guarantee whether the heuristic will lead the search down an infinite path.
- Time: $O(b^m)$, where m is the length of the longest path along the tree. `GBFS` can potentially search the entire tree.
- Space: $O(b^m)$, `GBFS` may potentially have to remember all the leaf nodes at level m.
- Optimal? No, since it is not even complete.

1.2 A*

The A* search algorithm defines the evaluation function differently:

$$f(n) = g(n) + h(n)$$

The evaluation function is the sum of the true path from the initial node to node n and the heuristic value at n. In this way A* selects nodes that optimize the estimate for the complete path cost. Figure 3 shows the operation of A*. In part (b), Sibiu is chosen as the next node to expand because it has the lowest evaluation function. This means that A* predicts that the path from the start to the goal node that goes through Sibiu is the shortest one.

As in the case of uninformed search, in informed search we can distinguish between two alternative ways on how to deal with repeated states. If we allow our search algorithm to revisit repeated states, then the resulting search data structure is a tree and the approach is referred to as `TREE-SEARCH`. Instead, if we use an additional data structure to store the nodes that have been already expanded, i.e., the closed list, perhaps by using a hash table, then the resulting data structure is a graph and the approach is referred to as `GRAPH-SEARCH`. As long as the state space is small enough so as to be feasible to remember all the states that the search technique visits, `GRAPH-SEARCH` is typically the preferred approach. For very large state-spaces and for problems were loops might be beneficial for the solution of the problem `TREE-SEARCH` is used. We will show in the following discussion the properties of A* for each of these two cases.

1.2.1 `TREE-SEARCH`

For A* to be optimal in the case of `TREE-SEARCH`, we must use an admissible (optimistic) heuristic.

Definition An admissible heuristic $h(n)$ always returns a value smaller than the true path cost $C^*(n)$:

$$h(n) \leq C^*(n), \quad \forall n$$

Thus, an admissible heuristic is always optimistic about the distance between the current node and the goal. It is very easy to come up with admissible heuristics, since even assigning a distance of 0 as the heuristic value to all states will be sufficient for the heuristic to be admissible (but we will see later that such a bad distance estimate effects the performance of A* considerably).

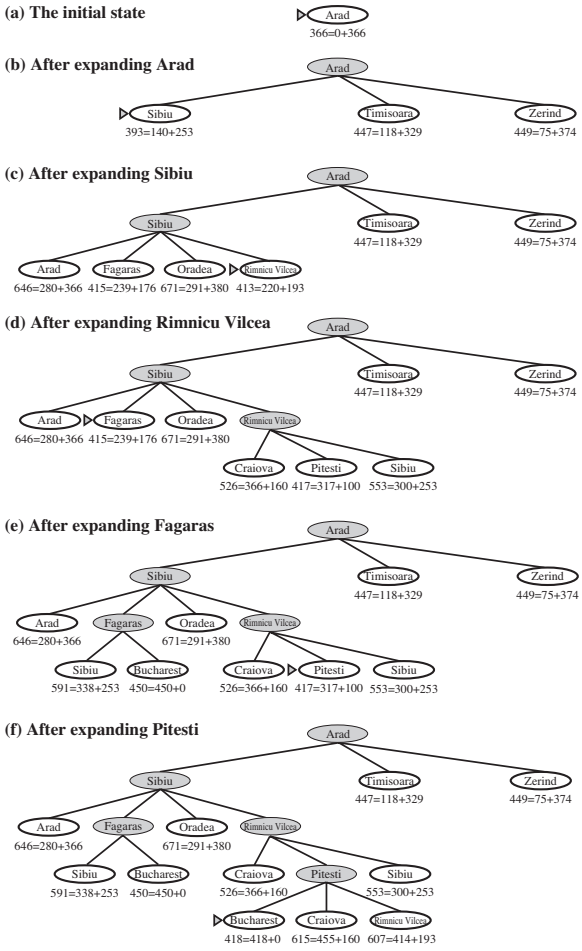

Figure 3: Progress of A*.

Lemma 1.1. If G_2 is a node that corresponds to a goal state but a suboptimal path, A* will select a reachable node n along the optimal path before selecting G_2 for expansion.

Proof. Note that a reachable node n along the optimal path always exists in the set of fringe nodes, since at least the root of the tree and one of its successors trivially belong to the optimal path.

Assume that the true optimal path from the root of the tree to the goal is C^*. Then in order to show that A* selects first a node n along the optimal path before selecting G_2 for expansion it is sufficient to show that there is a node n along the optimal path so that:

$$f(n) < f(G_2)$$

Notice however the following:

$$f(G_2) = g(G_2) + h(G_2) = g(G_2) + 0 > C^* \tag{1}$$

The heuristic value of G_2 is zero because G_2 is a goal node ($h(G_2) = 0$). Furthermore, the path cost from the

root $g(G_2)$ has to be greater than the optimum cost to the goal C^* because G_2 is a goal node that is connected to the root with a suboptimal path ($g(G_2) = 0$).

On the other hand:

$$f(n) = g(n) + h(n) \leq C^* \tag{2}$$

The above equation is due to the fact that $h(n)$ is optimistic and underestimates the distance of n from the goal node.

Given Equations 1 and 2, we can conclude that:

$$f(n) \leq C^* < f(G_2)$$

which proves that n will be selected before the suboptimal goal node G_2. □

Theorem 1.2. A* always finds the optimal path in `TREE-SEARCH` if the heuristic is admissible.

The proof of the theorem is based on the above lemma. As long as there are reachable nodes along the optimal path, A* with an admissible heuristic will always prefer to expand them before expanding a suboptimal goal node. Consequently, the first goal node reached by A* is the goal node along the optimal path.

1.2.2 `GRAPH-SEARCH`

Now we will focus on the case where we avoid repeated states. In `GRAPH-SEARCH`, an admissible heuristic is no longer sufficient for A* to be optimal. Since we avoid repeated states, after we have already generated a goal node, even if it corresponds to a suboptimal path, subsequent goal nodes will not be generated because they correspond to a repeated state. One way to alleviate this problem is to check whether the new path to that node is better than the path already stored but in general this approach requires a lot of book-keeping. We will show in the following discussion that given a new requirement for the heuristic function, the first goal node to be generated is the optimum goal node.

Definition A heuristic is consistent as long as:

$$h(n) <= c(n, \alpha, n') + h(n'),\ \forall n, n' \ \text{ so that: } \exists\ \alpha \text{ and } succ(n, \alpha) = n' \tag{3}$$

In the above definition:

- $h(n)$ is the heuristic of the parent node n
- $h(n')$ is the heuristic of the successor node n'
- $c(n, \alpha, n')$ is the path cost to move from n to n' when applying action α

Figure 4 illustrates a geometric representation of the consistency property and shows the similarity between consistency and the triangular inequality. A consistent heuristic is a more strict requirement than admissibility. Nevertheless, it is generally difficult to come up with an admissible heuristic that is not also consistent.

Theorem 1.3. A consistent heuristic is also an admissible heuristic.

Proof. To prove the theorem's statement we must show that given Equation 3 holds for all states, then $h(n) \leq C^*(n),\ \forall n$, where $C^*(n)$ is the true optimal cost from node n to the goal. We will prove this property by induction.

Base Case: For a goal state n_g, we have that trivially: $h(n_g) \leq C^*(n_g)$, since $h(n_g) = 0$ and $C^*(n_g) = 0$. We will now show that the property holds for nodes that are parents of goal nodes. We will denote such nodes as n_1 and their property is that $\exists \alpha_1$ so that: $succ(n_1, \alpha_1) = n_g$. Then if $c(n_1, \alpha_1, n_g)$ is the cost of the action α_1, the following statement holds:

$$C^*(n_1) = c(n_1, \alpha_1, n_g)$$

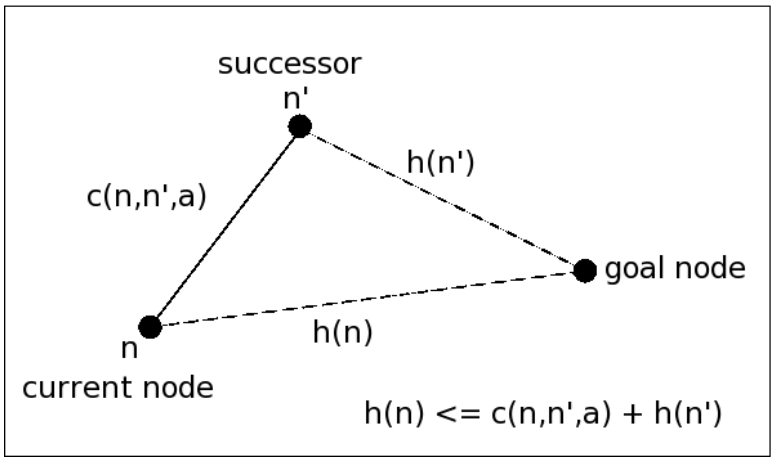

Figure 4: Triangular Inequality.

Furthermore, due to the fact that the heuristic is consistent (Equation 3):

$$h(n_1) \leq c(n_1, \alpha_1, n_g) + h(n_g) = c(n_1, \alpha_1, n_g)$$

since $h(n_g) = 0$, as n_g is a goal node. Therefore:

$$h(n_1) \leq c(n_1, \alpha_1, n_g) = C^*(n_1).$$

Thus the desired property holds for the parents of goal nodes as well.

Inductive Step: For the inductive step we assume that for a node n_{i-1} it holds that $h(n_{i-1}) \leq C^*(n_{i-1})$. We then want to show that for the parent node n_i $[succ(n_i, \alpha_i) = n_{i-1})]$ that the property also holds: $h(n_i) \leq C^*(n_i)$. Notice then that the:

$$C^*(n_i) = c(n_i, \alpha_i, n_{i-1}) + C^*(n_{i-1}). \tag{4}$$

Because the heuristic is consistent (Equation **??**):

$$h(n_i) \leq c(n_i, \alpha_i, n_{i-1}) + h(n_{i-1}) \Rightarrow \text{ (by the inductive assumption: } h(n_{i-1}) \leq C^*(n_{i-1}) \text{)}$$

$$h(n_i) \leq c(n_i, \alpha_i, n_{i-1}) + C^*(n_{i-1}) \tag{5}$$

Therefore from Equations 4 and 5: $h(n_i) \leq C^*(n_i)$, and the heuristic is admissible. □

Lemma 1.4. If $h(n)$: consistent, then $f(n)$ is non-decreasing along any path

Proof. Assume that n' is a successor node of n. Then:

$$g(n') = g(n) + c(n, \alpha, n')$$

And consequently we have that:

$$f(n') = g(n') + h(n') = g(n) + c(n, \alpha, n') + h(n')$$

Since $c(n, \alpha, n') + h(n') \geq h(n)$ (consistent heuristic), we have that:

$$f(n') \geq g(n) + h(n) = f(n),$$

which means that the evaluation function always increases as we traverse a path of the tree towards the goal. □

Corollary 1.5. A* expands nodes with a non-decreasing order of $f(n)$

The above corollary is based on the previous lemma. At each iteration of A*, the algorithm will pick the node n that minimizes $f(n)$. All the remaining nodes on the tree will have greater values that at least one node in the current fringe and they will be selected after that node for expansion. Consequently, A* always expands nodes in a non-decreasing order of $f(n)$.

Proposition 1.6. The first node that corresponds to the goal state also corresponds to the optimal path.

The above proposition is now obvious. Since A* expands nodes in non-decreasing order of the evaluation function, the first goal that will be expanded is the goal node with the minimum evaluation function, and thus the one that corresponds to the optimal path. Suboptimal goal nodes will be at least as expensive as the optimal one in terms of their evaluation function.

Corollary 1.7. A* expands all nodes with $f_n < C^*$.

At some point in the operation of A* the (optimal) goal node will be expanded. The evaluation function of the optimal goal node is $f(n_g) = C^*(n_g) = C^*$. Since A* expands nodes in non-decreasing order, then all the nodes with $f(n) \leq f(n_g) = C^*$ will be expanded.

A consequence of the above corollary is that the closer the heuristic is to C^* the fewer nodes will be expanded, making the search much more efficient. Consequently, although it is very easy to come up with optimistic heuristics, overoptimistic estimates will degrade the performance of A*, because the algorithm will be forced to expand a larger number of nodes before reaching the goal node. Different ways to design heuristics will be discussed later.

Corollary 1.8. A* is complete if the number of nodes for which $f(n) < C^*$ is finite.

The number of nodes for which $f(n) < C^*$ is finite if the branching factor is finite and if the minimum edge cost is greater than $\epsilon > 0$. The above corollary is true because if we try to search an infinite amount of nodes, we will never find the solution because there are still more nodes to expand.

Corollary 1.9. For the same heuristic (and ignoring ties) there is no other optimal algorithm that will expand fewer nodes than A*.

Every optimal algorithm must have the property that at least all the nodes with $f(n) < C^*$ are expanded. A* expands exactly this set.

2 Designing Heuristics

For some problems it is straightforward to come up with a good heuristic function. For example for the path finding problem in Figure 5 we can select the straight-line distance between two cities as the heuristic. The straight-line distance has the property that it is both optimistic and consistent.

Nevertheless, for many other problems the procedure to come up with a good heuristic is much more involved. For example, let's focus on the 8-puzzle problem in Figure 5. Two possible ideas for a heuristic are the following:

h_1: the number of tiles that has to be moved so that the current state looks like the final state,

h_2: the sum of the Manhattan distances for all tiles between the current state and the final one.

For the start state shown in Figure 5 the value of the two heuristics is: $h_1 = 8$ and $h_2 = 18$. The true number of steps that it takes to reach the goal from this state is 26. This is an indication that the two heuristics are admissible. It turns out that both heuristics are truly admissible and consistent. However, how can we evaluate which heuristic is the most appropriate to solve the problem with A* and how can we come up with heuristics in an automated way. The following discussion focuses on these two questions.

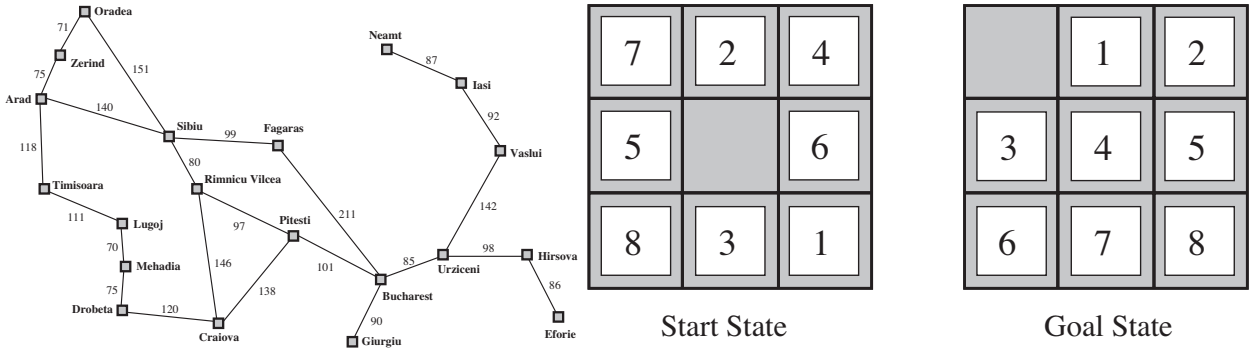

Figure 5: The path finding problem on the road network of Romania and the 8-puzzle.

2.1 Evaluating heuristics

Overall, we prefer heuristics that will make the A* algorithm to expand as few nodes as possible (thus making the search procedure faster).

One way to evaluate the number of nodes expanded by a search technique is the *effective branching factor* b^*. This factor is computed as follows: If an algorithm has visited N nodes, and the solution was at depth d, then b^* is the branching factor of the complete tree of length d with N nodes. Notice that the relationship between the three parameters is the following:

$$N + 1 = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d$$

e.g., if $d = 5$ and $N = 52$ then the branching factor is 1.92. This means that the algorithm on average branched out 1.92 times out of each node. The optimum and desired performance is for the effective branching factor to be 1, since this means that the algorithm never had to expand any redundant node.

If we experiment with the two possible heuristics for the 8-puzzle, we will discover that the second heuristic h_2 has on average a lower effective branching factor than h_1. This happens because $h_2(n) > h_1(n)$, $\forall n$. When this happens, we say that h_2 dominates h_1. From the properties of A*, we know that the algorithm expands all the nodes with $f(n) < C^*$. For a dominant heuristic the number of such nodes will be smaller than other heuristics.

2.2 Automatic Generation of Heuristics

Notice that the common characteristic between heuristics h_1 and h_2 is that both are optimal solutions to a simpler version of the original 8-puzzle. In particular:

h_1 : it is the cost of the optimal solution to the problem where we move a piece directly to its goal position,

h_2 : it is the cost of the optimal solution to the problem where we can move pieces while ignoring the fact that other pieces are already present in the same position.

In general, optimal solutions to problems with fewer restrictions than the original definition (called relaxed problems) are good candidates for a heuristic function. Such heuristics will be admissible and consistent as optimal solutions to a problem.

2.2.1 Using Subproblems

One way to come up with relaxed versions of the original problem is to use subproblems. In the case of the 8-puzzle, we can define a subproblem by looking only at four of the tiles and how to move those four into the correct position while ignoring the presence of the remaining tiles (Figure 6). The optimal solution found this way, is an admissible heuristic for the original problem.

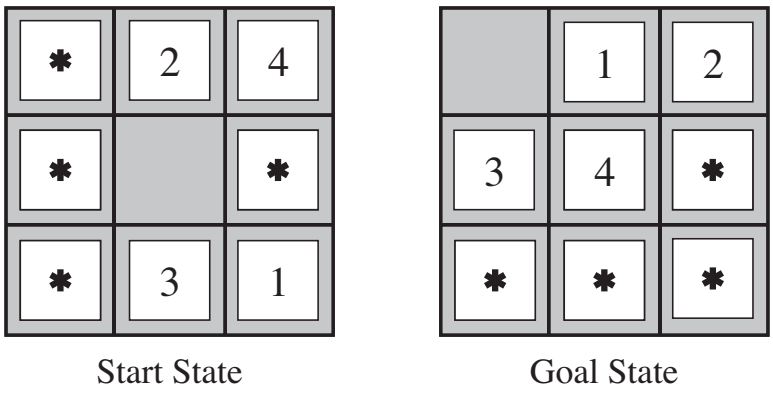

Figure 6: Finding a heuristic for the 8-puzzle.

2.2.2 Combining multiple heuristics

In many cases we can define multiple valid heuristics and the question that arises is which one should we select to use. For example, we can have heuristics for various subproblems of the original problem (e.g., different set of four tiles in the case of the 8-puzzle).

If we have a list of heuristics, $h_1, h_2, ..., h_n$ and there is not one that dominates every other heuristic all of the time, then we can use at each state the heuristic that dominates the others in that specific state. In this way, we are defining a new heuristic $h_{max}(n) = max\{h_1, h_2, ..., h_n\}$, which has the following properties:

- if $h_1, \ldots, h_n$ are admissible, then h_{max} is also admissible
- if $h_1, \ldots, h_n$ are consistent, then h_{max} is also consistent
- h_{max} is a dominant heuristic over this set.

It happens that for the 8-puzzle the best heuristic is to use the maximum over multiple heuristics defined over subproblems (e.g., optimal costs for different sets of four tiles).