Bidirectional A* Search with Additive Approximation Bounds

Michael N. Rice and **Vassilis J. Tsotras**
University of California, Riverside
{mrice,tsotras}@cs.ucr.edu

Abstract

In this paper, we present new theoretical and experimental results for bidirectional A* search. Unlike most previous research on this topic, our results do not require assumptions of either consistent or balanced heuristic functions for the search. Our theoretical work examines new results on the worst-case number of node expansions for inconsistent heuristic functions with bounded estimation errors. Additionally, we consider several alternative termination criteria in order to more quickly terminate the bidirectional search, and we provide worst-case approximation bounds for our suggested criteria. We prove that our approximation bounds are purely additive in nature (a general improvement over previous multiplicative approximations). Experimental evidence on large-scale road networks suggests that the errors introduced are truly quite negligible in practice, while the performance gains are significant.

1 Introduction

A* search (Hart, Nilsson, and Raphael 1968) is a classical graph-searching algorithm established in the late 1960s for finding minimum-cost (i.e., "shortest") paths. Since that time, much additional theoretical and experimental work has been focused on this algorithm and its useful properties, due to its many practical applications for pathfinding in the domains of robotics, game AI, transportation analysis, and combinatorial optimization, to name a few.

In recent years, due to the increasingly-ubiquitous presence of personal navigation systems, additional focus has also been given to efficiently solving shortest paths in large-scale, real-world road networks. Many of the most recent state-of-the-art techniques in this domain have involved some variant of A* search, either as the main algorithmic component (Goldberg and Harrelson 2005), or simply as one of several complimentary algorithmic ingredients (Bauer et al. 2008). One of the primary benefits of using A* search within these contexts comes from the fact that, unlike most other approaches, the A* lower-bounding technique often remains valid even in dynamic or semi-dynamic scenarios, in which the resulting shortest paths can change, depending on the state of the graph (e.g., see (Delling and Wagner 2007)) or the incoming query parameters (e.g., see (Rice and Tsotras 2010)), respectively. However, most work in this particular domain to date has focused solely on the use of consistent heuristic functions, as consistency has generally been assumed to be a necessity for good performance.

In this work, we further extend the existing theory behind the A* search algorithm. We present a new perspective on the worst-case behavior of the A* search algorithm for inconsistent heuristics, as well as several alternative termination criteria for speeding up bidirectional A* search in this regard. We further demonstrate the power behind this theory by presenting experimental results on one of the largest real-world road networks: the road network of North America.

2 Background and Related Work

2.1 Preliminaries

Let $G = (V, E)$ be a directed graph with node set V and edge set E, such that $n = |V|$ and $m = |E|$. Let $w : E \rightarrow \mathbb{Z}_+$ be a *weight function* mapping edges to positive integers. Let $P_{s,t} = \langle v_1, v_2, \ldots, v_q \rangle$ be any path in G from some starting node $s = v_1 \in V$ to some terminal node $t = v_q \in V$, such that, for $1 \leq i < q$, $(v_i, v_{i+1}) \in E$. When nodes s and t are implied, we shall refer to such a path as simply P. We compute the weight of any path P as $w(P) = \sum_{i=1}^{q-1} w(v_i, v_{i+1})$. We denote the s-t path with minimum overall weight as the *shortest path* $P^*_{s,t}$, or simply P^*. We denote the shortest path "distance" from s to t as $d(s,t) = w(P^*_{s,t})$. In general, $d(s,t) \neq d(t,s)$ may be true.

Additionally, we say that a given algorithm guarantees an (α, β)-approximate shortest path if the following inequality holds for the cost, μ, of the solution path returned by the algorithm: $\mu \leq \alpha \cdot d(s,t) + \beta$. We utilize the concept of (α, β)-approximation to universally classify the different categories of approximation introduced by each algorithm discussed within this paper. Using this classification, any optimal algorithm therefore guarantees a $(1, 0)$-approximation. Any algorithm which guarantees an approximation of the form $(\alpha, 0)$ for $\alpha > 1$ is considered to have purely *multiplicative* error. Conversely, any algorithm which guarantees an approximation of the form $(1, \beta)$ for $\beta > 0$ is considered to have purely *additive* error.

2.2 Unidirectional A* Search

The original concept of classical (unidirectional) A* search was first formalized and presented in (Hart, Nilsson, and Raphael 1968). Within the context of this seminal work, the objective of the presented search algorithm is to find the shortest path, P^*, from some starting node, s, to some terminal "goal" node, t, in a graph, G.

The search proceeds by assigning each reached node, v, an f-value, computed as $f(v) = g(v) + h(v)$, where $g(v)$ represents the current best-known path cost from s to v, and $h(v)$, defined as the *heuristic function*, represents an estimate on the shortest path distance from v to t. Initially, $g(v) = \infty$ for all $v \in V$. The algorithm begins by assigning $g(s) = 0$ and inserting s into the node set called $OPEN$ (this set represents the "fringe" of the search). At each iteration, the search then removes from $OPEN$ any node, u, with minimum f-value (computed as above) and adds this node to the set called $CLOSED$.

After *closing* a node u, the algorithm proceeds to *expand* node u as follows. For each outgoing edge, $e = (u, v) \in E$, the search *relaxes* the edge, by checking whether $g(v) > g(u) + w(e)$ is true. If so, the algorithm updates the value to $g(v) = g(u) + w(e)$, and if $v \notin OPEN$, then it is added to $OPEN$ for subsequent expansion (if $v \in CLOSED$, it is removed from $CLOSED$ before being added to $OPEN$). This process continues until either $OPEN$ becomes empty or until node t is removed for expansion. In the former case, this indicates that there is no valid path from s to t. In the latter case, the shortest path cost from s to t is the value $g(t)$.

The above algorithm will terminate with the correct shortest path cost under certain conditions. Specifically, any heuristic search algorithm which guarantees to find a shortest path, if one exists, is considered *admissible*. Within the context of A* search, the search is provably admissible if, for all nodes $v \in V$, $0 \leq h(v) \leq d(v, t)$ holds true. That is, as long as the heuristic function, h, never overestimates the true cost to the destination, then the search is guaranteed to terminate with a shortest path (Hart, Nilsson, and Raphael 1968). If this property does not hold true for one or more nodes, then the search is considered *inadmissible*.

An even stronger claim exists for heuristic functions which exhibit a property known as *consistency*. A heuristic function, h, is considered *consistent* if, for all edges $(u, v) \in E$, the property $h(u) \leq w(u, v) + h(v)$ holds true. If this property does not hold for one or more edges, then the heuristic function is considered *inconsistent*.

The property of consistency represents a form of *triangle inequality* which guarantees that the computed f-values of the closed nodes can only monotonically increase as the search progresses. Under these conditions, any node will be closed and expanded at most once during the search process (Hart, Nilsson, and Raphael 1968). Therefore, consistent heuristic functions guarantee to perform no more than $O(n)$ node expansions in the worst case.

On the other hand, when using inconsistent heuristic functions, a node may need to be closed and re-opened several times throughout the search. In particular, A* search with inconsistent heuristics may require up to $O(2^n)$ possible node expansions in the worst case (Martelli 1977).

However, more recent revelations have been made in regards to the true impact of inconsistent heuristic functions on the overall efficiency of the search, for most practical real-world cases. Specifically, it has been recently demonstrated in (Zhang et al. 2009) that this exponential growth of the search space can only occur for degenerate cases in which the edge weights grow exponentially with the graph size. For graphs whose edge weights are independent of the size of the graph (i.e., the maximum edge weight is bounded by some constant), A* search using inconsistent heuristic functions will require no more than $O(n^2)$ node expansions (Zhang et al. 2009).

2.3 Bidirectional A* Search

The original A* concept was later further extended to bidirectional A* search in (Pohl 1969), where two separate unidirectional A* searches are carried out simultaneously from both nodes s and t. A *forward* A* search is carried out from node s (toward goal node t) and a *backward*[1] A* search is carried out from node t (toward goal node s).

To keep track of two simultaneous A* searches, separate $OPEN/CLOSED$ sets are needed for both search directions. Let these sets be $OPEN_f/CLOSED_f$ and $OPEN_b/CLOSED_b$ for the forward and backward search, respectively. Similarly, each search direction must make use of its own distinct heuristic function, given that each direction is searching towards a different goal node. We denote the heuristic search functions as h_f and h_b for the forward and backward search directions, respectively. In the context of bidirectional A* search, $h_f(v)$ behaves as before by providing a lower-bound estimate on the cost $d(v, t)$, whereas $h_b(v)$ provides a lower-bound estimate on the cost $d(s, v)$. For forward search, the f-values are computed as $f_f(v) = g_f(v) + h_f(v)$. For backward search, $f_b(v) = g_b(v) + h_b(v)$.

In general, the bidirectional algorithm proceeds by iteratively switching between each search direction according to some predefined *alternating strategy* (e.g., choosing the search direction to expand as the direction with the minimum f-value in its open set or simply alternating between search directions). Any alternating strategy will guarantee optimality, given a valid *termination criterion* (Pohl 1969).

The termination criterion suggested in (Pohl 1969) is based on the shortest path distance, μ, seen thus far in the search. Initially, the bidirectional algorithm sets $\mu = \infty$. Whenever the search in a given direction closes a node v, such that v has already been closed in the opposite search direction, then the algorithm sets $\mu = min\{\mu, g_f(v) + g_b(v)\}$. If the search algorithm is terminated as soon as $max(k_f, k_b) \geq \mu$, where $k_f = min\{f_f(v) \mid v \in OPEN_f\}$ and $k_b = min\{f_b(v) \mid v \in OPEN_b\}$, this guarantees to terminate with an optimal shortest path (Pohl 1969). The optimality for this termination criterion holds for any admissible heuristic functions, h_f and h_b (even inconsistent ones, as long as node re-opening is allowed).

[1] A *backward* search in a directed graph $G = (V, E)$ is the equivalent of performing a standard (i.e., *forward*) search in the graph $G' = (V, E')$, where $E' = \{(v, u) \mid (u, v) \in E\}$.

2.4 Related Work

Despite the correctness of the proposed termination criterion for bidirectional A* search, experimental results from (Pohl 1969) suggested that this bidirectional variant of A* search can still often be even less efficient than the classical unidirectional A* search from (Hart, Nilsson, and Raphael 1968).

Theoretical and empirical results from (Kaindl and Kainz 1997) suggest that this problem manifests itself as a condition of the termination criterion presented earlier, forcing the search to continue for too long in both directions once they have met, resulting in duplicate work per search direction.

Additional optimizations to the bidirectional A* search algorithm from (Pohl 1969) were suggested in (Kwa 1989), in which the optimized algorithm attempts to aggressively prune nodes from both search directions where possible (e.g., by pruning a node, v, when $f(v) \geq \mu$, or by avoiding expansion from v when it has already been closed in the opposing direction). Such optimizations help to reduce the redundant search overhead of the opposing directions. However, some of the proposed optimizations are only possible when using consistent heuristic functions.

An alternative bidirectional A* algorithm is presented in (Ikeda et al. 1994). This requires an adjustment of the update procedure for μ as follows. After closing a node u in the forward direction, when attempting to relax any edge (u, v), if node v has already been closed in the backward direction, then set $\mu = min\{\mu, g_f(u) + w(u, v) + g_b(v)\}$. Similar logic holds for updating μ in the backward search direction. Using this update procedure, the authors prove that, by using a set of *balanced*[2] and consistent heuristic functions h'_f and h'_b such that, for all nodes $v \in V$, $h'_f(v) + h'_b(v) = c$ (for some constant, c), then the search may terminate as soon as the first node has been closed in both directions. This approach guarantees to find the shortest path, if one exists. The authors further demonstrate a simple approach for deriving such balanced heuristic functions by using $h'_f(v) = (h_f(v) - h_b(v))/2$ and $h'_b(v) = (h_b(v) - h_f(v))/2 = -h'_f(v)$, where h_f and h_b are arbitrarily-defined consistent functions (again, note the requirement of consistency).

In recent years, the experimental algorithms community has seen much progress in the way of establishing highly-efficient shortest path algorithms through the use of offline preprocessing algorithms. One of the most prominent of these approaches to incorporate A* search techniques is that of the ALT algorithm (Goldberg and Harrelson 2005). ALT involves preprocessing which selects a small subset of so-called *landmark* nodes, $L \subseteq V$, typically such that $|L| \ll |V|$. For each landmark, $l \in L$, the preprocessing step computes the shortest path costs from and to all other nodes, v, in the graph: $d(v, l)$ and $d(l, v)$, respectively. After preprocessing, a bidirectional A* search based primarily on the one presented in (Ikeda et al. 1994) is then carried out using heuristic functions derived from the preprocessed landmark costs: $h_f(v) = \max_{\forall l \in L'}\{max\{d(v, l) - d(t, l), d(l, t) - d(l, v)\}\}$ and $h_b(v) = \max_{\forall l \in L'}\{max\{d(s, l) - d(v, l), d(l, v) - d(l, s)\}\}$, where $L' \subseteq L$ is a landmark subset chosen at query time.

Other, similar preprocessing techniques have since incorporated variants of this ALT technique into their methodology as well (e.g., (Bauer et al. 2008; Geisberger, Kobitzsch, and Sanders 2010; Geisberger et al. 2012)), due primarily to the general flexibility of A* search within these domains. However, to date, very little related research has been focused on using alternate A* techniques other than ALT or on using techniques which might instead support the use of inconsistent functions.

As yet another alternative for dealing with the issues of A* search efficiency, approximate solutions have also been developed. One well-known variant on the classical A* technique is the so-called *weighted* A* search algorithm (Pohl 1969), in which the f-values are instead computed as $f(v) = g(v) + \alpha \cdot h(v)$, for some $\alpha \geq 1$. This additional weighting parameter allows for placing greater emphasis on the heuristic function value throughout the search, typically leading to much faster convergence to a valid (although possibly sub-optimal) solution path. Such weighted search is guaranteed to produce an $(\alpha, 0)$-approximate shortest path (i.e., it has purely-multiplicative error). This approach can also be easily extended to bidirectional weighted A* search (Köll and Kaindl 1993), giving similar approximation bounds.

Another approximation result from (Harris 1974) is the first to demonstrate a purely-additive error for *unidirectional* A* search in which the heuristic function can *over*estimate the true shortest path cost by as much as some error, β. For such inadmissible heuristics, unidirectional A* search guarantees a $(1, \beta)$-approximate shortest path.

Much research has also been devoted to the impact of heuristic function estimation error on the size of the resulting search space (Pohl 1969; Chenoweth and Davis 1991; Dinh, Russell, and Su 2007; Helmert and Röger 2008). However, this work focuses primarily on bounding only the number of nodes that can be reached by the search based on this error, and does not consider the number of times each reached node can be expanded for inconsistent heuristics.

2.5 Our Contributions

In this work, we present both theoretical and experimental results that provide the following contributions:

- We present an alternative perspective on the worst-case number of node expansions when using admissible, inconsistent heuristic functions by proving a pseudo-polynomial upper bound related to the underestimation error, ϵ, of the heuristic function(s). Specifically, we demonstrate an upper bound of $O(n\epsilon)$ node expansions, further justifying the effectiveness of accurate, but inconsistent heuristics, and improving upon the result from (Zhang et al. 2009) for certain classes of inconsistent heuristics.
- We present several alternative termination criteria for bidirectional A* search, the best of which guarantee $(1, \epsilon)$-approximate shortest paths.

[2] Such functions have also been commonly referred to as *consistent* heuristic functions in (Goldberg and Harrelson 2005). However, this conflicts with the original use of the term consistency in this context. Therefore, we have adopted the term *balanced* from (Pijls and Post 2009) to describe this property.

- We examine a straightforward preprocessing approach for constructing inconsistent heuristic functions for which we can readily prove bounds on the resulting underestimation errors, and whose results in practice show very small average estimation errors.
- We present an experimental analysis of this preprocessed heuristic function, using our newly-proposed termination criteria. A comparison of the results against the well-known ALT method shows that our new method can result in significantly faster query times when using the same memory overhead, while yielding little to no error.

3 Theoretical Results

We first begin with some theoretical results on unidirectional A* search which can then be further applied to bidirectional A* search. All subsequent proofs assume only admissibility of the chosen heuristic function(s). For the purposes of our discussion of unidirectional A* search, we shall assume that our heuristic function is ϵ-*bounded*, as defined by the following ϵ *inequality*:

$$d(v,t) - \epsilon \leq h(v) \leq d(v,t)$$

Lemma 1. (Hart, Nilsson, and Raphael 1968) *For any node $v \in OPEN$ and any optimal path $P^*_{s,v}$ in G, $\exists\, v' \in P^*_{s,v}$ such that $v' \in OPEN$ and $g(v') = d(s,v')$.*

Corollary 1. (Hart, Nilsson, and Raphael 1968) *At any point before termination, $\exists\, v' \in P^*_{s,t}$ such that $v' \in OPEN$ and $f(v') \leq d(s,t)$.*

Lemma 2. *If the heuristic function, h, is ϵ-bounded, then $\forall\, v \in CLOSED$, $g(v) \leq d(s,v) + \epsilon$.*

Proof. Since, by definition, the value $g(v)$ can only decrease during a given A* search, it suffices to consider the value of $g(v)$ the first time any node, v, is chosen from $OPEN$ to be added to $CLOSED$. Suppose that $g(v) > d(s,v) + \epsilon$. This gives us the following inequality:

$$\begin{aligned} f(v) = g(v) + h(v) &> d(s,v) + h(v) + \epsilon && (1)\\ &\geq d(s,v) + d(v,t) && (2)\\ &\geq d(s,t) && (3)\\ &\geq f(v') && (4) \end{aligned}$$

Inequality (1) holds by the initial assumption above, (2) holds by the ϵ inequality, (3) holds by the triangle inequality, and (4) follows from Corollary 1 for some $v' \in OPEN$. Since $f(v) > f(v')$, this contradicts the fact that v was chosen from $OPEN$ to be added to $CLOSED$ (because the algorithm always chooses a node with minimum f-value from $OPEN$). Thus, $v \in CLOSED \Rightarrow g(v) \leq d(s,v) + \epsilon$. □

Theorem 1. *Any A* search algorithm using an ϵ-bounded heuristic function, h, will expand no more than $O(n\epsilon)$ nodes during the search.*

Proof. In order for a node to be expanded, it must first be added to the $CLOSED$ set. By Lemma 2, we have that for any node $v \in CLOSED$, $g(v) \leq d(s,v) + \epsilon$ must be true.

As noted previously, $g(v)$ can only decrease during the search. Let γ be the greatest common divisor of all edge weights in the graph. Since all path costs in the graph must be a multiple of γ, any time we re-open a node for subsequent re-expansion, we must therefore decrease $g(v)$ by at least γ (Zhang et al. 2009). Because $g(v) \leq d(s,v) + \epsilon$, then we can only expand any node v at most $\lceil \epsilon/\gamma \rceil$ times. Since $\gamma \geq 1$, we therefore expand at most $O(n\epsilon)$ nodes. □

Assuming non-negative, admissible heuristic functions, then the largest possible error for any heuristic function on a search must therefore be upper-bounded by the diameter of the graph, $\Delta(G) = max\{d(s,t) \mid s,t \in V\}$. For $\overline{w} = max\{w(u,v) \mid (u,v) \in E\}$, it is easy to see that $\epsilon \leq \Delta(G) \leq \overline{w}(n-1)$. If the value of $\overline{w}$ is independent of the size of the graph (i.e., it is bounded by some constant), then this gives us a worst-case number of node expansions of $O(n^2)$, which is identical to the previous work of (Zhang et al. 2009) under similar conditions. However, for $\epsilon \in o(n)$, the worst-case number of node expansions improves even further. For example, for $\epsilon \in O(1)$, the number of node expansions is $O(n)$, which is asymptotically equivalent to using a consistent heuristic function.

We note that all previous claims for unidirectional A* search also apply directly to *both* search directions in a bidirectional A* search, for the remainder of this discussion.

Lemma 3. *Let P be the shortest path seen thus far in a bidirectional A* search. Let $w(P) = \mu$. Let $k_f = min\{f_f(v) \mid v \in OPEN_f\}$ and $k_b = min\{f_b(v) \mid v \in OPEN_b\}$. If $max(k_f, k_b) \geq \mu - \beta$, then P is a $(1,\beta)$-approximate shortest path.*

Proof. Let $max(k_f, k_b) = k_f$ (a symmetric argument holds if $max(k_f, k_b) = k_b$). By Corollary 1, we have that there exists a node $v' \in OPEN_f$ which lies on the shortest path from s to t with $f_f(v') \leq d(s,t)$. Therefore, we have the following inequality:

$$\begin{aligned} \mu - \beta &\leq max(k_f, k_b)\\ \Rightarrow \mu &\leq max(k_f, k_b) + \beta\\ &\leq f_f(v') + \beta\\ &\leq d(s,t) + \beta \end{aligned}$$

□

This lemma implies that one can apply any arbitrary additive error, β, to their search criteria by using this inequality as a valid termination criterion. The larger the value of β, the sooner the bidirectional A^* search can terminate (albeit with higher possible additive error). We shall use this property to prove approximation bounds on several different termination criteria presented later on in the paper.

Now, suppose that we are using heuristic functions h_f and h_b with provably-bounded estimation errors of ϵ_f and ϵ_b, respectively. That is, for all nodes, $v \in V$, the following inequalities hold:

$$\begin{aligned} d(v,t) - \epsilon_f \leq h_f(v) \leq d(v,t)\\ d(s,v) - \epsilon_b \leq h_b(v) \leq d(s,v) \end{aligned}$$

For the remainder of the discussion, we shall assume that both heuristic functions are bounded by the same value: $\epsilon = \epsilon_f = \epsilon_b$. However, for $\epsilon_f \neq \epsilon_b$, let $\epsilon = max\{\epsilon_f, \epsilon_b\}$.

We also enhance the original bidirectional A* logic slightly as follows. Instead of just updating the value of μ whenever we reach a node $v \in CLOSED_f \cap CLOSED_b$, we additionally try to update μ any time we relax edge (u, v) in the search and node u has been closed in one direction (e.g., $u \in CLOSED_f$) whereas node v has been closed in the other (e.g., $v \in CLOSED_b$). Here we set $\mu = min\{\mu,\ g_f(u)+w(u,v)+g_b(v)\}$. This is similar to the update procedure used by (Ikeda et al. 1994). It guarantees that μ represents a lower bound on all possible connecting paths seen thus far and helps to minimize the value of μ as much as possible, given our alternative termination criteria presented below.

Theorem 2. *Any bidirectional A* search algorithm using ϵ-bounded heuristic functions, h_f and h_b, which terminates after finding the first node $v \in CLOSED_f \cap CLOSED_b$ guarantees a $(1, min\{2\epsilon, d(s,t)\})$-approximate shortest path, and this bound is tight.*

Proof. For the purposes of our discussion, we shall assume that the node v is pulled from $OPEN_f$ when this termination criterion is first satisfied (a similar argument holds for the case where it is pulled from $OPEN_b$). First, we note that the following properties hold by definition of our proposed algorithm:

$$max(k_f, k_b) \geq g_f(v) + h_f(v) \quad (5)$$
$$g_f(v) + g_b(v) \geq \mu \quad (6)$$

Property (5) is guaranteed by our selection which chooses the node v with the smallest value $f_f(v) = g_f(v) + h_f(v)$ from $OPEN_f$, and (6) is guaranteed by the fact that μ defines a lower bound on all connecting paths seen thus far.

First, we prove a $(1, d(s,t))$-approximation (note that this also implies at worst a multiplicative $(2, 0)$-approximation bound). This follows from property (6) above and Corollary 1, which guarantees that $\mu \leq g_f(v) + g_b(v) \leq f_f(v) + f_b(v) \leq d(s,t) + d(s,t)$. However, this may indeed be further improved upon, depending on the value of ϵ. To demonstrate this, consider the following inequality:

$$\begin{aligned} max(k_f, k_b) &\geq g_f(v) + h_f(v) && (7)\\ &\geq \mu - g_b(v) + h_f(v) && (8)\\ &\geq \mu - (d(v,t) + \epsilon) + h_f(v) && (9)\\ &\geq \mu - (d(v,t) + \epsilon) + (d(v,t) - \epsilon) && (10)\\ &= \mu - d(v,t) - \epsilon + d(v,t) - \epsilon && (11)\\ &= \mu - 2\epsilon && (12) \end{aligned}$$

Inequality (7) holds by property (5) above, (8) holds by property (6) above, (9) follows from Lemma 2 for backward search, and (10) holds by the ϵ inequality. By Lemma 3, we have that $max(k_f, k_b) \geq \mu - 2\epsilon$ guarantees a $(1, 2\epsilon)$-approximate shortest path.

We further demonstrate that this approximation bound is tight for this particular termination criterion by providing a simple example where the error is exactly $2\epsilon = d(s,t)$. Consider the graph presented in Figure 1. In this graph, all nodes are labeled with their respective h_f and h_b function values. For each node, except for node v, the heuristic

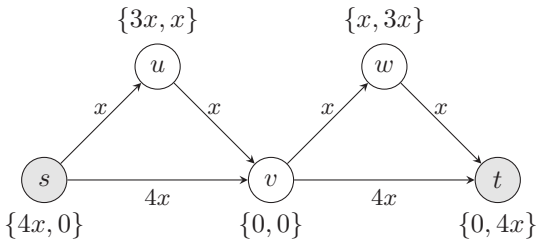

Figure 1: In the sample graph presented above, each edge, e, is labeled with its weight, $w(e)$, and each node, v, is labeled as $\{h_f(v), h_b(v)\}$.

functions return a *perfect* estimate of the shortest path costs in both directions. For node v, the heuristic functions underestimate their respective shortest path costs by exactly $\epsilon = 2x$. For this graph, we have that $d(s,t) = 4x$. However, assuming we break ties when choosing the minimum f-value node by preferring nodes with higher g values, as is typical, a bidirectional A* search which terminates using the above criterion will find a path $P = \langle s, v, t \rangle$, such that $w(P) = 8x = d(s,t) + 4x = d(s,t) + 2\epsilon$. □

For the remainder of the discussion, we shall omit the $min\{\bullet, d(s,t)\}$ notation for brevity (e.g., $(1, min\{2\epsilon, d(s,t)\})$ will be written as simply $(1, 2\epsilon)$), as the multiplicative $(2, 0)$ bound will be implied for all remaining criteria (it is similarly straightforward to prove).

While the above termination criterion guarantees a $(1, 2\epsilon)$-approximate shortest path, we can improve this approximation even further, if we know the value of ϵ, by using some simple properties of our ϵ-bounded heuristic functions. The additional logic is as follows. When choosing a node $u \in OPEN_f$ for expansion, we first relax all edges (u, v) to see if we can improve upon the value $g_f(v)$. If we do not improve $g_f(v)$, then we continue as before. However, if we do improve $g_f(v)$, we first check to see whether the inequality $g_f(v) - h_b(v) \leq \epsilon$ is true. If not, we may skip the node v (i.e., do not add it to $OPEN_f$), since $g_f(v) - h_b(v) > \epsilon$ implies that $g_f(v) > h_b(v) + \epsilon \geq d(s,v)$ by the ϵ inequality, and thus the current path through node u cannot be a shortest path from s to v. A similar argument holds for expanding nodes in the backward search direction. We shall call this procedure *ϵ-skipping*. As a corollary, we note that any node which subsequently belongs to either the open or closed sets of either search direction must satisfy this inequality; otherwise, it would have never been added to the open set for that respective search direction.

Theorem 3. *Using ϵ-skipping, any bidirectional A* search algorithm which terminates after finding the first node $v \in CLOSED_f \cap CLOSED_b$ guarantees a $(1, \epsilon)$-approximate shortest path.*

Proof. As in Theorem 2, we shall assume that the node v is pulled from $OPEN_f$ when this termination criterion is first satisfied. First, we note that, in addition to properties (5) and (6) from Theorem 2, the following additional property

is satisfied explicitly by our ϵ-skipping method (since $v \in CLOSED_b$):

$$g_b(v) - h_f(v) \leq \epsilon \tag{13}$$

These properties, taken together, allow us to derive the following inequality:

$$\begin{aligned} max(k_f, k_b) &\geq g_f(v) + h_f(v) && (14)\\ &\geq \mu - g_b(v) + h_f(v) && (15)\\ &= \mu - (g_b(v) - h_f(v)) && (16)\\ &\geq \mu - \epsilon && (17) \end{aligned}$$

Inequalities (14) and (15) are the same as (7) and (8), respectively, and (17) follows from (13) above. By Lemma 3, we have that $max(k_f, k_b) \geq \mu - \epsilon$ guarantees a $(1, \epsilon)$-approximate shortest path. □

As a simple exercise, one can easily verify that if we apply the concept of ϵ-skipping to the problem presented in Figure 1, the new algorithm would now terminate with the true shortest path from s to t for this particular example.

It is noteworthy to mention that the proof outlined above assumes that the search algorithm has *a priori* knowledge of the value of ϵ (to enforce ϵ-skipping). While this proves useful in establishing effective skipping criteria, we may not always know the true value of ϵ in advance. For such cases, we demonstrate yet another termination criterion which guarantees a $(1, \epsilon)$-approximate shortest path without requiring explicit knowledge of the error bound, ϵ.

Theorem 4. *Terminating any bidirectional A* search algorithm as soon as $k_f + k_b \geq \mu + h_f(s)$ will guarantee a $(1, \epsilon)$-approximate shortest path when using ϵ-bounded heuristic functions h_f and h_b.*

Proof.

$$\begin{aligned} k_f + k_b &\geq \mu + h_f(s)\\ &\geq \mu + d(s,t) - \epsilon \quad \text{(by } \epsilon \text{ inequality)}\\ \Rightarrow k_f + k_b - d(s,t) &\geq \mu - \epsilon \end{aligned}$$

By Corollary 1, $k_f \leq d(s,t)$ and $k_b \leq d(s,t)$, and thus $k_f - d(s,t) \leq 0$ and $k_b - d(s,t) \leq 0$, which gives us:

$$max(k_f, k_b) \geq k_f + k_b - d(s,t) \geq \mu - \epsilon$$

Again, by Lemma 3, we have a guaranteed $(1, \epsilon)$-approximate shortest path. □

4 Region-to-Region (R2R) Preprocessing

Given the theoretical results presented thus far regarding ϵ-bounded heuristic functions, we now must demonstrate a practical approach for deriving an admissible (but inconsistent) heuristic function for which we can readily prove the resulting error bounds. We may establish such a heuristic function by preprocessing the input graph as follows. Similar heuristic functions have previously been established in (Felner and Sturtevant 2009) and (Maue, Sanders, and Matijevic 2006). However, (Felner and Sturtevant 2009) is applied only to undirected, unweighted graphs and (Maue, Sanders, and Matijevic 2006) is not applied to A* search.

We start by partitioning the graph into a set of k disjoint regions $\mathcal{R} = \{R_1, R_2, \ldots, R_k\}$, where, for $1 \leq i \leq k$, $R_i \subseteq V$. Given a partitioning, we say that $d(R_i, R_j) = min\{d(v_i, v_j) \mid v_i \in R_i, v_j \in R_j\}$ represents the shortest path distance between the closest nodes from any two regions, R_i and R_j. For $1 \leq i \leq k$, let $\delta(R_i) = max\{d(s,t) | s,t \in R_i\}$ represent the *diameter* of region R_i. Let $\delta(\mathcal{R}) = \max_{1 \leq i \leq k} \{\delta(R_i)\}$ be the maximum partition diameter. Let $r : V \rightarrow \mathcal{R}$ be a function mapping nodes to their associated regions. After partitioning the graph, we therefore calculate a $k \times k$ cost matrix, C, such that each entry $C_{i,j} = d(R_i, R_j)$.

Lemma 4. *For all $s, t \in V$, $d(s,t) \leq d(r(s), r(t)) + 2\delta(\mathcal{R})$.*

Proof. By definition, there must exist nodes $v_s \in r(s)$ and $v_t \in r(t)$ where $d(v_s, v_t) = d(r(s), r(t))$. This gives us the following inequality:

$$\begin{aligned} d(s,t) &\leq d(s, v_s) + d(v_s, v_t) + d(v_t, t)\\ &= d(s, v_s) + d(r(s), r(t)) + d(v_t, t)\\ &\leq \delta(r(s)) + d(r(s), r(t)) + \delta(r(t))\\ &\leq d(r(s), r(t)) + 2\delta(\mathcal{R}) \end{aligned}$$

□

Corollary 2. *Bidirectional heuristic functions $h_f(v) = d(r(v), r(t))$ and $h_b(v) = d(r(s), r(v))$ have bounded error $\epsilon = 2\delta(\mathcal{R})$.*

This corollary suggests that the maximum ϵ value for a given partitioning is proportional to the largest diameter region in the partition. Therefore, if one can minimize the average or maximum diameter of the partitioning, one can effectively minimize the resulting error and thus the resulting search complexity. This problem is more commonly known as *Minimum-Diameter Partitioning* and it is known to be $\mathcal{NP}$-hard. In this paper, we shall not focus on minimizing the maximum partition diameter, as this is outside the scope of this work. In the next section, we present experimental results using a straightforward partitioning technique for establishing our heuristic functions, described as follows.

Given a target number of regions, k, we begin by selecting a subset of k nodes, $V' = \{v'_1, v'_2, \ldots, v'_k\}$, uniformly at random. We then add a temporary *super-source* node, s', to the graph, along with temporary edges (s', v'_i), for all $1 \leq i \leq k$, each with cost equal to zero. We then perform a single, shortest path search from the super-source node, s', until all nodes in the graph have been added to the shortest path tree. For each node, v, there exists exactly one *ancestor* node from V' on the shortest path from s' to v. All nodes which share the same ancestor node thus make up a single partition, or *region*. In practice, this simple approach typically results in tightly-clustered regions, centered around each node $v'_i \in V'$.

After partitioning the graph, we must next compute the cost matrix, C, representing the shortest path between each region, as described earlier. To achieve this, we now use a similar process to that performed during the original graph partitioning. For each region, R_i, of the graph, we temporarily add a super-source node, s'_i, to the graph, along with

temporary edges (s'_i, v_i) for all nodes $v_i \in R_i$, each with cost equal to zero. We then perform a shortest path search from s'_i, as before. Afterwards, for all $1 \leq j \leq k$, we set $C_{i,j} = d(R_i, R_j) = \min_{\forall v_j \in R_j} \{d(s'_i, v_j)\}$. This process is repeated for all regions.

5 Experimental Results

In this section, we present an experimental evaluation of the algorithms discussed so far.

5.1 Test Environment and Test Dataset

All experiments were carried out on a 64-bit server machine running Linux CentOS 5.3 with 2 quad-core CPUs clocked at 2.53 GHz with 18 GB RAM (although only one core was used per experiment). All programs were written in C++ and compiled using gcc version 4.1.2 with optimization level 3.

Shortest path queries were performed on one of the largest available real-world road networks: the North American[3] road network, with a total of $21,133,774$ nodes and $52,523,592$ edges. The edge weight function used for this dataset is based on the edge travel times, in minutes. This dataset was derived from NAVTEQ transportation data products, under their permission.

5.2 Preprocessing Results

The results of the preprocessing phase for both ALT and our R2R technique are presented in Table 1. For ALT with k landmarks, the preprocessing time is roughly proportional to the computation of $2k$ shortest path trees (since we must compute shortest paths to and from each landmark). This requires $O(kn)$ memory overhead to store the resulting landmark costs, and thus k must be chosen relatively small to achieve reasonable memory limits. For the R2R method with k regions, the preprocessing time is roughly proportional to computing only $k+1$ shortest path trees (one for partitioning, and one from each region to all others). However, in contrast to ALT, R2R requires only $O(k^2+n)$ memory overhead, so we can afford to choose a much larger value of k for R2R than we can for ALT, in general, to achieve similar levels of memory overhead. For these experiments, we have chosen k = 10,000 (50,000) for R2R and k = 4 (64) for ALT, which gives similar memory overhead from the resulting preprocessed data, albeit at the expense of much higher preprocessing times for the much larger values of k for R2R. Average partition diameters for the resulting 10k and 50k partitionings were 87 and 39 minutes, respectively.

Table 1: Preprocessing on the North American graph.

	Preprocessing	
Algorithm	Time [H:M]	Space [GB]
ALT-4	0:02	0.62
R2R-10k	27:00	0.45
ALT-64	0:58	10.08
R2R-50k	139:00	9.39

[3]This includes only the US and Canada.

5.3 Query Performance

We present the results of the query experiments in Table 2. All query performance results are averaged over 10,000 source-target pairs, chosen uniformly at random. For these results, we compare our new R2R method directly against the ALT method, as well as the standard, bidirectional Dijkstra search for a baseline comparison. For our R2R method, we consider results from both (U)nidirectional A* search and (B)idirectional A* search. For the bidirectional R2R approach, we further test three different termination criteria: the original (M)ax criterion (from (Pohl 1969)), the (I)ntersection criterion (from Theorem 3), and the (S)um criterion (from Theorem 4).

Table 2: Query Experiments on the North American graph.

	Queries			
Algorithm	Time [ms]	Expanded Nodes	Reopened Nodes	(α, β)
Dijkstra	3,513.29	6,938,720	0	$(1, 0)$
ALT-4	810.99	964,922	0	$(1, 0)$
R2R-10k-U	325.36	694,427	160,974	$(1, 0)$
R2R-10k-B-M	554.69	981,107	277,959	$(1, 0)$
R2R-10k-B-I	**53.18**	**162,744**	**37,654**	$(1, \epsilon)$
R2R-10k-B-S	91.55	270,312	65,583	$(1, \epsilon)$
ALT-64	74.68	91,967	0	$(1, 0)$
R2R-50k-U	108.09	219,255	32,393	$(1, 0)$
R2R-50k-B-M	143.66	288,824	49,751	$(1, 0)$
R2R-50k-B-I	**14.82**	**47,625**	**6,404**	$(1, \epsilon)$
R2R-50k-B-S	23.00	73,840	10,140	$(1, \epsilon)$

As expected, both methods clearly outperform the standard Dijkstra approach. Even for the lowest-memory configurations, the unidirectional R2R-10k-U method outperforms Dijkstra by an order of magnitude. Furthermore, R2R-10k-U outperforms the (bidirectional) ALT-4 method by nearly a factor of 2.5. However, for the highest-memory configurations, ALT-64 slightly outperforms R2R-50k-U.

As mentioned in the Related Work section, it is noteworthy that, using the original termination criterion for bidirectional A* search, our experiments show that both R2R-10k-B-M and R2R-50k-B-M perform even worse than their unidirectional counterparts. However, in contrast, both of our newly-proposed bidirectional termination criteria greatly improve upon the unidirectional variant, with the Intersection criterion performing the best overall, resulting in speed improvements by a factor of 15 and 5 over ALT for the lowest- and highest-memory configurations, respectively.

Of further interest is the fact that, despite these high levels of performance improvement over ALT, the Intersection criterion expands only a factor of 6 and 2 fewer nodes than ALT for the lowest- and highest-memory configurations, respectively. So why the much higher performance improvements over ALT? This comes primarily from the fact that the R2R heuristic function requires only two memory accesses (one to identify a node's region and one to lookup the cost matrix entry for that region to the target region), whereas ALT can require multiple memory accesses, due to the lookup of multiple landmark costs, as well as multiple arithmetic operations (to compute the differences in landmark costs). In-

Table 3: Approximation Errors on the North American graph.

	Error			
Algorithm	Avg. Relative Error (%)	Avg. Absolute Error (minutes)	Max Relative Error (%)	Max Absolute Error (minutes)
R2R-10k-B-I	0.44%	2.98	35.21%	108.77
R2R-10k-B-S	$<$ **0.01%**	$<$ **0.01**	**4.75%**	**8.98**
R2R-50k-B-I	0.15%	0.99	24.40%	34.59
R2R-50k-B-S	$<$ **0.01%**	$<$ **0.01**	**1.20%**	**1.14**

tuitively, this supports the expectation that not only does the accuracy of the heuristic play an important role in performance, but also the computational complexity of evaluating the heuristic as well. In this regard, R2R excels over ALT.

5.4 Solution Quality

In Table 3, we present the approximation errors achieved by our two $(1, \epsilon)$-approximate termination criteria for both the 10k and 50k region partitionings (all other algorithms from Table 2 are omitted, as they compute optimal solutions). This includes both the average and maximum relative errors (as the percentage difference from the true shortest path costs), as well as the average and maximum absolute errors (as the absolute difference from the true shortest path costs). Both methods easily achieve $< 1\%$ relative error and < 3 minutes absolute error, on average, for both partition sizes. For the faster Intersection criterion, despite good average solution quality, it has worst-case absolute errors of nearly 2 hours for the 10k partitioning and half an hour for the 50k partitioning. However, with only slightly slower query times, the Sum termination criterion is able to achieve near-optimal approximation results with the worst-case absolute error being < 9 minutes from the true shortest path for the 10k partitioning. This improves to < 2 minutes worst-case absolute error for the 50k partitioning.

6 Conclusion

We have presented a new theoretical outlook on the efficiency of A* search when using inconsistent heuristic functions with bounded estimation error. Experimental evidence further supports this theory, suggesting that inconsistent heuristic functions can sometimes outperform consistent heuristic functions in practice. The hope is that this work will serve as a catalyst to broaden the scope of ongoing research related to A* search techniques with inconsistent heuristic functions in the context of bidirectional search.

7 Acknowledgements

This work was partially supported by NSF IIS-1144158 and IIS-0803410. We also thank the reviewers for their feedback.

References

Bauer, R.; Delling, D.; Sanders, P.; Schieferdecker, D.; Schultes, D.; and Wagner, D. 2008. Combining Hierarchical and Goal-Directed Speed-Up Techniques for Dijkstra's Algorithm. In *WEA*, 303–318.

Chenoweth, S. V., and Davis, H. W. 1991. High-Performance A* Search Using Rapidly Growing Heuristics. In Mylopoulos, J., and Reiter, R., eds., *IJCAI*, 198–203. Morgan Kaufmann.

Delling, D., and Wagner, D. 2007. Landmark-Based Routing in Dynamic Graphs. In *WEA*, 52–65.

Dinh, H. T.; Russell, A.; and Su, Y. 2007. On the Value of Good Advice: The Complexity of A* Search with Accurate Heuristics. In *AAAI*, 1140–1145. AAAI Press.

Felner, A., and Sturtevant, N. R. 2009. Abstraction-Based Heuristics with True Distance Computations. In *SARA*.

Geisberger, R.; Rice, M. N.; Sanders, P.; and Tsotras, V. J. 2012. Route Planning with Flexible Edge Restrictions. *J. Exp. Algorithmics* 17(1):1.2:1.1–1.2:1.20.

Geisberger, R.; Kobitzsch, M.; and Sanders, P. 2010. Route Planning with Flexible Objective Functions. In *ALENEX*, 124–137.

Goldberg, A. V., and Harrelson, C. 2005. Computing the Shortest Path: A* Search Meets Graph Theory. In *SODA*, 156–165. SIAM.

Harris, L. R. 1974. The Heuristic Search Under Conditions of Error. *Artif. Intell.* 5(3):217–234.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. In *IEEE Transactions on System Science and Cybernetics*, volume 4.

Helmert, M., and Röger, G. 2008. How Good is Almost Perfect? In Fox, D., and Gomes, C. P., eds., *AAAI*, 944–949. AAAI Press.

Ikeda, T.; Hsu, M.-Y.; Imai, H.; Nishimura, S.; Shimoura, H.; Hashimoto, T.; Tenmoku, K.; and Mitoh, K. 1994. A Fast Algorithm for Finding Better Routes by AI Search Techniques. In *Proceedings of the Vehicle Navigation and Information Systems Conference (VNSI'94)*, 291–296. ACM Press.

Kaindl, H., and Kainz, G. 1997. Bidirectional Heuristic Search Reconsidered. *J. Artif. Intell. Res. (JAIR)* 7:283–317.

Köll, A. L., and Kaindl, H. 1993. Bidirectional Best-First Search with Bounded Error: Summary of Results. In *IJCAI*, 217–223.

Kwa, J. B. H. 1989. BS*: An Admissible Bidirectional Staged Heuristic Search Algorithm. *Artif. Intell.* 38(1):95–109.

Martelli, A. 1977. On the Complexity of Admissible Search Algorithms. *Artif. Intell.* 8(1):1–13.

Maue, J.; Sanders, P.; and Matijevic, D. 2006. Goal Directed Shortest Path Queries Using Precomputed Cluster Distances. In *WEA*, 316–327.

Pijls, W., and Post, H. 2009. A New Bidirectional Search Algorithm with Shortened Postprocessing. *European Journal of Operational Research* 198(2):363–369.

Pohl, I. 1969. *Bidirectional Heuristic Search in Path Problems*. Ph.D. Dissertation, Stanford University.

Rice, M. N., and Tsotras, V. J. 2010. Graph Indexing of Road Networks for Shortest Path Queries with Label Restrictions. *PVLDB* 4(2):69–80.

Zhang, Z.; Sturtevant, N. R.; Holte, R. C.; Schaeffer, J.; and Felner, A. 2009. A* Search with Inconsistent Heuristics. In *IJCAI*, 634–639.