

Heuristic Search on Graphs¹

A tree is a special case of a graph in which there is exactly one path to every node. Indeed an arbitrary graph can contain multiple paths to any given node. To extend heuristic search algorithms designed for trees to graphs, we maintain a *closed* list of nodes already visited, in addition to the open list of nodes yet to be visited. In best- h search, nodes are inserted into the open list as they are encountered, if they are not yet on either the open or the closed lists. In addition, in best- g search, nodes on the open list are promoted if they are re-encountered with lower priority. In addition, in A* search, nodes on the closed list are re-opened if they are re-encountered with lower priority.

Best- h Search on Graphs It is straightforward to extend best- h search to graphs: we maintain a closed list and do not open any nodes that already appear on either the open list or the closed list. Since the heuristic h is a function (i.e., each node's value is unique), nodes are never re-encountered with lower priority. Thus, open nodes need not ever be promoted and closed nodes need not be re-inserted into the open list. The best- h search algorithm for graphs is depicted in Table 1.

Best- g Search on Graphs The best- g search algorithm is outlined in Table 2. Like best- h search, best- g search maintains a closed list and never re-opens any nodes on this list: since nodes are visited in priority order, closed nodes can never be re-encountered with lower priority. But open nodes could be re-encountered with lower priority, since there can be multiple paths to a single node in a graph. Best- g search *promotes* nodes on the open list if ever they are re-encountered with lower priority.

A* Search on Graphs Table 3 describes A* search through arbitrary graphs. Like best- g search, A* search on graphs promotes nodes on the open list whenever such nodes are re-encountered with lower priority. In addition, A* search on graphs maintains a closed list and re-opens nodes on this list if ever they are re-encountered with lower priority. How can closed nodes be re-encountered with lower priority? Recall that A* search is guided by the evaluation function $f = g + h$: i.e., nodes are visited in *roughly* f -order. Unlike in best- g search, where nodes are visited in g -order, a closed node in A* search can be re-encountered with lower a g -value, and thus a lower priority f than its priority during any previous encounters.

¹Copyright© Amy Greenwald, 2001-05

BEST- $h(X, S, G, \delta, c, h)$	
Inputs	search problem heuristic function h
Output	(path to) optimal goal node
Initialize	$O = S$ is the list of open nodes $C = \emptyset$ is the list of closed nodes
<pre> while (O is not empty) do 1. delete node $n \in O$ s.t. $h(n)$ is minimal 2. if $n \in G$, return (path to) n 3. for all $m \in \delta(n)$ (a) compute $h'(m)$ (b) if $m \notin C$ and $m \notin O$ i. insert m into O with priority $h(m) = h'(m)$ 4. insert n into C fail </pre>	

Table 1: Best- h Search on Graphs.

1 Optimality

A heuristic function h is called *monotone* at node n iff $h(n) \leq c(n, m) + h(m)$ for all successor nodes $m \in \delta(n)$. A heuristic function h is said to be *consistent* iff (i) it is monotone at all non-goal nodes, and (ii) it is zero at all goal nodes.

Lemma If h is consistent, then f is monotonically, nondecreasing in depth: i.e., for all $n \in X$, for all $m \in \delta(n)$, $f(m) \geq f(n)$.

Proof Note that g (and therefore f) is not a function. For each value of $g(n)$,

$$\begin{aligned}
 f(n) &= g(n) + h(n) \\
 &= g(m) - c(n, m) + h(n) \\
 &\leq g(m) + h(m) \\
 &= f(m)
 \end{aligned}$$

Theorem A* search on graphs with a consistent heuristic is optimal.

BEST- $g(X, S, G, \delta, c)$	
Inputs	search problem
Output	(path to) optimal goal node
Initialize	$O = S$ is the list of open nodes $C = \emptyset$ is the list of closed nodes
while (O is not empty) do	
1. delete node $n \in O$ s.t. $g(n)$ is minimal	
2. if $n \in G$, return (path to) n	
3. for all $m \in \delta(n)$	
(a)	$g'(m) = g(n) + c(m, n)$
(b)	if $m \notin C$ and $m \notin O$
i.	insert m into O with priority $g(m) = g'(m)$
else if	$m \in O$ and $g'(m) < g(m)$
i.	promote m in O to priority $g(m) = g'(m)$
4. insert n into C	
fail	

Table 2: Best- g Search on Graphs.

2 Examples

A sample graph appears in Figure 1. The start state is S and the unique goal node is G , but there are two paths to this goal node: $SACG$ and $SBCG$.

Best- h Search Best- h search maintains the following sequence of priority queues in its search through this graph: $S_0, A_1B_2, C_0B_2, G_0B_2$, GOAL! The path returned is suboptimal: $SACG$.

Best- g Search Best- g search maintains the following sequence of priority queues in its search through this graph: $S_0, A_1B_5, B_5C_{21}, C_{10}, G_{10}$, GOAL! The path returned is optimal: $SBCG$.

Best- g search first encounters node C via node A , traversing a path of cost 21. Later, it encounters node C again via node B , traversing a path of cost only 10. At this point, node C 's priority is promoted from 21 to 10.

A*(X, S, G, δ, c, h)	
Inputs	search problem heuristic function h
Output	(path to) optimal goal node
Initialize	$O = S$ is the list of open nodes $C = \emptyset$ is the list of closed nodes
while (O is not empty) do	
1. delete node $n \in O$ s.t. $f(n)$ is minimal	
2. if $n \in G$, return (path to) n	
3. for all $m \in \delta(n)$	
(a) compute $h'(m)$	
(b) $g'(m) = g(n) + c(m, n)$	
(c) $f'(m) = g'(m) + h'(m)$	
(d) if $m \notin C$ and $m \notin O$	
insert m into O with priority $f(m) = f'(m)$	
else if $m \in O$ and $f'(m) < f(m)$	
promote m in O to priority $f(m) = f'(m)$	
else if $m \in C$ and $f'(m) < f(m)$	
re-open m in O with priority $f(m) = f'(m)$	
4. insert n into C	
fail	

Table 3: A* Search on Graphs.

A*(X, S, G, δ, c, h)	
Inputs	search problem heuristic function h
Output	(path to) optimal goal node
Initialize	$O = S$ is the list of open nodes $C = \emptyset$ is the list of closed nodes
<p>while (O is not empty) do</p> <ol style="list-style-type: none"> 1. delete node $n \in O$ s.t. $f(n)$ is minimal 2. if $n \in G$, return (path to) n 3. for all $m \in \delta(n)$ <ol style="list-style-type: none"> (a) compute $h'(m)$ (b) $g'(m) = g(n) + c(m, n)$ (c) $f'(m) = g'(m) + h'(m)$ (d) $f'(m) = \max\{f'(m), f(n)\}$ (e) if $m \notin C$ and $m \notin O$ <ol style="list-style-type: none"> insert m into O with priority $f(m) = f'(m)$ else if $m \in O$ and $g'(m) < g(m)$ <ol style="list-style-type: none"> promote m in O to priority $f(m) = f'(m)$ else if $m \in C$ and $g'(m) < g(m)$ <ol style="list-style-type: none"> re-open m in O with priority $f(m) = f'(m)$ 4. insert n into C <p>fail</p> 	

Table 4: A* Search on Graphs with PathMax.

A*(X, S, G, δ, c, h)	
Inputs	search problem heuristic function h
Output	(path to) optimal goal node
Initialize	$O = S$ is the list of open nodes $C = \emptyset$ is the list of closed nodes
<p>while (O is not empty) do</p> <ol style="list-style-type: none"> 1. delete node $n \in O$ s.t. $f(n)$ is minimal 2. if $n \in G$, return (path to) n 3. for all $m \in \delta(n)$ <ol style="list-style-type: none"> (a) compute $h'(m)$ (b) $g'(m) = g(n) + c(m, n)$ (c) $f'(m) = g'(m) + h'(m)$ (d) if $m \notin C$ and $m \notin O$ <ol style="list-style-type: none"> insert m into O with priority $f(m) = f'(m)$ else if $m \in O$ and $f'(m) < f(m)$ <ol style="list-style-type: none"> promote m in O to priority $f(m) = f'(m)$ 4. insert n into C <p>fail</p>	

Table 5: A* Search on Graphs with a Consistent Heuristic.

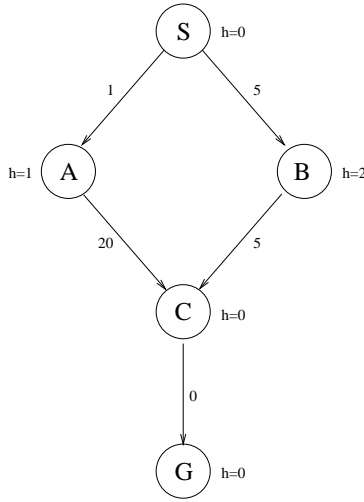


Figure 1: Sample DAG: edges are labeled with costs g ; nodes are labeled with heuristic values h . The start state is S and the unique goal node is G . The optimal path to this goal is $SBCG$. Only best- g search and A* search return this optimal path; best- h search returns the suboptimal path $SACG$.

A* Search A* search maintains the following sequence of priority queues in its search through the graph in Figure 1: $S_0, A_2B_7, B_7C_{21}, C_{10}, G_{10}, \text{GOAL!}$. The path to the goal that is returned is optimal: $SBCG$.

Figure 2 (LHS):

- Best- g : $S_0, D_1A_1, A_1G_{11}, B_2G_{11}, C_3G_{11}, G_{11}, \text{GOAL!}$ (After C_3 is popped, G_{13} is not pushed, since G_{11} is already on the queue. The optimal path SDG is returned.)
- Best- h : $S_0, A_3D_{10}, B_2D_{10}, C_1D_{10}, G_0D_{10}, \text{GOAL!}$ (The path returned, namely $SABCG$, is not optimal.)
- A*: $S_0, A_4D_{11}, B_4D_{11}, C_4D_{11}, D_{11}G_{13}, G_{11}, \text{GOAL!}$ (After D_{11} is popped, the node G_{13} is promoted to G_{11} . The optimal path SDG is returned.)

Figure 2 (RHS):

- Best- g : $S_0, D_1A_1, C_1A_1, A_1G_{11}, B_2G_{11}, C_3G_{11}, G_{11}, \text{GOAL!}$ (After C_3 is popped, G_{13} is not pushed, since G_{11} is already on the queue. The optimal path $SDCG$ is returned.)
- Best- h : $S_0, A_3D_{10}, B_2D_{10}, C_1D_{10}, G_0D_{10}, \text{GOAL!}$ (The path returned, namely $SABCG$, is not optimal.)

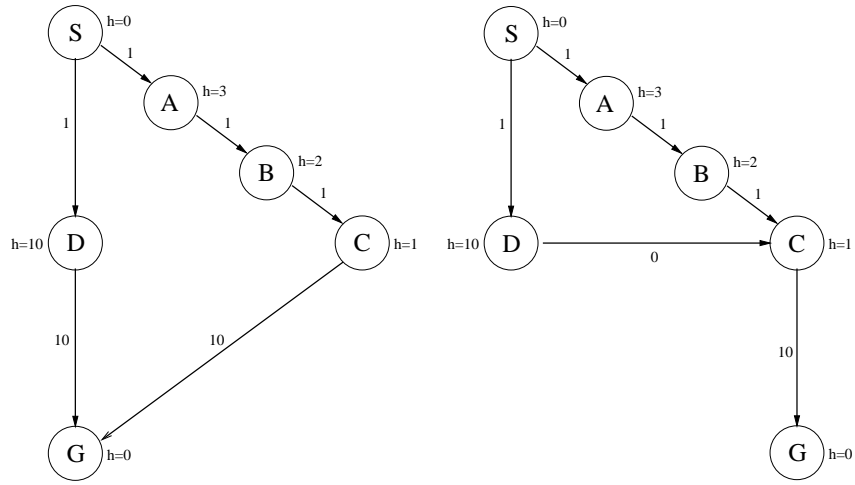


Figure 2: Sample DAGs: edges are labeled with costs g ; nodes are labeled with heuristic values h . The start state is S and the unique goal is G . (LHS) The optimal path to the goal is SDG . (RHS) The optimal path to the goal is $SDCG$.

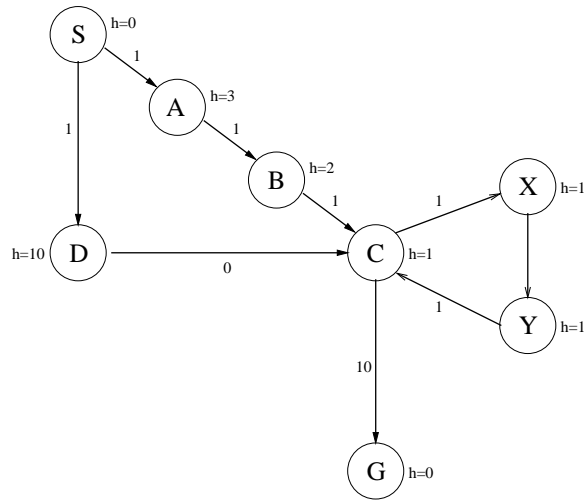


Figure 3: Sample Directed Graph with Cycle: edges are labeled with costs g ; nodes are labeled with heuristic values h . The start state is S and the unique goal is G . The optimal path to the goal is $SDCG$.

- A*: $S_0, A_4D_{11}, B_4D_{11}, C_4D_{11}, D_{11}G_{13}, C_2G_{13}, G_{11}$, GOAL! (After C with cost 4 is closed, it is re-opened with cost 2. And after C_2 is popped, the node G_{13} is promoted to G_{11} . The optimal path $SDCG$ is returned.)

Problems

#1 *Towers of Hanoi*: given a stack of n disks of distinct sizes on a peg with no larger disk on top of a smaller disk, move the disks one at a time from one peg to a second goal peg, making use of a third peg when necessary, but never stacking a larger disk on top of a smaller disk.

(a) Represent the Towers of Hanoi problem for $n = 2$ as a formal search problem by drawing the search space and highlighting the start and final states.

(b) State whether the following heuristics are admissible. Where admissible, state why, and where inadmissible, give counterexample. [Hint: The optimal number of moves from the start state is $2^n - 1$].

1. the number of disks on a peg other than the goal peg
2. the number of disks on top of the largest disk
3. $2^k - 1$, where k is the largest misplaced disk
4. 2^{k-1} , where k is the largest misplaced disk

#2 Prove that consistency implies admissibility.