1 Problem-Solving Problems

When your environment can be effectively modeled as having

- discrete states and actions
- deterministic, known world dynamics
- known initial state
- explicit goal or goal test; path cost

Treat it as a search problem. This formulation is particularly useful when different goals arise in the same world.

Also, problems are often presented in this form (scheduling, navigation), so either the agent or the designer has to do problem solving.

Build an abstraction of your domain:

- initial state
- successor function (maps states into possible “successor” states)
- goal test
- additive path cost, described as non-negative step costs

Objective will be to find the lowest-cost path to a goal state.

2 Uninformed search

This problem is intractable, in general. You apparently have to consider all possible paths. It’s not that bad, because of additivity, which means that, when considering going through some state $s$, it’s enough to find the cheapest path from start to $s$, and the cheapest path
from $s$ to a goal. You don’t have to think of all possible combinations of paths to and from $s$. We’ll take advantage of this later.

Search tree: root is start state; children are successor states. A single state may occur multiple times in the tree. States are not the same as nodes.

Measuring search complexity:

- $b$: branching factor
- $d$: depth of the shallowest solution in the search tree
- $m$: maximum depth of the tree

Basic search methods. Expand a node by checking to see if it’s a goal. If so, return it. If not, visit its successors, by putting them on the agenda. Methods differ in what order the next node to be expanded is chosen.

- depth-first: agenda is a stack; time $O(b^m)$ space $O(bm)$ not optimal
- breadth-first: agenda is a queue; time $O(b^d)$ space $O(b^d)$ optimal
- uniform-cost: agenda is a priority queue: expand node with least cost so far; optimal
- iterative deepending: do depth-first-search with increasingly deep cut-offs; time $b + b^2 + b^3 \ldots b^d = O(b^d)$, space $O(bd)$; optimal; This is often the uninformed method of choice.

3 Repeated states

*Algorithms that forget their history are doomed to repeat it.*

Consider a rectangular street grid. Search tree has $4^d$ leaves, but there are only about $2d^3$ leaves at depth $d$.

Rule 1: *In any search, never revisit a state that’s on the current path.*

In these uninformed methods, it’s okay to follow

Rule 2: *Never visit a node (add it to the agenda) that has already been expanded.*

We retain optimality, because all of our methods (so far) visit nodes via the best path first. Note, though, that we lose the space advantages of DFS if we do this.

4 Formulating problems

- Driving with gas stations
- Short order cook: 4 dishes, 2 burners, utility is temperature of all the dishes at the end
5 Informed search

We can do better by focusing search in a reasonable direction.

- \( f(n) \): evaluation function on nodes; choose node with lowest \( f(n) \) to expand
- \( h(s) \): heuristic function: estimated cost of cheapest path from \( s \) to a goal state
- \( h(s) \) is admissible if it never overestimates cost
- \( g(n) \): cost so far

\( A^* \) search sets \( f(n) = g(n) + h(s) \). It’s optimal if \( h \) is admissible and we don’t apply Rule 2 above.

Straight-line distance is a good heuristic. In general, you can often “relax” a problem, and use the solution cost in that problem as a lower-bound on the actual costs.

Beware avoiding repeated states! It is no longer necessarily the case that we expand states via the cheapest path first. So, if we find a new path to a state that’s on the agenda, discard the more expensive path.

Another strategy is to ensure that your heuristic is consistent: for every node \( n \) and successor \( n' \),

\[
h(n) \leq c(n, a, n') + h(n')
\]

You don’t have heuristic values that decrease dramatically as you follow a path.

If your heuristic is consistent, then heuristic values are always non-decreasing along a path. So the sequence of nodes expanded by \( A^* \) has non-decreasing values. So the first time we reach the goal, it must be via the cheapest path. So it’s okay to use Rule 2.

6 Memory-bounded informed search

A simple version is iterative-deepening \( A^* \), called IDA*. It’s like iterative deepening, but instead of a depth cutoff on each iteration, we use an \( f \)-cost cutoff. Set the cutoff to be the smallest cost of any node that exceeded the cutoff on the previous iteration. Not so good, because it tends to add too few nodes in each iteration.

**Recursive best-first search (RBFS)** Treat the \( f \) values a bit more generally. They will always be an underestimate of the total path cost of going through this node. We can, in general, improve these estimates by looking farther down the tree.

- keep track of \( f \)-cost of the best alternative path from any ancestor of current node
- if current node exceeds this limit, backtrack to that alternative
- replace all \( f \)-values along the path with the best \( f \)-value of the children

Better than IDA*, but still generates too many nodes. Optimal if \( h \) is admissible. Space \( O(bd) \). Too hard to characterize time; can’t make use of graph-search efficiencies.
Simplified Memory-Bounded A*  Uses all the memory you have available to avoid backtracking too much.

- Works like A* until memory is full
- Forget leaf node with highest $f$ value
- Back values up to parent: means that we only regenerate this subtree when all other subtrees have been shown to be worse

Complete if there is any reachable solution (depth less than memory). Returns best reachable solution.

7 Learning to Search Better

Can try to learn at the meta-level, a mapping from the state of a search problem to which node to expand next. More later.

8 Online Search

What if

- We don’t know the successor function
- We do have a set of actions that can be taken in each state
- The states are completely observable, so when we take an action, we can observe the successor state
- The world is safely explorable: no dead ends

Agent has to wander around the world in search of the goal. Could try a random walk, but they’re terrible, in general. Need to be more systematic. LRTA*

- Let $H(s)$ be the current best estimate of the cost to goal from $s$
- Agent in state $s$ should choose the $a$ that minimizes

$$c(s, a, s') + H(s')$$

- Untried actions have cost $h(s)$: optimism under uncertainty

Initial $h(s)$ has to be admissible. Guaranteed to find the goal in any safely-explorable finite space. Can be led astray in infinite spaces.
9 Exercises

• 3.7d, 3.9, 3.11a-d, 3.19a, e

• Construct a search example with an inadmissible heuristic and show that it can lead to a suboptimal solution.

• 4.2, 4.3, 4.5, 4.14

• What would happen in SMA* if you didn’t back up the values?

• Give an example (safely explorable) infinite space in which LRTA* would fail to find the goal.