Bidirectional A*: Comparing balanced and symmetric heuristic methods*

Wim Pijls† and Henk Post‡

Econometric Institute Report EI 2006-41

Abstract

A widely known algorithm for finding the shortest path in a network is Bidirectional A*. The version of bidirectional A* that is considered the most appropriate hitherto, uses so-called balanced heuristic estimates. In this paper, we focus on symmetric heuristic estimates. First, we show that bidirectional A* using the symmetric heuristic estimate provides us with a feasible approximation. Next a framework is introduced for solving the shortest path problem exactly. It turns out that both the balanced and the symmetric heuristic estimate are instances of a general bidirectional A* framework. The symmetric instance surpasses the balanced instance in space and time.

Keywords: Shortest path, Network flow, Graph theory, Operations Research, Search.

*This research is part of a Ph.D. project of the second author at Delft University of Technology, department of Electrical Engineering, Mathematics and Computer Science.

†Corresponding author. Econometric Institute, Erasmus University Rotterdam, P.O.Box 1738, 3000 DR Rotterdam, The Netherlands, email: pijls@few.eur.nl, tel: +31-10-4082588; fax: +31-10-4089162

‡Connexxion Taxi Services b.v., The Netherlands, email: h.post@connexxion.nl

1 Introduction

In the past few years large digital road maps have become available, to be used in car navigators. This have given rise to a revival of shortest path algorithms. We consider the point-to-point instance of the shortest path problem. The best-known algorithms in Operations Research are Bellman-Ford and Dijkstra [1]. In Artificial Intelligence the A* algorithm is widely known [8]. Both Dijkstra's algorithm and A* start from the origin point and continue their search process until the destination is reached. A new approach, bidirectional search, was introduced by Pohl [7], who proposed two simultaneous processes starting from either point. On two sides Dijkstra or A* is executed. The processes meet somewhere in the middle between the two points. Before a meeting point is obtained, we are in the so-called *main phase*. As soon as the processes meet, the remaining steps of bidirectional search are called the postprocessing phase or briefly the *post-phase*. Some variants of Pohl's approach were discussed in [4] and [6].

The difference between Dijkstra's algorithm and A* is the use of an heuristic estimate function. This function estimates for any node the remaining distance to the destination. Dijkstra may be regarded as an instance of A* with the estimate function $\equiv 0$.

Balanced and symmetric heuristics Bidirectonal search is implemented mostly using *symmetric* heuristic functions, where each process computes the remaining distance in a similar way. Another type is the heuristic which is called *balanced* in the current paper. This type of heuristic has been introduced in [3]. The exact definitions of *symmetric* and *balanced* are found in Section 3.2 and Section 4.1 respectively.
The balanced heuristic estimate is incorporated in recently published experiments with large-scale real road networks, see [2, 5]. According to those papers the balanced heuristic is very beneficial, because the post-phase is very short. However, we return to *symmetric* heuristic functions. First, we found out that, when the post-phase is skipped, the symmetric heuristics still give a tight approximation to the shortest path distance. Second, it turns out that the benefits of the balanced heuristic can be carried over to the symmetric heuristic. To that end a new framework covering both methods is defined. We focus on a version using a symmetric heuristic. This algorithm outperforms in space and time the versions with balanced heuristics in aforementioned papers.

Which estimating method? The obvious estimate is the astronomical distance between two nodes. Many papers discussing large-scale road networks have been published in the past few years, most of which propose novel heuristic estimates. In this paper we refrain from discussions about the quality of the different estimates. The current paper aims at optimizing the search space and the runtime of the process, given a certain heuristic.

Overview. Section 2 gives some general preliminaries. In Section 3.1 we recall the classical A* algorithm along with some properties relevant to discuss the bidirectional properties later on. Section 3.2 contains an elementary but effective and useful bidirectional A* algorithm. Section 4 presents our new framework for bidirectional A* along with several instances therein. The properties of the search space of the new framework are treated in Section 5. Experimental results are shown in Sections 3.2 and 4.3.

2 Preliminaries.

Let a directed graph or network G be given by a pair (V, E) with V the set of nodes and E the set of edges. A path is a sequence of nodes without duplicate elements. We assume that two particular nodes are given, an origin o and a destination node d respectively. The shortest path and its length from the origin to the destination is looked for. The weight or length of an edge (u, v) is denoted by $d(u, v)$, whereas $d^*(u, v)$ denotes the length of a shortest path from u to v.
We define a heuristic estimate h as a function from V into $\mathbb{R}$. An estimate h is called *consistent* if h obeys the inequality $h(u) - h(v) \leq d^*(u, v)$ for any two vertices $u, v \in V$. In some textbooks a different definition is found: $h(u) - h(v) \leq d(u, v)$ for any edge $u, v \in V$. The two definitions are equivalent, as can readily be shown. If a function h is consistent, so is $h + c$ for any constant c.

Heuristic estimates. Let $\delta(v, w)$ denote a underestimate of $d^*(v, w)$ with the property that δ obeys the triangle inequality, i.e. $\delta(v, w) \leq \delta(v, u) + \delta(u, w)$ for any triple u, v, w. Then $h(v) = \delta(v, t)$ with t a fixed node in the graph is a consistent heuristic function, because applying the triangle inequality we have $h(v) - h(w) = \delta(v, t) - \delta(w, t) \leq \delta(v, w) \leq d^*(v, w)$. As said before, the most obvious choice for $\delta(v, w)$ is the astronomical distance between u and v.
Apart the above heuristic $h(v) = \delta(v, t)$, we can derive other heuristic functions from δ. More generally, we can define a heuristic function h for given a distance function δ as:

$$h = \alpha \cdot \delta(v, t) - \beta \cdot \delta(s, v) \tag{1}$$

with α and β two non-negative numbers such that $\alpha + \beta = 1$. In the remainder of this paper we pay special attention to three options for the combination α-β:

$$\begin{array}{ll} \alpha = 1 \;\; \beta = 0: & h_1 = \delta(v, t) \\ \alpha = \frac{1}{2} \;\; \beta = \frac{1}{2}: & h_2 = \delta(v, t)/2 - \delta(s, v)/2 \\ \alpha = 0 \;\; \beta = 1: & h_3 = -\delta(s, v). \end{array}$$

Experiments We conducted multiple experiments on the road network of the Netherlands and Belgium, including the border regions of Germany. This network is part of the Multinet version 2006 provided by TeleAtlas[10]. Speaking formally the network is a directed graph consisting of 3,304,638 nodes and 6,942,109 directed edges. When one can drive in a road into two directions, this road corresponds to two edges. A one-way road corresponds to one edge. In all experiments the astronomical distance acts as heuristic estimate. The experiments were conducted on a Dell Latitude D600 with a 2 Ghz processor and 2 Gb memory, where the programs are developed using Borland Delphi 2006. Experiments with the same environment were shown in [5].

3 The A* algorithm

In this section we recall A*. First unidirectional A* is treated and next we discuss a version of bidirectional A* without the post-processing steps.

Algorithm 1 The Unidirectional A* algorithm

1: **for all** $v \in V$ **do**
2: $g(v) = \infty$;
3: **end for**
4: $S = \emptyset$;
5: $g(s) = 0$;
6: **while** $t \notin S$ **do**
7: $C = \{v \mid v \notin S\}$;
8: $u_0 = \arg\min\{g(v) + h(v) \mid v \in C\}$
9: $F = g(u_0) + h(u_0)$;
10: $S = S + \{u_0\}$;
11: **for all** edges (u_0, v) with $v \notin S$ **do**
12: **if** $g(v) > g(u_0) + d(u_0, v)$ **then**
13: $g(v) = g(u_0) + d(u_0, v)$;
14: $\text{pred}(v) = u_0$;
15: **end if**
16: **end for**
17: **end while**

3.1 The description of the A* algorithm

Before discussing a new bidirectional A* algorithm, we elaborate on the classical A* algorithm. To distinguish this algorithm from its bidirectional counterpart, we call this algorithm unidirectional A*. The pseudo-code is displayed in Algorithm 1. The values $g(v)$ and *pred*(v) are called the label and the predecessor respectively of v. We see that, whenever a node u_0 assigns a new value $g(v)$ to a node v (line 13), u_0 becomes the predecessor of v. In unidirectional A* node s denotes the origin node o and t denotes the destination d. The h-function involved conforms to formula (1). Notice that $h(t) \neq 0$ if $\beta > 0$. With each finite label value $g(v)$ a path can be associated which goes from s to v, where each node is the predecessor of the next node in the path. Therefore this path is called the *predecessor path* of v. The set S is the set of nodes with a permanent label.
In view of Section 4 we have two variables C and F. The set C contains the candidates for insertion into S. The variable F is a lower bound of the set $\{g(v) + h(v) | v \notin S\}$ after line 9 has been executed. When line 13 is executed, this property is preserved, because $F = g(u_0) + h(u_0) \leq g(u_0) + d(u, v) + h(v) = g(v) + h(v)$. The inequality in this relation is a result of the consistency of h. The variable F will play an important role in Section 4.

In order to discuss the features of bidirectional A* we first give the following lemma referring unidirectional A*.

Lemma 1 *The A* algorithm has two invariant properties:*

a) if $v \in S$, then $d^*(s, v) = g(v)$.

b) if $v \notin S$ but $g(v)$ is finite, then $g(v)$ is equal to the length of the shortest path through S from s to v, so the shortest path with the restriction that any intermediate node between s and v is in S.

Proof a) Suppose u_0 is to be added to S. Consider a path P from s to u_0 with minimal length. We denote the first node beyond S on this path by p (maybe $p = u_0$). As a result of part b) $g(p) = d^*(s, p)$ holds. The consistency of h says: $d^*(p, u_0) \geq h(p) - h(u_0)$. Because $g(u_0) + h(u_0)$ is minimal outside S by the selection criterion, $g(u_0) + h(u_0) \leq g(p) + h(p)$. The following (in)equalities apply:

$$\begin{aligned} g(p) + h(p) &= d^*(s, p) + h(p) \\ &\leq d^*(s, p) + d^*(p, u_0) + h(u_0) \\ &\leq g(u_0) + h(u_0) \\ &\leq g(p) + h(p). \end{aligned}$$

We conclude that the above inequalities are equalities and consequently $g(u_0) = d^*(s, u_0)$. b) For a node $v \notin S$ the shortest path through S may change when u_0 is added to S. A shorter path may arise by inserting u_0 in front of v in the existing path for v. To check whether this is the case, lines 11 through 16 are executed. □

Note. Suppose that multiple consistent h-functions are available. The above proof shows that it is allowed to switch randomly from one h-function to another. Both parts of Lemma 1 keep valid in that case.

The nodes in S are called *scanned* and the nodes outside S but with a finite label are called *labeled*. A node with an infinite label is called *unreached*. These names are adopted from [2] and [4]. Notice that each node that is a predecessor of another node, is included in S. At any time we have a set S surrounded by labeled nodes.

3.2 Bidirectional A*, an elementary version

Now we study the bidirectional version. The code of Algorithm 1 may run simultaneously in a forward and a backward direction. In the forward search s denotes the origin and t denotes the destination; in the backward process s and t play inverse roles. The backward search runs on the original set of nodes with a new edge set. The original set E is replaced with the set E_b, such that $(i, j) \in E$ with length $d(i, j)$ corresponds to an edge $(j, i) \in E_b$ with $d_b(j, i) = d(i, j)$. We apply *symmetric* heuristic functions. This means that formula (1) takes the following shape in the forward and backward search respectively.

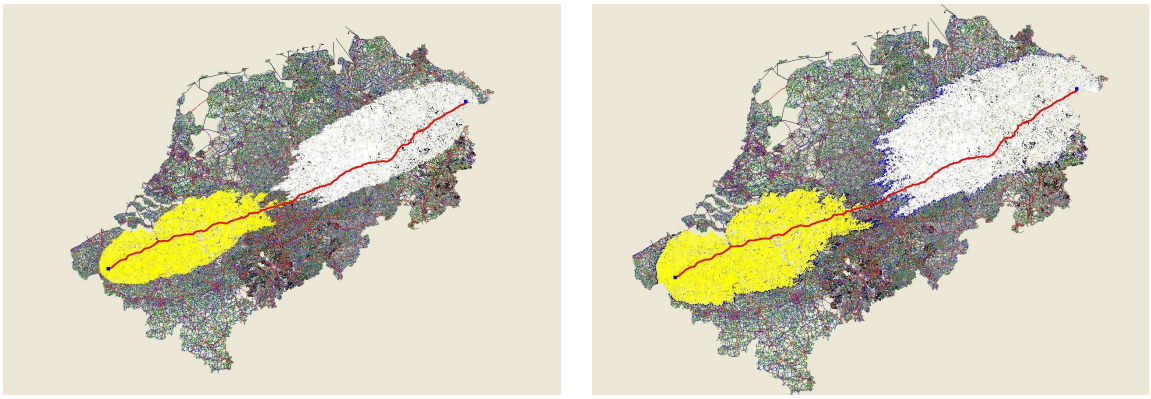
$$h_f(v) = \alpha \cdot \delta(v, d) - \beta \cdot \delta(o, v) \quad (2)$$

$$h_b(v) = \alpha \cdot \delta_b(v, o) - \beta \cdot \delta_b(d, v) = \alpha \cdot \delta(o, v) - \beta \cdot \delta(v, d) \quad (3)$$

where o and d denote the origin and the destination respectively. We can define h_1, h_2 or h_3, each corresponding to a combination of α and β as listed in Section 2. The definition of *symmetric* also assumes that either side takes the same combination α-β.

For this elementary version, we decide that the algorithm stops as soon as the forward and the backward S set have one node in common. Then a path from the origin to the destination has been found, which is not necessarily a shortest path.

Table 1 shows the experimental results of bidirectonal A*. In each experiment we considered 100 random origin-destination pairs. Alternately the forward and the backward search executed one iteration. The figures in the tables show the summation of the results of the 100 runs. In each run the results of the two search processes are added. The times are



a) after the main phase using h_1. b) after the main phase using h_2.

Figure 1: Search spaces.

	scanned	finite label	CPU time	distance
h_1	18,936,207	19,180,997	32.891	23,487,238
h_2	25,708,396	26,034,428	51.039	23,499,216
h_3	71,569,899	72,586,139	180.666	24,452,889

Table 1: The algorithm stops when the processes meet.

measured in seconds, the distances in meters. An illustration for one randomly chosen pair is shown in Figure 1. The bright areas represent the two S sets on termination.
The exact distance is 23,486,861 meters. The closed approximation to the distance is reached by h_1 The deviance for h_1 is less than 0.01%!. Our conclusion is that the elementary bidirectional A* implementation using h_1 is a feasible algorithm for practical applications. For h_2 the deviance is slightly greater. The table shows that h_1 dominates by far h_2 and h_3 in space and time.

4 The extended bidirectional description

The process studied in Section 3.2 stops as soon as the two searches have one node in common. The path from the origin to the destination provided at that time needs not to be the shortest path. To obtain the exact shortest distance we have to continue the process. The process after the first meeting point has been achieved, has been called the *post-phase* in Section 1.

4.1 Postprocessing for balanced heuristics

When the A* algorithm is running on a certain side, it can benefit from intermediate results in the opposite process. The variables on the opposite process are denoted using a tilde. So we have the set $\tilde{S}$ and for any v we have the functions: $\tilde{g}(v)$ and $\tilde{h}(v)$. Notice that $t = \tilde{s}$

and $s = \tilde{t}$. An heuristic is called *balanced* if for any $v \in V$: $h(v) + \tilde{h}(v) = 0$. (In [2] this type of heuristic was confusingly called 'consistent'. By far most of the literature uses the term *consistent* in the same sense as we do). An example of a balanced heuristic is h_2 applied in a symmetric setting: if $h_2(v) = \delta(v,t)/2 - \delta(s,v)/2$ then $\tilde{h}_2(v) = \tilde{\delta}(v,\tilde{t})/2 - \tilde{\delta}(\tilde{s},v)/2 = \delta(s,v)/2 - \delta(v,t)/2$.

For a balanced heuristic the post-phase is short, as has been pointed out in [2] and [3]. The post-phase is short, because only nodes that are labeled in the own search and scanned in the opposite search, need to be inspected. Let $\mathcal{L}$ denote the length of the path found at the end of the main phase. We show that any path through a labeled node that is not scanned in the opposite search is at least as long as $\mathcal{L}$. See Figure 2 (maybe $u = v$) with u and v labeled nodes. As F is a lower bound to the $g + h$-values outside S, $g(u) + h(u) \geq F$ and $\tilde{g}(v) + \tilde{h}(v) \geq \tilde{F}$. Consider a shortest path P from the the origin to the destination with u and v the nodes immediately outside S and $\tilde{S}$ respectively. Using Lemma 1 we state that the length of P equals $g(u) + \tilde{g}(v) + d^*(u,v)$. A lower bound to this value is obtained as follows:

$$\begin{aligned} g(u) + \tilde{g}(v) + d^*(u,v) &\geq \\ g(u) + \tilde{g}(v) + h(u) - h(v) &\geq \\ g(u) + h(u) + \tilde{g}(v) + \tilde{h}(v) &\geq \\ F + \tilde{F}. & \end{aligned}$$

Both F and $\tilde{F}$ are non-decreasing values and $F + \tilde{F} \geq \mathcal{L}$ is an invariant relation in the post-phase. It follows that the value $\mathcal{L}$ cannot be improved by any path through u or v. So only nodes that are scanned on one side, are eligible to be scanned on the alternate side. An equivalent statement is: only edges connecting two nodes scanned in either side need to be inspected in the post-phase.

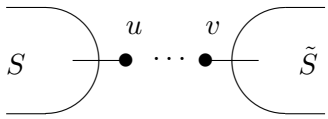


Figure 2: Pruning nodes using $h(u) - h(v)$ or $\tilde{h}(v) - \tilde{h}(u)$.

4.2 The new description

The above reasoning can be generalized to an arbitrary heuristic. See again Figure 2. Similarly to Section 4.1 we can derive that any shortest path P has length $\geq g(u) + \tilde{g}(v) + \tilde{h}(v) - \tilde{h}(u)$. This value is $\geq g(u) + \tilde{F} - \tilde{h}(u)$. This implies that any node $u \notin \tilde{S}$ satisfying

$$g(u) + \tilde{F} - \tilde{h}(u) \geq \mathcal{L}$$

can be eliminated. Those nodes are excluded from being scanned. This feature is implemented in Algorithm 2.
Algorithm 2 is an extended version of Algorithm 1. It it designed to run simultaneously on

two sides, like Algorithm 1, starting from the origin ($s = o, t = d$) and from the destination ($s = d, t = o$). The code contains a number of additional statements related to the post-phase. The novel feature is included in lines 13 and 14. The entire process stops when one side stops, i.e. when one side has an empty candidate set. The variable $\mathcal{L}$ is equal at any time to the length of a path from o to d through a doubly scanned node. On termination this path is the desired path.
Algorithm 2 may be viewed as a *framework* covering arbitrary heuristics. Apart from the consistency property, nothing is required of the heuristic estimates.
The algorithm utilizes the variables $\tilde{S}, \tilde{F}$ and $\tilde{h}$ of the opposite search. These variables are read-only variables and can only be set by the opposite search process. The variable $\mathcal{L}$ is a shared variable, which is a read/write variable for both running instances.

Instead of three states for a node, as was the case in unidirectional A*, we now have four states. Next to the states *scanned*, *labeled* and *unreached*, we also have *rejected*. A candidate node in C may come into the state *scanned* (line 16) or *rejected* (line 14) respectively. Like *scanned* the state *rejected* for a node v is kept until the end. Notice that $g(v) + h(v) \leq F$ is an invariant property for a such a node. The union of the sets of *labeled* and *rejected* nodes respectively is called the *open* set. Notice that rejected nodes do not contribute to the F-value.

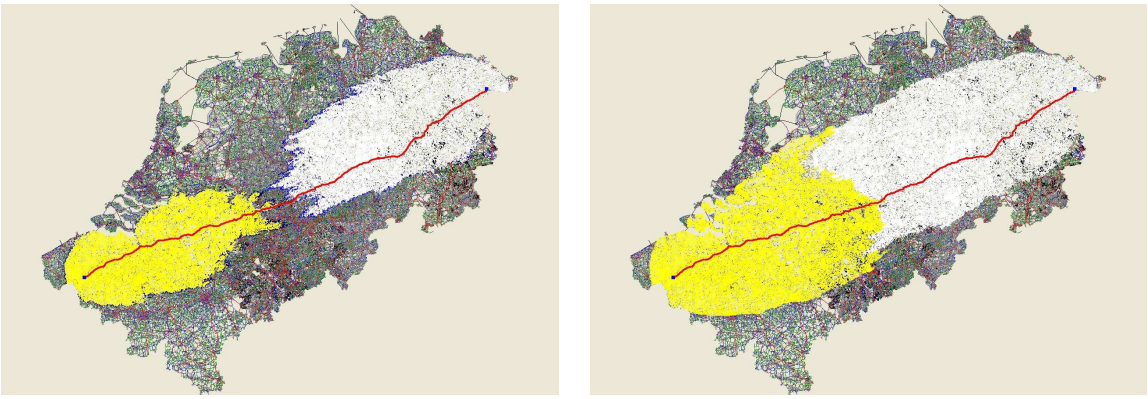
The eliminating method in lines 13 and 14 reduces the search space considerably. Two other methods to restrict the search space, already mentioned in [6], are also applied. Any node v with $g(v) + h(v) - h(t) \geq \mathcal{L}$ should not to be scanned, because a path through v is not able to improve the shortest path found so far (This was called 'trimming' in [6]). A node that becomes doubly scanned does not expand any new node ('nipping'), so for such a node the lines 25 through 30 are skipped. If a node u_0 has a doubly scanned direct or indirect predecessor, u_0 needs not to be scanned ('pruning'). The code of Algorithm 2 does not reflect this idea because checking the condition does not outweigh the savings. We show in Section 5.2, that, even when this idea is not implemented explicitly, a node v with a doubly scanned predecessor will not be scanned, if the condition $v \notin \tilde{S}$ holds.

Does Lemma 1 still hold? Because a node u that becomes doubly scanned does not update neighbor labels, Lemma 1b) is not valid for a node v, if v has a doubly-scanned node on its shortest path from s. Fortunately, such a node v is not relevant for improving the value of $\mathcal{L}$
Apart from this exception, Lemma 1 is still correct, but the proof needs a small extension. Suppose a node u_0 is selected. We have to consider the case that p, as defined in the proof of Lemma 1, is rejected. Then $g(u_0) \geq d^*(s, u_0) = d^*(s, p) + d^*(p, u_0) = g(p) + d^*(p, u_0)$. By the definition of the graph used in the opposite search $d^*(p, u_0) = \tilde{d}(u_0, p)$. The latest value is $\geq \tilde{h}(u_0) - \tilde{h}(p)$ due to the consistency. It follows that $g(u_0) + \tilde{F} - \tilde{h}(u_0) \geq g(p) + d^*(p, u_0) + \tilde{F} - \tilde{h}(u_0) \geq g(p) + \tilde{F} - \tilde{h}(p)$. Because p is rejected, u_0 also satisfies $g(u_0) + \tilde{F} - \tilde{h}(u_0) \geq \mathcal{L}$ and therefore u_0 is not selected. We conclude that p cannot be rejected, when u_0 is to be scanned.

The correctness proof of Algorithm 2 is rather complex. As rejected nodes do not contribute to the F-value (rejected nodes u have $g(u) + h(u) \leq F$), $\tilde{F}$ is not a lower bound to the $g + h$-values of the open nodes in the opposite search. Consequently $g(u) + \tilde{F} - \tilde{h}(u)$ is not a lower bound to the path lengths between u and t. In Section 5 we complete the proof.

Algorithm 2 The Bidirectional A* algorithm

1: **for all** $v \in V$ **do**
2: $g(v) = \infty$;
3: **end for**
4: $S = \emptyset$;
5: $\mathcal{L} = \infty$;
6: $g(s) = 0$; // s becomes labeled
7: **boolean** cand-found=true; //stands for 'candidate found'
8: **while** cand-found==true **do**
9: $C = \{v \mid v$ is *labeled* and $g(v) + h(v) - h(t) < \mathcal{L}\}$;
10: cand-found=false;
11: **while** $C \neq \emptyset$ and cand-found==false **do**
12: $u_0 = \arg\min\{g(v) + h(v) \mid v \in C\}$;
13: **if** $u_0 \notin \tilde{S}$ and $g(u_0) + \tilde{F} - \tilde{h}(u_0) \geq \mathcal{L}$ **then**
14: $C = C - \{u_0\}$ // u_0 becomes *rejected*
15: **else**
16: cand-found=true; // a suitable candidate is found
17: **end if**
18: **end while**
19: **if** cand-found==true **then**
20: $S = S + \{u_0\}$; //u_0 becomes *scanned*
21: $F = g(u_0) + h(u_0)$;
22: **if** $u_0 \in \tilde{S}$ **then**
23: $\mathcal{L} = \min(\mathcal{L}, g(u_0) + \tilde{g}(u_0))$; // u_0 becomes doubly scanned, $\mathcal{L}$ is updated.
24: **else**
25: **for all** edges $(u_0, v) \in E$ with v *labeled* or *unreached* **do**
26: **if** $g(v) > g(u_0) + d(u_0, v)$ **then**
27: $g(v) = g(u_0) + d(u_0, v)$; //$g(v)$ becomes (re-)labeled
28: pred$(v) = u_0$;
29: **end if**
30: **end for**
31: **end if**
32: **end if**
33: **end while**



a) after the post-phase using h_1. b) without rejected nodes using h_1.

Figure 3: Search spaces.

4.3 Experiments

The experimental outcomes are shown in Table 2. The same instances as in Section 3.2 are used. As mentioned there, the results of the two search processes are added. This means that doubly scanned or doubly labeled nodes are counted twice. The search space for one instance is depicted in Figure 3.
The table in combination with Table 1 conforms to our conclusion in Section 4.1, viz. the post-phase for h_2 generates only new doubly scanned nodes and does not generate new labeled nodes.
The advantage of h_1 in space has diminished, but still it is. In time it far defeats h_2.

	scanned	finite label	CPU time	distance
h_1	25,465,177	25,793,947	43.234	23,486,861
h_2	25,708,620	26,034,428	51.542	23,486,861
h_3	74,975,861	75,998,025	188.401	23,486,861

Table 2: After the post-phase.

According the experiments in [2] and [5] the best implementation of bidirectional A* is achieved, when the balanced heuristic h_2 is applied, due to its short post-phase. We now see that we are not bound to balanced functions. The symmetric heuristic function h_1 performs better. The somewhat awkward function h_2 which also takes more time to compute than h_1 or h_3, is no longer needed.

The tables 1 and 2 and the maps in Figure 3 exhibit another phenomenon. The major differences between the h-functions arise during the main phase. h_1 has a short main phase but looses its benefit in the post-phase. For h_3 the other way round holds: the post-phase is very short compared to the main phase. Moreover, we found that the number of updates of the variable $\mathcal{L}$ in the post-phase is small. So we may regard the post-phase as a certifying process, generating a certificate for the (nearly) shortest path length obtained in the main phase.

To show the effectiveness of the eliminating rule, we have also conducted experiments with lines 13 and 14 omitted. The algorithm obtained this way is identical to bidirectional A* with the symmetric heuristic, as discussed in [2] next to the version with the balanced heuristic. However, leaving out the criterion of line 13 deteriorates the search space dramatically, as we see in Table 3.

	scanned	finite label	CPU time	distance
h_1	45,229,047	45,377,809	67.601	23,486,861
h_2	173,358,453	173,475,559	276.661	23,486,861
h_3	301,975,719	301,999,525	662.230	23,486,861

Table 3: After the post-phase without eliminating rule.

For a node $v \notin \tilde{S}$ we have shown in Section 4.2 that $\tilde{F} - \tilde{h}(v)$ is a lower bound to the length of the paths from v to t. This bound may be used as the heuristic estimate governing the selection of the node to be scanned. This was suggested in [4]. The heuristic $\tilde{F} - \tilde{h}(v)$ is equivalent to $-\tilde{h}(v)$. The experiments in this section show that this a weak estimate. Therefore the lower bound $\tilde{F} - \tilde{h}(v)$ is only suitable to eliminate nodes from the search, as we do in our algorithm.

5 The search space

In this section we study the shape of the search space of Bidirectonal A*. By the search space we mean the set of visited nodes during execution. The properties of the search space also imply the correctness of the algorithm.

5.1 Some properties of the search space

Some formal properties First of all we present two Lemma's. Next we draw some conclusions about the shape of the search space. Finally we show that the shape is almost independent of the choice of α-β.

Lemma 2 *The bidirectional A* algorithm has the following invariant: for any open u and any v that is open in the opposite search (maybe $u = v$) at least one of the following conditions holds:*

a) (u is labeled) $\wedge$ (v is labeled in the opposite search)

b) (u is rejected) $\wedge$ $g(u) + \tilde{g}(v) + \tilde{h}(v) - \tilde{h}(u) \geq \mathcal{L}$

c) (v is rejected in the opposite search) $\wedge$ $g(u) + \tilde{g}(v) + h(u) - h(v) \geq \mathcal{L}$.

Proof. We distinguish between the primary process and the opposite process.
The invariant holds when the algorithm starts. There are three events, which may affect the invariant.
Suppose u becomes rejected in the primary process, while v is labeled on the opposite side.

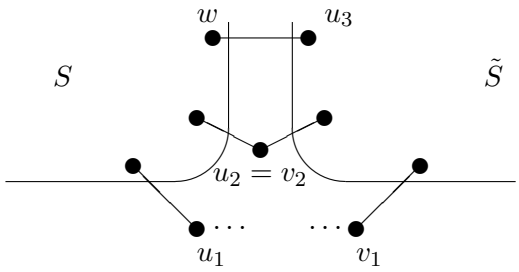


Figure 4: The search space on termination.

Apparently the condition in line 13 holds and b) becomes valid, because $g(u)+\tilde{g}(v)+\tilde{h}(v)-\tilde{h}(u) \geq g(u)+\tilde{F}-\tilde{h}(u) \geq \mathcal{L}$.
Suppose u becomes rejected in the primary process, while v is already rejected in the opposite search. This is the mirror of b), expressed by c). Notice that the inequality in line 13 of the code does not imply the inequality in b), due to the relation $\tilde{g}(v)+\tilde{h}(v) \leq \tilde{F}$ on the opposite side. So b) needs not to hold.
Suppose a node u_0 becomes scanned and generates a new label for u, while v is rejected. Then $g(u)+h(u) = g(u_0)+d(u_0,u)+h(u) \geq g(u_0)+h(u_0)$. As invariant c) applied to the pair u_0, v, it now applies to u and v. □

If $u = v$ in the Lemma 3b) can transform into: if a node u is rejected and u is open on the opposite side, then $g(u)+\tilde{g}(u) \geq \mathcal{L}$.

Lemma 3 *If v is rejected, then $v \notin \tilde{S}$.*

Proof By the first condition in line 13 a node $v \in \tilde{S}$ will not be rejected in the primary process. Conversely, assume that v has taken the status rejected in the primary process. We show that v will not be added to $\tilde{S}$. Because v is rejected, $g(v)+h(v) \leq F$ and, as a consequence of Lemma 3b): $g(v)+\tilde{g}(v) \geq \mathcal{L}$. Now we have: $\tilde{g}(v)+F-h(v) \geq \tilde{g}(v)+g(v)+h(v)-h(v) \geq \mathcal{L}$ and thus v will not be added to $\tilde{S}$. □

For illustration see Figure 4 which displays a configuration with open nodes. Suppose the process on the left-hand side has an empty candidate set, causing the entire algorithm to terminate. Apparently none of the three open u nodes is a suitable candidate on the left side for getting scanned. As Lemma 3 implies that $u_3 \in \tilde{S}$ is not rejected, u_3 must satisfy the relation:

$$g(u)+h(u)-h(t) \geq \mathcal{L}. \tag{4}$$

According to Lemma 2, each of the pairs u_2-v_2 and u_1-v_1 obeys at least one of the following relations:

$$g(u)+\tilde{g}(v)+\tilde{h}(v)-\tilde{h}(u) \geq \mathcal{L} \tag{5}$$
$$g(u)+\tilde{g}(v)+h(u)-h(v) \geq \mathcal{L} \tag{6}$$

provided that the u-nodes are rejected; otherwise (4) holds for u-nodes.

Now that we have derived some properties of the search space, we prove the correctness of the algorithm and we pose a statement on the shape of the search of the space.

Correctness of the Algorithm 2 Suppose a shortest path P from the origin to the destination has zero, one or multiple intermediate nodes between S and $\tilde{S}$. This corresponds to situations with respectively the combinations w-u_3, w-u_3 or u_1-v_1 on P. In each case relation (4), (5) or (6) holds, implying that a path with length $= \mathcal{L}$ is minimal. This shows that Algorithm 2 is correct.

The influence of α-β on the search space In practice the relation (4) mostly occurs in the areas farthest from target t. A non-rejected open node in $\tilde{S}$, as is the case with u_3, is very rare. Such a situation nearly always leads to a doubly scanned node. When the algorithm stops, mainly (5) or (6) holds in the area where the S-sets are close to each other.

If u and v have a short distance between themselves, which is relatively short compared with the distance to their start node, we may state that $\delta(u,v) \approx \delta(s,v) - \delta(s,u)$ and $\delta(u,v) \approx \delta(u,t) - \delta(v,t)$. Then the following derivation is allowed:

$$\begin{aligned} h(u) - h(v) &= \alpha \cdot \delta(u,t) - \beta \cdot \delta(s,u) - \alpha \cdot \delta(v,t) + \beta \cdot \delta(s,v) \\ &= \alpha \cdot (\delta(u,t) - \delta(v,t)) + \beta \cdot (\delta(s,v) - \delta(s,u)) \\ &\approx \alpha \cdot \delta(u,v) + \beta \cdot \delta(u,v) = \delta(u,v). \end{aligned}$$

A similar derivation leads to: $\tilde{h}(v) - \tilde{h}(u) \approx \tilde{\delta}(v,u) = \delta(u,v)$.
For nodes close to the opposite S set, the relations (5) and (6) mostly apply. Using the above derivations we can transform (5) and (6) into:

$$g(u) + \tilde{g}(v) + \delta(u,v) \geq \mathcal{L}, \tag{7}$$

avoiding the h-function and hence avoiding α and β.

The consequence is that in the region of the network where the S sets are close to each other, the shape of the search space hardly depends on the choice of the combination α-β. The experiments in Section 4.3 support this statement. This also explains why the space advantage of h_1 over h_2 in the main phase reduces in the post-phase.

5.2 Doubly scanned nodes

Pruning a node u with a doubly scanned predecessor is not expressed by the code of Algorithm 2, as we mentioned in Section 4. However, a special case of pruning is carried out implicitly by this code. Assume a labeled node $u \notin \tilde{S}$ is selected to be scanned and u has a doubly scanned predecessor a. When u is to be scanned, its predecessors including a make up a shortest path from s to u. Clearly $\mathcal{L} \leq g(a) + \tilde{g}(a)$ and $\tilde{g}(a) + \tilde{h}(a) \leq \tilde{F}$. Lemma 1 tells us that $g(a) = d^*(s,a)$. The following relations are invariant relations.

$$\begin{aligned} g(u) + \tilde{F} - \tilde{h}(u) &\geq d^*(s,u) + \tilde{g}(a) + \tilde{h}(a) - \tilde{h}(u) \\ &\geq d^*(s,a) + d^*(a,u) + \tilde{g}(a) + \tilde{h}(a) - \tilde{h}(u) \\ &= d^*(s,a) + \tilde{d}^*(u,a) + \tilde{g}(a) - (\tilde{h}(u) - \tilde{h}(a)) \\ &\geq d^*(s,a) + \tilde{g}(a) = g(a) + \tilde{g}(a) \geq \mathcal{L}. \end{aligned}$$

We conclude that $u \notin \tilde{S}$ with a doubly scanned predecessor is not selected to be scanned.

Figure 5 illustrates a situation in which a node inside $\tilde{S}$ with a doubly scanned predecessor is scanned. Suppose node a is labeled in the right-hand process with b as predecessor.

Likewise b and c are labeled in the left-hand process with a as predecessor. Next, suppose that a (or perhaps a left-hand predecessor of a) becomes scanned in the right-hand process. Then b and c have doubly scanned predecessors in the left-hand process, but presumably

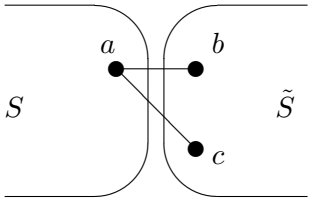


Figure 5: Scanning nodes with a doubly scanned predecessor.

they are scanned in one of the subsequent iterations of that process.
We also see a somewhat curious phenomenon: b pulls successor a into $\tilde{S}$, whereas a pulls successor b into S. This embrace can be prevented by a small addition to the code of Algorithm 2.

6 Concluding remarks

In this paper we have first introduced a useful method closely approximating the shortest path distance. Next a framework has been introduced with the quick algorithm of [2] and [3] as special cases. Another instance viz. the instance using the symmetric heuristic h_1 turns out to be slightly more efficient in space and far more efficient in time. Therefore we recommend this instance for practical applications.
Besides, studying the generalized form has provided more insight into the working of bidirectional search.

References

[1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin, *Network Flows, Theory, Algorithms and Applications,* Prentice Hall, 1993.

[2] Goldberg A.V. and C.Harrelson. Computing the Shortest Path: A* Search Meets Graph Theory, In: 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'05) 2005.

[3] Ikeda T.K., M.-Y. Hsu, H. Inai, S. Nishimura, H. Shimoura, T. Hashimoto, K. Tenmoku, and K. Mitoh. A Fast Algorithm for Finding Better Routes by AI Search Techniques, In: Proceedings Vehicle Navigation and Information Systems Conference. IEEE, 1994.

[4] Kaindl H. and G. Kainz. Bidirectional Heuristic Search Reconsidered, In: Journal of Artificial Intelligence Research 7, 1997, p. 283-317.

[5] Klunder G.A. and H.N. Post. The Shortest Path Problem on Large Scale Real Road Networks, In: Networks, 48:4, 2006, p. 182-194.

[6] Kwa J. B. H. BS*: An Admissible Bidirectional Staged Heuristic Search Algorithm, In: Artificial Intelligence 38:1, 1989, p. 95-109.

[7] Pohl I. Bi-Directional Search, Machine Intelligence 6, 1971, p. 124-140.

[8] S.J. Russell and P. Norvig, *Artificial Intelligence, A modern Approach*, Prentice Hall, 2003.

[9] Yamaguchi K. and S. Masuda. An A* Algorithm with a New Heuristic Distance Function for the 2-Terminal Shortest Path Problem, In: IEICE Transactions on Fundamentals of Electronics E89-A, 2006, p. 544-550.

[10] `www.teleatlas.com`.