A new bidirectional algorithm for shortest paths

Wim Pijls* Henk Post†

Econometric Institute Report EI 2008-25

November 14, 2008

Abstract

For finding a shortest path in a network the bidirectional A* algorithm is a widely known algorithm. An A* instance requires a heuristic estimate, a real-valued function on the set of nodes. The version of bidirectional A* that is considered the most appropriate in literature hitherto, uses so-called balanced heuristic estimates. This means that the two estimates of the two directions are in balance, i.e., their sum is a constant value. In this paper, we do not restrict ourselves any longer to balanced heuristics. A generalized version of bidirectional A* is proposed, where the heuristic estimate does not need to be balanced. This new version turns out to be faster than the one with the balanced heuristic.
Keywords: Shortest path, Road network search, Bidirectional search.

1 Introduction

In the last decade large digital road maps have become available, to be used in car navigators. This has given rise to a revival of shortest path algorithms. Digital road maps contain millions of nodes and edges, whereas previous experiments used small artificial networks. We consider the point-to-point instance of the shortest path problem. The best-known algorithms in Operations Research are Bellman-Ford[1, 3] and Dijkstra[2]. In Artificial Intelligence the A* algorithm[5], which assumes the availability of a heuristic estimate, is widely known. These classical algorithms are nowadays applied in a bidirectional setting. This idea was introduced in [10]. Two simultaneous search processes starting from either endpoint meet somewhere in the middle between the endpoints.
For the A* algorithm one needs to define some heuristic function h estimating the remaining distance from a node to the target. Since there are two processes, two estimates h and $\tilde{h}$ are used, each belonging to one process. The heuristic is called *balanced* if $h(v) + \tilde{h}(v)$ equals a constant value C for any node v.
The algorithm that is considered the most efficient of all shortest path solutions hitherto is *bidirectional A** utilizing a balanced heuristic, cf. [4, 6, 7]. The efficiency of this algorithm is due to a short post-phase. We found that this benefit of a balanced heuristic can be carried over to arbitrary heuristics. So a new version of bidirectional A* is proposed. This algorithm is a slight modification of the one in [9].

*Econometric Institute, Erasmus University Rotterdam, P.O.Box 1738, 3000 DR Rotterdam, The Netherlands, e-mail: pijls@few.eur.nl

†Connexxion Taxi Services, The Netherlands, e-mail henk@ftonline.nl

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, such that two consecutive nodes are connected by an edge. We assume that two particular nodes are given, an origin node and a destination node. The shortest path 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.
As aforementioned, A* assumes a heuristic estimate h, defined as 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 nodes $u, v \in V$. 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'. In this paper h denotes a consistent heuristic function.

2 A new algorithm

Algorithm 1 is a new search algorithm akin to A*. It is assumed that the code runs simultaneously on two sides. Each side has a start node s and an end node t. On one side the origin is chosen as s and the destination as t, on the other side s and t play inverse roles. In the process starting from the origin we use the original graph and the original distance function $d(u, v)$. In the alternate process the so-called reverse graph is used, where every original edge (u, v) is replaced with edge (v, u) of equal distance. Speaking on bidirectional search we refer to the two processes as the *primary* and the *opposite* process. Either side, whether it has the origin or the destination as start node, may be appointed as primary. In the opposite process everything is denoted by a tilde.
The execution of Algorithm 1 stops when one side stops, i.e., when one side has an empty candidate set.

The value $g(v)$ for any $v \in V$ is called the *label* of a node v. The set S is the set of nodes with a permanent label. The code utilizes the variables $\tilde{S}, \tilde{F}$ and $\tilde{h}$ from the opposite search. These variables are read-only variables and can only be set by the opposite search process. The variables $\mathcal{L}$ and $\mathcal{R}$ are shared variables, which are read/write variables for both sides. A meeting point of the two processes arises when a node v has two finite labels, cf. line 28. When $\mathcal{L}$ has a finite value, this values is equal to the length of a path from the origin to the destination. On termination this path is the desired path.

When we run Algorithm 1 in an unidirectional setting, the variable $\mathcal{L}$ keeps its value ∞ and $\mathcal{R}$ remains empty. Then the algorithm is equivalent to the traditional A* algorithm using a consistent heuristic. When $h \equiv 0$ holds, the algorithm is equivalent to Dijkstra's algorithm.

The correctness of Algorithm 1 is proved by Theorem 1. Before we need four Lemmas. The results expressed by the first three lemma are not new, but have been published, sometimes in an implicit way, in earlier papers on A*. Algorithm 1 contains one main loop (*while candidate-found=true...*). Some lemmas present an invariant of this main loop. An invariant of a while loop is an assertion, which holds at the start as well as at the end of each iteration. An invariant is proved by mathematical induction. First, one shows that the invariant holds at the start of the first iteration. Second, under the assumption that the invariant holds at the start of an iteration, one proves that it holds at the end. In our

proofs, we restrict ourselves to the second induction step, the first step being always trivial.

Lemma 1 *The following invariant holds:* $g(u) + h(u) \leq F \leq g(v) + h(v)$ *for any* $u \in S$ *and* $v \notin S$.

Proof When a new node u_0 is selected for insertion into S (cf. line 13), the inequality $g(u) + h(u) \leq F \leq g(u_0) + h(u_0)$ holds due to the invariant itself. Establishing a new value $F = g(u_0) + h(u_0)$ in line 23, preserves the left inequality.
Since $F = g(u_0) + h(u_0)$ was minimal outside S, the right inequality is maintained for nodes v which do not take a new label. For a node v which does take a new label, the following relations hold: $g(v) + h(v) = g(u_0) + d(u_0, v) + h(v) \geq g(u_0) + h(u_0)$, where the latter inequality holds due to the consistency of h. □

Lemma 2 *The following invariant holds:* $g(v) \leq g(u) + d(u, v)$ *for any* $u \in S$ *and* $v \in V$.

Proof When a node u_0 is inserted into S, the invariant holds for any $v \notin S$, due to line 27. By Lemma 1, a node $v \in S$ satisfies $g(v) + h(v) \leq g(u_0) + h(u_0)$ and hence, by the consistency of h, $g(v) \leq g(u_0) + h(u_0) - h(v) \leq g(u_0 + d(u_0, v)$. □

Lemma 3 *Let* P *denote the shortest path from* s *to an arbitrary node* v. *If the intermediate nodes between* s *and* v *belong to* S *, then* $g(v) = d^*(s, v)$.

Proof At any time, $g(v)$ equals the length of path from s to v. (This is proved formally in [8]). Let P be given by the series $(s = p_0, p_1, p_2, \ldots, p_n = v)$ with $p_i \in S$ for $0 \leq i < n$. Then Lemma 2 states: $g(p_{i+1}) \leq g(p_i) + d(p_i, p_{i+1})$ for $0 \leq i < n$. Using the relation $g(s) = 0$, we conclude that $g(v) \leq \sum_{i=0}^{i=n-1} d(p_i, p_{i+1})$. This sum is equal to $d^*(s, v)$. Since $g(v)$ cannot be smaller than $d^*(s, v)$, we have $g(v) = d^*(s, v)$. □

Lemma 4 *The following invariant holds: if* $v \in \mathcal{R}$, *then any path from* s *to* t *through* v *has length* $\geq \mathcal{L}$.

Proof Suppose u_0 is inserted into $\mathcal{R}$ in line 16 of the code. Let P denote a path from s to t through u_0.
If P already contains a node $v \in \mathcal{R}$ with $v \neq u_0$, then the invariant holds.
Let p be the first node beyond S and q the first node beyond $\tilde{S}$. Such nodes p and q can be found since $u_0 \notin S \cup \tilde{S}$. These nodes p and q have finite labels. The (in)equalities in the schema below hold. Here the distances and the labels on the opposite side are denoted by $\tilde{d}$ and $\tilde{g}$ respectively, by analogy with the denotations $\tilde{h}, \tilde{F}$ and $\tilde{S}$.

$$
\begin{aligned}
length(P) &= d^*(s,p) + d^*(p,u_0) + \tilde{d}(t,q) + \tilde{d}^*(q,u_0) \\
&= g(p) + d^*(p,u_0) + \tilde{g}(q) + \tilde{d}^*(q,u_0) && \textit{(by Lemma 3)} \\
&\geq g(p) + h(p) - h(u_0) + \tilde{g}(q) + \tilde{h}(q) - \tilde{h}(u_0) && \textit{(by the consistency of } h\textit{)} \\
&\geq g(p) + h(p) - h(u_0) + \tilde{F} - \tilde{h}(u_0) && \textit{(by the right inequality of Lemma 1)} \\
&\geq g(u_0) + h(u_0) - h(u_0) + \tilde{F} - \tilde{h}(u_0) && (g(u_0) + h(u_0) \textit{ is minimal outside } S) \\
&= g(u_0) + \tilde{F} - \tilde{h}(u_0) \\
&\geq \mathcal{L} && \textit{(since } u_0 \textit{ is inserted into } \mathcal{R}\textit{)}
\end{aligned}
$$

□

Theorem 1 *On termination* $\mathcal{L}$ *is equal to the length of the shortest path from the source to the destination.*

Algorithm 1 New bidirectional A* algorithm

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

Proof Suppose the primary process stops. (The alternate case is symmetrical.) Since the g-labels correspond to path lengths, $\mathcal{L}$ is equal to the length of a path from the source to the destination. Let L denote the length of a shortest path P from s to t. We show that L cannot be smaller than $\mathcal{L}$.
Suppose that P lies entirely in $S \cup \tilde{S}$. Then P includes at least one node with a finite g-label on either side and hence, by line 28 of the code, $L \geq \mathcal{L}$.
Suppose that P lies not entirely in $S \cup \tilde{S}$ and let p denote the first node beyond S on P. If $p \in \mathcal{R}$, then $L \geq \mathcal{L}$ by Lemma 4. If $p \notin \mathcal{R}$, p must violate the inequality in line 10, otherwise the candidate set would not be empty. Lemma 3 states that $g(p) = d^*(s,p)$. It holds that $L = d^*(s,p) + d^*(p,t) = g(p) + d^*(p,t)$. Taking into account the inequality $g(p) + h(p) - h(t) \geq \mathcal{L}$ of line 10 and the consistency property $d^*(p,t) \geq h(p) - h(t)$, we obtain $L \geq \mathcal{L}$. □

In the original bidirectional A* working with a balanced heuristic, a value $\mathcal{L} = g(u_0) + \tilde{g}(u_0)$ is established only when u_0 is included in both S-sets. Then the algorithm enters the postprocessing stage. If any node v between the two S-sets is selected in the new stage, the following (in)equalities hold:

$$\begin{aligned}
\mathcal{L} &= g(u_0) + \tilde{g}(u_0) \\
&= g(u_0) + h(u_0) + \tilde{g}(u_0) + \tilde{h}(u_0) - C && \textit{(h is balanced)} \\
&\leq g(u_0) + h(u_0) + \tilde{F} - C && \textit{(by Lemma 1, left inequality)} \\
&\leq g(v) + h(v) + \tilde{F} - C && \textit{(by Lemma 1)} \\
&= g(v) + \tilde{F} - \tilde{h}(v) && \textit{(h is balanced)}
\end{aligned}$$

Consequently any node $v \notin S \cup \tilde{S}$, when selected, would be inserted into $\mathcal{R}$. Therefore in the postprocessing stage, one inspects only nodes from the opposite set $\tilde{S}$, or equivalently: one inspects edges connecting the two S sets.

3 Implementations

For an efficient implementation, we introduce a slight modification. The shared set $\mathcal{R}$ is divided into two subsets, viz. a subset R for the nodes inserted on the primary side and a subset $\tilde{R}$ its counterpart on the opposite side. Either process considers only its own subset R in line 10 of the code. The correctness is not violated by this modification, as can readily be derived from the proof of Theorem 1. The aim of our modification is to speed up the execution of line 10. The composition of C proceeds more quickly, when R is inspected instead of $\mathcal{R} = R \cup \tilde{R}$. Theoretically, the set C of the primary process might take a node $v \in \tilde{R} \subseteq \mathcal{R}$. In practice, this hardly occurs, since almost any node in $\tilde{R}$ has an infinite g-label. Its inclusion in $\tilde{R}$ is due to its finite $\tilde{g}$-label.
Introducing the above modification recovers the algorithm proposed in [9].

We conducted multiple experiments with Algorithm 1 on the road network of the Netherlands and Belgium, including the border regions of Germany. This network is part of the Multinet version 2007 provided by TeleAtlas. The network is a directed graph consisting of 3,097,648 nodes and 6,559,643 directed edges. We have compared our algorithm with the bidirectional algorithm using a balanced heuristic as described in [4] and [6]. It turns out that our new algorithm clearly beats the other one in running time. For details and figures we refer to [9].

References

[1] R. Bellman, On a routing problem, Quaterly of Applied Mathematics 16(1) (1958) 87-90.

[2] E.W. Dijkstra, A note on two problems in connexion with graphs, Numerische Mathematik 1 (1959) 269-271.

[3] L.R. Ford, Network Flow Theory, Technical Report P-923 Rand Corporation, Santa Monica CA 1956.

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

[5] E. P. Hart, N.J. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE Transaction, System Science and Cybernetics SSC(4)-2 (1968) 100-107

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

[7] G.A. Klunder, H.N Post, The Shortest Path Problem on Large Scale Real Road Networks, Networks, 48(4) (2006) 182-194.

[8] W. Pijls, Heuristic estimates in shortest paths, Statistica Neerlandica, 61(1) (2007) 61-74.

[9] W. Pijls and H. Post, A new bidirectional search algorithm with shortened postprocessing, forthcoming in: European Journal of Operational Research.

[10] I. Pohl, Bi-Directional Search, Machine Intelligence 6 (1971) 124-140.