Switching from Bidirectional to Unidirectional Search

Hermann Kaindl
Gerhard Kainz
Siemens AG Osterreich
Geusaugasse 17
A-1030 Vienna
Austria

Roland Steiner
Kranzgasse 4/17
A - l 150 Vienna
Austria

Andreas Auer
Penzingerstr. 18/12
A - l 140 Vienna
Austria

Klaus Radda
Weinwurmweg 1/53
A-1220 Wien
Austria

Abstract

Recently, we showed that for traditional bidirectional search with "front-to-end" evaluations, it is not the meeting of search fronts but the cost of proving the optimality of a solution that is problematic. Using our improved understanding of the problem, we developed a new approach to improving this kind of search: switching to unidirectional search after the search frontiers meet for the first time (with the first solution found). This new approach shows improvements over previous bidirectional search approaches and (partly) also over the corresponding unidirectional search approaches in different domains. Together with a special-purpose improvement for the TSP, this approach showed better results than the standard search algorithms using the same knowledge.

1 Introduction

Linear-space search algorithms such as DFBB (depth-first branch-and-bound) [Lawler and Wood, 1966] or IDA* (iterative-deepening- A*) [Korf, 1985) can solve very difficult problems because they face no real space limitations. However, they have large search overheads if there are many distinct cost values, and if the problem graph is not a tree. So, whenever sufficient memory is available, traditional best-first search may be the best choice, since it expands the fewest nodes among all admissible algorithms using the same cost function [Zhang and Korf, 19931. Since machines with larger and larger memories are becoming available, this approach can be applicable for many problems in practice (e.g., route planning).

Due to the optimality result of A* over unidirectional competitors in [Dechter and Pearl, 1985], the question may arise what to improve here. However, there is some renewed interest in *bidirectional* heuristic search, which can both expand fewer nodes and be faster than A* [Kaindl and Kainz, 1997]. Still, there is some drawback involved with this kind of search: the big effort for proving the optimality of solutions found. The issue is to satisfy the specific termination condition of such searches. We address this issue by proposing a new approach that switches to unidirectional search immediately after the bidirectional search finds the first solution.

This paper is organized as follows. In order to make it self-contained, we sketch some background material on bidirectional heuristic search. Then we propose our new approach to bidirectional search that switches to unidirectional search once a solution is found. After describing a more special-purpose improvement, we finally demonstrate both the practical relevance of some theoretical bounds derived in [Kaindl and Kainz, 1997) and the efficiency of our new search approach through presenting experimental results.

2 Background

First, let us be precise on what constitutes a bidirectional search. When there is one goal node t explicitly given, such a search proceeds both in the forward direction from the start node s to t and in the backward direction from t to s [Pohl, 1971). Bidirectional search is possible if for a given node n the set of parent nodes p_i can be determined for which there exist operators that lead from p_i to n. Searching backwards means generating parent nodes successively from the goal node t (see, e.g., [Russell and Norvig, 1995]). In other words, backward search implements *reasoning* about the operators in the backward direction.

Bidirectional search also works correctly in cases where the costs of inverse arcs between any two nodes are different: the backward search implements reasoning in the backward direction but takes account of the cost of going in the forward direction. More formally, $k_1(m,n) = k_2(n,m)$ is the cost of an optimal path from m to n. So, $k_2(m,n)$ is the cost of an optimal path from n to m, and is used for notational convenience only.[1] All the bidirectional search algorithms dealt with in this paper work correctly under these conditions and do *not* require that the operators are reversible or that the cost of a path is the same in either direction.

In this paper we focus on the kind of traditional bidirectional search with "front-to-end" evaluations: the heuristic evaluation functions $f_d(n)$ estimate the cost of an optimal path to the appropriate endpoint (i.e., $f_1(n)$ uses t as the target for the forward search, and $f_2(n)$ uses s as the target for the backward search). So, we do not technically deal here with those algorithms with "front-to-front" evaluations, that perform a *wave-shaping* strategy: neither with

[1]The notation we use in this paper is summarized in the appendix.

the traditional BHFFA [de Champeaux and Sim, 1977] nor with the more recent approach to non-traditional bidirectional search called *perimeter search* [Dillenburg and Nelson, 1994; Manzini, 1995]. Also the more recently proposed non-traditional bidirectional search with "front-to-end" evaluations [Kaindl *el al.,* 1995; Kaindl and Kainz, 1997] is not dealt with in this paper.

So, it is sufficient here to sketch the essentials of traditional bidirectional search with "front-to-end" evaluations. We can view such a search essentially as two A*-type searches in opposite directions.[2] These are performed quasi-simultaneously, i.e., on a sequential machine one node is expanded after another, but the search direction is changed at least from time to time. The decision for searching in the forward or backward direction is made anew for each node expansion, most often according to the *cardinality criterion* [Pohl, 1971]: search in the direction with fewer open nodes.

The typical representatives of traditional bidirectional heuristic search with "front-to-end" evaluations are the two algorithms BHPA [Pohl, 1971] and BS* |Kwa, 19891. While BHPA explores part of the search space twice, BS* can avoid this due to its major improvements over BHPA.[3] Both BHPA and BS* are *admissible* if h_d is *consistent.*[4] We assume the availability of a consistent heuristic evaluation function h_d in both directions.

Whenever the search frontiers meet at some node n, a solution is found. Its cost is $g_1(n) + g_2(n)$, i.e., the cost of the path found by the forward search from *s* to n, plus the cost of the path found by the backward search from *n* to /. Even when the two parts of such a solution of the forward and the backward search are optimal, however, the concatenated solution path is not necessarily optimal. Therefore, such an algorithm requires a special termination condition for guaranteeing optimal solutions. The termination condition of BHPA is as follows:

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

This condition essentially means that the cost L_{min} of the best (least costly) complete path from s to t found so far is not larger than an estimate computed from the f_d-values in both search frontiers. If the heuristic used for these estimates is admissible, this path must already be an optimal solution in order to satisfy this termination condition. Implicitly this is also the condition for successful termination of the improved algorithm BS* [Kwa, 1989], which removes all nodes n whose f_d-values are $\geq L_{min}$ and terminates when OPEN_1 or OPEN_2 is empty.

Since understanding this condition is important for this paper, we discuss shortly how it can be achieved (this issue is dealt with more formally in (Kaindl and Kainz, 1997]). If there is a solution path from .s to /, BHPA or BS* will terminate successfully (i.e., by finding an optimal path) if and only if both the following conditions are satisfied:

1. an optimal solution must have been found, that is, $L_{\min} = C^*$; and
2. at least in one of the search frontiers *d* of BHPA or BS* the minimum /-value must have been raised at least to the value of an optimal solution C*, that is. $\min_{x \in \text{OPEN}_d} f_d(x) \geq C^*$.

The minimum f-values of OPEN_d are at first the values $f_1(s)$ and $f_2(t)$, respectively- S i n (f_d i s consistent they do not exceed C^*. The minimum f-values of OPEN_d increase only gradually until all the nodes with f-values $< C^*$ of at least one search frontier are expanded (or nipped or pruned by BS*). Since the maximum of those values is used, only one of them must become $\geq C^*$. During the search, $L_{\min} \geq C^*$ always holds, and when an optimal solution is found, $L_{\min} = C^*$.

3 Switching to Unidirectional Search

Because of the issue of satisfying the termination condition, we devised an approach of switching from bidirectional search to unidirectional search once a solution is found. Depending on the concrete instantiation of this approach, the termination condition may be satisfied earlier, or a different condition of the unidirectional search algorithm used may apply directly.

The first instantiation actually amounts to a simple modification of BS*. When the search frontiers meet for the first time, the search direction *d* with

$$\min_{x \in \text{OPEN}_d} f_d(x) > \min_{x \in \text{OPEN}_{3-d}} f_{3-d}(x) \quad (2)$$

is selected and kept until the end of the search. According to this condition, it chooses the OPEN set with the higher value, and directs the whole effort of the search after the first meeting towards achieving a minimum f-value of C^* in *that* OPEN set. In case of equality, the cardinality criterion is used as a tie-breaker. We call this algorithm Switch-A*.

The second instantiation is a variant of Switch-A* that utilizes the *Max* method for dynamically improving heuristic values according to I Kaindl and Kainz, 19971. So, we call it Max-Switch-A*. In order to make good use of this method, the differences in one frontier used for improving the heuristic values in the other should be as high as possible. Therefore, it is useful to have reached the next level of f-values for improving the search in the other direction. That is why Max-Switch-A* docs not change its search direction according to the cardinality criterion, but whenever the next level of

[2]The origin is clearly BHPA [Pohl, 19711, which can be intuitively viewed to consist of two HPA searches in opposing directions. For *admissible* but not *consistent* heuristic functions, the option to move nodes back from CLOSED to OPEN is important, if a new better g_d-value is found. As long as the heuristic function used is consistent, we can ignore the specific differences for our discussion here.

3These improvements are the following:
(i) *nipping:* If a node is selected for expansion which is already closed in the opposite search tree, it can just be closed *without* expansion.
(ii) *pruning:* In the same situation, descendants of this node in the opposing OPEN list can be removed.

[4]h_d is said to be consistent if $h_d(m) \leq h_d(n) + k_d(m, n)$ for all nodes *m* and *n.* If for any goal node *t* $h_d(t) = 0$, this implies that h_d is admissible, i.e., the heuristic function never overestimates the minimal cost.

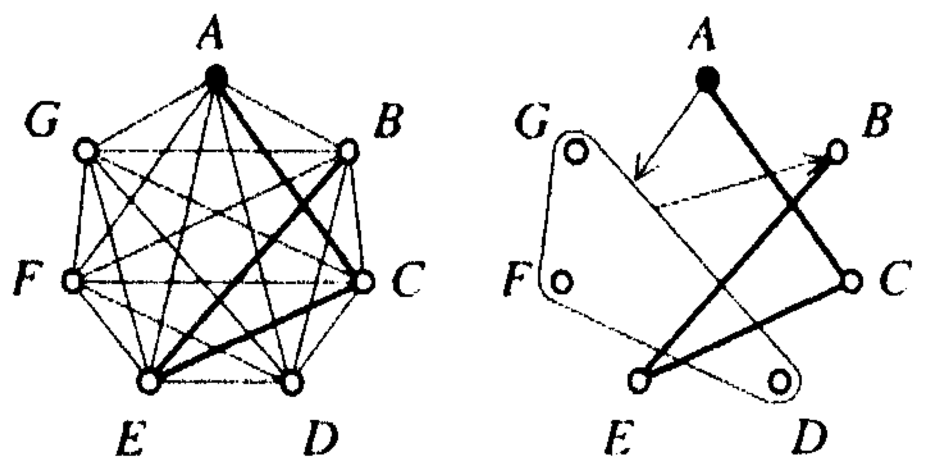

Figure 1: An example of complementary nodes.

/-values is reached in the current search direction. Also after the first meeting of the search frontiers, the current search direction is kept until this condition is fulfilled. In effect, this amounts to a delay of switching. Only then, the final decision for the search direction is made by Max-Switch-A*.

The third instantiation can be implemented as follows (we call it Switch-DFBB-Trans):

1. Perform A*-type searches in opposite directions as in BHPA or BS*, i.e., change the search direction according to the cardinality criterion.
2. When the search frontiers meet for the first time, determine the search direction *d* according to condition (2).
3. Switch to DFBB-Trans[5] and initialize its upper bound with L_{min}.
4. Perform a search with DFBB-Trans in direction *d* (utilizing the hash table entries from the previous searches and starting from the frontier of OPEN_d) until its normal termination condition holds.

This instantiation strives to be *faster,* since DFBB-Trans does not have to maintain a priority queue like A*. In addition, DFBB-Trans does not *expand* nodes (in the sense of generating all the children of the expanded node at once), so it can save the generation of certain nodes that A* generates during node expansion.

Other instantiations are easy to construct, since in principle this approach may switch to any unidirectional search. When, for instance, "plain" DFBB or IDA* is used, only linear storage is needed for the remaining search. Still, the information already stored in the hash table of the bidirectional search can be used to improve efficiency (e.g., as in BAI [Kaindl *el al*., 19951).

4 Bidirectional Search for the TSP

We also present here another approach for improving bidirectional search that is more specific for domains where a fixed number of items is to be sequenced. In the TSP (traveling salesman problem) these items are the cities. In such a domain, it is possible to save the expansion of nodes through recognizing complementary paths in the search graph. In the following, we explain this idea specifically for the TSP While it is possible with some additional book-keeping to apply this approach to the asymmetric TSP [Steiner, 19961, we explain here the simpler case of the symmetric TSP.

[5]DFBB-Trans was first mentioned in [Kaindl *el al*., 19951. It simply utilizes the approach of improving heuristic values through a *transposition table* (Reinefeld and Marsland, 19941 in DFBB.

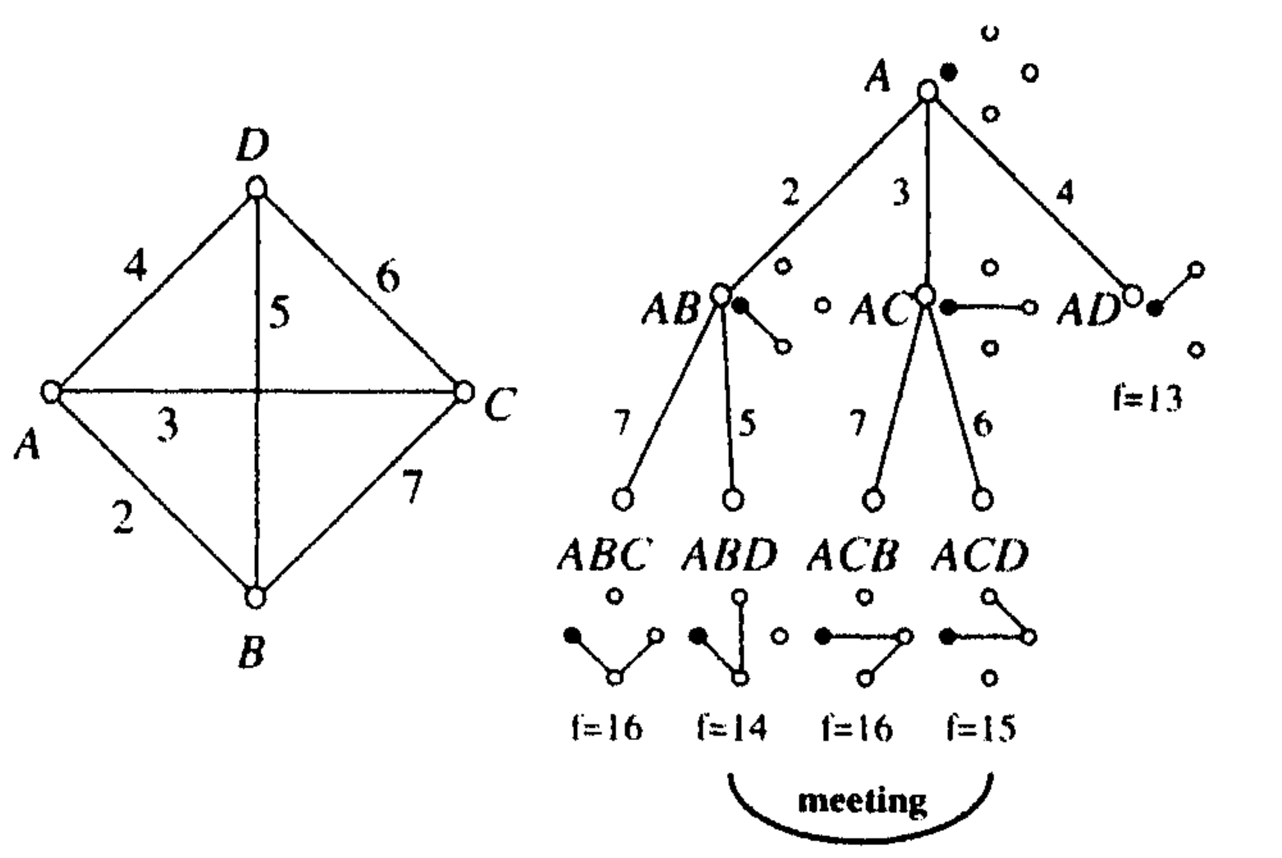

Figure 2: A graph of bidirectional search for a TSP example.

The key idea is to perform a unidirectional search (that stores nodes) and to interpret every node both as a partial tour in the forward direction that it represents directly, and concurrently as a complementary tour in the backward direction. So, in just *one* explicitly stored search frontier, implicitly *two* frontiers are contained. The search must be able, of course, to identify those nodes that represent complementary tours — we call them complementary nodes.

The example in Figure 1 illustrates complementary nodes. Let us assume that some node represents the best partial route found so far between some start city *A* and some currently visited city, say B, that goes through certain other cities, e.g., $\{C, E\}$. Let us take the set of all these cities, e.g., $\{A, B, C, E\}$ and compute its set complement, e.g., $\{D, F, G\}$. Then the complementary node is the one that represents the best partial route found so far between the same start city ,4 and the same currently visited city, in this example *13,* that goes through this complementary set of cities, in this example $\{D, F, G\}$.

Figure 2 illustrates an example of a TSP search that utilizes this idea. The graph in the left part of this figure defines the problem, and the graph in the right shows a snapshot of a corresponding search.[6] The important point is that the node *ABD* is complementary to the node *ACD,* and so the search can already recognize the meeting of the corresponding partial tours and therefore a complete solution.

Apart from admissibility, the most important theoretical result about this approach is *dominance* over A*. That is, all the nodes expanded by either BHPA/TSP or BS*/TSP (that include this approach) must be expanded by A*. The corresponding proofs can be found in [Steiner, 19961.

5 Experimental Results

Our experiments showed promising results in several domains: mazes (like those used in [Kaindl and Kainz, 1997]), route planning (like in [Kwa, 1989J), TSP, and 15-Puzzle. In all these domains, the new "switching" approach gives better results than BS* in terms of both generated nodes and running time. These results are statistically significant. In

[0]In this paper we use the *minimum spanning tree* of the cities not yet visited as the heuristic function h_d.

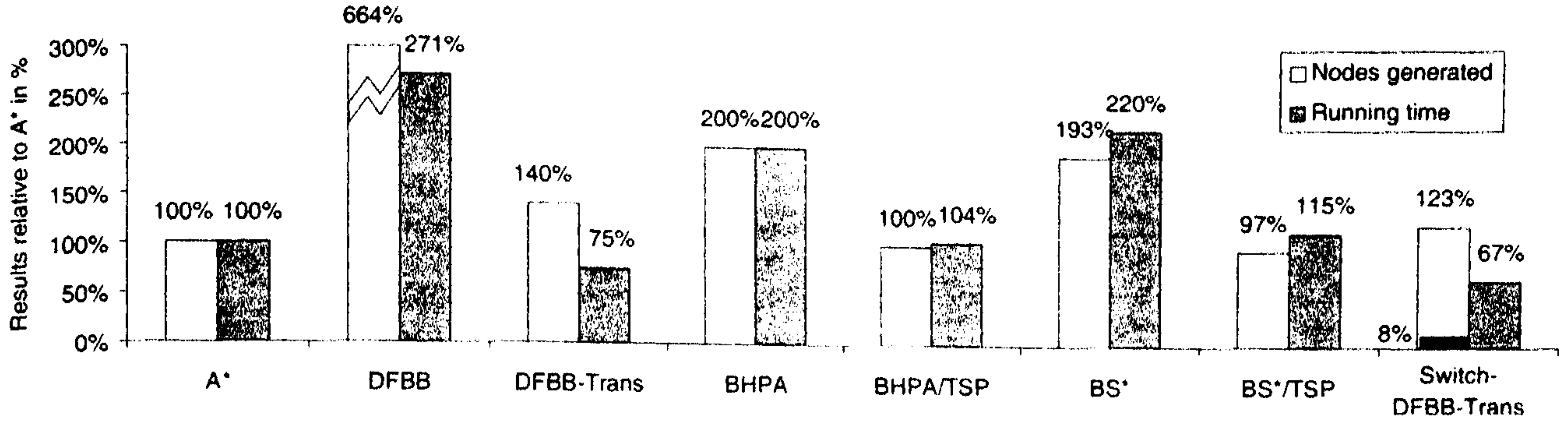

Figure 3: Comparison on the TSP (results relative to A* in %).

addition, we gained empirical evidence that the conjecture in [Kwa, 1989] that BS* would become generally superior to A* for more difficult problems is invalid. In the following, we focus on the domains TSP and 15-Puzzle, which are also very interesting in regard to the theoretical results presented in (Kaindl and Kainz, 1997], TSP is a domain where the "naive" approach of implementing BHPA leads to nearly twice of the node expansions compared to A*, the theoretical upper bound. The 15-Puzzle with the Manhattan distance heuristic, however, is a domain with very few different /-values, where according to the theory in I Kaindl and Kainz, 1997] comparably many node expansions can be saved by the traditional approach of bidirectional search using "front-to-end" evaluations. So, our results are consistent with this theory and they illustrate that the bounds derived there are reasonable in practice.

5.1 Euclidean TSP

First we present here our experimental results on the Euclidean TSP These experiments did not attempt to achieve the best ever results in this particular domain, but they were intended to compare our new algorithms with the standard heuristic search algorithms. All the compared algorithms use no domain-specific knowledge about the Euclidean TSP other than the minimum spanning tree heuristic. Figure 3 shows the results for 19-city problems.[7] We selected these particular results for presentation since they are reproducible using any standard PC with 48MBytes of main memory. On average, A* generated 73,650 nodes in 7.0 seconds for solving one problem instance.

On such problem instances, both BHPA (result is rounded to 200%) and BS* expand close to twice the number of nodes compared to A*. BS* can save only *1%* of the node expansions compared to BHPA. In fact, the naive implementations explore the same space twice, and so they treat the TSP as a *perfectly A*-symmetric* domain as defined in [Kaindl and Kainz, 1997].

The versions BHPAATSP and BS*/TSP that recognize complementary nodes as described above expand half of the nodes compared to the "naive" implementations. So, they are slightly better in terms of node expansions than A* (as they dominate A*'), hut they are actually slower than A*.

The best results in terms of running time achieves Switch-DFBB Trans as presented above. After 8% of its overall node generations, it switches to DFBB-Trans. At this time it has on average found a solution with cost 1.02 x *C* which is used as the initial upper bound by DFBB-Trans. Note, that due to its depth-first search DFBB-Trans cannot make use of complementary nodes and really works unidirectionally. While overall Switch-DFBB-Trans generates more nodes than A* or BS*/TSP, it is faster than both of them due to the fact that DFBB-Trans is faster per node searched.

The only other algorithm competing here is DFBB-Trans itself. However, the initial bound gained through the nearest-neighbor heuristic is worse than the bound gained through the bidirectional approach running first, and within the time required for finding the better bound by Switch-DFBB-Trans, DFBB-Trans alone cannot achieve such a good bound on its own.

It is intersting to compare this bidirectional approach with unidirectional approaches that first find an approximate solution and continue the search until they find an optimal solution IGent and Walsh, 1997; Korf, 19951. For instance, in the number partitioning problem such an algorithm may often know from the solution itself that it is already optimal. Our approach does not need such a domain -specific property. It gradually raises the minimum f-values of $OPEN_d$ until the termination condition (1) is satisfied. While this is time consuming, our "switch" approach improves the efficiency.

In summary, Switch DFBB-Trans is better in terms of running time than all these competitors, which include the standard search algorithms that have the same knowledge available. This result is statistically significant. For example, the probability that the improvement of the running time over DFBB-Trans is due to chance fluctuation is 0.66 percent according to a test that compares the means of the paired samples of the absolute running times, and it is even much smaller according to the same test for the data relative to the difficulty of each instance (0.02 percent) as well as according to the sign test (0.06 percent).[8] In comparison with A*, this chance is even smaller than 0.05 percent for all these statistic tests.

We do not include here results from perimeter search, since according to our experiments this approach appears not to work satisfactorily in this domain (like for maze problems as shown in I Kaindl and Kainz, 1997]).

[8] For more details on the statistic tests used we refer the interested reader to [Kaindl and Smetana. 19941.

5.2 15-Puzzle

Now let us have a look on specific experimental results for finding optimal solutions to a set of (sliding-tile) 15-Puzzle problems.[9] We compare algorithms that achieve the best published results in this domain with IDA* and BS* as well as Switch-A* and Max-Switch-A* (as presented above). All the compared algorithms use no domain-specific knowledge about the puzzle other than the Manhattan distance heuristic.[10] The main storage available on a Convex C3220 was up to 256 Mbytes.

Within this given amount of storage. A*, BHPA, BS* and Max-BAI-Trans IKaindl and Kainz, 19971 can store a maximum of 5 million nodes in our implementations. BIDA* [Manzini, 1995] requires more storage per node, so it can store a maximum of 1 million perimeter nodes within 256 Mbytes.[11]

While Max-BAI-Trans and BIDA* can find optimal solutions to all of the given instances, the traditional best-first searches cannot (with this amount of storage and the Manhattan distance as the only knowledge source): A* solves 34 and BS* 59 from the given set of 100 problem instances (for BHPA we did not gather the complete data since they would be clearly worse than those of BS*).

Switch-A* solves 63 instances of this set under these constraints, so in this regard it is better than both A* and BS*. On the set of 34 problem instances solvable by A*, Switch-A* just expands 42.3% of the nodes expanded by A* and needs 55.5% of its time. Max-Switch-A* solves 79 problems of the set of 100 instances.

On the set of 56 problem instances solvable by both BS* and Switch-A*, the latter expands 93.1% of the nodes expanded by BS* and needs 93.6% of its time.[12] On average, Switch-A* finds the first solution after 7%; of its overall time, and the average cost of these solutions is 1.06 x *C**. For the overall data see Table 1. This table shows that on this set of instances, our new algorithm Switch-A* is as good as the best published approaches so far IKaindl and Kainz, 1997; Manzini, 19951.[13] While the precise mean data might even suggest a slight improvement, statistic tests tell us that there is a high chance fluctuation. So, we cannot reject the null hypothesis that Switch-A* and Max-BAI-Trans are equally good. In fact, the difference amounts to less than 1% (in effect, less than 1 second) per problem instance on average.

Table 1: Comparison on the 15-Puzzle (results relative to BS* in %).

	Nodes generated	Running time
BS*	100.0	100.0
IDA*	1628.4	206.2
BIDA* (depth 16)	19.6	112.1
Max-BAI-Trans	108.2	94,4
Switch-A*	93.1	93.6
Max-Switch-A*	62.1	78.8

Max-Swilch-A*, however, improves the performance on these problems. It expands only 62.1 % of the nodes expanded by BS*, but due to the overhead of applying the *Max* technique, it needs 78.8% of its time. Still, this result is better than the results of the previously best methods BIDA* and Max-BAI-Trans both in terms of the mean and the median values. It is statistically significant according to the sign test and a test that compares the means of the paired samples of the absolute running times.

Still, the question may arise, whether the application of the *Max* technique directly in BS* could not be better. This application was also studied in [Kainz, 1996] and achieved improvements, but for a good utilization of this technique the search direction should stay constant at least for a while. Finally, it turned out that the switching approach presented in this paper is the limit of what can be achieved by the application of *Max* in the context of BS*.

The major issue with this approach is that it was not able to solve all the problem instances. However, machines with larger and larger memories are becoming available now. In contrast to several approaches to memory-bounded search like MA* [Chakrabarti *et al.,* 1989], MREC [Sen and Bagchi, 1989], SMA* [Russell, 19921 and ITS [Ghosh *et al.,* 19941, our approach showed clear improvements in running time over IDA*.

Moreover, using much improved heuristic functions like those developed by [Culberson and Sehaeffer, 1996; Korf and Taylor, 1996] would make it possible to run Switch-A* and Max-Switch-A* even with smaller available memory. We conjecture that also Switch-A* would be better under these conditions than both BIDA* and Max-BAI-Trans, since these approaches perform dynamic improvements of the heuristie during their search, which may not be as effective for such improved heuristics as it is for the Manhattan distance heuristic.

[9]We used the set of 100 instances from IKorf, 19851.

[10]With much improved heuristic functions, much more efficient searches result [Culberson and Sehaeffer, 1996] and even solving 24-Puzzle instances has become feasible IKorf and Taylor, 1996).

[1 'Since a given machine has a fixed storage size, it would not be fair to compare algorithms with the same number of nodes when these require different amounts of storage. However, BIDA* has an optimum in running time for a smaller perimeter size that does not even require that available amount of storage IKaindl and Kainz, 1997].

[12]A* cannot be compared on this set. Just to give an idea of the overall difficulty of this problem set, note that BS* generates some 2.25 million nodes on average, which needs slightly less than 90 seconds on a Convex C3220.

[13]BIDA*'s result here is worse than the data reported in (Manzini, 1995]. This is primarily due to the use of a different machine and a different implementation that is based on the very efficient code of IDA* for the puzzle provided to us by Korf that we are using. In such an implementation the overhead especially of wave shaping shows up more clearly even when using the runtime optimizations described in [Manzini, 1995]. While we had no access to the implementation by Manzini, in E-mail communication with him we were given some hints about it, and there was agreement about the overall effect on the relative running times due to the different implementations of IDA*. The data for BIDA* and Max-BAI-Trans in this table are also not exactly those reported in [Kaindl and Kainz, 1997], since not all of the problem instances are included.

6 Conclusion

The major problem of traditional bidirectional search with "front-to-end" evaluations as exemplified by BHPA and BS* is the cost of satisfying the termination condition. So, we addressed this problem and developed a new approach to improving this kind of search: switching to unidirectional search after the search frontiers meet for the first time (with the first solution found). This new approach shows improvements over previous bidirectional search approaches and partly also over the corresponding unidirectional search approaches in different domains.

Appendix, Glossary of Notation

s, t	Start node and goal node, respectively.
d	Current search direction index; when search is in the forward direction $d = 1$, and when in the backward direction $d=2$.
C^*	Cost of an optimal path from s to t.
$k_d(m,n)$	Cost of an optimal path from m to n if $d = 1$, or from n to m if $d = 2$.
$g^*_d(n)$	Cost of an optimal path from .s? to n if $d = 1$, or from n to t if $d = 2$.
$h^*_d(n)$	Cost of an optimal path from n to t if d — 1, or from s to n if $d = 2$.
$g_d(n), h_d(n)$	Estimates of $g^*_d(n)$ and $h^*_d(n)$, respectively.
$f_d(n)$	Static evaluation function: $g_d(n) + h_d(n)$.
$L_{\min}$	Cost of the best (least costly) complete path found so far from s to t.
OPEN_d	The set of open nodes in search direction d.
CLOSED_d	The set of closed nodes in search direction d.

References

[Chakrabarti et al., 1989] P.P. Chakrabarti, S. Ghose, A. Acharya. and S.C. DcSarkar. Heuristic search in restricted memory. *Artificial intelligence,* 41(2): 197-221, 1989.

[Culberson and Schaeffer, 1996] J. Culberson and J. Schaeffer. Searching with pattern databases. In G. McCalla, editor, *Advances in Artificial Intelligence,* pages 402-416. Springer-Verlag. Berlin, 1996.

[de Champeaux and Sim, 1977] D. de Champeaux and L. Sim. An improved bidirectional heuristic search algorithm. *J. ACM,* 24(2):177-191, 1977.

iDechter and Pearl, 1985] R. Dechter and J. Pearl. Generalized best-first strategies and the optimality of ,4*. *J. ACM,* 32(3):505-536, 1985.

iDillenburg and Nelson, 1994] J.F. Dillenburg and PC. Nelson. Perimeter search. *Artificial Intelligence,* 65(1): 165-178, 1994.

[Gent and Walsh, 1997] I.P Gent and T. Walsh. From approximate to optimal solutions: constructing pruning and propagation rules. In *Proc. Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97),* pages 1396-1401. San Francisco, CA: Morgan Kaufmann Publishers, 1997.

iGhosh *et al.,* 1994] S. Ghosh, A. Mahanti, and D.S. Nau. ITS: an efficient limited-memory heuristic tree search algorithm. In *Proc. Twelfth National Conference on Artificial Intelligence (AAAI-94),* pages 1353-1358. Menlo Park, CA: AAAI Press / The MIT Press, 1994.

IKaindl and Kainz, 1997] H. Kaindl and G. Kainz. Bidirectional heuristic search reconsidered. *Journal of Artificial Intelligence Research (JAIR),* 7:283-317, 1997.

IKaindl and Smetana, 1994] H. Kaindl and H. Smetana. Experimental comparison of heuristic search algorithms. In *AAAI-94 Workshop on Experimental Evaluation of Reasoning and Search Methods,* pages 11-14, 1994.

IKaindl *et ai,* 1995] H. Kaindl, G. Kainz, A. Leeb, and H. Smetana. How to use limited memory in heuristic search. In *Proc. Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95),* pages 236-242. San Francisco, CA: Morgan Kaufmann Publishers, 1995.

[Kainz, 1996] G. Kainz. Neue Algorithmen fur die bidirektionale heuristische Suche. Doctoral dissertation, Technische Universitat Wien, Vienna, Austria, 1996.

[Korf and Taylor, 1996] R.E. Korf and L.A. Taylor. Finding optimal solutions to the Twenty-Four Puzzle. In *Proc. Thirteenth National Conference on Artificial Intelligence (AAAI-96),* pages 1202-1207. Menlo Park, CA: AAAI Press / The MIT Press, 1996.

I Korf, 19851 R.E. Korf. Depth-first iterative deepening: An optimal admissible tree search. *Artificial Intelligence,* 27(1):97—109, 1985.

[Korf, 1995] R.E. Korf. From approximate to optimal solutions: a case study of number partitioning. In *Proc. Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95),* pages 266-272. San Francisco, CA: Morgan Kaufmann Publishers, 1995.

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

lLawler and Wood, 1966] E.L. Lawler and D. Wood. Branch-and-bound methods: a survey. *Operations Research,* 14(4):699-719, 1966.

fManzini, 19951 G. Manzini. BIDA*: an improved perimeter search algorithm. *Artificial Intelligence,* 75(2):347-360, 1995.

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

[Reinefeld and Marsland, 1994) A. Reinefeld and T.A. Marsland. Enhanced iterative-deepening search. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMl),* 16(12):701-709, July 1994.

I Russell and Norvig, 1995] S.J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, Englewood Cliffs, NJ, 1995.

[Russell, 1992] S. Russell. Efficient memory-bounded search methods. In *Proc. Tenth European Conference on Artificial Intelligence (ECAI-92),* pages 1-5. Chichester, England: Wiley, 1992.

[Sen and Bagchi, 1989] A.K. Sen and A. Bagchi. Fast recursive formulations for best-first search that allow controlled use of memory. In *Proc. Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89),* pages 297-302. San Francisco, CA: Morgan Kaufmann Publishers, 1989.

[Steiner, 1996] R. Steiner. Back jumping in Zustandsraumen und bidirektionale Suche beim TSP. Diplomarbeit, Technische Universitat Wien, Vienna, Austria, 1996.

[Zhang and Korf, 19931 W. Zhang and R.E. Korf. Depth-first vs. best-first search: new results. In *Proc. Eleventh National Conference on Artificial Intelligence (AAAI-93),* pages 769-775. Menlo Park, CA: AAA! Press /The MIT Press, 1993.