Best-First Search with a Maximum Edge Cost Function

P. Alex Dow and **Richard E. Korf**
Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90095
alex_dow@cs.ucla.edu, korf@cs.ucla.edu

Abstract

Best-first search is a general search technique that uses an evaluation function to determine what nodes to expand. A* is a well-known best-first search algorithm for finding least-cost solution paths on search problems where the cost of a solution path is the sum of the edge costs. In this paper, we focus on search problems where the cost of a solution path is the maximum edge cost. We present an algorithm, MaxBF, that is analogous to A* but meant to solve these maximum edge cost problems.

We show that the evaluation function used by MaxBF does not meet a condition for the admissibility of best-first search algorithms given by Dechter & Pearl (1985). Additionally, we show that that condition can be loosened to include the MaxBF evaluation function without sacrificing admissibility.

Another result shows that, while many choices of heuristic function may require A* to reopen closed nodes, a heuristic need only be optimistic to guarantee that it is never beneficial for MaxBF to reopen closed nodes.

Finally, we show that, although MaxBF never needs to reopen closed nodes, it may find an alternate path to a closed node that appears better than the original path. This implies that a naive version of MaxBF could unnecessarily reopen closed nodes. We give a specification for MaxBF that carefully avoids this inefficiency.

1 Introduction and Overview

Best-first search is a general search technique that uses an evaluation function to decide which nodes to expand. At any point in time, a node that the evaluation function deems "best" will be expanded. A classic best-first search algorithm is Dijkstra's single-source shortest-path algorithm, which uses, as an evaluation function, the sum of the edge costs from the source node to the current node (Dijkstra, 1959).

Dechter & Pearl (1985) present an algorithm, BF*, which implements best-first search without specifying a specific evaluation function. BF* places two conditions on the choice of evaluation function which allow it to take several shortcuts and still return an optimal solution. These shortcuts increase its efficiency over a completely general best-first search algorithm. The well-known best-first search algorithm A* is a specialization of BF*, where the evaluation function is $f(n) = g(n)+h(n)$, the function $g(n)$ is the sum of the edge costs on the path taken from the root to n, and the function $h(n)$ is estimate of the sum of the edge costs on a shortest path from n to a goal node.

In this paper, we present an algorithm, MaxBF, that is a slightly altered specialization of BF* with the evaluation function $f(n) = \max(g(n), h(n))$. Whereas A* is typically employed on problems where a complete solution path, from the root node to a goal node, has a cost equal to the sum of the edge costs, MaxBF is meant to be employed on problems where the cost of a complete solution path is the maximum edge cost. Therefore, for MaxBF, the function $g(n)$ denotes the maximum edge cost on the path taken from the root node to n, and $h(n)$ is an estimate of the maximum edge cost on any path from n to any goal node.

An optimistic heuristic is sufficient for guaranteeing that A* meets the conditions for admissibility set forth by Dechter & Pearl (1985). We show that this is not the case with MaxBF. Although, by demonstrating that these admissibility conditions can be weakened to include more algorithms, we show that MaxBF with an optimistic heuristic is, in fact, admissible.

Additional contributions in this paper relate to MaxBF's behavior with respect to reopening closed nodes. Since a node is considered closed when it has already been expanded, reopening closed nodes amounts to allowing the same nodes to be expanded multiple times. Any graph search algorithm, including A* and MaxBF, that reopens closed nodes runs the risk of searching large portions of the search space multiple times. We show that it is never necessary for MaxBF with an optimistic heuristic to reopen a closed node in order to find an optimal solution. We also show that, although it is unnecessary, a naive version of MaxBF may reopen closed nodes. Additionally, we show how to avoid this inefficiency.

Finally, we describe a real-world problem space for which the cost of a solution path is the maximum edge cost. This problem, finding the treewidth and an optimal vertex elimination order of a general graph, can be solved by MaxBF.

2 Notation and Preliminaries

The purpose of best-first search is to find a least-cost sequence of transitions from a start, or *root*, state to a *goal* state. A graph is used to represent the space that is searched, where vertices are states and edges are transitions. In many problems, a transition is associated with a cost. In the search graph, this cost is attached to the corresponding edge. A *path* through the search graph corresponds to a sequence of transitions applied to successive states. A *solution path*, from the root to a goal state, corresponds to a solution to the problem. A *node* refers to a state when reached by a particular path from the root node. If a search graph is a tree, then there is exactly one node for each state in the problem space. If a search graph is not a tree, then there is more than one path to at least some states in the problem space, thus there are multiple nodes that correspond to the same state. Two nodes that include distinct paths to the same state are referred to as *duplicate nodes*. A tree expansion of a search graph is the tree that results from representing each node as a distinct vertex.

When best-first search applies a transition to a node, it *generates* a new node. When a node is generated, it saves a pointer back to its parent node. Thus, at any time in the search, the path that is found by following parent pointers from a generated node back to the root is referred to as the node's *current path*.

To remain consistent with Dechter & Pearl (1985) we use the following notational conventions. One significant difference is that we define $g(n)$ in terms of a maximum edge cost function as opposed to an additive edge cost function.

$C(\cdot)$	Cost function, defined over solution paths.
$c(n,m)$	The cost of an edge between n and m.
$f(n)$	Evaluation function, defined over nodes.
$g(n)$	The maximum of the edge costs along the current path from the root node to a node n.
$h(n)$	A cost estimate of a cheapest path remaining between n and any goal state.
$k(n,m)$	The cost of the cheapest path from n to m.
r	Root node.

Since a path includes the node it ends on, we can substitute a path for a node in our notation. For example, if P is a path from r to n, then we say that $f(P) = f(n)$.

We now state several lemmas that we will use to prove results later in the paper. The variables in the following lemmas are any real numbers.

Lemma 1
If $a = x$ *and* $b \leq y$, *then* $\max(a,b) \leq \max(x,y)$.

Lemma 2
$\max(a, b_1, b_2, \ldots, b_n) < \max(a,c)$ *if and only if* $a < c$ *and* $b_i < c$ *where* $1 \leq i \leq n$.

Lemma 3
If $\max(a,b) \geq \max(x,y)$ *and* $a \geq b$, *then* $a \geq x$ *and* $a \geq y$.

Lemma 4
$a \geq \max(b,c)$ *if and only if* $a \geq b$ *and* $a \geq c$.

3 Prior Work

Dechter & Pearl (1985) describe a general best-first search algorithm called BF* (see also Pearl, 1984). It uses a *closed list* to store nodes that have been expanded, and an *open list* to store nodes that have been generated but not yet expanded.

Algorithm 1 BF*

```
1: Insert $r$ in OPEN
2: while OPEN is not empty do
3:     Remove a node $n$ with minimal $f(n)$ from OPEN and insert it in CLOSED
4:     if $n$ is a goal node then
5:         Exit successfully, solution obtained by tracing pointers from $n$ to $r$
6:     end if
7:     Expand $n$, generating children with pointers to $n$
8:     for all children $m$ of $n$ do
9:         Calculate $f(m)$
10:        if $m \notin$ OPEN and $m \notin$ CLOSED then
11:            Insert $m$ in OPEN
12:        else if $m \in$ OPEN and $f(m) <$ $f$-value of duplicate in OPEN then
13:            Remove duplicate from OPEN and insert $m$
14:        else if $m \in$ CLOSED and $f(m) <$ $f$-value of duplicate in CLOSED then
15:            Remove duplicate from CLOSED and insert $m$ in OPEN
16:        else
17:            Discard $m$
18:        end if
19:    end for
20: end while
21: Exit with failure, no solution exists
```

An *evaluation function*, denoted $f(n)$, is used to estimate the cost of any complete solution path that extends the current path of a node n. BF* begins with only the root node on the open list, and it progresses with a loop that chooses a node with the *best* f-value in the open list for expansion. The node is removed from the open list and each of its children are generated. For each child node that is generated, BF* checks the open and closed lists to ensure that it is not a duplicate of a previously generated node or, if it is, that it represents a better path than the previously generated duplicate. If that is the case, then the node is inserted into the open list. After every child node has been generated, the parent node is inserted into the closed list and a new node is chosen for expansion. This continues until a goal node is chosen for expansion. See Algorithm 1 for pseudocode

BF* is referred to as a general best-first search algorithm, because the evaluation function, $f(\cdot)$, is left unspecified. Thus it is up to specialized algorithms to add an evaluation function for determining in what order to expand nodes. Dechter & Pearl (1985) ensure that BF* is admissible when $f(\cdot)$ is optimistic and strongly order preserving.[1]

Definition Let n be any node in the search graph, and P_n be a least-cost solution path from the root to a goal node among those that include the current path to n. An evaluation function $f(\cdot)$ is called *optimistic* if $f(n) \leq C(P_n)$.

[1]Dechter & Pearl (1985) just call this "order preserving"; we add "strongly" to differentiate from the upcoming definition of "weakly order preserving".

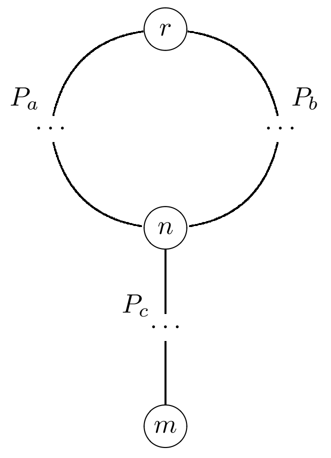


Figure 1: The nodes and paths referred to in the definition of strong order preservation. This also applies to the definition weak order preservation, where m is γ and P_c is P_γ.

An optimistic evaluation function allows BF* to terminate the first time a goal node is chosen for expansion.

Definition Let n and m be any two nodes in the search graph; and let P_a, P_b, and P_c, be any paths such that P_a and P_b are two paths from r to n, and P_c is a path from n to m. See Figure 1. An evaluation function $f(\cdot)$ is called *strongly order preserving (SOP)* if the following holds

$$f(P_a) \geq f(P_b) \Rightarrow f(P_aP_c) \geq f(P_bP_c)$$

where P_iP_j is the concatenation of P_i and P_j.

An SOP evaluation function ensures that when BF* finds a least-cost path from the root to a node, that path can be extended into a least-cost path from the root to any other node. This allows BF* to save only a single best path found to a node, since no other path can lead to a better solution.

In Section 5 we introduce a specialized version of BF* with the evaluation function $f(n) = \max(g(n), h(n))$. This specialization has been studied in prior work (Dow & Korf, 2007). The relationship between that prior work and the results in this paper is discussed in Sections 5 and 9.

4 A Weaker Condition for Admissibility

An SOP evaluation function is actually stronger than it needs to be. While an SOP evaluation function guarantees that a least-cost path from the root to a node can be extended into a least-cost path from the root to any other node, we only care about least-cost paths from the root to goal nodes. It is actually acceptable for an evaluation function to disregard duplicate nodes that offer better paths to a node's descendants, as long as those descendants are not goal nodes.

Definition Let n be any node in the search graph; γ be any goal node; and P_a, P_b, and P_γ be any paths such that P_a and P_b are two paths from r to n, and P_γ is a path from n to γ. To illustrate this see Figure 1, but let m and P_c in the figure be γ and P_γ, respectively. An evaluation function $f(\cdot)$ is called *weakly order preserving (WOP)* if the following holds

$$f(P_a) \geq f(P_b) \Rightarrow f(P_aP_\gamma) \geq f(P_bP_\gamma)$$

where P_iP_j is the concatenation of P_i and P_j.

BF* with a WOP evaluation function will only discard a node if it is storing a duplicate node that could result in a solution that is as good or better than the one being discarded. Thus, to show that BF* is admissible with a given evaluation function, it is sufficient to show that the evaluation function is optimistic and weakly order preserving.

Since a goal node γ qualifies as the node m in the definition of SOP, SOP implies WOP. In the next section, we will present an evaluation function that is WOP but not SOP.

5 NaiveMaxBF

Here we describe a version of BF* that is meant to search a problem space where the cost of a solution path is the maximum edge cost, that is, for a solution path $P = (r, n_1, \ldots, \gamma)$

$$C(P) = \max(c(r, n_1), c(n_1, n_2), \ldots, c(n_k, \gamma)). \quad (1)$$

We will refer to search over a space with this cost function as a *max-edge-cost* search. The algorithm, which we call NaiveMaxBF, is the result of using the following evaluation function in line 9 of Algorithm 1:

$$f(n) = \max(g(n), h(n)) \quad (2)$$

where n is reached from the root node via the path $(r, n_1, n_2 \ldots, n)$, and

$$g(n) = \max(c(r, n_1), c(n_1, n_2), \ldots, c(n_{i-1}, n)). \quad (3)$$

We will frequently analyze the behavior of NaiveMaxBF when the heuristic function, $h(\cdot)$, is optimistic. That is,

$$h(n) \leq k(n, \gamma) \text{ for all goal nodes } \gamma, \quad (4)$$

where $k(a, b)$ is the cost of a least-cost path from node a to node b. Notice that a heuristic is optimistic when it underestimates the cost of paths from a node to any goal, whereas an evaluation function is optimistic when it underestimates the cost of any complete solution path through a node.

The significance of the word "naive" in the algorithm's name reflects the fact that it merely adds an evaluation function to BF*, whereas, in later sections, we show how a minor modification to the algorithm can improve its efficiency.

To demonstrate that NaiveMaxBF is admissible, we must show that (2) is optimistic and, at least, weakly order preserving. Additionally, to demonstrate that the WOP property applies to a larger set of evaluation functions than SOP, we show conditions under which (2) is not SOP.

Lemma 5 *If the heuristic function $h(n)$ is optimistic, then the evaluation function $f(n) = \max(g(n), h(n))$ is optimistic.*

Proof Let $P_n = P_{r-n}P_{n-\gamma}$, where $P_{r-n} = (r, n_1, \ldots, n)$ and $P_{n-\gamma} = (n, n_{i+1}, \ldots, \gamma)$. We slightly abuse notation by saying that

$$\begin{aligned} c(P_{r-n}) &= \max(c(r, n_1), \ldots, c(n_{i-1}, n)), \text{ and} \\ c(P_{n-\gamma}) &= \max(c(n, n_{i+1}), \ldots, c(n_k, \gamma)). \end{aligned}$$

Notice that $g(n) = c(P_{r-n})$, by (3), and

$$\begin{aligned} h(n) &\leq k(n,\gamma') \text{ for all goal nodes } \gamma'\text{, and} \\ k(n,\gamma') &\leq c(P_{n-\gamma}) \end{aligned}$$

because a least cost path from n to any goal node cannot cost more than any specific path. Thus $h(n) \leq c(P_{n-\gamma})$. Furthermore,

$$\begin{aligned} f(n) &= \max(g(n), h(n))\text{, thus} \\ f(n) &\leq \max(c(P_{r-n}), c(P_{n-\gamma})) \text{ by Lemma 1, and} \\ f(n) &\leq C(P_n) \end{aligned}$$

because $C(P_n) = \max(c(P_{r-n}), c(P_{n-\gamma}))$, by (1). ∎

In previous work, we have shown that (2) is strongly order preserving if the heuristic function satisfies a version of the triangle inequality, referred to as *max-consistency* (Dow & Korf, 2007). A heuristic that is max-consistent is necessarily optimistic, therefore this result allows for the possibility that heuristics that are optimistic but not max-consistent cause the algorithm to return suboptimal solutions. We now show that this is not the case, because (2) with an optimistic heuristic is weakly order preserving.

Lemma 6 *If the heuristic function $h(n)$ is optimistic, the the evaluation function $f(n) = \max(g(n), h(n))$ is weakly order preserving.*

Proof We assume that the lemma is incorrect and show a contradiction. Let n^a correspond to n when reached by P_a, and n^b correspond to n when reached by P_b. Also, let $P_\gamma = (n, n_{i+1}, \ldots, \gamma)$ lead from n to some goal node γ. As a slight abuse of notation, we say that $c(P_\gamma) = \max(c(n, n_{i+1}), \ldots, c(n_k, \gamma))$. Thus, we assume

$$\begin{aligned} f(P_a) &\geq f(P_b)\text{, i.e.,} \\ \max(g(n^a), h(n^a)) &\geq \max(g(n^b), h(n^b)) \qquad (5) \end{aligned}$$

and

$$\begin{aligned} f(P_aP_\gamma) &< f(P_bP_\gamma)\text{, i.e.,} \\ \max(g(n^a), c(P_\gamma)) &< \max(g(n^b), c(P_\gamma)), \qquad (6) \end{aligned}$$

because the optimism of $h(\cdot)$ implies that $h(\gamma) \leq 0$. From Lemma 2 we see that (6) implies that

$$\begin{aligned} c(P_\gamma) &< g(n^b)\text{, and} \qquad (7) \\ g(n^a) &< g(n^b). \qquad (8) \end{aligned}$$

We now consider two exhaustive cases and show that both lead to contradictions.

Case 1: $\mathbf{g(n^a) \geq h(n^a)}$ In this case, with Lemma 3 and (5), it follows that $g(n^a) \geq g(n^b)$, which contradicts (8).

Case 2: $\mathbf{g(n^a) < h(n^a)}$ In this case, with Lemma 3 and (5), it follows that

$$\begin{aligned} g(n^b) &\leq h(n^a)\text{, and} \\ g(n^b) &\leq k(n,\gamma')\text{, for all goal nodes } \gamma', \end{aligned}$$

because $h(\cdot)$ is optimistic. This implies that $g(n^b) \leq c(P_\gamma)$, which contradicts (7). ∎

Since an optimistic heuristic is sufficient for guaranteeing that the evaluation function given in (2) is optimistic and weakly order preserving, NaiveMaxBF with an optimistic heuristic function is admissible. The next lemma shows that (2) with an optimistic heuristic function is not necessarily strongly order preserving. Therefore, the distinction between strong and weak order preservation in the previous section is significant.

Lemma 7 *An optimistic heuristic $h(\cdot)$ is insufficient for making the evaluation function $f(n) = \max(g(n), h(n))$ strongly order preserving.*

Proof To show that (2) with an optimistic heuristic is not necessarily an SOP evaluation function, we will give a counter-example in which the SOP condition is violated. We will use the same notation as in the proof of Lemma 6. Also, let $P_c = (n, n_{i+1}, \ldots, m)$ lead from n to some node m. Let m^a be m reached by P_aP_c, and m^b be m reached by P_bP_c.

Consider the graph shown in Figure 1 with the following path costs

$$\begin{aligned} c(P_a) &= 1 \\ c(P_b) &= 2 \\ c(P_c) &= 1 \end{aligned}$$

and the following heuristic function values

$$\begin{aligned} h(n) &= 3 \\ h(m) &= 1 \end{aligned}$$

For this example to be valid, the heuristic function must be optimistic. Thus, assume that any path to any goal node that follows from node m has at least one edge with cost at least 3. Notice that this heuristic function violates a max-cost version of the triangle inequality. Although this is typically not desirable, there are certainly heuristic functions for which it is the case.

First, notice that

$$\begin{aligned} f(n^a) &= \max(1,3) = 3\text{, and} \\ f(n^b) &= \max(2,3) = 3. \end{aligned}$$

Thus,

$$f(n^a) \geq f(n^b)$$

Second, notice that

$$\begin{aligned} f(m^a) &= \max(1,1) = 1\text{, and} \\ f(m^b) &= \max(2,1) = 2. \end{aligned}$$

Thus,

$$f(m^a) < f(m^b)$$

which contradicts the definition of SOP. ∎

6 NaiveMaxBF Does Not Need to Reopen Closed Nodes

When BF* generates a node that is a duplicate of a node in the closed list, it is possible that the new node represents a better partial solution path than the old node. This is detected when the new node's f-value is less than the old node's f-value. Since it is possible that the new node will lead to better solutions than the old node, it is necessary to remove the old node from the closed list and insert the new node in the open list. This process is referred to as reopening a closed node, and it is seen in lines 14–15 of Algorithm 1.

An optimal solution in a search with an additive cost function, as is the case with A*, must include a shortest subpath to every node on a complete solution path. For max-edge-cost search, only a single edge cost (the maximum) determines the cost of the entire solution path. Thus, the subpaths from the root to each node on the solution path do not need to be least-cost, as long as they cost no more than the maximum. Consider two distinct paths from the root to a node. If the optimal solution cost is greater than the costs of either of these partial paths, then it doesn't matter which of the two paths is saved and which is discarded. We can detect this with an optimistic heuristic function that lower-bounds the cost of paths from a node to any goal node. Thus, the algorithm MaxBF with an optimistic heuristic function guarantees that once a node is chosen for expansion, no other path through that node can lead to a better solution. Showing this formally is the main result of this section. We begin with several helpful lemmas.

At all times in a best-first search, each node either has not been generated yet, is in OPEN, or is in CLOSED. The following lemma gives a result about the status of certain nodes when a given node is chosen for expansion.

Lemma 8 *During the execution of NaiveMaxBF, for every path P from r to any node n either (1) every node preceding n on P is on the closed list, or (2) there is at least one node preceding n on P that is on the open list.*

Proof Shown by induction on the length of paths from r to n. For the basis, consider all paths from r to n of length one. Since r is the only node preceding n on these paths, and r is always on either OPEN or CLOSED, the lemma holds for paths of length one.

For the inductive hypothesis, assume that the lemma holds for all paths from r to n of length i. Given a path from r to n of length $i+1$, $P_n = (r, n_1, \ldots, n_{i-1}, n_i, n)$, there is a path of length i from r to node n_i: $P_{n_i} = (r, n_1, \ldots, n_{i-1}, n_i)$. Since the lemma holds for paths of length i, we know that either all of the nodes preceding n on P_{n_i} are on CLOSED, or at least one is on OPEN. If there is at least one on OPEN, then the lemma holds for P_n. If all are on CLOSED, then n_i has been generated (when n_{i-1} was expanded) and now resides on either OPEN or CLOSED. Thus the lemma holds for all paths from r to n. ▮

Thus, there is no node preceding n that has not yet been generated and is, itself, only preceded by closed nodes.

Lemma 9 *During the execution of NaiveMaxBF, given two nodes on the open list, n and m, if n is chosen for expansion, then the following holds*

$$\max\left(g(m), h(m)\right) \geq g(n)$$

Proof Line 3 of Algorithm (1) specifies that if n was chosen for expansion, then

$$\begin{aligned} f(m) &\geq f(n), \text{ i.e.,} \\ \max(g(m), h(m)) &\geq \max(g(n), h(n)). \end{aligned}$$

Thus, the lemma follows from this and Lemma 4. ▮

Definition A node n_p is called a *predecessor* of a node n if there is a path from r to n that includes n_p. Note that n_p is not necessarily on the current path to n, and that n may also be a predecessor of n_p.

Lemma 10 *During the execution of NaiveMaxBF, consider a node n on the open list and one of its predecessors n_p also on the open list. Let P be a least-cost solution path among those that include the current path to n. Let P' be a solution path that includes the current path to n_p, and includes n following n_p. If $C(P) > C(P')$, then the following hold*

$$\begin{aligned} g(n) &> g(n_p) && (9) \\ g(n) &> k(n_p, n) && (10) \\ g(n) &> k(n, \gamma), \text{ for all goal nodes } \gamma. && (11) \end{aligned}$$

Proof Observe that

$$\begin{aligned} C(P) &= \max(g(n), k(n, \gamma)), \text{ and} \\ C(P') &\geq \max(g(n_p), k(n_p, n), k(n, \gamma)). \end{aligned}$$

Thus, from the lemma's condition, $C(P) > C(P')$, it follows that

$$\max(g(n), k(n, \gamma)) > \max(g(n_p), k(n_p, n), k(n, \gamma)).$$

The lemma follows from this and Lemma 2. ▮

Finally, we show the main result of this section.

Theorem 11 *The first time a node n is chosen for expansion by NaiveMaxBF, let P be a least-cost solution path among those that include the current path to n. If the heuristic function is optimistic, then there is no solution path P' that includes n such that $C(P') < C(P)$.*

Proof To show by contradiction, assume there exists P' that includes n such that $C(P') < C(P)$.

Let P'_{r-n} be the subpath of P' from r to n. First of all, notice that P'_{r-n} cannot be the current path to n, because that would make $C(P') \geq C(P)$, which is not the case. Also, notice that all of the nodes on P'_{r-n} preceding n cannot be on CLOSED. If they were, that would imply that n had been previously generated with P'_{r-n} as its current path. Since P'_{r-n} is not n's current path now, the old version of n would have had to have been replaced with a superior duplicate in the open list. This only happens if $f(\text{new } n) < f(\text{old } n)$. Since $f(\cdot)$ is weakly order preserving, that would mean that $C(P) \leq C(P')$, which is not the case. Therefore, all of the nodes on P'_{r-n} cannot be on CLOSED. Furthermore, by Lemma 8, at least one node must be on OPEN; call this node n_p.

Notice that both Lemma 9 and Lemma 10 apply to this situation, because both nodes n and n_p are on OPEN, node n is chosen for expansion, and we are assuming that some solution path through n_p and n is better than the best solution path that continues the current path to n. Also, n follows n_p on that path.

We now consider two exhaustive cases and show that they both lead to contradictions.

Case 1: $\mathbf{g(n_p) \geq h(n_p)}$ From Lemma 9, it follows that

$$\max(g(n_p), h(n_p)) \geq g(n).$$

Thus, in this case, $g(n_p) \geq g(n)$, which contradicts (9).

Case 2: $\mathbf{g(n_p) < h(n_p)}$ In this case, with Lemma 9, it follows that

$$\begin{aligned} g(n) &\leq h(n_p), \text{ and} \\ g(n) &\leq \max(k(n_p, n), k(n, \gamma)) \end{aligned}$$

because $h(\cdot)$ is optimistic. This contradicts the fact that both (10) and (11) hold. ∎

Corollary 12 *Once a node has been expanded by NaiveMaxBF with an optimistic heuristic, it never needs to be reopened.*

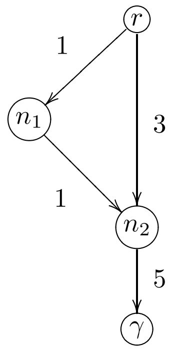
Proof The reason BF* allows a closed node to be reopened is that a new path has been found to a node that may lead to a better solution path than the duplicate node in the closed list. Since Theorem 11 shows that this isn't possible for NaiveMaxBF, there is never a need to reopen a closed node. ∎

7 NaiveMaxBF May Reopen Closed Nodes

In the previous section we showed that NaiveMaxBF with an optimistic heuristic does not need to reopen closed nodes in order to find an optimal solution. Nevertheless, we now will show that, in certain situations, it will reopen closed nodes. Recall that a closed node is reopened when a new path is found to a node on the closed list, such that the new duplicate node has a lesser f-value than the closed node. This is tested for on line 14 of Algorithm 1. Now we show that this can occur during a NaiveMaxBF search.

Theorem 13 *On a problem space where the cost of a solution path is the maximum edge cost, NaiveMaxBF with an optimistic heuristic may find a new path to a node on the closed list that has a lesser f-value than a previous path.*

Proof We demonstrate that this is possible by giving an example where it occurs. Consider the following search graph and corresponding heuristic function.[2]



node	$h(\cdot)$
n_1	4
n_2	2
γ	0

One can easily verify that the given $h(\cdot)$ is an optimistic heuristic. We now trace through several steps of NaiveMaxBF.

[2]This example uses a directed graph, but an undirected graph would lead to the same result.

- OPEN = $\{\mathbf{r}\}$, CLOSED = $\{\}$
  Initially r is chosen for expansion, removed from OPEN, and inserted in CLOSED. Nodes n_1 and n_2 are generated and inserted in OPEN with the following f-values
  $$g(n_1) = 1, \quad f(n_1) = \max(1,4) = 4$$
  $$g(n_2) = 3, \quad f(n_2) = \max(3,2) = 3$$
- OPEN = $\{\mathbf{n_1}, \mathbf{n_2}\}$, CLOSED = $\{\mathbf{r}\}$
  Node n_2 is chosen for expansion, removed from OPEN, and inserted in CLOSED. Node γ is generated and inserted in OPEN with the following f-value
  $$g(\gamma) = \max(3,5) = 5, \quad f(\gamma) = \max(5,0) = 5$$
- OPEN = $\{\mathbf{n_1}, \gamma\}$, CLOSED = $\{\mathbf{r}, \mathbf{n_2}\}$
  Node n_1 is chosen for expansion, removed from OPEN, and inserted in CLOSED. A duplicate of node n_2 is generated, which we will call n_2', with the following f-value
  $$g(n_2') = \max(1,1) = 1, \quad f(n_2') = \max(1,2) = 2$$

Notice that $f(n_2') < f(n_2)$, thus NaiveMaxBF has found a path to node n_2 that has a lesser f-value than the current path of n_2 when it was expanded. ∎

If we continue the example in the proof, n_2 will be reopened, and, in the next step, it will be expanded for a second time. In the next section we will show how a minor modification to the algorithm can make it avoid this sort of redundant search.

8 MaxBF

Theorems 11 and 13 say that, although it is never beneficial for NaiveMaxBF with an optimistic heuristic to reopen closed nodes, it may still do so. When a closed node is reopened, it is possible that the algorithm will re-search the entire subtree rooted at that node. The amount of effort wasted on node reopening and reexpansion is problem dependent, though in pathological cases the algorithm would unnecessarily search the tree expansion of the search graph.

We can eliminate this inefficiency simply by never reopening closed nodes. This is done by removing lines 14 to 15 from Algorithm 1. We refer to this algorithm as MaxBF and include the pseudocode in Algorithm 2.

Whereas NaiveMaxBF simply added a specific evaluation function to BF*, as does A*, MaxBF accounts for the fact that, with (2) as an evaluation function and an optimistic heuristic, allowing closed nodes to be reopened can only lead to unnecessary work.

9 Application: Treewidth

Treewidth is a fundamental property of an undirected graph that is meant to measure how much a graph is like a tree. A tree itself has a treewidth of one, while a clique with n vertices has a treewidth of $n - 1$. One way of finding the treewidth of a graph is by searching over the space of vertex elimination orders. As we will see, this search space uses a maximum edge cost function.

Eliminating a vertex from a graph is defined as the process of adding an edge between every pair of the vertex's neighbors that are not already adjacent, then removing the vertex

Algorithm 2 MaxBF

```
1:  Insert $r$ in OPEN
2:  **while** OPEN is not empty **do**
3:      Remove a node $n$ with minimal $f(n)$ from OPEN and insert it in CLOSED
4:      **if** $n$ is a goal node **then**
5:          Exit successfully, solution obtained by tracing pointers from $n$ to $r$
6:      **end if**
7:      Expand $n$, generating children with pointers to $n$
8:      **for all** children $m$ of $n$ **do**
9:          Calculate $f(m) \leftarrow \max(g(m), h(m))$
10:         **if** $m \notin$ OPEN and $m \notin$ CLOSED **then**
11:             Insert $m$ in OPEN
12:         **else if** $m \in$ OPEN and $f(m) < f$-value of duplicate in OPEN **then**
13:             Remove duplicate from OPEN and insert $m$
14:         **else**
15:             Discard $m$
16:         **end if**
17:     **end for**
18: **end while**
19: Exit with failure, no solution exists
```

and all of its incident edges from the graph. A *vertex elimination order* is a total order over the vertices in a graph. The *width* of an elimination order is defined as the maximum degree of any vertex when it is eliminated from the graph. Finally, the *treewidth* of a graph is the minimum width over all possible elimination orders, and any order whose width is the treewidth is an *optimal vertex elimination order*.

Finding the treewidth of an undirected graph is central to many queries and operations in a variety of areas, including probabilistic reasoning and constraint satisfaction. Determining the treewidth of a general graph is NP-complete (Arnborg, Corneil, & Proskurowski, 1987), therefore it is a natural candidate for heuristic search techniques.

The treewidth of a graph can be found by searching over the space of vertex elimination orders. For a simple example, consider searching for an optimal elimination order for the graph in Figure 2. The corresponding search space is shown in Figure 3. Eliminating a set of vertices from a graph leads to the same resulting graph, regardless of the order in which the vertices are eliminated. Thus, the state in this search space can be represented by the unordered set of vertices eliminated from the graph. At the root node, no vertices have been eliminated, and, at the goal node, all three vertices have been eliminated. To transition from one node to another, a vertex is eliminated from the graph. The cost of a transition, which labels the corresponding edge in the search space, is the degree of the vertex at the time it is eliminated. A solution path represents a particular elimination order, and the width of that order is the maximum edge cost on the solution path.

This search space is an example of a max-edge-cost search space for which MaxBF is intended. An existing algorithm, BestTW (Dow & Korf, 2007), is an enhanced

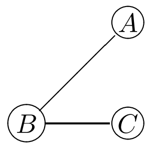


Figure 2: An undirected graph.

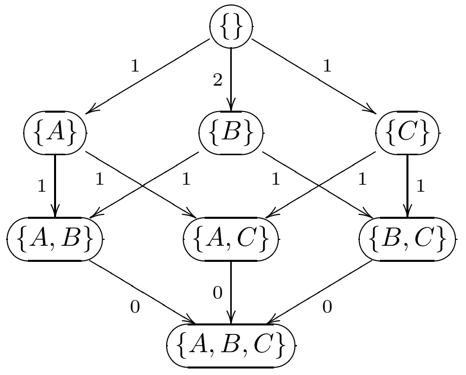


Figure 3: The treewidth search space for the graph in Fig. 2.

version of NaiveMaxBF for finding treewidth. As mentioned in Section 5, Dow & Korf (2007) established the admissibility of this algorithm by showing that a so-called max-consistent heuristic implies that the evaluation function, $f(n) = \max(g(n), h(n))$, is strongly order preserving. Additionally, Dow & Korf (2007) incorrectly stated that the heuristic used by BestTW was max-consistent, when, in fact, it is not. Fortunately, we have shown that only an optimistic heuristic is required for $f(n) = \max(g(n), h(n))$ to be weakly order preserving. Thus, BestTW is admissible and the error is insignificant. Nevertheless, given the results in this paper, it is possible that BestTW could save some wasted effort by not reopening closed nodes, as is done with MaxBF.

10 Conclusions

While best-first search is a well studied heuristic search technique, most existing research has focused on applying it to problems with additive cost functions. In this paper, we have applied best-first search to problem spaces where the cost of a solution path is the maximum edge cost. We have given an algorithm, MaxBF, that is analogous to A* and designed for these maximum edge cost problem spaces.

While Dechter & Pearl (1985) demonstrated that a best-first search algorithm is admissible if its evaluation function is optimistic and strongly order preserving, we have shown that the evaluation function used by MaxBF is not strongly order preserving if the heuristic function is merely optimistic. We have also shown that a weakly order preserving evaluation function is sufficient for admissibility, and that MaxBF's evaluation function meets that condition if the heuristic function is optimistic.

It is well-known that A* requires a consistent heuristic function to avoid reopening closed nodes. We have shown that, with just an optimistic heuristic function, MaxBF never needs to reopen closed nodes. This does not mean that the first time a node is chosen for expansion a least-cost path to it has been found, thus a naive version of MaxBF may reopen and reexpand closed nodes. In pathological cases it would search the tree expansion of the search graph. We have shown how to ensure that MaxBF avoids this inefficiency.

Finally, we have described an important problem, treewidth, that is critical to many active research areas, and to which MaxBF can be applied.

Acknowledgments

Thanks to the anonymous reviewers for some very helpful comments and recommendations.

References

Arnborg, S.; Corneil, D. G.; and Proskurowski, A. 1987. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic and Discrete Methods* 8(2):277–284.

Dechter, R., and Pearl, J. 1985. Generalized best-first search strategies and the optimality of A*. *Journal of the Association of Computing Machinery* 32(3):505–536.

Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik* 1:269–271.

Dow, P. A., and Korf, R. E. 2007. Best-first search for treewidth. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence*, 1146–1151. Vancouver, British Columbia, Canada: AAAI Press.

Pearl, J. 1984. *Heuristics*. Reading, MA: Addison-Wesley.