Searching with Partial Information

- Above we (unrealistically) assumed that the environment is fully observable and deterministic
- Moreover, we assumed that the agent knows what the effects of each action are
- Therefore, the agent can calculate exactly which state results from any sequence of actions and always knows which state it is in
- Its percepts provide no new information after each action
- In a more realistic situation the agent’s knowledge of states and actions is incomplete
- If the agent has no sensors at all, then as far as it knows it could be in one of several possible initial states, and each action might therefore lead to one of several possible successor states

An agent without sensors, thus, must reason about sets of states that it might get to, rather than single states
- At each instant the agent has a belief of in which state it might be
- Hence, the action happens in the power set \( \mathcal{P}(\Sigma) \) of the state space \( \Sigma \), which contains \( 2^{\vert \Sigma \vert} \) belief states
- A solution is now a path to that leads to a belief state, all of whose members are goal states
- If the environment is partially observable or if the effects of actions are uncertain, then each new action yields new information
- Every possible contingency that might arise during execution need considering
The cause of uncertainty may be another agent, an adversary. When the states of the environment and actions are uncertain, the agent has to explore its environment to gather information.

In a partially observable world one cannot determine a fixed action sequence in advance, but needs to condition actions on future percepts. As the agent can gather new knowledge through its actions, it is often not useful to plan for each possible situation. Rather, it is better to interleave search and execution.

### 3.5 Informed (Heuristic) Search Strategies

We now consider informed search that uses problem-specific knowledge beyond the definition of the problem itself. This information helps to find solutions more efficiently than an uninformed strategy. The information concerns the regularities of the state space. An evaluation function $f(n)$ determines how promising a node $n$ in the search tree appears to be for the task of reaching the goal. Best-first search chooses to expand the node that appears by evaluation function to be most promising among the candidates. Traditionally, one aims at minimizing the value of function $f$. 
Heuristic Search Strategies

- A key component of an evaluation function is a heuristic function $h(n)$, which estimates the cost of the cheapest path from node $n$ to a goal node.
- In connection of a search problem “heuristics” refers to a certain (but loose) upper or lower bound for the cost of the best solution.
- Goal states are nevertheless identified: in a corresponding node $n$ it is required that $h(n) = 0$.
- E.g., a certain lower bound — bringing no information — would be to set $h(n) \equiv 0$.
- Heuristic functions are the most common form in which additional knowledge is imported to the search algorithm.

3.5.1 Greedy best-first search

- Greedy best-first search tries to expand the node that is closest to the goal, on the grounds that this is likely to lead to a solution quickly.
- Thus, the evaluation function is $f(n) = h(n)$.
- E.g. in minimizing road distances a heuristic lower bound for distances of cities is their straight-line distance.
- Greedy search ignores the cost of the path that has already been traversed to reach $n$.
- Therefore, the solution given is not necessarily optimal.
- If repeating states are not detected, greedy best-first search may oscillate forever between two promising states.
Because greedy best-first search can start down an infinite path and never return to try other possibilities, it is incomplete. Because of its greediness the search makes choices that can lead to a dead end; then one backs up in the search tree to the deepest unexpanded node. Greedy best-first search resembles depth-first search in the way it prefers to follow a single path all the way to the goal, but will back up when it hits a dead end. The worst-case time and space complexity is $O(b^m)$. The quality of the heuristic function determines the practical usability of greedy search.

3.5.2 A* search

- A* combines the value of the heuristic function $h(n)$ and the cost to reach the node $n$, $g(n)$.
- Evaluation function $f(n) = g(n) + h(n)$ thus estimates the cost of the cheapest solution through $n$.
- A* tries the node with the lowest $f(n)$ value first.
- This leads to both complete and optimal search algorithm, provided that $h(n)$ satisfies certain conditions:
  - $h(n)$ is admissible if it never overestimates the cost to reach the goal.
Optimality of A*

- Provided that \( h(n) \) is admissible, then in tree search A* gives the optimal solution
  - Let \( G_2 \) be a suboptimal goal node generated to the tree
  - Let \( C^* \) be the cost of the optimal solution
  - Because \( G_2 \) is a goal node, it holds that \( h(G_2) = 0 \), and we know that \( f(G_2) = g(G_2) > C^* \)
  - On the other hand, if a solution exists, there must exist a node \( n \) that is on the optimal solution path in the tree
  - Because \( h(n) \) does not overestimate the cost of completing the solution path, \( f(n) = g(n) + h(n) \leq C^* \)
  - We have shown that \( f(n) \leq C^* < f(G_2) \), so \( G_2 \) will not be expanded and A* must return an optimal solution

- For example straight-line distance is a heuristic that never overestimates road distance between cities
- In graph search finding an optimal solution requires taking care that the optimal solution is not discarded in repeated states
- A particularly important special case are consistent (or monotonic) heuristics for which the triangle inequality holds in form
  \[
  h(n) \leq c(n,a,n') + h(n'),
  \]
  where \( n' \in S(n) \) (the chosen action is \( a \)) and \( c(n,a,n') \) is the step cost
- Straight-line distance is also a monotonic heuristic
• A* using a consistent heuristic $h(n)$ is optimal also for graph search
• If $h(n)$ is consistent, the values of $f(n)$ along any path are non-decreasing
• Suppose that $n'$ is a successor of $n$ so that $g(n') = g(n) + c(n,a,n')$ for some action $a$

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

• Hence, the first goal node selected for expansion (in graph search) must be an optimal solution

• In looking for a solution, A* expands all nodes $n$ for which $f(n) < C^*$, and some of those for which $f(n) = C^*$
• However, all nodes $n$ for which $f(n) > C^*$ get pruned
• It is clear that A* search is complete

• A* search is also optimally efficient for any given heuristic function, because any algorithm that does not expand all nodes with $f(n) < C^*$ runs the risk of missing the optimal solution
• Despite being complete, optimal, and optimally efficient, A* search also has its weaknesses
• The number of nodes for which $f(n) < C^*$ for most problems is exponential in the length of the solution
3.5.3 Memory-bounded heuristic search

- Once again the main drawback of search is not computation time, but rather space consumption
- Therefore, one has had to develop several memory-bounded variants of A*
- IDA* (Iterative Deepening A*) adapts the idea of iterative deepening
- The cutoff used in this context is the $f$-cost ($g + h$) rather than the depth
- At each iteration the cutoff value is the smallest $f$-cost of any node that exceeded the cutoff on the previous iteration
- Subsequent more modern algorithms carry out more complex pruning

3.6 Heuristic functions

- In 8-puzzle we can define the following heuristic functions, which never overestimate:
  - $h_1$: the number of misplaced tiles: any tile that is out of place must be moved at least once to obtain the desired configuration
  - $h_2$: The sum of Manhattan distances of tiles from their goal position: the tiles need to be transported to their goal positions to reach the desired configuration

- In the initial configuration all tiles are out of their place: $h_1(s_1) = 8$
- The value of the second heuristic for the example is:
  
  \[ 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18 \]
Heuristic $h_2$ gives a stricter estimate than $h_1$: $h_2(n) \geq h_1(n)$

We say that the former dominates the latter.

$A^*$ expands every node for which

$$f(n) < C^* \iff h(n) < C^* - g(n)$$

Hence, a stricter heuristic estimate directly leads to more efficient search.

The branching factor of 8-puzzle is about 3.

Effective branching factor measures the average number of nodes generated by $A^*$ in solving a problem.

For example, when $A^*$ applies heuristic $h_1$, the effective factor in 8-puzzle is on average circa 1.4, and using heuristic $h_2$ circa 1.25.