1 A fun puzzle

Consider the following search problem: you are trying to get from point A to point B on the plane via a polygonal path, but a line segment (of arbitrary size and position) is blocking the straight-line path. This is a search problem with an infinite number of states: in one step you may go from any point X to any other point Y, provided there is nothing blocking the straight path between them. The step cost is the Euclidean distance between X and Y.

Using the usual heuristic (Euclidean distance), imagine running $A^*$ on this problem. What shape is the locus of points $A^*$ expands? (If you’re bothered by the infinite number of states, suppose everything is contained in a rectangle and the states are lattice points.) What is the shape of the locus (roughly) if the obstacle is a circle rather than a line segment?

2 Notes

2.1 Search

Parts of a search problem: start state, transition model, goal test, path cost

What search algorithms have in common: can be thought of as building trees

1. Root node is annotated with start state. Initial fringe is root node.
2. Successively expand nodes (pop them off the fringe, adding children to fringe).
3. When you expand (not enqueue!) a goal node, you’re done.

Dimensions of variation

1. Whether you remember states you’ve visited before
2. How you decide which node to expand
   • Uninformed: breadth-first, depth-first, uniform cost, iterative deepening
   • Informed: you have a function $f(n)$ defined for all nodes of search tree
     - Expand lowest $f(n)$
       * $A^*$: $f(n) = g(n) + h(n)$ ($h$ estimates forward cost; $h$ only a function of state!)
       * Greedy (sometimes also called best-first): $f(n) = h(n)$ ($h$ as above)
     - Beam search (like greedy, but prune paths)

Admissibility: $h(n)$ never more than true best cost from $n$. Implies $A^*$ is optimal (but remember caveat about graph search).

Consistency: for all edges $n \rightarrow p$ in search tree, $h(n) \leq cost(n, p) + h(p)$. Implies admissibility. Also implies $A^*$ visits nodes in order of increasing $f$.

Constructing heuristics: bigger is better; try true cost for a relaxed problem; try max of multiple heuristics.

Local Search: hill-climbing (gets stuck on local minima, ridges), random restarts, stochastic beam search, simulated annealing (Monte Carlo descent: $p \propto e^{-\Delta h/T}$), genetic algorithms
2.2 Constraint Satisfaction Problems

Parts of a CSP: variables $X_i$, domains $D_i$, constraints $C_i$ dealing with subsets of $\{X_i\}$

Concepts: CSPs as search, incremental vs. complete-state formulations, constraint graphs

Fail-first/minimum remaining values (MRV) heuristic: assign the variable with the smallest number of values it could take on (while preserving consistency)

Least-constraining value heuristic: assign a value to $X_i$ that maximizes the minimum over neighbors $V$ (of $X_i$) of |consistent values for $V$|

Forward checking: prune domains when a variable is assigned

Arc consistency on $(X,Y)$: prune domain of $X$ (initially and/or anytime domain of $Y$ is reduced)

$k$-consistency of a CSP: given any consistent assignment to a subset (of size $k-1$) of variables, it is possible to consistently assign any $k$th variable. 2-consistency is arc-consistency. 3-consistency is called path-consistency. A CSP is strongly $k$-consistent if it is $j$-consistent for $j$ from 1 to $k$.

Cycle cutset: a set of nodes which, if removed, make a graph into a tree

Methods exploiting structure of constraint graphs

Constraint graphs that are trees can be solved in $O(n)$ time. In cutset conditioning, we loop over consistent assignments to a cutset, solving the tree CSP inside the loop. In tree decomposition, we build a junction tree that represents the constraints and solve the resulting tree CSP.

3 Questions

1. True or false: hill-climbing is a specialization of beam search.
2. Are the heuristics described for CSPs the same sort of heuristics as used in informed search?
3. Explain why 2-consistency and arc-consistency are the same.
4. Why is 3-consistency called path-consistency?
5. Why is a cycle cutset called a cycle cutset?
6. Where does the term cutset conditioning come from?
7. Find a way to describe Sudoku as a CSP.
8. Find another way which uses a completely different set of constraints.
9. Based on the last two question, does it sometimes make sense to use redundant constraints?
10. Suppose we write a Sudoku-solving backtracking algorithm which (a) performs forward checking (b) makes an assignment to a variable anytime the size of the domain of that variable is cut down to one. Can arc-consistency buy us anything when solving Sudoku as compared to the algorithm above? Why or why not? (You can assume we use only the binary constraints.)
11. Construct an example where arc-consistency prunes the search space more than merely forward checking.
12. Write a backtracking CSP algorithm which maintains arc-consistency.
13. Why isn’t a $k$-consistent CSP automatically strongly $k$-consistent? Find an example.
14. Round-robin scheduling: suppose there are $n$ soccer teams which must each play each other team within $m$ days. A team can only play once a day. Formulate the problem of scheduling these games as a CSP. If you want practice with CSPs, try picking $m$ and $n$ and running through backtracking search with various heuristics.