

Heuristic Search on Trees¹

This lecture expands our notion of basic search problems to incorporate costs. We generalize the notion of optimality discussed in the previous lecture from depth (i.e., all edges are of cost 1) to edges with arbitrary costs. In this extended framework, we introduce heuristic functions, which estimate the cost of reaching a goal node from a state in the search space. Based on these heuristic functions, we present several examples of heuristic search algorithms: best-first search, including greedy search and A* search, and iterative deepening A* search.

1 Search Problem

A *search problem* is a 5-tuple $\langle X, S, G, \delta, c \rangle$, where

- $\langle X, S, G, \delta, \rangle$ is a basic search problem
- $c : X \times X \rightarrow \mathbb{R}$ is a cost function

For $y \in \delta(x)$, $c(x, y)$ denotes the cost of reaching y from x . Now given path $\{n_0, \dots, n_i, n_{i+1}, \dots, n_{k+1}\}$, where $n_0 \in S$, $n_{k+1} = n$, and $n_{i+1} \in \delta(n_i)$ for all $0 \leq i \leq k$, $g(n)$ denotes the cost of reaching node n :

$$g(n) = \sum_{i=0}^k c(n_i, n_{i+1}) \quad (1)$$

Examples of cost functions include: $g(n) = \text{depth}(n)$ and $g(n) = \text{distance}(n)$.

Note that search problems can be stated in terms of cost, with $g(x) \geq 0$, for all $x \in X$, (or value, with $g(x) \leq 0$, for all $x \in X$), in which case the problem is one of minimization (or maximization).

2 Best-First Search

The main idea of the best-first search class of algorithms is to expand the lowest-cost node on the fringe, according to some evaluation function $e : X \rightarrow \mathbb{R}$.

¹Copyright© Amy Greenwald, 2001–05

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

Table 1: Best-First Search. Best- g search is the special case of best-first search in which $e = g$. Best- h search is the special case of best-first search in which $e = h$. A* search is the special case of best-first search in which $e = f = g + h$.

BFS is the special case of best-first search in which the evaluation function $e(n) = \text{depth}(n)$ for node n . Therefore, the complexity of best-first search in the worst-case is at least that of BFS: exponential in the depth of the goal for both time and space. Best-first search visits nodes in depth-first search order, when the evaluation function e dictates the following of paths until the algorithm dead ends. Best-first search is not complete; nor is best-first search optimal.

3 Best- g Search

The main idea of best- g search is to expand the lowest-cost node on the fringe, according to cost function g , defined in Equation 1. Best- g is complete, except in search spaces that contain infinitely many nodes n with $g(n) < g^*$ (e.g., an infinite path with finite cost), where g^* is the optimal cost. Best- g is also optimal: that is, it is guaranteed to find the lowest-cost goal, whenever g is a monotonically, nondecreasing function of depth: i.e., for all $n \in X$, for all $m \in \delta(n)$, $g(m) \geq g(n)$. Monotonicity is violated whenever $c(n, m) < 0$ for some node n and its successor m . Figure 1 depicts two search trees. In both spaces, S is the start state, Y and Z are goal nodes, and Z is optimal. On the LHS, the search tree contains an infinite path of finite cost. Best- g search never reaches either goal node. On the RHS, the search tree contains an edge of negative cost. Best- g search proceeds directly to the suboptimal goal node Y .

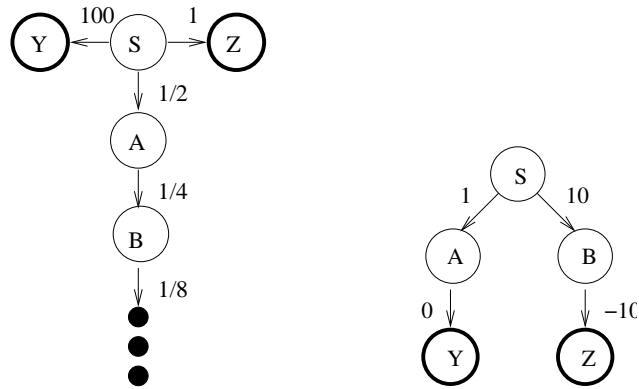


Figure 1: (LHS) A search space that contains an infinite path of finite cost. (RHS) A search space that contains an edge of negative cost. In both search spaces, S is the start state, Y and Z are goal nodes, and Z is optimal.

4 Best- h Search

The main idea of best- h search is to expand the lowest-cost node on the fringe, according to some heuristic function $h : X \rightarrow \mathbb{R}$. The degree of optimality of best- h search depends on the quality of the heuristic function.

A heuristic function $h : X \rightarrow \mathbb{R}$ computes an estimate of the distance from node n to a goal node. Heuristics are used to guide the search process. In the sliding tiles puzzle, one heuristic function $h_1(n)$ is simply the number of misplaced tiles. A second heuristic function $h_2(n)$ is the Manhattan distance: i.e., the number of moves required to place each tile correctly, summed over all misplaced tiles.

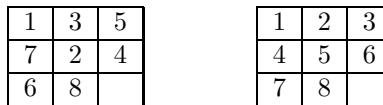


Figure 2: (LHS) Start State. (RHS) Goal State. $h_1(n) = 6$ and $h_2(n) = 10$.

Figure 2 depicts an arbitrary state n and the goal of the 8-puzzle—the sliding tiles puzzle with 8 tiles. In this state n , there are 6 misplaced tiles, and the Manhattan distance evaluates to 10.

Exercise Give other examples of heuristics for the sliding tiles puzzle.

5 Admissible Heuristics

Let $h^*(n)$ be the true cost from node n to the nearest goal node. A heuristic function $h(n)$ is said to be *admissible* iff $h(n) \leq h^*(n)$, for all nodes n . In other words, admissible heuristics are optimistic: in minimization problems, admissible heuristics never overestimate the distance to a goal; in maximization problems, admissible heuristics never underestimate the value of a goal.

The sample heuristics h_1 and h_2 in the sliding tiles puzzle are both admissible. The heuristic function h_1 is admissible since it requires at least one move to move each misplaced tile to its correct position. The heuristic function h_2 is admissible since, more accurately, it requires at least the Manhattan distance to move each misplaced tile to its correct position.

The most useful admissible heuristics are those which most closely approximate $h^*(n)$ *without going over*. An admissible heuristic h *dominates* an alternative admissible heuristic h' iff $h(n) \geq h'(n)$ for all nodes n . Intuitively, a dominant heuristic is more informed than the heuristic it dominates. For example, the Manhattan distance h_2 dominates h_1 .

Exercise Given two admissible heuristics h' and h'' , it need not be the case that one dominates the other. In this case, one can construct *composite* heuristics of the form $h(n) = \max\{h'(n), h''(n)\}$ for all n . The new heuristic h is admissible and it dominates the individual heuristics h' and h'' . Prove this claim.

One “heuristic” for constructing admissible heuristics is to remove one or more of the problem’s constraints. In the sliding tiles puzzle, moves are constrained in three ways: a tile can only be moved into the blank space; a tile must be moved along the grid; and, a tile can only be moved into an adjacent cell. If we relax only the first constraint, this yields the Manhattan distance (h_2). If we relax the first and the second constraints, this yields another heuristic function—Euclidean distance—call it h' . If we relax all three constraints, this yields the heuristic function h_1 . Clearly, h_2 dominates h' dominates h_1 , since h_2 enforces more constraints than h' ; and, h' dominates h_1 , since h' enforces more constraints than h_1 .

6 A* Search

Let $f(n) = g(n) + h(n)$, where $g(n)$ is the cost of reaching node n from the start state and $h(n)$ is an heuristic estimate of the distance from node n to the nearest goal node. The main idea of A* search is to expand the lowest-cost node on the fringe, according to the evaluation function f . Like best- g and best- h searches, A* is a special case of the best-first search algorithm. Nonetheless, we present the A* algorithm in its entirety in Table 2.

A*(X, S, G, δ, c, h)	
Inputs	search problem heuristic function <i>h</i>
Output	(path to) optimal goal node
Initialize	$O = S$ is the list of open 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) insert m into O with priority $f(m)$	
fail	

Table 2: A* Search.

6.1 Optimality

We now prove that A* search is optimal, assuming the heuristic function h is admissible. As in best- g search (the special case of A* search in which $h = 0$), the proof relies on the assumption that g is a monotonically, nondecreasing function of depth. To ensure that this property holds, we assume all costs are nonnegative; in particular, there exists $\alpha \geq 0$ s.t. $c(n, m) \geq \alpha$ for all $n, m \in \delta(n)$. Now it suffices to consider heuristic functions whose cost estimates exceed α .

Definition For some $\epsilon \geq 0$, an heuristic function h is said to be ϵ -admissible iff $h(n) \leq h^*(n) + \epsilon$, for all nodes n .

Theorem If h is ϵ -admissible, then A* search is ϵ -optimal: i.e., if A* returns goal node m^* , then $g(m^*) \leq g(n^*) + \epsilon$, where n^* is an optimal goal node.

Proof Suppose A* returns goal node m^* before it returns optimal goal node n^* . It follows that there exists node n on the priority queue that is also on the path to n^* s.t. $f(m^*) \leq f(n)$. But then $g(m^*) \leq g(n^*) + \epsilon$ (i.e., m^* is ϵ -optimal), by the following reasoning:

$$\begin{aligned}
g(m^*) &\leq g(m^*) + h(m^*) && \text{since } h(m^*) \geq \alpha \\
&= f(m^*) && \text{by definition} \\
&\leq f(n) && \text{by assumption} \\
&= g(n) + h(n) && \text{by definition} \\
&\leq g(n) + h^*(n) + \epsilon && \text{by } \epsilon\text{-admissibility} \\
&= g(n^*) + \epsilon && \text{distance to optimal goal}
\end{aligned}$$

Corollary If the heuristic function h is admissible, then A* search is optimal.

Proof Let $\epsilon = 0$.

6.2 PathMax

The *pathmax* equation is an optimization routine: $\hat{f}(m) = \max\{f(n), f(m)\}$, for all $m \in \delta(n)$. For example, if $g(n) = 1$ and $h(n) = 10$, while $g(m) = 3$ and $h(m) = 3$ for some child m of n , then n 's f -cost is 11, while m 's f -cost is 6. The pathmax equation increases m 's f -cost to 11.

Lemma The pathmax operation preserves admissibility: i.e., if h is admissible, then $\hat{h}(m) = \max\{h(m), h(n) - c(n, m)\}$, for all $n, m \in \delta(n)$ is also admissible.

Proof Fix an arbitrary $n \in X$. To show $\hat{h}(m) \leq h^*(m)$, for all $m \in \delta(n)$, it suffices to show: (i) $h(m) \leq h^*(m)$, and (ii) $h(n) - c(n, m) \leq h^*(m)$. Condition (i) follows immediately by the admissibility of h . Condition (ii) also follows by admissibility: $h(n) - c(n, m) \leq h^*(n) - c(n, m) = h^*(m)$.

Figure 3:

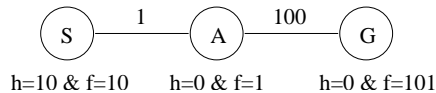


Figure 4: Simple search tree: S is the start node; A is an intermediate node; G is the goal node. Edges are labeled with costs c . Nodes are labelled with h -costs and f -costs. Note that f is *not* monotonically nondecreasing in depth, but A* search visits nodes in order of depth. Thus, A* search does not visit nodes in f -order. With pathmax, however, $f(A) = 10$, and nodes are visited in f -order.

7 IDA* Search

Iterative deepening A* (IDA*) is an optimal search algorithm with the performance properties of A*—it is complete and optimal—and the space requirements of DFS—(essentially) linear in depth. The main idea of iterative deepening A* is to repeatedly search in depth-first fashion, over subgraphs with f -cost less than α , less than 2α , less than 3α , and so on, until a goal is found, where α is a lower bound on the cost between nodes and their successors throughout the search space: i.e., $\alpha \leq c(n, m)$, for all $n, m \in \delta(n)$.

Recall that the space complexity of ID is $O(bd)$, where d is the depth of the goal node. Similarly, the space complexity of IDA* is $O(bg^*/\alpha)$, where g^* is the optimal cost. The time complexity of IDA*, however, can exceed that of A*. In particular, in search spaces where the f -cost is different at every state, only one additional state is expanded during each iteration. In such a search space, if A* expands N nodes, IDA* expands $1 + \dots + N = O(N^2)$ nodes. The typical solution to this problem is to fix an increment $\beta > \alpha$ such that several nodes n have cost $f_i < f(n) \leq f_i + \beta$, where f_i is the i th incremental value of the f -cost. This strategy reduces search time, since the total number of iterations is proportional to $1/\beta < 1/\alpha$, and returns solutions that are at worst β -optimal: i.e., if the algorithm returns m^* , then $g(m^*) < g^* + \beta$.

IDA*(X, S, G, β, c, h)	
Inputs	search problem admissible heuristic h
Output	(path to) optimal goal node
Initialize	$i = 0$ is the cut-off f -value $O = S$ is the list of open nodes
<pre> while (1) do 1. while (O is not empty) do (a) delete <i>first</i> node $n \in O$ (b) if $n \in G$, return (path to) goal n (c) for all $m \in \beta(n)$ i. compute $h(m)$ ii. $g(m) = g(n) + c(m, n)$ iii. $f(m) = g(m) + h(m)$ iv. if $f(m) \leq i$, insert m in <i>front</i> of O 2. increment i by β, $O = S$ </pre>	

Table 3: Iterative Deepening A*.

8 Examples

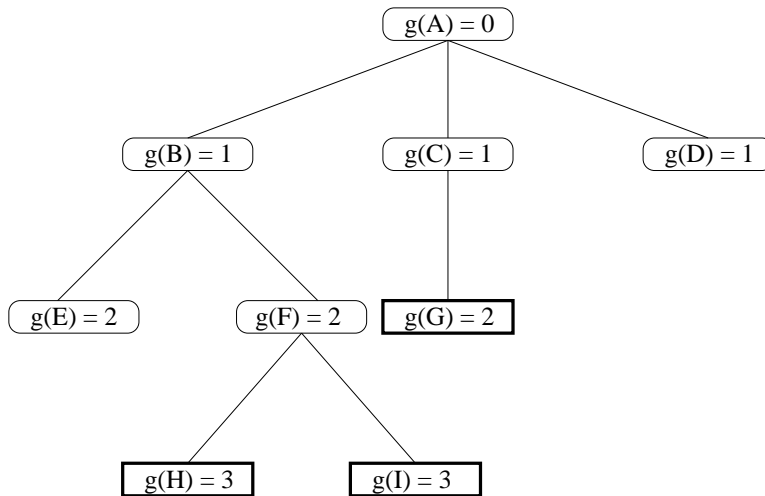


Figure 5: Sample search tree, labeled with costs g . Boxes indicate goal nodes. Best- g returns the optimal goal node G.

Best- g Search The tree shown in Figure 5 has cost function $g(n) = \text{depth}(n)$. Best- g on this search space is precisely BFS: it finds the optimal goal node G. Nodes are expanded as follows: A, BCD, CDEF, DEFG, EFG, FG, GHI, GOAL!

Best- h Search The tree depicted in Figure 6 has cost function $h(n)$. Best- h search returns the suboptimal goal node H in this example. The priority queue is maintained as follows: A, BCD, EFCD, FCD, HICD, GOAL!

A* and IDA* Search The tree depicted in Figure 7 has cost function $f(n) = g(n) + h(n)$. A* search returns the optimal goal node G in this example. Nodes are expanded as follows: A, BCD, ECFD, CFD, GFD, GOAL! Or, if ties are broken otherwise, nodes could be expanded in an alternative order: A, BCD, CEDF, EGDF, GDF, GOAL! Since h is admissible, A* is optimal. IDA* expands nodes as follows, for $\beta = 1$: $f = 0$: A; $f = 1$: AB; $f = 2$: ABECG, GOAL!

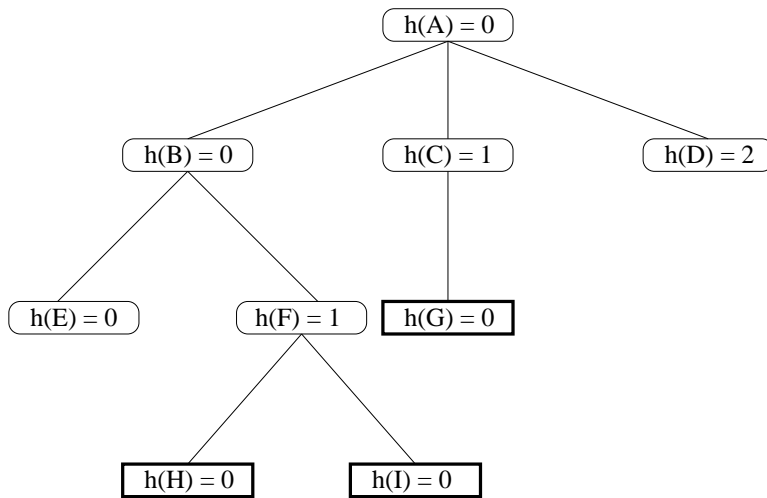


Figure 6: Sample search tree, labeled with heuristic values h . Boxes indicate goal nodes. best- h search returns the suboptimal goal node H.

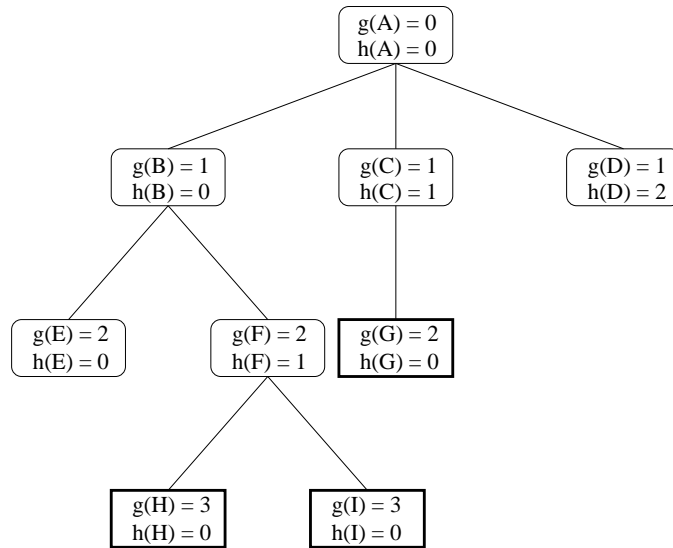


Figure 7: Sample search tree, labeled with costs g and heuristic values h . Boxes indicate goal nodes. A* search returns the optimal goal node G.

9 Summary

Criteria	Best- g
Time	$O(b^d)$: BFS, if $g = \text{depth}$
Space	$O(b^d)$: BFS, if $g = \text{depth}$
Completeness	YES, if there do not exist ∞ -many nodes n s.t. $g(n) < g^*$
Optimality	YES, if g is monotonically nondecreasing in depth

Criteria	Best- h
Time	$O(b^d)$: BFS, if $h = \text{depth}$
Space	$O(b^d)$: BFS, if $h = \text{depth}$
Completeness	NO, if nodes are visited in DFS order
Optimality	NO, if nodes are visited in DFS order

Criteria	A*
Time	$O(b^d)$: BFS, if $g = \text{depth}$ and $h = 0$
Space	$O(b^d)$: BFS, if $g = \text{depth}$ and $h = 0$
Completeness	YES, if there do not exist ∞ -many nodes n s.t. $f(n) < f^*$
Optimality	YES, if h is admissible and g is monotonically nondecreasing in depth

Criteria	IDA*
Time	$O(N^2)$, if f -costs differ at all states and A* expands N nodes
Space	$O(bg^*/\beta)$, if f is monotonically nondecreasing in depth and if g^* optimal is the optimal cost
Completeness	YES, if there do not exist ∞ -many nodes n s.t. $f(n) < f^* + \beta$
β -Optimality	YES, if h is admissible and g is monotonically nondecreasing in depth

Problems

#1 Which of the following are admissible, given admissible heuristics h_1, h_2 ?

- $h(n) = \min\{h_1(n), h_2(n)\}$
- $h(n) = wh_1(n) + (1 - w)h_2(n)$, where $0 \leq w \leq 1$
- $h(n) = \max\{h_1(n), h_2(n)\}$

#2 Consider the algorithm wA^* , which is a variant of A* search that uses the following weighted cost function: for some $w \geq 1$,

$$f_w(n) = g(n) + wh(n)$$

As usual, g is the cost from the root node to n and h is an admissible heuristic.

(a) Prove that the goal node m^* returned by wA^* search on trees is within a factor of w of the optimal goal n^* : i.e., $g(m^*) \leq wg(n^*)$.

(b) The wA^* algorithm uses a weighted cost function that increases the value of the heuristic function h , hoping to proceed more directly towards a goal. An alternative is to ignore g entirely, simply letting $f = h$. Give two advantages of wA^* over this alternative.

(c) An *anytime* algorithm is one whose solution quality improves over time, but can nonetheless be interrupted to return a (perhaps suboptimal) solution at any time. A^* search is not an anytime algorithm. Design an anytime search algorithm based on wA^* .

#3 The pseudocode in Table 4 implements beam search (on trees), a memory-bounded heuristic variant of breadth-first search.

BEAM(X, S, G, δ, c, w, h)	
Inputs	search problem beam width w heuristic h
Output	(path to) goal node
Initialize	$O = S$ is the list of open nodes P is the beam (i.e., priority queue)
<pre> while (O is not empty) do $P = O, O = \emptyset$ while (P is not empty) do 1. delete node $n \in P$ 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) insert node m into O with priority $f(m)$ (e) truncate O to maximum beam width w fail </pre>	

Table 4: Beam Search.

(a) Give the time and space complexity of beam search in terms of branching factor b , depth d , and beam width w .

(b) Argue whether or not beam search is optimal and complete.

BNB(X, S, T, G, δ, c, h)	
Inputs	search problem heuristic function h
Output	optimal goal node's value
Initialize	$v^* = +\infty$
for all $s \in S$	
1. $v = \text{REC-BNB}(X, S, T, G, \delta, c, +\infty, h, s)$	
2. if $v < v^*$, let $v^* = v$	
return v^*	
REC-BNB($X, S, T, G, \delta, c, \beta, h, n$)	
Inputs	search problem upper bound β heuristic function h subtree's root node n
Output	optimal goal node's value
1. if $n \in G$, return $g(n)$	
2. if $n \in T$, return $+\infty$	
3. $f(n) = g(n) + h(n)$	
4. if $f(n) \geq \beta$, return β	
5. for all $m \in \delta(n)$	
(a) $\beta = \min\{\beta, \text{REC-BNB}(X, S, T, G, \delta, c, \beta, h, m)\}$	
6. return β	

Table 5: Branch and Bound.

#4 *Branch-and-Bound* is an optimization algorithm that relies on an heuristic function, much like A* search. The pseudocode shown in Table 5 is a recursive implementation of branch-and-bound (on trees of finite depth with terminal nodes $T \subseteq X$).

This implementation returns only the optimal value, not the path to an optimal goal. This value is initialized as $\beta = +\infty$, which serves as an upper bound on the value of an optimal goal throughout search. In step 4, the subtree rooted at node n is pruned, if ever $f(n) \geq \beta$. Branching occurs within the recursive call in step 5. The (global) upper bound is refined in step 5a based on the values returned as goals are encountered. Ultimately, the algorithm returns the value β , the minimum value among all goals encountered.

- (a) Argue that branch-and-bound returns an optimal value if h is admissible.
- (b) Comment on the space complexity of BNB vs. the space complexity of A*.
- (c) Comment on the total number of nodes expanded by BNB vs. the total number of nodes expanded by A*.