Example of A^* with different g estimates

Stephen D. Boyles

October 15, 2018

The $A*$ algorithm is a refinement of Dijkstra's algorithm, with potential to run more efficiently when finding a path from a single origin to a single destination. The key is the selection of lower bounds g_i^s on the shortest path cost from each node i to the destination s. As a label setting algorithm, once a node is finalized we know its labels will never change again, and the best path found thus far is indeed shortest among all possible paths. If we are only finding a path to one destination, we can therefore stop as soon as we scan the destination node. (This is *not* true of a label correcting method; no labels are final until the entire algorithm is done.)

This document gives three examples of how these lower bounds might be chosen, and the corresponding effect on the performance of A^*. The network for these examples is shown in Figure 1. The origin is node 5, and the destination is node 3. Notice that all links are bidirectional (i.e., each line in the figure actually corresponds to two links, one in each direction). Costs are the same for both directions of a link.

The progress of each algorithm is shown in Tables 1 to 2. These tables show, at each iteration, the node currently scanned (i) the cost labels for the nine nodes, and the sets of eligible and finalized nodes E and F. The backnode labels are omitted for brevity. For reference, the g_i^s values are shown at the top of each row of the table. At each iteration, the $A*$ algorithm chooses i to be the eligible node (i.e., in E) with the least value of $L_i + g_i^s$. The three methods of choosing g_i^s are:

1. The trivial lower bound of $g_i^s = 0$ for all i. This results in A^* running exactly the same as Dijkstra's algorithm (which simply chooses the eligible node with the least L_i). All nine nodes must be scanned before the destination is reached. See Table 1.
2. All links in the network have a cost of at least 2; therefore, the shortest path cost from each node to the destination is at least twice the number of links in the most direct path from each node to the

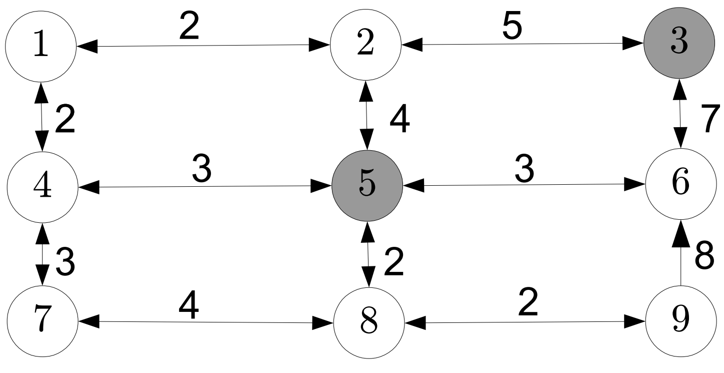

Figure 1: Network for A^* example, with link costs shown.

Table 1: Naïve A^* with the trivial lower bound (identical to Dijkstra's).

	g^3	0	0	0	0	0	0	0	0	0		
Iteration	i	L_1	L_2	L_3	L_4	L_5	L_6	L_7	L_8	L_9	E	F
0	—	∞	∞	∞	∞	0	∞	∞	∞	∞	$\{5\}$	$\emptyset$
1	5	∞	4	∞	3	0	3	∞	2	∞	$\{2,4,6,8\}$	$\{5\}$
2	4	5	4	∞	3	0	3	6	2	∞	$\{1,2,6,7,8\}$	$\{4,5\}$
3	8	5	4	∞	3	0	3	6	2	4	$\{1,2,6,7,9\}$	$\{4,5,8\}$
4	6	5	4	10	3	0	3	6	2	4	$\{1,2,3,7,9\}$	$\{4,5,6,8\}$
5	2	5	4	10	3	0	3	6	2	4	$\{1,3,7,9\}$	$\{2,4,5,6,8\}$
6	9	5	4	10	3	0	3	6	2	4	$\{1,3,7\}$	$\{2,4,5,6,8,9\}$
7	1	5	4	9	3	0	3	6	2	4	$\{3,7\}$	$\{1,2,4,5,6,8,9\}$
8	7	4	4	9	3	0	3	6	2	4	$\{3\}$	$\{1,2,4,5,6,7,8,9\}$
9	3	4	4	9	3	0	3	6	2	4	$\emptyset$	$\{1,2,3,4,5,6,7,8,9\}$

Table 2: Cleverer A^* with lower bounds based on number of links to destination.

	g^3	4	2	0	6	4	2	8	6	4		
Iteration	i	L_1	L_2	L_3	L_4	L_5	L_6	L_7	L_8	L_9	E	F
0	—	∞	∞	∞	∞	0	∞	∞	∞	∞	$\{5\}$	$\emptyset$
1	5	∞	4	∞	3	0	3	∞	2	∞	$\{2,4,6,8\}$	$\{5\}$
2	6	∞	4	10	3	0	3	∞	2	11	$\{2,3,4,8,9\}$	$\{5,6\}$
3	2	6	4	9	3	0	3	∞	2	11	$\{1,3,4,8,9\}$	$\{2,5,6\}$
4	8	6	4	9	3	0	3	6	2	4	$\{1,3,4,7,9\}$	$\{2,5,6,8\}$
5	9	6	4	9	3	0	3	6	2	4	$\{1,3,4,7\}$	$\{2,5,6,8,9\}$
6	3	6	4	9	3	0	3	6	2	4	$\{1,4,7\}$	$\{2,3,5,6,8,9\}$

destination (the path with the least number of links). With this choice, only six nodes are scanned, and we can find the shortest path to node 3 without having to scan nodes 1, 4, or 7: A^* focuses the search on nodes in the general direction of the destination. See Table 2.

3. The tightest possible lower bound is the exact shortest path cost from each node to the destination. With this choice, A^* *only* scans nodes along the shortest path, and only three scans are needed. This is the best possible performance, but in practice the only way to know these values of g_i^s is to already have solved shortest path. See Table 3.

You can further check that choosing g values in between the second and third cases will result in performance between the two; the closer the bounds are to the true values (the third case), the fewer nodes will be scanned. It is also instructive to check that if the g values are *not* lower bounds, the algorithm may not find the correct shortest path — if, say, $g_2^3 = 20$, stopping as soon as node 3 is finalized gives the wrong path.

Table 3: Omniscient A^* with perfect lower bounds.

	g^3	7	5	0	9	9	7	12	11	13		
Iteration	i	L_1	L_2	L_3	L_4	L_5	L_6	L_7	L_8	L_9	E	F
0	—	∞	∞	∞	∞	0	∞	∞	∞	∞	$\{5\}$	$\emptyset$
1	5	∞	4	∞	3	0	3	∞	2	∞	$\{2,4,6,8\}$	$\{5\}$
2	2	6	4	9	3	0	3	∞	2	∞	$\{1,3,4,6,8\}$	$\{2,5\}$
3	3	6	4	9	3	0	3	∞	2	∞	$\{1,4,6,8\}$	$\{2,3,5\}$