Bidirectional Best-First Search with Bounded Error: Summary of Results

Andreas L. Koll
Kaisers 13
A-6655 Steeg
Austria, Europe

Hermann Kaindl
Siemens AG Osterreich
Geusaugasse 17
A-1030 Wien
Austria, Europe

Abstract

For more than two decades there has been consensus that bidirectional heuristic search is afflicted by the problem of search wavefronts missing each other. However, our results indicate that a different problem appears to be of primary importance. The fronts typically meet rather early even without using wave-shaping techniques. Especially when aiming for optimal solutions, however, much effort has to be spent for subsequently improving the solution quality, and finally for proving that there is indeed no better solution possible. Therefore, only slightly relaxing the requirements on the solution quality already leads to strong improvements in efficiency.

We describe several new *e-admissible* bidirectional search algorithms which do *not* use wave-shaping techniques. The most efficient of these use a novel termination criterion designed to address the suspected primary problem of bidirectional heuristic search. We prove £-admissibility and a *dominance* result based on this termination criterion. In summary, we show that and how bidirectional best-first search can be more efficient than the corresponding unidirectional counterpart without using computationally very demanding wave-shaping techniques.

Notation

s, t	Start node and goal node, respectively.
$\Gamma_1(n)$	Successors of node n in the problem graph.
$\Gamma_2(n)$	Parents of node n in the problem graph.
d	Current search direction index; when search is in the forward direction $d = 1$, and when in the backward direction $d = 2$.
d'	$3 - d$; it is the index of the direction opposite to the current search direction.
$c_i(m, n)$	Cost of the direct arc from m to n if $i = 1$, or from n to m if $i = 2$.
$k_i(m, n)$	Cost of an optimal path from m to n if $i = 1$, or from n to m if $i = 2$.
$g_i^*(n)$	Cost of an optimal path from s to n if $i = 1$, or from t to n if $i = 2$.
$h_i^*(n)$	Cost of an optimal path from n to t if $i = 1$, or from n to s if $i = 2$.
$g_i(n), h_i(n)$	Estimates of $g_i^*(n)$ and $h_i^*(n)$, respectively.
h_{F_i}	FOCAL-heuristic.
$f_i(n)$	Static evaluation function.
C^*	Cost of an optimal path from s to t.
$L_{\min}$	Cost of the best (least costly) complete path found so far from s to t.
TREE₁	The forward search tree.
TREE₂	The backward search tree.
OPEN$_i$	The set of open nodes in TREE$_i$.
FOCAL$_i$	$\{m \mid (m \in \text{OPEN}_i) \wedge (g_i(m) + h_i(m)) \leq (1+\delta)(\min_{n \in \text{OPEN}_i} (g_i(n) + h_i(n)))\}$.
$\|$OPEN$_i\|$	Number of nodes in OPEN$_i$.
CLOSED$_i$	The set of closed nodes in TREE$_i$.
$p_i(n)$	Parent of node n in TREE$_i$.
$p_d^i(n)$	$\underbrace{p_d(p_d \ldots (p_d(n)) \ldots)}_{i-times}$.
Ω_m	The set of nodes in OPEN$_{d'}$ which are descendants of m in TREE$_{d'}$.
MeetingNode	Node where TREE₁ met TREE₂ and yielded the best complete path found so far.

* Andreas Koll was with Siemens AG Osterreich for a summer internship.

1 Introduction and Background

Much research in heuristic search has been devoted to algorithms finding *optimal* solutions. In the unidirectional case *A** [4] and its derivatives received much attention. Especially related to NP-complete problems, an important approach in relaxing the search for optimality is to provide a bound for the worst case quality of a solution compared to an optimal one [5], A search algorithm is said to be *e-admissible* if it guarantees that the costs of its solutions are bounded by $(1+\epsilon)C^*$ [14, 13]. An early example is *dynamic weighting* in the form used in [18]. It uses an upper bound N on the depth of the deepest node to be evaluated, which has to be given as a parameter.[1] With $N \to \infty$, asymptotically the heuristic evaluation function with constant weighting of [15] arises $(f = g + wh)$.[2] Such an evaluation function is used by *EPA* [16].[3] A more recent approach to dynamic weight-

[1] In fact, besides using an admissible heuristic component h the algorithm must have the possibility to reopen nodes, i.e., to move nodes back from CLOSED to OPEN whenever a new better path is found to them.

[2] For constant w, this is equivalent to $f = (1 - w')g + w'h$ with $w' = \frac{w}{1+w}$.

[3] For our comparisons, we use a slight variant of *HPA*, which includes the option to move nodes back from CLOSED

ing [7] compensates for the error made by the heuristic component using learnt weights, and it is ^-admissible without the need for a given upper bound like *N*. The e-admissible algorithm *A** [14] is designed to allow for incorporating an additional heuristic to estimate the effort for finishing a search, which is used as FOCAL-heuristic. All these algorithms perform unidirectional search, and for $\varepsilon > 0$ they are typically more efficient than *A**.

BHPA[4] [15, 17] was the first bidirectional algorithm using a heuristic evaluation function for finding optimal solutions. While *BS** [10] improved *BHPA* technically, its performance was only nearly as good as the unidirectional *A**. There was consensus that the main reason for the superiority of unidirectional search is that the bidirectional heuristic search is afflicted by the problem of search wavefronts missing each other (cf. the *missile metaphor* in [15]).

Wave-shaping techniques like those used in *BHFFA* [3], *BHFFA2* [2] and d-node retargeting [19] showed that bidirectional heuristic search can be rather efficient, especially in terms of the number of expanded nodes. Davis [1] presented a generalized algorithm (containing *BHPA* and *BHFFA2)* and analyzed it theoretically. However, these algorithms are either excessively computationally demanding, or they have no restriction on the solution quality.

Considering the missile metaphor of Pohl, the use of ε-admissible approaches in bidirectional search should be dangerous without wave-shaping, since the chance of search wavefronts missing each other could be even increased. However, we found that in bidirectional searches the first meeting of the fronts typically occurred early, compared to the effort for finally satisfying the termination condition.

Fig. 1 illustrates this phenomenon.[5] These data are normalized in the sense that every data point represents the ratio of the number of nodes expanded by the corresponding algorithm to that expanded by *A**. In the average, the fronts meet rather early even without using wave-shaping techniques, requiring only slightly more than half the effort of complete *A** on the 8-Puzzle for $\varepsilon = 0$. On the much more difficult 15-Puzzle, the fronts meet already when about 3 orders of magnitude fewer nodes are generated than by complete *IDA** [9] searches (data for *A** are unavailable here).

Especially when aiming for optimal solutions, however, much effort has to be spent for subsequently improving the solution quality. (In the average, this is achieved with less effort than that of a complete *A** search on the 8-Puzzle.) Finally, for proving that there is indeed no better solution possible, many more nodes have to be expanded. Therefore, we were interested in investigating e-admissible bidirectional search and in comparing its efficiency to the corresponding unidirectional case.

to OPEN, if a new better g-value is found. Moreover, we include a check whether OPEN has become empty. Since this is like *A** [4], we call this variant *HPA**.

[4]The earlier version named *VGHA* in [15] did not move nodes back from CLOSED to OPEN. Hence, it needed a *consistent* heuristic function for being *admissible.*

[5]For $\varepsilon = 0$, IBS^*_ε — one of our new algorithms as introduced below — is identical to *BS**.

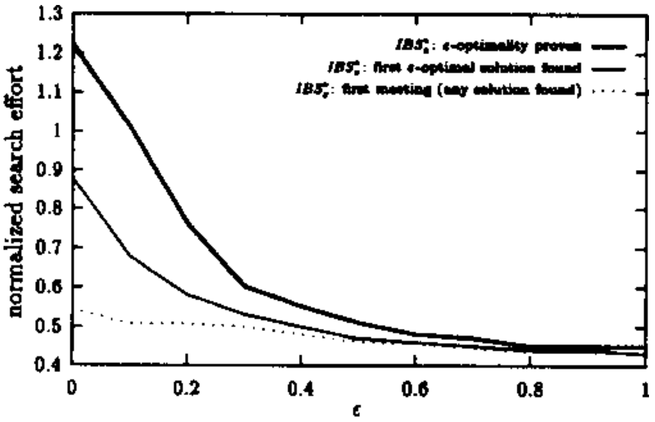

Figure 1: Phases of IBS^*_ε (on the 8-Puzzle).

2 Improved Bidirectional Search

In the following, we describe several ^-admissible bidirectional search algorithms which do *not* use wave-shaping techniques. We relate them to the earlier known algorithms in pointing out the differences. The origin is clearly *BHPA* [15, 17], which can be intuitively viewed to consist of two *HPA* searches in opposing directions. The termination condition compares the length of the best path found so far to an optimistic estimate computed from the nodes in both search frontiers (see also below). While *BHPA* only considered meeting points between CLOSED nodes from both sides, Kwa [10] showed that meeting of a newly generated node with any node in the opposing front is sufficient. All the algorithms described below use this improvement. This technical change together with the incorporation of checks for empty OPEN lists results in a version we call *BHPA'*.

2.1 Constant weighting ($IBHPA_\varepsilon$ and IBS^*_ε)

There are two ways of achieving an ^-admissible variant of *BHPA'*:

- $BHPA_w$ using an evaluation function of the form $f = g + wh$, where f can be rewritten as $f = g + (1 + \varepsilon)h$. The original termination condition of *BHPA*

  $$L_{min} \le max[min_{x\in \mathrm{OPEN}_1} f_1(x), min_{x\in \mathrm{OPEN}_2} f_2(x)] \quad (1)$$

  turns out to include $(1+\varepsilon)h$ implicitly in $BHPA_w$:

  $$L_{min} \le max[min_{x\in \mathrm{OPEN}_1}(g_1(x) + (1+\varepsilon)h_1(x)), min_{x\in \mathrm{OPEN}_2}(g_2(x) + (1+\varepsilon)h_2(x))]. \quad (2)$$

  In fact, Pohl [17] proposed such a change in the evaluation function, when not concerned with finding a shortest path. For this reason, he suggested to simplify the termination condition to just meeting. In contrast, $BHPA_w$ using (1) or equivalently (2) as termination condition is ε-admissible when *h* is admissible.

- $BHPA_\varepsilon$ using $f = g + h$ (like *BHPA* or a bidirectional admissible version of A^*), but using the termination criterion

  $$L_{min} \le (1+\varepsilon)max[min_{x\in \mathrm{OPEN}_1}(g_1(x) + h_1(x)), min_{x\in \mathrm{OPEN}_2}(g_2(x) + h_2(x))]. \quad (3)$$

  $BHPA_\varepsilon$ is also ε-admissible when h is admissible.

These ideas can be combined in an improved version as follows: $IBHPA_\epsilon$ uses $f = g + (1+\epsilon)h$ with $\epsilon \geq 0$ and the termination criterion (3) still guaranteeing e-admissibility.

Another approach is to modify BS* [10] similarly, in order to utilize its technical improvements. When a node is selected for expansion which is already closed in the opposite search tree, it can just be closed without expansion (nipping). Moreover, it is possible in such a situation to remove the descendants of this node in the opposing open set (pruning). Likewise, it is possible to remove all those open nodes (in both directions) whose /-values are greater or equal than L_{min} (trimming). Of course, such newly generated nodes are not placed in the open sets at all (screening). The interested reader will find a pseudo-code formulation of these improvements in Appendix A.

Generally, the provision for reopening nodes must be incorporated, since otherwise e-admissibility cannot be guaranteed. In particular, without this provision there could be no guarantee that the shallowest open node n' on an optimal path always satisfies $g(n') = g^*(n')$. In addition, we have to assume a consistent h because of the BS* specific actions nipping and pruning.[6]

- BS^*_w using $f = g + wh$ with $w \geq 1$ like HPA (in step 14 of [10, p. 101]).[7]
- BS^*_ϵ using $f = g + h$ and the following termination criterion (in step 3 of [10, p. 100]): OPEN$_1$ or OPEN$_2$ is empty or (3).

Again, these modifications can be combined, resulting in an improved algorithm we call IBS*.

2.2 Derivatives of A^*_ϵ ($BA^*_{\delta,\epsilon}$ and $BSA^*_{\delta,\epsilon}$)

Especially, when there is another heuristic function available that estimates the effort for completing a search, it can serve as a FoCAL-heuristic in the unidirectional A_e* [14].[8] Using A^*_ϵ type of search in both directions and the termination condition (3), BA^*_ϵ arises, which is e-admissible.

In contrast to the bidirectional algorithms $IBHPA_\epsilon$ and IBSb* described above, BA^*_ϵ is normally less efficient than the corresponding unidirectional algorithm (A^*_ϵ). We found out that its searches are too greedy, leading to comparably bad solutions at the first meeting and consequently to much effort for subsequent satisfaction of the termination condition guaranteeing e-adrnissible solutions. Generally, we conjecture that bidirectional searches without wave-shaping tend to meet first with a solution quality which is worse than that achieved by the respective evaluation function in a corresponding unidirectional search.

[6] Despite consistency, reopening of nodes is necessary because of the $(1+\epsilon)$-factor in the evaluation function.

[7] This variant is only of interest for showing the genesis of the improved algorithm below, since by itself it is very inefficient due to lacking an appropriate termination condition.

[8] Pearl and Kim propose to use the same heuristic for both purposes if no estimator for the effort of completion is available. We wanted to compare all the algorithms with the same domain-specific knowledge, and consequently used the available admissible heuristic h also as a FoCAL-heuristic.

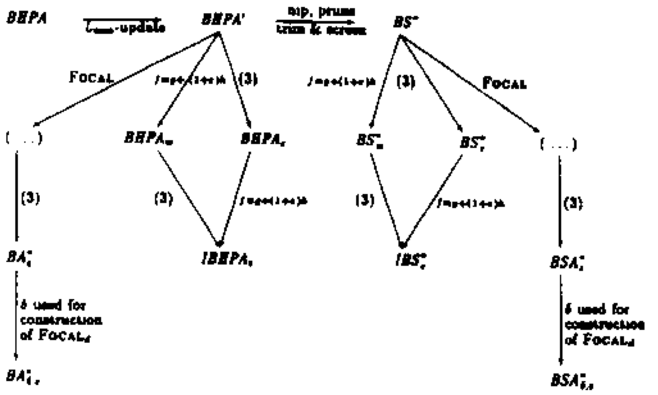

Figure 2: Overview of e-admissible bidirectional search algorithms.

Therefore, we found an improvement by adding another parameter δ. $BA^*_{\delta,\epsilon}$ uses 6 for creating the FOCAL-lists, and £ for termination. Much better results were achieved, for instance, with $\delta = \frac{\epsilon}{2}$ and $\delta = \frac{\epsilon}{4}$.

Of course, it is also possible to incorporate the approach of A^*_ϵ into the framework of BS*. BSA* arises from BS^* by the following changes:

- reopening of closed nodes if a new better path to them has been found,
- the termination condition (in step 3 of [10, p. 101]) analogously to BS^*_ϵ (see above),
- the selection of a node m for expansion (in step 7 of [10, p. 100]) as usual for A^*_ϵ.

Moreover, another parameter can be used, $BSA^*_{\delta,\epsilon}$ arises from BSA^*_ϵ analogously to $BA^*_{\delta,\epsilon}$ from BA^*_ϵ. Since this algorithm is already somewhat different from BS*, we present a complete pseudo-code description in Appendix A.[9]

Fig. 2 provides an overview of all the presented algorithms and their genesis.

3 Theoretical Results

Due to lack of space we cannot present here all the proofs for the various algorithms introduced above (the interested reader is referred to [6]). Since the proofs for BSA^*_ϵ and $BSA^*_{\delta,\epsilon}$ are the ones most different from similar proofs for A^*_ϵ, BHPA and BS*, we selected them for presentation below. Moreover, we will show an interesting result about dominance between bidirectional algorithms which is due to the improved termination condition (3). We assume that the branching degrees of all nodes are finite, the arc costs $c_i(m,n) \geq e > 0$ for some c, and a consistent heuristic estimator is used.

Definition 3.1 A path P from s to t is ϵ-optimal, if $cost(P) < (1+\epsilon)C^*$.

Definition 3.2 A search algorithm is said to be ϵ-admissible, if it terminates with an ϵ-optimal solution whenever a solution exists [14,13].

Definition 3.3 An algorithm A\ is said to dominate A_2 if every node expanded by A\ is also expanded by A_2 [13].

[9] Due to the inclusion of reopening of nodes in our algorithms and a technical bug in step 22 of the BS* description in [10, p. 101], we restructured the code.

Lemma 3.1 *Before $BSA^*_{\delta,\epsilon}$ $(\delta \leq \varepsilon)$ terminates either an ε-optimal path P' has already been found or there exist n_i and n_j in* OPEN$_1$, OPEN$_2$ *respectively, such that n_i and n_j are shallowest open nodes along an optimal path $P = (s = n_1, \ldots, n_i, \ldots, n_j, \ldots, t = n_t)$ with $g_1(n_i) = g_1^*(n_i)$ and $g_2(n_j) = g_2^*(n_j)$.*

Proof: *Except for the BS^* specific actions such nodes must exist at any time before termination, since we account for reopening of nodes (cf. [13]), and before the first meeting no such actions can occur. Furthermore, $i < j$. Otherwise the optimal path would have been found. There are four cases where theses nodes could be removed from* OPEN$_d$ *without installing a successor substituting them along P. We consider the case of n_i to be removed from* OPEN$_1$ *(the case of n_j to be removed from* OPEN$_2$ *is analogous).*

1. *Nipping: Node n_i is* nipped *in* TREE$_1$, *i.e. n_i must be in* CLOSED$_2$. *n_k is a node in* OPEN$_2$ *having the smallest f_2 value. Since n_i is closed in* TREE$_2$, *n_i was selected for expansion before n_j (while searching in the backward direction) and we have*
$$\begin{aligned} f_2(n_i) &\leq (1+\delta) f_2(n_k) && n_i \text{ from } \textsc{Focal}_2 \\ &\leq (1+\epsilon) f_2(n_k) && \delta \leq \epsilon \\ &\leq (1+\epsilon) f_2(n_j) && f_2(n_k) \leq f_2(n_j) \qquad (4) \end{aligned}$$
We distinguish two cases
(1.1) $(1+\epsilon)[g_2^*(n_j)+k_2(n_i,n_j)]+\epsilon g_1^*(n_i) \geq g_2(n_i)$
$$\begin{aligned} cost(P') &= g_1^*(n_i) + g_2(n_i) \\ &\leq g_1^*(n_i) + \epsilon g_1^*(n_i) \\ &\quad +(1+\epsilon)[g_2^*(n_j) + k_2(n_i,n_j)] \\ &\leq (1+\epsilon)[g_1^*(n_i) + g_2^*(n_j) + k_2(n_i,n_j)] \\ &\leq (1+\epsilon)C^* \end{aligned}$$
This implies $L_{min} \leq (1+\epsilon)C^*$.
(1.2) $(1+\epsilon)[g_2^*(n_j)+k_2(n_i,n_j)]+\epsilon g_1^*(n_i) < g_2(n_i)$
Due to consistency $h_2(n_j) \leq k_2(n_i,n_j) + h_2(n_i)$.
$$\begin{aligned} g_2(n_j) + h_2(n_j) &\leq g_2(n_j) + k_2(n_i,n_j) + h_2(n_i) \\ (1+\epsilon)(g_2(n_j) + h_2(n_j)) &\leq (1+\epsilon)[g_2^*(n_j) + k_2(n_i,n_j)] \\ &\quad +(1+\epsilon)h_2(n_i) \\ &\quad +\epsilon g_1^*(n_i) - \epsilon g_1^*(n_i) \\ (1+\epsilon) f_2(n_j) &< g_2(n_i) + h_2(n_i) \\ &\quad -\epsilon\underbrace{[g_1^*(n_i) - h_2(n_i)]}_{\geq 0} \\ &< f_2(n_i) \text{ contradicting (4)} \end{aligned}$$

2. *Pruning: Node n_i has an ancestor node n_x in* TREE$_1$ *which was nipped in* TREE$_2$. *n_x must be on an optimal path (say P), otherwise n_i cannot have its parent pointer set to n_x. Therefore, $g_1(n_x) = g_1^*(n_x)$ and the proof is analogous to case 1, substituting n_x for n_i.*

3. *Trimming: If n_i would be trimmed from* OPEN$_1$, *then*
$$g_1(n_i) + h_1(n_i) \geq L_{min} \qquad (5)$$
Since n_i is on an optimal path and $g_1(n_i) = g_1^*(n_i)$, and since h_1 is consistent and consequently admissible, $h_1(n_i) \leq h_1^*(n_i)$, this implies $g_1(n_i) + h_1(n_i) \leq g_1^*(n_i) + h_1^*(n_i) = C^*$. If no path P' with $cost(P') \leq (1+\epsilon)C^*$ was found yet, $C^* \leq (1+\epsilon)C^* < L_{min}$, contradicting (5).

4. *Screening: If n_i is expanded and not nipped, there is one of its successors n_{i+1} substituting it as the shallowest open node on P. If n_{i+1} would be screened and $C^* < L_{min}$, this would lead to an analogous contradiction as in case 3.* □

Lemma 3.2 *If a path exists from s to t, $BSA^*_{\delta,\epsilon}$ will not terminate before finding an ε-optimal path.*

Proof: *The proof is analogous to the one in [17, p. 133], but using our termination condition (3).* □

Lemma 3.3 // a path exists from s to t, $BSA^*_{\delta,\epsilon}$ terminates.

Proof: Either the graph is finite and exhausted, or it is infinite and the costs become unbounded (analogous proofs can be found in [4, 12, IS, 10]). D

Theorem 3.1 If a path exists from s to t, $BSA^*_{6,c}$ terminates with a solution whose cost does not exceed the optimal cost by more than a factor $(1 + \varepsilon)$, *i.e.*, $BSA^*_{\delta,\epsilon}$ is ϵ-admissible.

Proof: This result follows from the lemmas above. □

Corollary 3.1 BSA^*_{ϵ} *is* ϵ-admissible.

Theorem 3.2 A bidirectional algorithm B\ using the termination condition (S) dominates another bidirectional algorithm B_2 differing from B\ only in using the condition (2) instead.

Proof: Since the right side of (S) is always greater than or equal to the right side of (2), algorithm B\ cannot terminate later than B_2. D

Corollary 3.2 $IBHPA_\epsilon$ dominates $BHPA_w$.

4 Experimental Results

Experiments were conducted to compare the performance of these bidirectional algorithms. The test domains used were sliding-tile puzzles and route planning.[10] These puzzles have been often used in the literature for explanations and comparisons of search algorithms in AI. Therefore, we also used them in order to make it easier to compare our results with published ones. Since finding optimal solutions is NP-complete, it is interesting to study finding approximate solutions, and the effects of scaling up.

However, while the puzzle is illustrative, it is just a game. In contrast, the route planning domain we selected has more real world flavor (though it is easier to scale up). We used randomly generated route planning problems in a map of the Viennese network of public transportation [6] rather than synthetically generated problems as used in [10].

4.1 Constant weighting

We were primarily interested in comparing the bidirectional best-first search algorithms with their corresponding unidirectional counterparts. Let us first summarize the 8-Puzzle results. The case of $\varepsilon = 0$ is identical to a comparison of HPA^* with BS^* and $BHPA'$. There, the unidirectional HPA* is clearly more efficient. However, already for $\epsilon \geq 0.2$ the bidirectional search algorithms are in terms of the number of generated nodes more efficient than the corresponding unidirectional algorithm. The linear-space algorithms WIDA* (an e-admissible extension of IDA*) and RBFS [8] showed relatively bad performance in terms of generated nodes. However, since node generation and evaluation is very efficient for the puzzle, they are still competitive in terms of running time, due to avoiding the overhead for managing the lists used for implementing classical best-first search.

These domains are not especially selected to fulfill the conditions under which classical best-first search is best. In particular, the puzzles have uniform cost and the usual evaluation functions there do not have many distinct values.

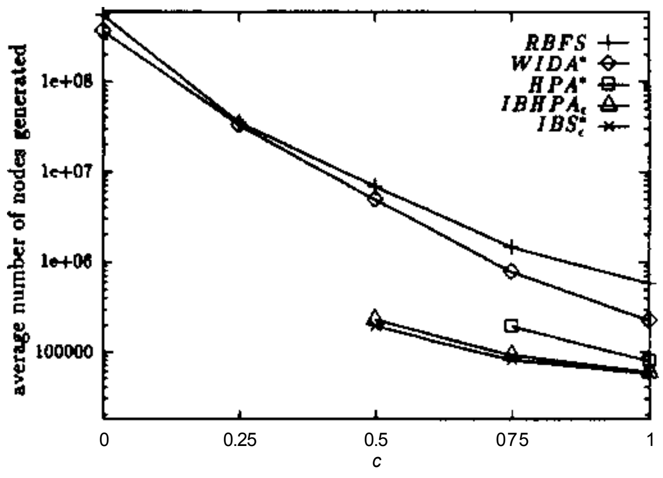

Figure 3: Comparison on the 15-Puzzle (100 instances).

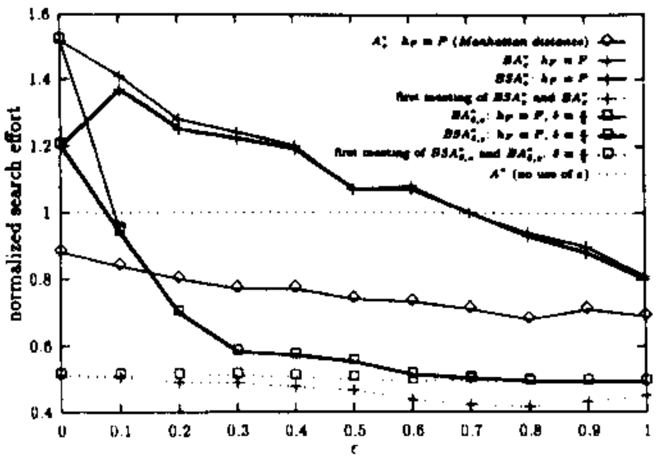

Figure 4: Comparison on the 8-Puzzle (500 instances).

An important question is how significant the results are. In addition to comparing the numbers of nodes searched, we look at the empirical data from a different viewpoint. Instead of summing up the numbers of generated nodes for each sample, we compare the performance on each instance. We count how often one or the other algorithm "wins" in the sense that it expands fewer nodes than its competitor when solving the same instance. The statistical details of using the *sign test* can be found in [6]. Although it is statistically significant according to this test that for $\epsilon \leq 0.1$ HPA^* expands more often fewer nodes than IBS^*_ϵ on the same instance, the inverse is true for $\epsilon \geq 0.3$.

Since the sliding-tile puzzle is NP-complete, it is interesting to investigate scaling up. In particular, it is interesting to see the effect of relaxing the solution quality [5]. Fig. 3 compares *1BHPA*$_\epsilon$ and IBS^*_ϵ with *HPA**, *WIDA** and *RBFS* on the 15-Puzzle (using the sample of 100 instances in [9]). Due to their memory problem, it is infeasible for $IBHPA_\epsilon$, IBS^*_ϵ and HPA^* to find or even to guarantee optimal solutions here.

For $\epsilon = 1$, HPA^* can solve all the 100 instances of [9], but it requires memory for nearly 1 million nodes in the maximum. In contrast, $IBHPA_\epsilon$ requires only a storage of slightly more than 300k nodes in maximum, and *IBS** slightly more than 250k nodes. Both are clearly more efficient than the others in terms of nodes generated. Interestingly, *HPA** "wins" against both $IBHPA_\epsilon$ and IBS^*_ϵ slightly more often, but the results arenot statistically significant according to the sign test. While the worst case bound $\epsilon = 1$ would allow solutions with twice the cost of optimal ones, the solutions actually found by these best-first algorithms are in the average just about 20 percent worse.

For $\epsilon = 0.75$, $IBHPA_\epsilon$ and IBS^*_ϵ require storage of slightly more than 600k nodes in maximum. In contrast, *HPA** requires to store nearly 2 million nodes for this task. The advantage of our bidirectional algorithms over *HPA** in terms of the total number of nodes searched is highly statistically significant (testing the means), but it is not according to the sign test. From this we can conclude that they are better especially on the difficult problems. Generally, the advantage of $IBHPA_\epsilon$ and IBS^*_ϵ is even clearer than for $\epsilon = 1$, generating an order of magnitude fewer nodes than $WIDA^*$.

For $\epsilon = 0.5$, $IBHPA_\epsilon$ and IBS^*_ϵ require less than 1.5 million nodes in maximum for the whole sample. While this worst case bound would allow solutions that are 50 percent worse than optimal ones, in the average these algorithms find solutions that are about 10 percent worse. HPA^* cannot solve 8 instances with this amount of storage. The number of nodes generated by $IBHPA_\epsilon$ (IBS^*_ϵ) is in the average smaller by a factor of 21 (25) than that generated by $WIDA^*$. Therefore, it appears that the performance advantage of our bidirectional algorithms steadily increases with decreasing values of ϵ (at least in the range where memory suffices). Moreover, their advantage is much more pronounced on the much more difficult 15-Puzzle than on the smaller 8-Puzzle.

4.2 Derivatives of A^*_ϵ

In contrast, the bidirectional algorithms BA^*_ϵ and BSA^*_ϵ are normally less efficient than the corresponding unidirectional A^*_ϵ (see Fig. 4 for the results on a random set of 500 8-Puzzles).[11] Therefore, we designed $BA^*_{\delta,\epsilon}$ and $BSA^*_{\delta,\epsilon}$, and Fig. 4 also shows a comparison vs. A^*_ϵ in terms of the total number of expanded nodes, using $\delta = \frac{\epsilon}{4}$. In total, $BA^*_{\delta,\epsilon}$ and $BSA^*_{\delta,\epsilon}$ expanded fewer nodes than A^*_ϵ for $\epsilon \geq 0.2$.

It is statistically significant that for $\epsilon \leq 0.2$, A^*_ϵ expands more often fewer nodes on the same instance than $BSA^*_{\delta,\epsilon}$ with $\delta = \frac{\epsilon}{4}$. However, using $\epsilon = 0.3$ the inverse is true. For $\epsilon \geq 0.5$ the bidirectional search algorithm did significantly better than the corresponding unidirectional one.

Moreover, it is interesting to have a look at the data of the first meeting in Fig.4. The worse bidirectional algorithms in the average met *earlier* than the better ones (but with worse solutions). After that, the latter could terminate rather quickly (or even immediately), while the former needed much time to satisfy the termination condition.

[11]The performance of A^*_ϵ with $\epsilon = 0$ is better than that of A^* since it uses its FOCAL-heuristic as a tie-breaking rule.

4.3 Cross-domain comparison

Since we conducted experiments in two different domains, the question arises whether there are major differences in the results. For $\varepsilon = 0$, IBS^*_ε (equivalent to BS^* there) did relatively much better in the route planning domain than for the 8-Puzzle. Most interestingly, BS^* was even 3 percent *more efficient* than A^* in terms of overall nodes expanded. However, with high statistical significance it was beaten by the latter in 63 percent of the cases which were not "draws". This observation reinforces the conjecture in [10] that BS^* is relatively more efficient than A^* on difficult problems.

Moreover, we found that BSA^*_ε was not that bad in comparison to A^*_ε in the route planning domain as it was for the puzzle. Generally, for both domains used for our experiments the better bidirectional search algorithms were more efficient than their corresponding unidirectional counterpart when the goal of finding optimal solutions is slightly relaxed.

4.4 Which is the best algorithm?

Since we investigated several bidirectional algorithms, the question arises which one is the best. Generally, the bidirectional algorithms using the improved termination condition (3) are more efficient than the others.

In particular, let us compare IBS^*_ε vs. $IBHPA_\varepsilon$. For $\varepsilon = 0$, IBS^*_ε was clearly superior to $IBHPA_\varepsilon$ on the 500 8-Puzzle instances. Although already for $\varepsilon \geq 0.1$ the total number of expanded nodes was nearly the same, the BS^* derivative was significantly better for $\varepsilon \leq 0.9$ in terms of "winning", but with increasing values of ε the vast majority of cases were "draws". On the 100 15-Puzzle instances, the advantage of IBS^*_ε over $IBHPA_\varepsilon$ was statistically significant over the whole range investigated, and according to both statistic tests. In the route planning domain, IBS^*_ε was superior to $IBHPA_\varepsilon$ for $\varepsilon \leq 0.3$ both regarding the total number of expanded nodes and in terms of "winning". However, for larger values of ε the majority of cases were "draws". In most of these cases both algorithms terminate immediately after first meeting, and up to this point they do the same. Generally, the difference appears to be more pronounced for more difficult problems.

Although the *BS* derivatives do not strictly *dominate* the corresponding derivatives of *HP A* (especially because of the *cardinality criterion* for selecting the search direction), they generally "lose" very seldom. Therefore, *IBS** and *BSA** $_e$ with *6* = e/4 are candidates for being the best bidirectional heuristic search algorithms. We could not find that one of these is generally better than the other

In general, it strongly depends on the properties of the given problem which algorithm is "best". Of course, linear-space algorithms must be used when there is insufficient memory available. However, whenever there is, our bidirectional algorithms should be considered. Of course, running time is another important issue. Again, it strongly depends on the domain, whether the overhead of maintaining the lists deteriorates the performance or not. In particular, it depends on the effort for computing heuristic values. Finally, of course, also the efficiency of implementation plays an important role regarding the running time.

5 Conclusion

For more than two decades there has been consensus that the main reason for the superiority of unidirectional search is that the bidirectional heuristic search is afflicted by the problem of search wavefronts missing each other. However, our results show that the missile metaphor of Pohl [15] is not really applicable. The primary problem appears to be that after the first meeting especially an algorithm aiming for optimal solutions has to spend much effort for subsequently improving the solution quality, and finally for proving that no other open node can give a better solution. Therefore, only a slight relaxation of solution quality already leads to strong improvements in efficiency, and in particular to even more than in the unidirectional case. Our novel termination criterion contributes strongly to this result because of the importance of this problem.

However, a serious limitation of classical best-first search is its memory problem. In contrast, *WIDA** and *RBFS* have only linear space requirements. Rao *et al.* [20] compared depth-first with best-first search generally. While the latter has the serious disadvantage of its memory problem, there are conditions, where classical best-first search is still most efficient. In particular, it is useful when both density of solutions and heuristic branching factor are very low (and there is sufficient memory available). Moreover, both *IDA** and *RBFS* have large search overhead when there are many distinct values, and when the problem graph is not a tree [11].

Our bidirectional algorithms are designed to improve the performance under these conditions. While there are some differences, for both domains used for our experiments the main result about these algorithms is the same. We showed that and how bidirectional best-first search can be more efficient than the corresponding unidirectional counterpart without expensive wave-shaping Therefore, even for only slightly relaxing the requirements on the solution quality, the best of our algorithms are the most efficient e-admissible algorithms known for those domains where sufficient memory is available. Such domains are not the most difficult ones addressable by search techniques like *WIDA** and *RBFS.* But in those areas dominated by classical best-first search we found even better algorithms, based on a paradigm that was thought for long to be bad.

Currently, we are investigating a bidirectional combination of best-first search with linear-space search, in order to mitigate the issue of storage requirements. We hope that this will renew the interest in bidirectional search, which has additional potential for efficient implementation on parallel hardware.

Acknowledgements

The original implementation of our algorithms was based on a program written by Max Hofferer. A more efficient implementation was done by Angelika Gangl-Ehrenwerth and Harald Smetana, and the most efficient implementation based on code provided by Richard Korf is due to Ah Khorsand. The help of Harald Smetana and Ah Khorsand in running the 15-Puzzle experiments is highly appreciated. The experiments requiring very large memory were run on a machine of the computing center of the Technical University of Vienna.

A Pseudo Code of $BSA^*_{\delta,\epsilon}$

```
   procedure $BSA^*_{\delta,\epsilon}(s,t)$
1.   $g_1(s) \leftarrow g_2(t) \leftarrow 0; f_1(s) \leftarrow f_2(t) \leftarrow h_1(s); L_{min} \leftarrow \infty;$
2.   $\text{OPEN}_1 \leftarrow \{s\}; \text{OPEN}_2 \leftarrow \{t\};$
3.   $\text{CLOSED}_1 \leftarrow \text{CLOSED}_2 \leftarrow \emptyset;$
4.   until $\text{OPEN}_1 = \emptyset$ or $\text{OPEN}_2 = \emptyset$ or
           $L_{min} \leq (1+\epsilon)\max[\min_{x \in \text{OPEN}_1}(g_1(x)+h_1(x)),$
                 $\min_{x \in \text{OPEN}_2}(g_2(x)+h_2(x))]$ do
5.     if $|\text{OPEN}_1| \leq |\text{OPEN}_2|$ /* cardinality criterion */
6.       then $d \leftarrow 1$ else $d \leftarrow 2$
       endif
7.     $d' \leftarrow 3 - d$; /* set the opposite search direction */
8.     Select $m \in \text{FOCAL}_d$ with lowest $h_{F_d}$-value;
9.     $\text{OPEN}_d \leftarrow \text{OPEN}_d \setminus \{m\};$
10.    $\text{CLOSED}_d \leftarrow \text{CLOSED}_d \cup \{m\};$
11.    if $m \in \text{CLOSED}_{d'}$
         then /* nip $m$ in $\text{TREE}_d$ and prune $\text{TREE}_{d'}$ */
12.        $\Omega_m \leftarrow \{n \mid n \in \Gamma_{d'}(m) \wedge p_{d'}(n) = m\};$
13.        $\text{OPEN}_{d'} \leftarrow \text{OPEN}_{d'} \setminus \Omega_m;$
         else
14.          $EXPAND(m);$
       endif
     enduntil
15.  if $L_{min} = \infty$
       then no path exists
       else
16.      the solution path with costs $L_{min}$ is
         $(s, \ldots, p_1^2(MeetingNode), p_1(MeetingNode),$
         $MeetingNode, p_2(MeetingNode), \ldots, t).$
     endif
   endprocedure.

   procedure $EXPAND(m)$
1.   $TrimFlag \leftarrow false;$
2.   foreach $n \in \Gamma_d(m)$ do
3.     $g \leftarrow g_d(m) + c_d(m,n);$
4.     if $g + h_d(n) < L_{min}$
         then /* no screening */
5.         $f \leftarrow g + h_d(n);$
6.         if $n \notin \text{TREE}_d$ /* $n$ is a new node */
             then
7.             $g_d(n) \leftarrow g; f_d(n) \leftarrow f; p_d(n) \leftarrow m;$
8.             $\text{OPEN}_d \leftarrow \text{OPEN}_d \cup \{n\};$
9.           elsif $g < g_d(n)$ /* better path to $n$ found */
             then
10.            $g_d(n) \leftarrow g; f_d(n) \leftarrow f; p_d(n) \leftarrow m;$
11.            if $n \in \text{CLOSED}_d$
                 then /* reopen $n$ */
12.                $\text{CLOSED}_d \leftarrow \text{CLOSED}_d \setminus \{n\};$
13.                $\text{OPEN}_d \leftarrow \text{OPEN}_d \cup \{n\};$
               endif
           endif
14.        if $n \in \text{TREE}_{d'}$ and $g_1(n) + g_2(n) < L_{min}$
             then /* update $L_{min}$ */
15.            $L_{min} \leftarrow g_1(n) + g_2(n);$
16.            $MeetingNode \leftarrow n;$
17.            $TrimFlag \leftarrow true;$
           endif
       endif
     endforeach
18.  if $TrimFlag$
       then /* trim the open lists */
19.      Remove from $\text{OPEN}_1$ and $\text{OPEN}_2$ those nodes $n$
         $with(g+h)$-values $\geq L_{min}$ $(d = 1, 2)$ and which
         are not source nodes (for $\text{OPEN}_1$ the source node
         is $s$; for $\text{OPEN}_2$ it is $t$)
     endif
   endprocedure.
```

References

[1] H.W. Davis, R.B. Pollack, and T. Sudkamp. Towards a better understanding of bidirectional search. In Proc. of AAAI-84, pages 68-72, Austin, TX, 1984. Los Altos, CA.: Kaufmann.

[2] D. de Champeaux. Bidirectional heuristic search again. J. ACM, 30:22-32, 1983.

[3] D. de Champeaux and L. Sint. An improved bidirectional heuristic search algorithm. J. ACM, 24:177-191, 1977.

[4] P.E. Hart, N.J. 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):100-107, 1968.

[5] E Horowitz and S. Sahni. Fundamentals of Computer Algorithms. Springer-Verlag, New York, 1978.

[6] A.L. K611. Heuristische Suche mit begrenzbarem Fehler. Diplomarbeit, Technische Universitat Wien, 1992.

[7] A.L. K611 and H. Kaindl. A new approach to dynamic weighting. In Proc. EC A1-92, pages 16-17, Vienna, 1992. Chichester, Wiley.

[8] R.E. Korf. Linear-space best-first search. Artificial Intelligence. To appear.

[9] R.E. Korf. Depth-first iterative deepening: An optimal admissible tree search. Artificial Intelligence, 27(1):97-109, 1985.

[10] J.B.H. Kwa. BS*: An Admissible Bidirectional Staged Heuristic Search Algorithm. Artificial Intelligence, 38(2):95-109, 1989.

[11] A. Mahanti, S. Gosh, D.S. Nau, A.K. Pal, and L. Kanal. Performance of IDA* on tress and graphs. In Proc. Tenth National Conference on Artificial Intelligence, pages 549 544, San Jose, 1992. Los Altos, CA.: Kaufmann.

[12] N.J. Nilsson. Principles of Artificial Intelligence. Palo Alto, CA, Tioga, 1980.

[13] J. Pearl. Heuristics: Intelligent Search Strategies for Computer Problem Solving. Addison-Wesley, Reading, MA, 1984.

[14] J. Pearl and J.H. Kim. Studies in semi-admissible heuristics. IEEE Transactions on Pattern Analysis and Machine Intelligence, 4(4):392-399, 1982.

[15] I. Pohl. Bi-directional and heunsttc search in path problems. PhD thesis, Stanford University, 1969. SLAC Rept. No. 104.

[16] I. Pohl. First results on the effect of error in heuristic search. In B. Meltzer and D. Michie, editors, Machine Intelligence 5, pages 219-236. Edinburgh University Press, Edinburgh, 1970.

[17] I. Pohl. Bi-directional search. In Machine Intelligence 6, pages 127-140, Edinburgh, 1971. Edinburgh University Press.

[18] I. Pohl. The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving. In Proc. International Joint Conference on Artificial Intelligence 1973, pages 12-17, Los Altos, CA, 1973.

[19] G. Politowski and I. Pohl. D-node retargeting in bidirectional heuristic search. In Proc. of AAAI-84, pages 274-277, Austin, TX, 1984. Los Altos, CA.: Kaufmann.

[20] V.N. Rao, V. Kumar, and R.E. Korf. Depth-first vs best-first search. In Proc. Ninth National Conference on Artificial Intelligence, pages 434-440, Anaheim, 1991. Los Altos, CA.: Kaufmann.