Intro to AI Assignment 1 — Heuristic Search

Kenneth Bambridge — kmb394

February 18, 2017

Contents

1 Introduction

2 Algorithms

2.1 Uniform Cost Search

2.2 A*

2.3 Weighted A*

2.4 Sequential A*

2.4.1 Proof that Sequential A* is w_1 subomptimal

Given that the g value of any state s expanded by the Weighted A* algorithm is at most w_1 suboptimal for an admissible and consistent heuristic, consider a state s_i in the $OPEN_0$ queue that is on the least cost path to s_{goal}. During the initialization loop starting at line 13, all of the $OPEN_i$ fringes are populated with the start node, which is trivially on the shortest path to s_{goal}. For the iterations starting at line 19, the only time the $OPEN_0$ data structure is inserted or updated is on line 11, where it is inserted with the value $g_0(s) + w_1 * h_0(s)$ for all neighbors where their g values are less then the existing keys in the fringe.

Since during every `ExpandState` call, all neighbors are considered for the node that was originally on the shortest path (the start node), there must also be a node s_n considered every time $OPEN_0$ is updated that is on the shortest path to s_{goal}. This node's g value is also updated if it is less then the existing value. The key of this value is $g(s_n) + w_1 * h(s_n)$. The heuristic is admissible, so $h(s_n) \leq c^*(s_n) \implies w1 * h(s_n) \leq w1 * c^*(s_n)$. Since s_n is on the shortest path to goal, $g(s_n)$ represents the shortest path from $s_{start} \rightarrow s_n$, therefore $g(s_n) + w_1 * h(s_n) \leq w_1 * c * (s_n)$. Since this 'minimum' node exists for the initialization and is maintained in every loop, it exists for the duration of the algorithm.

The heuristic is admissible and consistent, therefore s_n must also be the minimum key in $OPEN_0$ because

2.4.2 Proof that Sequential A* is $w_1 * w_2$ subomptimal

The program can exit in one of two ways, either with the anchor search or via an non-anchor heuristic. If all the non-anchor heuristics have minimum keys greater than the anchor, the anchor will be run and the goal returned if $g_0(s_{goal})$ is less then the minimum key in $OPEN_0$ and infinity. In this case the output is w_1-suboptimal as proven above.

When the program exits with an non-anchor heuristic, it is run when the minimum key is less then $w_2 * OPEN_0$.Minkey $< w_2 * w_1 * c^*(s_goal)$. Therefore when the algorithm exits in this way the solutions is $w_1 * w_2$-suboptimal.

2.5 Integrated A*

2.5.1 Proof i

No state expanded more than twice other than the start node.

1. All $OPEN_i$ begin with the start node inserted.
2. When a state is expanded, it is inserted into one of the CLOSED data structures. (a) In the inadmissible branch, it is inserted into $CLOSED_{inad}$ (line 36). (b) In the admissible branch, it is inserted into $CLOSED_{anchor}$ (line 44).
3. A state is never inserted into $OPEN_i, \forall i$ while it is in the $CLOSED_{anchor}$ data structure (line 12).
4. A state is never inserted into $OPEN_i, \forall i \neq 0$ while it is in the $CLOSED_{inad}$ data structure (line 14).
5. A state can only be expanded after being inserted into an $OPEN_i$ data structure (line 29, 34, 42).
6. By combining (1a), (2) and (4), a state will be expanded at most once in the inadmissible branch (line 35).
7. By combining (1b), (2) and (4), a state will be expanded at most once in the anchor branch (line 43).

Since a non-start state will only be expanded at most once in each branch, therefore no state is expanded more than twice other than the start node.

2.5.2 Proof ii

State expanded in the anchor search is never re-expanded.

1. When the anchor search expands a state, the state is added to $CLOSED_{anchor}$ (line 44).
2. Only the anchor search branch can expand nodes in $OPEN_0$ (line 42).
3. The anchor search branch only expands nodes in $OPEN_0$ (lines 42–44).
4. A state is never inserted into $OPEN_i, \forall i$ while it is in the $CLOSED_{anchor}$ data structure (line 12).
5. After a state is expanded in the anchor search, it is not in $OPEN_i, \forall i$ (line 4).
6. A state can only be expanded while in an $OPEN_i$ data structure (line 29, 34, 42).
7. Therefore, a state expanded in the anchor search is never re-expanded.

2.5.3 Proof iii

State expanded in an inadmissible search can only be re-expanded in the anchor search if its g-value is lowered

1. A state expanded in an inadmissible can add the the state to $OPEN_0$ (line 13).
2. it probably has to do with line 16

3 Heuristics

3.1 Chebychev Distance

3.2 Manhattan Distance

3.3 Diagonal Distance

3.4 Euclidian Distance

4 Implementation Notes

4.1 Map Generation

4.2 Algorithm Implementation

4.3 GUI Visualizer

5 Benchmarks

5.1 Overall Comparison

5.2 Comparison of Heuristic Modifiers

5.3 Uniform Cost Search

5.4 A*

5.5 Weighted A*

5.6 Sequential A*

5.7 Integrated A*

5.8 Conclusion