Homework 1

COMP 557 - Articial Intelligence

Afsaneh Rahbar (ar72)
Linus Shih (yls2)

September 4, 2018

1 Search problems with negative path costs (25 points)

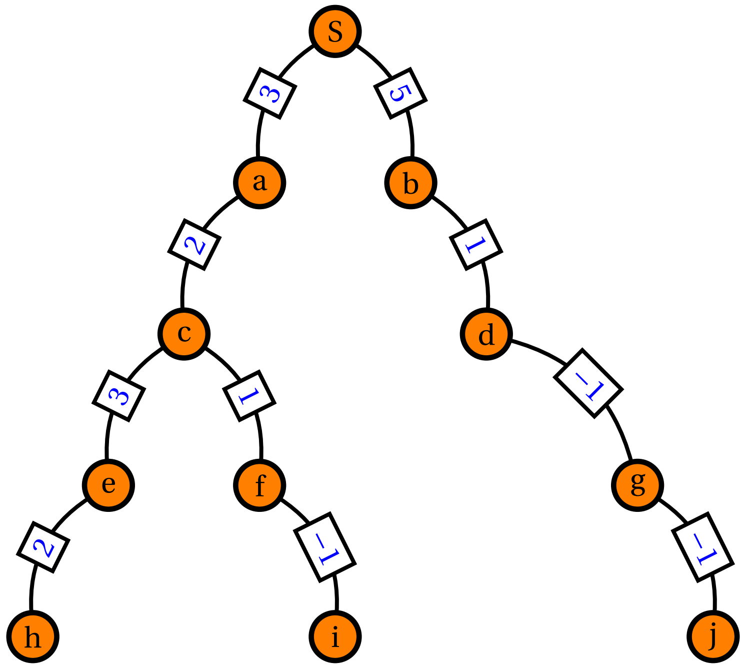
1.1 (5 points)

A path A whose first N edges might seem suboptimal in terms of edge cost may turn out to have a sufficiently high negative edge cost in its next M edges such that they offset the high edge cost of the first N edges. This would cause such a path to be the optimal path in the state space. However, an algorithm that does not explore all paths would ignore this optimal path due to the initially high edge cost of the first N edges. Therefore, an algorithm that would be able to recognize this hypothetical path as the optimal one is one that would have to search all paths completely in order to find the optimal one.

1.2 (5 points)

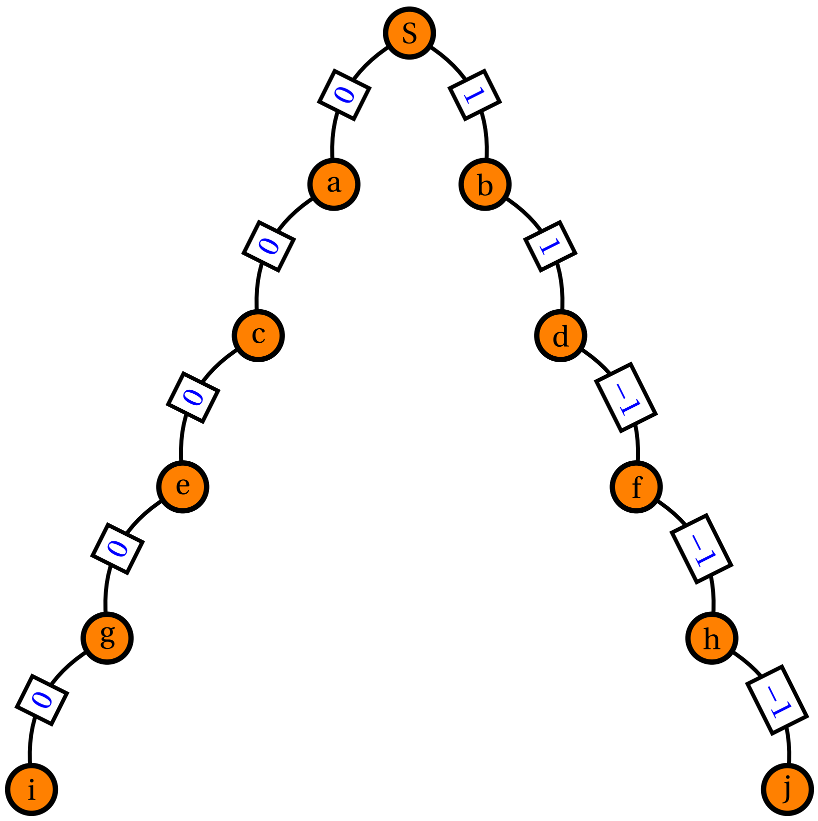
This approach would work in a state space with no cycles or loops. By placing a constraint C on the minimum value that any edge cost can have, we know how much any given path's total cost can be reduced by at any given point along the traversal of that path. As long as we know the maximum depth D of the state space, we know how many steps remain as we traverse down each path. Thus, at a certain step X on each path, we know that the total cost of each path can only be reduced by at most $(D - X) * C$ units (assuming that all remaining steps on all paths are of lowest possible cost C). We calculate the temporary optimal path as of step X and its corresponding total path cost. Any other paths that have a total path cost higher than $(D - X) * C$ compared to the temporary optimal path can be eliminated since, at this step, we know that these paths and their costs will never be able to fall below the temporary optimal path's total path cost. By eliminating these paths, we allow the algorithm to avoid expanding unnecessary nodes, thus improving its efficiency. However, in a state space with cycles, this approach would not work because the algorithm would be able to continuously accumulate a negative reward of C each time it passes through a loop, resulting in a total path cost of negative infinity.

The below example demonstrates how a constraint on the minimum edge cost could allow an algorithm to be optimized further to avoid expanding all nodes.



X	P1	P2	P2	$(D-X)*C$	Temp. Opt. Path	Path Elim.
1	3	3	5	-3	P1,P2	-
2	5	5	6	-2	P1,P2	-
3	8	6	5	-1	P3	P1
4	10	5	4	0	P3	P2

In the case presented below, an optimal algorithm still needs to explore every path in the state space despite having a constraint placed on the minimum edge cost.



X	P1	P2	$(D-X)*C$	Path Elim.
1	0	1	-5	-
2	0	2	-4	-
3	0	1	-3	-
4	0	0	-2	-
5	0	-1	-1	-
6	0	-2	-2	-

1.3 (5 points)

Because the total path cost is negative for the cycle, then the agent's optimal behavior would be expected to continuously loop throughout the cycle infinitely. The total path cost of the cycle is negative, so it represents the most optimal path in the state space, and because it can loop on itself, an infinite loop would allow the total path cost to become $\infty * (negative\ cycle\ edge\ cost)$, which would approach a total path cost of $-\infty$ over time.

1.4 (5 points)

Humans do not drive around scenic loops indefinitely because doing so would eventually diminish the joy we feel from seeing the view. We get used to driving past the same scenery everyday, and we begin to ignore it because we become familiar with what it looks like. In other words, our memory of the beautiful scenery is enough to satisfy us such that we do not need to continuously go see it in person. The concept of memory would help artificial agents prevent looping when applied to state space models. By defining a data structure containing a memory of states previously visited by the agent, it will then know what other states within the model remain unexplored. Furthermore, by applying a penalty such that the reward for visiting a state is reduced if it has been previously visited, we can incentivize the agent to visit previously unexplored states instead. Such an implementation would be effective to prevent looping.

1.5 (5 points)

Cycles with negative costs can be thought of as positive feedback loops. For example, when training a dog to sit, we can give it food each time it responds properly to our command for it to sit. This positive feedback/reinforcement/reward teaches the dog to repeat this behavior on command because it knows we will give it a positive reward (food) for doing so. In this manner, we establish a positive feedback loop where the cycle of the dog sitting on command is driven and reinforced by a negative cost or positive reward of food.

2 Analyzing Search Algorithms: dynamic A* (20 points)

2.1 (5 points)

Dynamic A* is complete on locally finite graphs with admissible and monotonic heuristics. Admissibility keeps the algorithm's cost expectations from the start to the goal within a finite limit and prevents it from overshooting the goal. If it overshoots the goal, the algorithm will fail to return a path because it will continue its search and not terminate. Monotonicity ensures that consistent progress is made towards the goal and that no regression or backtracking occurs, which prevents looping and non-terminating behavior. With these conditions, dynamic A* will always return a path if there is one from the start to the goal, or else return none. In other words, it will always terminate while giving a definite answer (yes there is a path or no there is none).

If a path from start state S to goal state G exists, it will be constructed. If no path exists, it will be reported in a finite amount of time. Proof: Each time a state is expanded, it places its neighbors on the open list. Unless a state in the sequence is never selected for expansion, if the sequence S exists, it will be constructed. The k value of a stated does not change after it has been placed on the open list. Because of the monotonicity of k_{min}, all states in the sequence will eventually be selected for expansion (Stentz, 1994).

2.2 (10 points)

In the worst case D* will need to explore all the edges O(b^d) or O($size\_of\_the\_state\_space$). D* is most efficient when changes are detected near the starting point in the search space.

2.3 (5 points)

- Dynamic A* (D*) performs better than other approaches (such as A*) in unknown/changing environments. When the initial route gets blocked, D* only needs to modify the path locally whereas other algorithms need to generate a new global trajectory. The local trajectories remain constant in complexity, but the global trajectories increase in complexity when the environment size increases. Thus, D* needs less time and resources.
- D* only needs to update the heuristic values of states in its immediate vicinity in order to find the new optimal path, whereas other algorithms need to recompute these values for all states in the full state space.
- D* can change edge cost parameters during the problem solving process while other approaches such as A* can not, this lead to D* being able to generate optimal trajectories.

3 Search space formulations and admissible heuristics (20 points)

3.1 (5 points)

O(n^{2n})

3.2 (5 points)

5^n

3.3 (5 points)

One heuristic can be the Manhattan distance between the vehicles current location $(x_i; y_i)$ and the goal $(n, n-i+1)$. $h_i = |x_i - y| + |y_i - (n-i+1)|$

3.4 (5 points)

From the options above, $max(h_1;...;h_n)$ and $min(h_1;...;h_n)$ are admissible heuristics for the problem of moving n vehicles to their destinations.

4 Pacman Search

4.1 Eating All The Dots

The algorithm completed the medium search in about 3 seconds.

4.2 Suboptimal Search

If we use a greedy approach that does not do both micro and macro environment optimization, then always eating the closet food may not lead to the optimal path.

References