heuristic function. If we want to find the shortest solutions, we need a heuristic function that never overestimates the number of steps to the goal. Here are two candidates:

- h_1 = the number of tiles that are in the wrong position. For Figure 4.7, seven of the eight tiles are out of position, so the start state would have $h_1 = 7$. h_1 is an admissible heuristic, because it is clear that any tile that is out of place must be moved at least once.
- h_2 = the sum of the distances of the tiles from their goal positions. Because tiles cannot move along diagonals, the distance we will count is the sum of the horizontal and vertical distances. This is sometimes called the **city block distance** or **Manhattan distance**. h_2 is also admissible, because any move can only move one tile one step closer to the goal. Tiles 1 to 8 in the start state give a Manhattan distance of

MANHATTAN DISTANCE

$$h_2 = 2 + 3 + 3 + 2 + 4 + 2 + 0 + 2 = 18$$

The effect of heuristic accuracy on performance

EFFECTIVE BRANCHING FACTOR

One way to characterize the quality of a heuristic is the **effective branching factor** b^*. If the total number of nodes expanded by A* for a particular problem is N, and the solution depth is d, then b^* is the branching factor that a uniform tree of depth d would have to have in order to contain N nodes. Thus,

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

For example, if A* finds a solution at depth 5 using 52 nodes, then the effective branching factor is 1.91. Usually, the effective branching factor exhibited by a given heuristic is fairly constant over a large range of problem instances, and therefore experimental measurements of b^* on a small set of problems can provide a good guide to the heuristic's overall usefulness. A well-designed heuristic would have a value of b^* close to 1, allowing fairly large problems to be solved. To test the heuristic functions h_1 and h_2, we randomly generated 100 problems each with solution lengths $2, 4, \ldots, 20$, and solved them using A* search with h_1 and h_2, as well as with uninformed iterative deepening search. Figure 4.8 gives the average number of nodes expanded by each strategy, and the effective branching factor. The results show that h_2 is better than h_1, and that uninformed search is much worse.

DOMINATES

One might ask if h_2 is *always* better than h_1. The answer is yes. It is easy to see from the definitions of the two heuristics that for any node n, $h_2(n) \geq h_1(n)$. We say that h_2 **dominates** h_1. Domination translates directly into efficiency: A* using h_2 will expand fewer nodes, on average, than A* using h_1. We can show this using the following simple argument. Recall the observation on page 98 that every node with $f(n) < f^*$ will be expanded. This is the same as saying that every node with $h(n) < f^* - g(n)$ will be expanded. But because h_2 is at least as big as h_1 for all nodes, every node that is expanded by A* search with h_2 will also be expanded with h_1, and h_1 may also cause other nodes to be expanded as well. *Hence, it is always better to use a heuristic function with higher values, as long as it does not overestimate.*