15-210 Assignment 05

Karan Sikka
ksikka@cmu.edu
Section C
October 19, 2012

1: Task 2.3

The euclidean distance satisfies the consistency property. Consider a pair of adjacent vertexes A, B, on an an arbitrary graph with target T, where WLOG $h(A) \leq h(B)$. It's easy to see visually that a straight line between A and B would have a distance of at least $h(A) - h(B)$. Therefore, $h(A) - h(B) \leq w(A, B)$. Additionally, $h(T)$ is 0 since the minimum distance from a target to another target is 0.

2: Task 2.4

A heuristic that makes A* perform exactly as Dijkstra's is $h(v) = 0$. Then, the A* property becomes exactly Dijkstra's property, and everything else is the same, so the algorithm would be equivalent to Dijkstra's.

3: Task 2.5

Since $h(v) - h(w) \leq w(v, w)$ and $h(T) = 0$, $h(n)$ evaluates to at most the cost of the path from n to T. This can be proved by induction. In the base case, $h(T) = 0$ because the cost from the target to the target is 0. Now we assume that $h(w)$ is at most the cost from w to T.

$$h(v) \leq w(v, w) + h(w) \leq \text{cost from v to w} + \text{cost from w to T}$$

Clearly, $h(v)$ is at most the cost from v to the target. The expression $d(v) + w(v, w) + h(w)$ represents at least the cost from the source to w and at most the cost from the source to the target. Say, for sake of contradiction, that A* pops the target node suboptimally before it pops some node n contained in the optimal path to the target. Then we know the cost of the suboptimal path is less than the estimated cost of the optimal path containing n. However, this is a contradiction since the path containing n is optimal, and it's estimated cost is less than or equal to its actual cost (due to consistency). Therefore, A* will always find the shortest path when it pops a target.

4: Task 2.7

No, you can't stop early. This is because with negative edge weights, you're no longer guaranteed that the distance in the table for some vertex will be the shortest path from the source to that vertex.

Instead, you must stop once you've used Dijkstra's to find the minimum path out of the paths with positive edges leading to the target, **and** once you've explored **all** paths containing negative edges which lead to the target.

If there is a negative edge, you can't necessarily skip a node if it has already been visited. Since dijkstra's property doesn't hold here, you have to check if the priority assigned to the node is less than the distance in the table (a normally impossible outcome when edges are nonnegative). If the priority is less than the distance in the distance table, you must re-visit the node.

5: Task 2.8

It visits each node at most once, so it visits at most n nodes. However, there is a good chance it will visit fewer than that because it stops as soon as it sees a target node. It will visit at least l nodes where l is the number of edges in the path from the source to the target, and at most n nodes where n is the number of vertices in the graph.