50.021 -AI

Alex

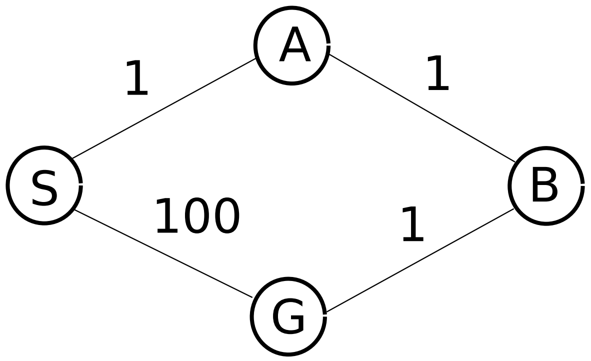
Week 08: Search

[The following notes are compiled from various sources such as textbooks, lecture materials, Web resources and are shared for academic purposes only, intended for use by students registered for a specific course. In the interest of brevity, every source is not cited. The compiler of these notes gratefully acknowledges all such sources.]

Search - What should you know at least?

- state - an abstract world state, i.e. position in a maze
- node - a data structure, contains usually not only the current world state but also information about all the sequence of actions taken and the path from the start state to the current world state
- leaf state - the current (=last) world state of a node. the current end of the encoded path
- plain tree search (always inserts a node into the frontier, no matter what leaf state it has)
- tree search with cycle pruning (does not insert a node into the frontier if the whole path has a cycle)
- tree search that checks the frontier if a node exists that has the same leaf state (and keeps only one node with same leaf state)

  Optional
- tree search that never pushes a the same state twice on the fringe
  - this is ok if path costs do not matter
  - with UCS or other cost functions it may miss the cheapest path. Example:



Suppose one does UCS in this graph . Start is S, Goal is G.
If one never pushes the same state twice on the fringe, then one will miss the cheap path S-A-B-G with cost 3, and choose the expensive path S-G with cost 100.
In conclusion, the tree search "that never pushes a the same state twice on the fringe" with UCS may not find the optimal cost solution if actions costs can be anything. This version is simple to remember but not to be used with UCS (but if costs dont matter, you can use that with BFS, DFS, iterative deepening)
For that reason I showed a tree search algorithm (m3-search.ppt, slide 22) that inserts a node always or that checks the frontier if a node exists that has the same leaf state. Obviously that algorithm is incompatible with "never pushes a the same state twice on the fringe".

- graph search
  - maintains an explored set. A state s is added to the explored set when a nodes, which has s as leaf state, is taken off the fringe. Note: stated are **NOT** added at the time when nodes are added to the fringe.
  - checks the frontier if a node exists that has the same leaf state (and keeps only one node with same leaf state - when costs matter, then the path to the leaf state with the cheapest cost).
  - does not add a node if its leaf state is in the explored set (means if the leaf state was one)
  - this automatically includes cycle pruning

Modes of search (how to order the fringe)

- breadth-first search - memory problem $O(b^d)$ at least, where d is the depth of the shallowest solution
- depth-first search
  - can miss shallow solutions by design
  - can descend forever in infinite trees.
  - Search time complexity is not in terms of depth of solution d, but in terms of depth of the tree.
  - memory efficient
- iterative deepening search (wastes a bit of time, but memory efficient), does not care about costs
- bidirectional breadth first graph search $O(b^{d/2})$ space and time

- uniform cost search (UCS)
  - finds path with lowest cost
  - UCS with constant action cost is same as BFS, and thus inherits its potential memory problems.
- greedy best first search: like UCS, but criterion is not path cost so far, but a heuristic of cost from current node to a goal state
- A* search like UCS, criterion is sum of UCS criterion and
  - A* tree search is optimal if the heuristic is admissible, that is never over estimates the true cost from the current node to the goal
  - A* graph search is optimal if the heuristic is consistent (a stronger condition).
  - of course know these definitions
  - understand the idea of hill-climbing search, beam search,
  - and of simulated annealing which has a temperature parameter that is decreased over loop iterations, and that sometimes makes a step in the wrong direction of the objective based on a probability distribution which depends on the temperature, and the difference ΔE between the objective in the current state, and the objective in a proposed state.

    The idea of simulated annealing is: go sometimes into the wrong direction to escape local minima. but decreases the probability of steps in the wrong directions over time

    Example: you want to **maximize an objective**, then, if the proposed state has a higher objective value than the current state, then $\Delta E < 0$. In simulated annealing, if the proposed state is worse (and thus $\Delta E > 0$), then with a certain probability you still chose the worse proposed state. The probability decreases as temperature decreases, and as $|\Delta E|$ increases. If $\Delta E > 0$, then $\Delta E > 0$ is a measure on how much worse the new state is. So you accept much worse states with lower probabilities.

Further reading

- Memory bounded A*