4 INFORMED SEARCH AND EXPLORATION

- 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 \( f \).

4.1 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 \( h(n) = 0 \).
- Heuristic functions are the most common form in which additional knowledge is imported to the search algorithm.
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.
**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

**Proof**

- Provided that \(h(n)\) never overestimates the cost to reach the goal, then in tree search A* gives the optimal solution
  - Suppose \(G_2\) is 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 monotonic (or consistent) 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 monotonic heuristic \( h(n) \) is optimal also for graph search.
- If \( h(n) \) is monotonic, the values of \( f(n) \) along any path are nondecreasing.
- Suppose that \( n' \) is a successor of \( n \) so that

\[ g(n') = c([n,a], n') + g(n), \]

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

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

• 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
4.2 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 \( h_1 \): \( h_2(n) \preceq h_1(n) \)
- We say that the former dominates the latter
- \( A^* \) expands every node for which \( f(n) < C^* \Leftrightarrow h(n) < C^*-g(n) \)
- 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 \( h_2 \) c. 1.25
To come up with heuristic functions one can study *relaxed* problems from which some restrictions of the original problem have been removed.

The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem (does not overestimate).

The optimal solution in the original problem is, by definition, also a solution in the relaxed problem.

E.g., heuristic $h_1$ for the 8-puzzle gives perfectly accurate path length for a simplified version of the puzzle, where a tile can move anywhere.

Similarly $h_2$ gives an optimal solution to a relaxed 8-puzzle, where tiles can move also to occupied squares.

If a collection of admissible heuristics is available for a problem, and none of them dominates any of the others, we can use the composite function

$$h(n) = \max\{ h_1(n), \ldots, h_m(n) \}$$

The composite function dominates all of its component functions and is monotonic if none of the components overestimates.

One way of relaxing problems is to study subproblems.

E.g., in 8-puzzle we could study only four tiles at a time and let the other tiles wander to any position.

By combining the heuristics concerning distinct tiles into one composite function yields a heuristic function, that is much more efficient than the Manhattan distance.
4.3 Local Search Algorithms

- Above we studied systematic search that maintains paths in memory
- In many problems, however, the path to the goal is irrelevant
- It suffices to know the solution to the problem
- Local search algorithms operate using a single current state and generally move to neighbors of that state
- Typically, the paths followed by the search are not retained
- They use very little memory, usually a constant amount
- Local search algorithms lead to reasonable results in large or infinite (continuous) state spaces for which systematic search methods are unsuitable

- Local search algorithms are also useful for solving pure optimization problems
  - In optimization the aim is to find the best state according to an objective function
  - Optimization problems are not always search problems in the same sense as they were considered above
  - For instance, Darwinian evolution tries to optimize reproductive fitness
  - There does not exist any final goal state (goal test)
  - Neither does the cost of the path matter in this task
Optimization of the value of objective function can be visualized as a state space landscape, where the height of peaks and depth of valleys corresponds to the value of the function.

A search algorithm giving the optimal solution to a maximization problem comes up with the global maximum.

Local maxima are higher peaks than any of their neighbors, but lower than the global maximum.

Hill climbing search

- In hill-climbing one always chooses a successor of the current state \( s \) that has the highest value for the objective function \( f \):
  \[ \max_{s' \in S(s)} f(s') \]

- Search terminates when all neighbors of the state have a lower value for the objective function than the current state has.
- Most often search terminates in a local maximum, sometimes by chance, in a global maximum.
- Also plateaux cause problems to this greedy local search.
- On the other hand, improvement starting from the initial state is often very fast.
• Sideways moves can be allowed when search may proceed to states that are as good as the current one
• *Stochastic* hill-climbing chooses randomly one of the neighbors that improve the situation
• Neighbors can, for example, be examined in random order and choose the first one that is better than the current state
• Also these versions of hill-climbing are incomplete because they can still get stuck in a local maximum
• By using *random restarts* one can guarantee the completeness of the method
  • Hill-climbing starts from a random initial state until a solution is found

**Simulated annealing search**

• A *random walk* — moving to a successor chosen uniformly at random from the set of successors independent of whether it is better than the current state — is a complete search algorithm, but when unsupervised also extremely inefficient
• Let us allow "bad" moves with some probability $p$
• The probability of transitions leading to worse situation decreases exponentially with time ("temperature")
• We choose a candidate for transition randomly and accept it if
  1. the objective value improves;
  2. otherwise with probability $p$
• If temperature is lowered slowly enough, this method converges to a global optimum with probability $\rightarrow 1$