Performance of the Classical Bidirectional Heuristic Search Reconsidered

Stefan Lewandowski

Abstract: The classical bidirectional heuristic search was introduced in 1971 by Pohl. In practice it did not turn out to be as good as supposed – in fact, in most cases, running times are worse than using the unidirectional heuristc search variant, which was introduced 1968 by Hart et. al., well known as the A* algorithm. In this article, we show that one of the unidirectional searches must always be faster than the bidirectional one. This result also extends to some heuristic improvements made to the bidirectional heuristic search during the past 35 years.

Keywords: heuristic search, bidirectional search, shortest path

1 Introduction

Dijkstra [1] introduced his single source shortest path algorithm for graphs with non-negative weights in 1959. Hart et. al. [3] suggested well known heuristic improvements with their A* algorithm in 1968, using estimates of the distances to the target node to perform a directed search towards the target. When solving the single pair shortest path problem the A* algorithm almost always outperforms Dijkstra's algorithm as it is optimal in the sense that it takes only those nodes into account that can be part of the shortest path (based on the information given by the estimates) ([2], [10]). The bidirectional variant of Dijkstra's algorithm (introduced by Nicholson [9]) in 1966 usually outperforms Dijkstra's algorithm. So the bidirectional heuristic version of A* (introduced by Pohl [11] in 1971) was supposed to be better than the unidirectional variant as well. Dijkstra's bidirectional variant can be stopped as soon as the forward and backward search meet somewhere in the middle (and a shortest path can then be determined with linear additional time), but this does not apply to the classical bidirectional heuristic search, and some more complicated termination criteria are necessary. Neither the classical version nor improvements (made e.g. by Kwa [6] in 1989) ever performed well in practice. Ikeda et. al. [4] introduced a more efficient variant that uses Dijkstra's bidirectional search and takes into account the estimates using potential functions in 1994, but it was never well understood why the classical version was less efficient in practice. It is known ([6], [5]) that big parts of the graph are often considered twice (once during the forward and once during the backward search), but even the improvement by Kwa [6] that prevents these double calculations did not perform well.

In this paper, we show the reason why: Either the forward or the backward heuristic search will consider less nodes than the bidirectional version (assuming that we are choosing nodes in the direction of the target node first if more than one node can be chosen next – this can affect running times only very near to the end of the search – we will describe this after giving the proof of our main theorem). Usually, the time needed to compute the forward and backward search do not differ significantly. Hence, using heuristic search a random decision for either forward or backward search will most often perform better than the bidirectional search – in correspondence with the practical observations.

This paper is organized as follows: Section 2 introduces the basic definitions and a brief review of the referenced algorithms. Section 3 presents some lemmas and our main theorem including the proofs.

2 Definitions, notation, and a brief review of the algorithms

Let $\mathbb{R}^+$ be the set of non-negative real numbers. A directed weighted graph $G = (V, E, \gamma)$ consists of the set of nodes V, the set of edges $E \subseteq V \times V$, the weight function $\gamma : E \to \mathbb{R}^+$. Undirected graphs are included because they can be regarded as directed graphs with $(u, v) \in E \Leftrightarrow (v, u) \in E$. The shortest distance from source node s to any node v is denoted by $\mathrm{d}_s(v)$ and upper bounds (used in the algorithms) by $\mathrm{D}_s(v)$. The estimates of the distances from node u to target node t used in the A* algorithm are denoted by $\widehat{\mathrm{ed}}_t(u)$. When calculating shortest paths to a target node t we denote the shortest distance and the upper bounds with $\mathrm{d}_t(u)$ and $\mathrm{D}_t(u)$ (so here $\mathrm{d}_s(t)$ will be identical to $\mathrm{d}_t(s)$).

In the following, we will describe the algorithms without prooving their correctness. Proofs can be found in the original papers.

2.1 Dijkstra's algorithm

This algorithm partitions the nodes in tree nodes T_s ($v \in T_s \Rightarrow \mathrm{D}_s(v) = \mathrm{d}_s(v)$), their neighbours N_s (the nodes to be considered in the next iteration of the algorithm) and unvisited nodes. We initialize $T_s := \{s\}$, $N_s := \{u \mid (s, u) \in E\}$, and accordingly $\mathrm{D}_s(s) := 0$, $\mathrm{D}_s(u) := \gamma(s, u)$, $(s, u) \in E$, and $\mathrm{D}_s(u) := \infty$, $(s, u) \notin E$. In every iteration take one node $u \in N_s$ with minimal value $\mathrm{D}_s(u)$ (for this node Dijkstra proofed $\mathrm{D}_s(u) = \mathrm{d}_s(u)$), move u to T_s, and update N_s (for every edge $(u, v) \in E$ if

$v \in N_s$ we set $\mathrm{D}_s(v) := \min\{\mathrm{D}_s(v), \mathrm{D}_s(u) + \gamma(u,v)\}$ else if $\mathrm{D}_s(v) = \infty$ we move v to N_s and set $\mathrm{D}_s(v) := \mathrm{D}_s(u) + \gamma(u,v)$). We call one iteration a dijkstra_forward_iteration(s). For solving the single target shortest path problem to target node t, we execute the corresponding steps in inverse direction of the edges, and we call such an iteration a dijkstra_backward_iteration(t).

2.2 The A* algorithm

The A* algorithm is usually used to solve the single pair shortest path problem using estimates of the distances to the target node t, but it also can be used to calculate the distances to all other nodes. We consider only monotone estimates $\widehat{\mathrm{ed}}_t : V \to \mathbb{R}^+$, i.e. $\widehat{\mathrm{ed}}_t(t) = 0$ and to all edges $(u,v) \in E$ applies $\widehat{\mathrm{ed}}_t(u) \geq \gamma(u,v) + \widehat{\mathrm{ed}}_t(v)$ [10] – otherwise nodes may have to be moved back from T_s to N_s and running times may become exponential [8]. The A* algorithm differs from Dijkstra's only in choosing a node $u \in N_s$ with minimal value $\mathrm{D}_s(u) + \widehat{\mathrm{ed}}_t(u)$ (for this also holds $\mathrm{D}_s(u) = \mathrm{d}_s(u)$). All other steps are identical and we denote one iteration by astar_forward_iteration(s) and astar_backward_iteration(t).

2.3 Bidirectional search with Dijkstra's algorithm

If we want to solve a single pair shortest path problem with source s and target t, we can use a bidirectional search. This performs independent forward and backward searches from the source node and to the target node, respectively. In each iteration, we choose to do either a dijkstra_forward_iteration(s) or a dijkstra_backward_iteration(t). As soon as we move a node to T_s or T_t such that $T_s \cap T_t \neq \emptyset$, we can stop and find the distance from s to t with the help of the following lemma.

Lemma 1 *Dijkstra's bidirectional search satisfies*

$$T_s \cap T_t \neq \emptyset \;\Rightarrow\; \mathrm{d}_s(t) = \min_{u \in T_s \cap (T_t \cup N_t)} \{\mathrm{D}_s(u) + \mathrm{D}_t(u)\}$$

and

$$T_s \cap T_t \neq \emptyset \;\Rightarrow\; \mathrm{d}_s(t) = \min_{u \in (T_s \cup N_s) \cap T_t} \{\mathrm{D}_s(u) + \mathrm{D}_t(u)\}$$

The calculation can be stopped as soon as $T_s \cap T_t \neq \emptyset$, and the correct distance can be found with linear additional time.

2.4 The classical bidirectional heuristic search

Using heuristics (i.e. using A* and two monotone estimates $\widehat{\mathrm{ed}}_t$ and $\widehat{\mathrm{ed}}_s$), we also choose in each iteration to perform either an astar_forward_iteration(s) or an astar_backward_iteration(t).

Unfortunately, lemma 1 does not hold when using heuristics[1]. When a new node in $T_s \cap T_t$ is found, we only update a value $\mathrm{D}_{\min}$ (giving the minimum length of the paths found so far, i.e. $\mathrm{D}_{\min} = \min\{\mathrm{D}_s(u) + \mathrm{D}_t(u) \mid u \in T_s \cap T_t\}$). We have to continue calculations until we can be sure that either the forward search or the backward search cannot find a path with length less than $\mathrm{D}_{\min}$ (this is sufficient as $\mathrm{d}_s(t) = \mathrm{d}_t(s)$). This leads to the following lemma first stated by Pohl ([11]):

Lemma 2 *The classical bidirectional heuristic search satisfies*

$$\max\{\min_{v \in V \setminus T_s}\{\mathrm{D}_s(v) + \widehat{\mathrm{ed}}_t(v)\}, \min_{v \in V \setminus T_t}\{\mathrm{D}_t(v) + \widehat{\mathrm{ed}}_s(v)\}\} \geq \mathrm{D}_{\min}$$
$$\Rightarrow \mathrm{D}_{\min} = \mathrm{d}_s(t)$$

The bidirectional search using Dijkstra's algorithm most often outperforms the standard Dijkstra algorithm, and also the unidirectional heuristic search almost always outperforms standard Dijkstra. Hence, normally we would expect that the bidirectional heuristic search also outperforms the bidirectional search using Dijkstra's algorithm. Surprisingly, this is not the case.

3 Main theorem

In this section, we proof that for every bidirectional heuristic search the number of nodes moved to T_s and T_t is not less than the number of nodes considered in the better case of either the unidirectional heuristic search from s to t or the search from t to s. In order to show this, we need two lemmas about the order of the nodes moved to T_x during the unidirectional heuristic search, and one lemma about the order of choosing the A* iteration in forward or backward direction in the bidirectional heuristic search.

Lemma 3 *The A* algorithm with monotone estimates* $\widehat{\mathrm{ed}}_t(\cdot)$ *moves the nodes to* T_s *in increasing order of the values* $\mathrm{d}_s(\cdot) + \widehat{\mathrm{ed}}_t(\cdot)$.

Proof: For the chosen node u (which has minimum value $\mathrm{D}_s(u) + \widehat{\mathrm{ed}}_t(u)$ of nodes in N_s) we know $\mathrm{D}_s(u) = \mathrm{d}_s(u)$. So in the update process for all other nodes in N_s either $\mathrm{D}_s(v)$ is unchanged and still satisfies $\mathrm{D}_s(v) + \widehat{\mathrm{ed}}_t(v) \geq \mathrm{D}_s(u) + \widehat{\mathrm{ed}}_t(u)$. Otherwise $\mathrm{D}_s(v)$ is changed to $\mathrm{D}_s(v) := \mathrm{D}_s(u) + \gamma(u, v)$, and therefore

$$\mathrm{D}_s(v) + \widehat{\mathrm{ed}}_t(v) = \mathrm{D}_s(u) + \gamma(u, v) + \widehat{\mathrm{ed}}_t(v) \geq \mathrm{D}_s(u) + \widehat{\mathrm{ed}}z(u)$$

[1]If the estimates meet the condition $\widehat{\mathrm{ed}}_{s/t}(\cdot) \leq \mathrm{d}_{s/t}(\cdot) \leq (1 + \varepsilon) \cdot \widehat{\mathrm{ed}}_{s/t}(\cdot)$ we can use lemma 1 and guarantee a distance $\leq (1 + \varepsilon/(2 + \varepsilon)) \cdot \mathrm{d}_s(t)$ as shown in [7]. Worst case examples exist even for graphs with edge lengths and estimates all being Euclidean. In such Euclidean graphs another approximation given by [7] may give better results: if $\mathrm{d}_s(t) = (1 + \varepsilon') \cdot \widehat{\mathrm{ed}}_s(t)$, the distance is guaranteed to be $\leq (1 + \varepsilon'/(1 + \varepsilon')) \cdot \mathrm{d}_s(t)$.

because we have monotone estimates $\widehat{\mathrm{ed}}_t(\cdot)$. So the next u' to be chosen will have a value $\geq \mathrm{D}_s(u) + \widehat{\mathrm{ed}}_t(u)$. $\square$

Lemma 4 *Let $w = v_0 v_1 \cdots v_k$ be a shortest path from $s = v_0$ to $t = v_k$, then the function $f : \{0, \ldots, k\} \to \mathbb{R}^+$ with $f(i) := \mathrm{d}_s(v_i) + \widehat{\mathrm{ed}}_t(v_i)$ is monotonic increasing.*

Proof: With the monotone property of the estimates $\widehat{\mathrm{ed}}_t(\cdot)$ we have $\mathrm{d}_s(v_i) + \widehat{\mathrm{ed}}_t(v_i) \leq \mathrm{d}_s(v_i) + \gamma(v_i, v_{i+1}) + \widehat{\mathrm{ed}}_t(v_{i+1}) = \mathrm{d}_s(v_{i+1}) + \widehat{\mathrm{ed}}_t(v_{i+1})$. $\square$

Lemma 5 *Let $a = a_1, a_2, \ldots, a_k \in \{f, b\}^k$ give the sequence of choosing a forward or backward iteration in bidirectional heuristic search. Let $\hat{a} = \hat{a}_1, \hat{a}_2, \ldots, \hat{a}_k \in \{f, b\}^k$ be another sequence with the same number of forward iterations and the same number of backward iterations. Then the sets T_s and T_t and the value $\mathrm{D}_{\min}$ will be identical at the end of both runs of the bidirectional heuristic search.*

Proof: The forward search has no influence on any steps made during the backward search and vice versa. Hence, we will move the same nodes to T_s and T_t, respectively, in both runs of the bidirectional heuristic search, because the number of forward iterations and backward iterations is identical. $\mathrm{D}_{\min}$ is set to $\min\{\mathrm{D}_s(u) + \mathrm{D}_t(u) \mid u \in T_s \cap T_t\}$, so we will end up with the same value $\mathrm{D}_{\min}$ – note that $\mathrm{D}_x(u) = \mathrm{d}_x(u)$ for $u \in T_x$. $\square$

Corollary 6 *In the classical bidirectional heuristic search we can execute all forward iterations first, then all backward iterations, and will end up with the same result as with any other sequence that has the same number of forward and backward iterations.*

We are now ready to proof our main theorem.

Theorem 7 *For every run of a classical bidirectional heuristic search with monotone estimates $\widehat{\mathrm{ed}}_t(\cdot)$ and $\widehat{\mathrm{ed}}_s(\cdot)$ solving the single pair shortest path problem between nodes s and t moving a total number of $r := |T_s| + |T_t|$ nodes to the tree sets, either a unidirectional search from s to t will have $|T'_s| \leq r$ nodes moved to T'_s or a unidirectional backward search from t to s will have $|T'_t| \leq r$ nodes moved to T'_t – both using the same monotone estimates $\widehat{\mathrm{ed}}_t(\cdot)$ or $\widehat{\mathrm{ed}}_s(\cdot)$, respectively.*

Proof: In order to satisfy the termination criterion of the classical bidirectional heuristic search (lemma 2) we must set $\mathrm{D}_{\min}$ to $\mathrm{d}_s(t)$ (this is done as soon as one node of a shortest path from s to t is in $T_s \cap T_t$ – remember that $\mathrm{d}_s(v) + \mathrm{d}_t(v) = \mathrm{d}_s(t)$ for any v on a shortest path from s to t, this node v may be s or t), and we must satisfy $\min_{v \in V \setminus T_s}\{\mathrm{D}_s(v) + \widehat{\mathrm{ed}}_t(v)\} \geq \mathrm{d}_s(t)$ (or the corresponding condition in the backward search – we will focus on the forward search here and assume $|T'_s| \leq |T'_t|$). The sketch of the proof is that we either will terminate the algorithm by moving t to T_s (case 1 below) or by trying to move as few nodes as possible to T_s and T_t after having moved as many nodes as necessary to T_s during the forward search (case 2 below). In both cases we implicitely use corollary 6 and execute all forward iterations first.

Case 1: If the bidirectional heuristic search moves the node t to T_s the criterion of lemma 2 applies: $\mathrm{D}_{\min} = \mathrm{D}_s(t) + \mathrm{D}_t(t) = \mathrm{d}_s(t) + 0 = \mathrm{d}_s(t)$ (note, we just moved t to T_s and $t \in T_t$ right from the start) and from lemma 3 we get $\min_{v \in V \setminus T_s}\{\mathrm{D}_s(v) + \widehat{\mathrm{ed}}_t(v)\} \geq \mathrm{D}_{\min} = \mathrm{d}_s(t)$. So all steps made in the calls of astar_backward_iteration(t) were done in vain. Analogously, if s is moved to T_t in the backward search, all calls of astar_forward_iteration(s) were unnecessary.

Case 2: There must be at least one node of a shortest path from s to t that is moved to both T_s and T_t in order to set $\mathrm{D}_{\min} := \mathrm{d}_s(t)$. We focus on the forward search here (otherwise we have the same argument for the backward search). Due to lemma 3 all nodes v with $\mathrm{d}_s(v) + \widehat{\mathrm{ed}}_t(v) < \mathrm{d}_s(t)$ will have to be moved to T_s first because these are candidates for shorter paths from s to t.

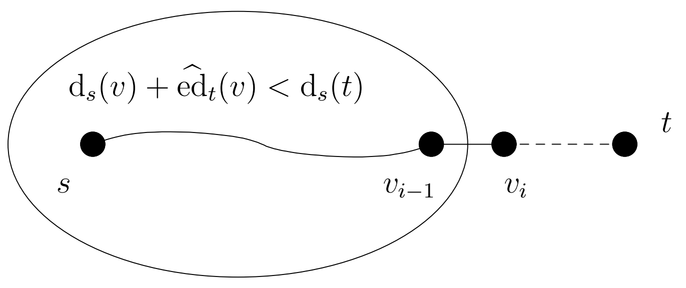
Let $w = v_0 v_1 \cdots v_k$ be a shortest path from $s = v_0$ to $t = v_k$ with the minimum number of nodes v_j having the property $\mathrm{d}_s(v_j) + \widehat{\mathrm{ed}}_t(v_j) = \mathrm{d}_s(t)$ – the reason for choosing a shortest path with this additional property is given at the end of this proof. Let v_i be the first node on path w with $\mathrm{d}_s(v_i) + \widehat{\mathrm{ed}}_t(v_i) = \mathrm{d}_s(t)$. From lemma 4 we get $\mathrm{d}_s(v_j) + \widehat{\mathrm{ed}}_t(v_j) = \mathrm{d}_s(t)$ for all $v_j \in \{v_i, \ldots, v_k\}$.

We observe that when v_{i-1} was moved to T_s we set $\mathrm{D}_s(v_i) := \mathrm{d}_s(v_{i-1}) + \gamma(v_{i-1}, v_i) = \mathrm{d}_s(v_i)$. Thus $\mathrm{D}_s(v_i) + \widehat{\mathrm{ed}}_t(v_i) = \min_{v \in V \setminus T_s}\{\mathrm{D}_s(v) + \widehat{\mathrm{ed}}_t(v)\} = \mathrm{d}_s(t)$, and we can choose v_i to be moved to T_s next – and then $v_{i+1}, \ldots, v_k$ afterwards. Hence, we would terminate the algorithm again by moving $v_k = t$ to T_s as in case 1. If $v_i = s$ we can move $v_{i+1}, \ldots, v_k$ to T_s in this order by the same argument.

Otherwise, for a suitable j ($i - 1 \leq j \leq k$), we must at least move $v_i, \ldots, v_j$ to T_s and $v_j, \ldots, v_k$ (in reverse order) to T_t to be able to terminate the algorithm – observe that the termination condition is then satisfied: $\mathrm{D}_{\min}$ is set to $\mathrm{d}_s(t)$ and $\min_{v \in V \setminus T_s}\{\mathrm{D}_s(v) + \widehat{\mathrm{ed}}_t(v)\} \geq \mathrm{D}_{\min}$. This way the number of nodes in $|T_s| + |T_t|$ is $|\{v \mid \mathrm{d}_s(v) + \widehat{\mathrm{ed}}_t(v) < \mathrm{d}_s(t)\}| + k - i + 2$, but the unidirectional heuristic search from s to t would only move $|\{v \mid \mathrm{d}_s(v) + \widehat{\mathrm{ed}}_t(v) < \mathrm{d}_s(t)\}| + k - i + 1$ nodes to T'_s. Furthermore, the nodes u (not on the path w) with $\mathrm{d}_t(u) + \widehat{\mathrm{ed}}_s(u) < \mathrm{d}_t(v_j) + \widehat{\mathrm{ed}}_s(v_j)$ would have to be moved to T_t before v_j. Hence, the unidirectional search is preferable.

The reason for choosing the path w with the minimum number of nodes v_j having the property $\mathrm{d}_s(v_j) + \widehat{\mathrm{ed}}_t(v_j) = \mathrm{d}_s(t)$ is only necessary to make sure that we move as few nodes as possible to T'_s in the unidirectional search.

$\square$



$\mathrm{d}_s(v) + \widehat{\mathrm{ed}}_t(v) < \mathrm{d}_s(t)$
t
s
v_{i-1}
v_i

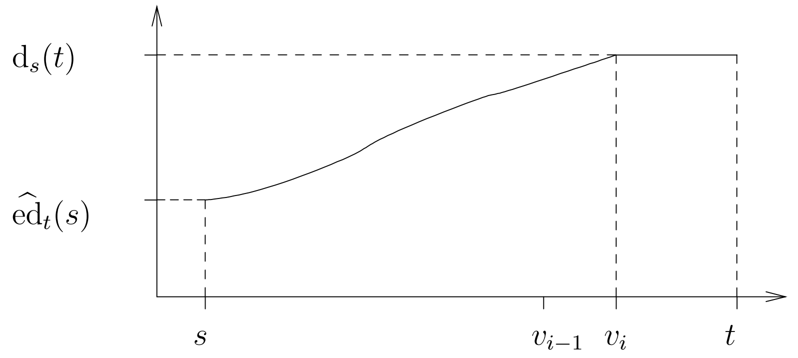


Figure 1: Illustration of lemma 4 and case 2 in the proof of theorem 7

4 Conclusion

We proofed that either the heuristic forward or the heuristic backward search will be at least as fast as the classical bidirectional heuristic search. It will not use more iterations and needs less additional time due to a much simpler termination criterion.

In practice, we cannot choose specific nodes from the set of nodes $\{v \mid \mathrm{d}_s(v) + \widehat{\mathrm{ed}}_t(v) = \mathrm{d}_s(t)\}$. Therefore, the bidirectional variant may be faster if there are many such nodes and moving nodes to T_s or T_t is decided in favour of the bidirectional search. When solving the single pair shortest path problem in real transportation networks, Euclidean distances are often used as estimates $\widehat{\mathrm{ed}}_{s/t}$. Likely, only a few nodes v will meet $\mathrm{d}_s(v) + \widehat{\mathrm{ed}}_t(v) = \mathrm{d}_s(t)$, and hence, bidirectional heuristic search will not speed up calculations. In case the costs for the calculation of the shortest path from s to t and backward from t to s differ significantly, it is still preferable to calculate those two searches independently and in parallel.

The main problem of the classical bidirectional heuristic search is that most parts of the graph are considered twice. Kwa [6] gave an improved algorithm called BS* by ignoring neighbours in the update of a forward iteration that are already in T_t (and analogously the nodes in T_s in a backward iteration). This keeps $T_s \cap T_t$ small and prevents unnecessary calculations. Though examples exist with lower running times than both unidirectional heuristic searches, these are not very easy to find, and on an average [6] observed slightly longer running times than with the unidirectional heuristic searches. When we compare BS* with the unidirectional A* on a grid (e.g. with Euclidean distances as estimates $\widehat{\mathrm{ed}}_{s/t}$), we can use a similar proof as in our main theorem to show that BS* cannot be faster than the unidirectional A* algorithm – the BS* will prevent double calculations, but any node not moved to T_s will have to be moved to T_t and vice versa. Hence, the overall performance of the BS* algorithm on Euclidean grids will always be worse than by using the simple A* algorithm.

References

[1] E.W. Dijkstra, *A Note on Two Problems in Connexion with Graphs*, Numerische Mathematik 1 (4) (1959) 269–271.

[2] D. Gelperin: *On the Optimality of A**, Artificial Intelligence 8 (1) (1977) 69–76.

[3] P. Hart, N. Nilsson and B. Raphael, *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*, IEEE Transactions on Systems Science and Cybernetics SSC-4 (2) (1968) 100–107.

[4] T. Ikeda, M.-Y. Hsu and H. Imai, *A Fast Algorithm for Finding Better Routes by AI Search Techniques*, in: *1994 Vehicle Navigation & Information Systems Conference Proceedings*, 1994, pp. 291–296.

[5] H. Kaindl and G. Kainz, *Bidirectional Heuristic Search Reconsidered*, Journal of Artificial Intelligence Research 7 (1997) 283–317.

[6] J.B.H. Kwa, *BS*: An Admissable Bidirectional Staged Heuristic Search Algorithm*, Artificial Intelligence 38 (1) (1989) 95–109.

[7] S. Lewandowski, *Vereinheitlichte Darstellung von Techniken zur effizienten Kürzeste-Wege-Suche*, Dissertation, Universität Stuttgart, Der Andere Verlag, Tönning, Lübeck, Marburg, 2005, ISBN 3-89959-339-1.

[8] A. Martelli, *On the Complexity of Admissible Search Algorithms*, Artificial Intelligence 8 (1) (1977) 1–13.

[9] T.A.J. Nicholson, *Finding the shortest route between two points in a network*, Computer Journal 9 (3) (1966) 275–280.

[10] J. Pearl, *Heuristics : intelligent search strategies for computer problem solving*, The Addison-Wesley series in artificial intelligence, Addison-Wesley, Reading, Mass., 1984.

[11] I. Pohl, *Bi-Directional Search*, in: B. Meltzer (ed.), *Machine intelligence 6*, University Press, Edinburgh, 1971, pp. 127–140.