

# Lecture 03: Heuristic Search on Trees

*TBA*

## Contents

<b>1</b>	<b>Overview</b>	<b>1</b>
<b>2</b>	<b>Search Problem</b>	<b>1</b>
<b>3</b>	<b>Best-First Search</b>	<b>2</b>
<b>4</b>	<b>Best-<math>g</math> Search</b>	<b>3</b>
<b>5</b>	<b>Best-<math>h</math> Search</b>	<b>3</b>
<b>6</b>	<b>A* Search</b>	<b>4</b>
6.1	Admissibility . . . . .	4
6.2	Optimality . . . . .	5
6.3	Completeness . . . . .	6
<b>7</b>	<b>Examples</b>	<b>6</b>
<b>8</b>	<b>Summary</b>	<b>8</b>

## 1 Overview

In this lecture, we expand 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. Note that unlike blind search algorithms, which are equally applicable to any basic search problem, heuristic search algorithms make use of domain-specific knowledge.

## 2 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 all  $y \in \delta(x)$ ,  $c(x, y)$  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  $0 \leq i \leq k$ , we write  $g(n)$  to denote 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  $g(n) = \text{distance}(n)$ .

### 3 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  $e : X \rightarrow \mathbb{R}$ .

BEST-FIRST( $X, S, G, \delta, c, e$ )	
Inputs	search problem evaluation function $e$
Output	(path to) goal node
Initialize	$O = S$ is the priority queue of open nodes
<pre> while (<math>O</math> is not empty) do   1. delete node <math>n \in O</math> s.t. <math>e(n)</math> is minimal   2. if <math>n \in G</math>, return (path to) <math>n</math>   3. for all <math>m \in \delta(n)</math>       (a) compute <math>e(m)</math>       (b) insert <math>m</math> into <math>O</math> with priority <math>e(m)</math> 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  $n$ . For this choice of evaluation function, the complexity of best-first search in the worst-case is that of BFS: exponential in the depth of the goal for both time and space.

Best-first search can also visit nodes in depth-first search order: *e.g.*, let  $e(n) = 0$ , for all  $n \in X$ . For this choice of evaluation function, best-first search is not complete; nor is it optimal.

### 4 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. (See Figure 3.)

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. The search tree on the LHS contains an infinite path of finite cost. Best- $g$  search never reaches either goal node. The search tree on the RHS contains an edge of negative cost. Best- $g$  search proceeds directly to the suboptimal goal node  $Y$ .

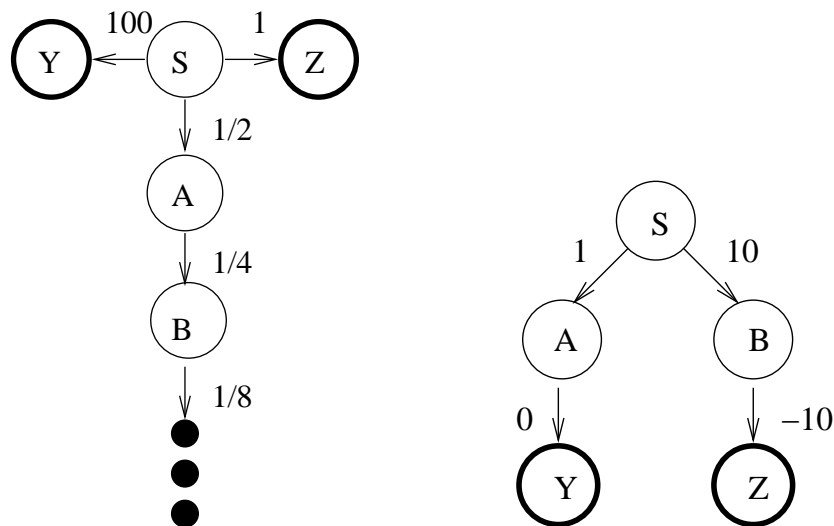


Figure 1: (LHS) A search space that contains an infinite path of finite cost. (RHS) A search space that contains an edge of negative cost. In both search spaces,  $S$  is the start state,  $Y$  and  $Z$  are goal nodes, and  $Z$  is optimal.

### 5 Best- $h$ Search

The main idea of best- $h$  search is to expand the lowest-cost node on the fringe, according to some domain-specific heuristic function  $h$ . Heuristics are used to guide the search process. The degree of optimality of best- $h$  search depends on the quality of the heuristic, as do any completeness guarantees. (See Figure 4.)

A heuristic function  $h : X \rightarrow \mathbb{R}$  computes an estimate of the distance from node  $n$  to a goal node. 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.

1	3	5
7	2	4
6	8	

1	2	3
4	5	6
7	8	

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 ( $h_2(3) = 1$ ,  $h_2(5) = 2$ ,  $h_2(7) = 1$ ,  $h_2(2) = 1$ ,  $h_2(4) = 2$ ,  $h_2(6) = 3$ ).

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

## 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 a 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  $f$ . (See Figure 5.) 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.

$A^*(X, S, G, \delta, c, h)$	
Inputs	search problem heuristic function $h$
Output	(path to) optimal goal node
Initialize	$O = S$ is the priority queue of open nodes
<b>while</b> ( $O$ is not empty) <b>do</b>	
1. delete node $n \in O$ <i>s.t.</i> $f(n)$ 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 $f(m)$	
<b>fail</b>	

Table 2: A\* Search.

### 6.1 Admissibility

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  $h_1$ .

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

**Exercise:** 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  $h_1$ .

## 6.2 Optimality

We now prove that A\* search is optimal, assuming the heuristic function  $h$  is nonnegative and admissible.

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

**Theorem:** If  $h$  is nonnegative and  $\epsilon$ -admissible, then A\* search is  $\epsilon$ -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  $n$  (possibly  $n^*$  itself) on the priority queue that is also on the path to  $n^*$  *s.t.*  $f(m^*) \leq f(n)$ . But then  $m^*$  is  $\epsilon$ -optimal (*i.e.*,  $g(m^*) \leq g(n^*) + \epsilon$ ), by the following reasoning:

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

◇

**Corollary:** If the heuristic function  $h$  is nonnegative and admissible, then A\* search is optimal.

**Proof:** Let  $\epsilon = 0$ .

As long as we assume that all costs are nonnegative (i.e.,  $c(n, m) \geq 0$ , for all nodes  $n$  and their successors  $m$ ), then any reasonable heuristic  $h$  will also be nonnegative. Why admit negative heuristic values if costs themselves are never negative?

Note also that nothing in the algorithm’s specification precludes running A\* search with an inadmissible heuristic. Indeed, it is often implemented in this way in the interest of time, but then it is not guaranteed to return an optimal goal.

### 6.3 Completeness

A\* search is complete in search spaces that do not contain infinitely many nodes  $n$  with  $f(n) < f^*$  (e.g., an infinite path of finite value), where  $f^*$  is the  $f$ -cost of an optimal goal.

## 7 Examples

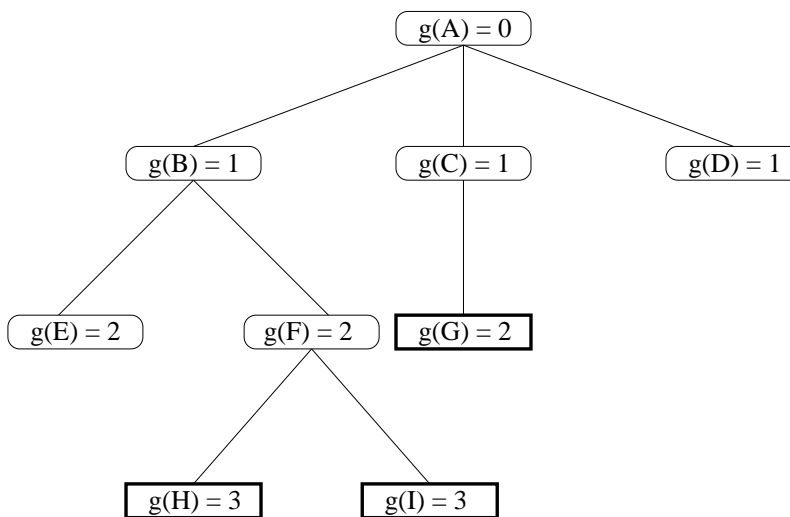


Figure 3: Sample search tree, labeled with costs  $g$ . Boxes indicate goal nodes. Best- $g$  returns the optimal goal node G.

**Best- $g$  Search** The tree shown in Figure 3 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:  $A_0, B_1C_1D_1, C_1D_1E_2F_2, D_1E_2F_2G_2, E_2F_2G_2, F_2G_2, G_2H_3I_3, \text{GOAL!}$

**Best- $h$  Search** The tree depicted in Figure 4 has cost function  $h(n)$ . Best- $h$  search returns the suboptimal goal node H in this example. The priority queue is maintained as follows:  $A_0, B_1C_1D_1, E_2F_2C_1D_1, F_2C_1D_1, H_3I_3C_1D_1, \text{GOAL!}$

**A\* and IDA\* Search** The tree depicted in Figure 5 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:  $A_0$ ,  $B_1C_2D_3$ ,  $E_2C_2F_3D_3$ ,  $C_2F_3D_3$ ,  $G_2F_3D_3$ , GOAL! Or, if ties are broken otherwise, nodes could be expanded in an alternative order:  $A_0$ ,  $B_1C_2D_3$ ,  $C_2E_2D_3F_3$ ,  $E_2G_2D_3F_3$ ,  $G_2D_3F_3$ , GOAL! Since  $h$  is admissible, A\* is optimal. IDA\* expands nodes as follows, for  $\beta = 1$ :  $f = 0$ :  $A_0$ ;  $f = 1$ :  $A_0B_1$ ;  $f = 2$ :  $A_0, B_1C_2, E_2C_2, C_2, G_2$ , GOAL!

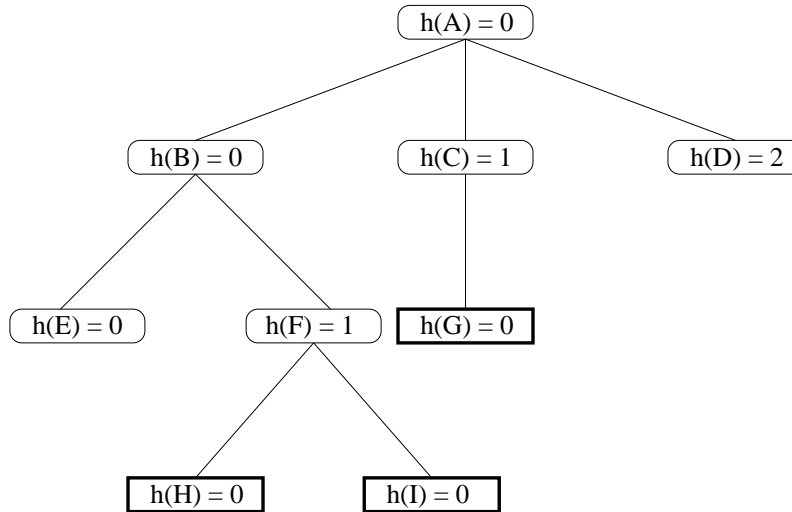


Figure 4: Sample search tree, labeled with heuristic values  $h$ . Boxes indicate goal nodes. Best- $h$  search returns the suboptimal goal node H.

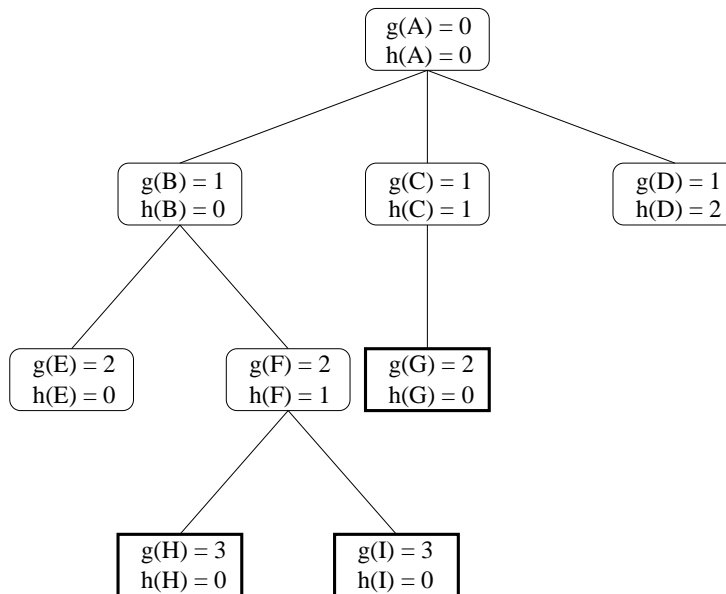


Figure 5: Sample search tree, labeled with costs  $g$  and heuristic values  $h$ . Boxes indicate goal nodes. A\* search returns the optimal goal node G.

## 8 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 $\infty$ -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 $\infty$ -many nodes $n$ s.t. $f(n) < f^*$
Optimality	YES, if $h$ is nonnegative and admissible, which makes sense if $c \geq 0$ , which implies $g$ is monotonically nondecreasing in depth