STAIRS 2012
K. Kersting and M. Toussaint (Eds.)

doi:10.3233/978-1-61499-096-3-240

A two-phase bidirectional heuristic search algorithm[1]

Francisco Javier Pulido L. Mandow J.L. Pérez de la Cruz [a]
[a] *francis,lawrence,perez@lcc.uma.es*

Abstract. This work describes a new best-first bidirectional heuristic search algorithm with two phases. The new algorithm is based on a critical review of the basic search reduction operations in previous algorithms like BS* or Switch-A*. The general guideline is to let search fronts meet as close to midground as possible. In a first phase, search is discontinued at nodes as soon as the opposite frontier is met, terminating when one of the fronts runs out of open nodes. In a second phase, unidirectional search is conducted from the discontinued nodes until an optimal solution can be guaranteed. The new algorithm is tested on random instances of the 15-puzzle and on path-finding problems. A significant improvement in efficiency is observed when compared with other bidirectional algorithms.

1. INTRODUCTION

Bidirectional search is an alternative to classical unidirectional graph search. It can be applied in cases where the goal is known, and both predecessors and successors of a node in the graph can be calculated.

Bidirectional blind best-first search is often considered a good alternative to blind unidirectional search, achieving important reductions in search effort. However, the development of efficient bidirectional heuristic search algorithms has proven to be a difficult task.

The general bidirectional heuristic search idea is to perform two A* searches in opposite directions. This basic *front-to-end* approach has been developed in algorithms like BHPA [1], BS* [2], or Switch-A* [3], which incorporate different techniques aimed at balancing the search between the two directions, avoiding the repeated expansion of nodes, or achieving a fast termination after the optimal solution has been found. Several works have also explored the opportunity offered by these algorithms to dynamically improve heuristic evaluations [3][4][5].

The belief that search fronts can pass each other without meeting until the very final stages of the search (the so-called *crossing missiles metaphor* [1]) prompted the development of *front-to-front* search algorithms. Rather than aiming at the goal from the start node (and viceversa), each search front is aimed at meeting the front of the opposite search. Representative algorithms include BHFFA [6][7] or d-node retargeting [8]. In general, these techniques achieved important reductions in the number of nodes con-

[1] This work is partially funded by / Este trabajo está financiado por:
Consejería de Innovación, Ciencia y Empresa. Junta de Andalucía (España), P07-TIC-03018

sidered, but at the cost of increasing time requirements. Additionally, arguments were presented against the missile metaphor [4]. Nevertheless, the development of *perimeter search* [9][10] achieved good results in certain domains, although limited to small perimeters.

More recently, a single-frontier bidirectional search schema (SFBDS) has also been proposed [11][12], which makes bidirectional search amenable to depth-first search techniques. The effectiveness of the approach depends on the determination of adequate *jumping policies* for different classes of problem domains. These eventually decide in which direction to explore the state space at each step.

This paper proposes a new front-to-end best-first bidirectional heuristic search algorithm. The new algorithm (2PBS*) divides the search into two phases. The first one lets both search fronts meet naturally in a balanced way. Once a common line of encounter has been clearly defined, the algorithm turns to unidirectional search in a second phase in order to guarantee that an optimal solution is returned. The algorithm is evaluated in two different problem domains (15-puzzle and path finding) and compared to previous front-to-end algorithms.

The next section carries out a critical review of search reduction techniques used in previous best-first front-to-end algorithms, and outlines the new algorithm. Section 3 presents the algorithm in detail and proves its admissibility. Section 4 describes the experimental analysis and results. A discussion is presented in section 5. Finally, some conclusions and future work are described.

2. REDUCTION TECHNIQUES IN BIDIRECTIONAL HEURISTIC SEARCH

The idea of a basic bidirectional heuristic search amounts to performing two A* searches in opposite directions, as shown in the BHPA [1] and BS* [2] algorithms. However, a number of problems have prevented bidirectional search from achieving the performance gains expected by researchers. Research effort has been directed over the years towards an adequate diagnosis of the workings of bidirectional search algorithms, as well as to new algorithmic techniques to improve their performance.

One interesting feature of bidirectional search is that many of the effort-saving techniques proposed over the years are *heuristic* in the deepest sense of the term. They can perform very well in certain cases, and not so well in others. Particularly, it is frequently possible to provide examples where a given technique saves search effort, as well as examples where the very same technique wastes search effort. Eventually, the value of the proposed techniques has to be evaluated experimentally on average terms. This evaluation is additionally complicated by the fact that bidirectional search can perform very differently in different problem domains.

The first important decision in bidirectional search deals with the way the search effort is distributed in both directions. It is generally acknowledged that the *cardinality criterion* [1] is a good solution to the problem. Basically, this calls for searching in the direction with fewer open nodes. The rationale is to let both search fronts grow approximately equally, and meet as close to midground as possible.

In his influential early work, Kwa [2] claimed that, if proper care is not taken, search fronts can easily go *through* each other, duplicating search effort. Therefore, the BS* algorithm incorporated several special operations (nipping, pruning, trimming, and screen-

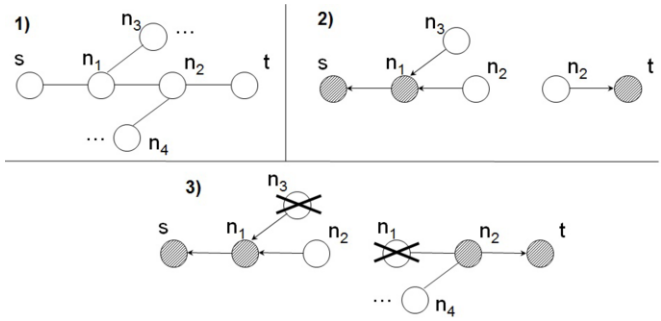

Figure 1. (1) Sample graph from nipping and pruning operations in BS*. (2) Search fronts meeting at node n_2. (3) Search fronts after nipping node n_1 and pruning node n_3.

ing) to avoid unnecessary exploration and prevent the repeated expansion of nodes in both fronts.

Let us consider for example the case of *nipping* and *pruning* operations, which are central to BS*. Let s and t denote the start and goal nodes of the search respectively. Let the forward search proceed from s to t, and the backward search in the opposite direction. Let us consider the graph depicted in Figure 1, and a BS* search situation where the forward search has expanded nodes s and n_1, while the backward search has expanded only t. Both frontiers have already met at node n_2 (second image in Figure 1). Let us assume that the backward search selects for expansion nodes n_2 and n_1. When node n_1 is selected for expansion, BS* discontinues search at that node (nipping), since the optimal path joining s and t through n_1 has already been found. For the same reason, search can also be discontinued in the forward search at n_3 (pruning) (see the third image in Figure 1). However, reaching n_1 in the backward direction also opened n_4, which could trigger additional expansions, at least until n_2 is selected in the forward direction and nipping and pruning come into action again. In other words, the relative benefits of nipping and pruning may depend on the particular situation at hand.

While BS* clearly improved over the performance of BHPA, its performance was still worse than that of A* in many cases. One of the most cited explanations for the poor performance of BS* in some domains is the so-called *crossing missiles metaphor*, which claims that bidirectional searches can pass through each other without meeting until the final stages of the search. This could easily double the search effort when compared to unidirectional search. This explanation leads to the development of *front-to-front* algorithms, like perimeter search [9] or BIDA* [10], as opposed to traditional *front-to-end* ones.

However, the analysis of Kaindl and Kainz [4] discredited the missiles metaphor and claimed that the frontiers of bidirectional search meet quite early, putting the focus again on *front-to-end* algorithms. According to these authors, the real problem would lie then in the *termination condition* of the algorithm, which basically has to guarantee that the optimal solution has been found. The new algorithm, Switch-A* [3], proposed switching from bidirectional to unidirectional search as soon as search frontiers meet for the first time. Earlier termination was expected to occur with this technique.

The directional switch idea is a clear abandonment of the *cardinality criterion* described above. Although some improvements were reported over BS* [3], the idea was later claimed to be less interesting than continued bidirectional search, due to its reduced

capability to properly exploit dynamically improved heuristics [5]. The experimental results presented in section 4 will show that early switching to an unidirectional search can in fact be worse than BS* in certain cases.

This paper proposes a new front-to-end bidirectional heuristic search algorithm based on a reconsideration of some of the above mentioned techniques, as well as on our own experience with them over different problem domains.

Our research is guided by the well established heuristic that the best way to save effort is to let search frontiers meet as close to midground as possible. To be precise, we propose a two-phase algorithm. The first phase lets both frontiers meet naturally and define a common line of encounter. During this phase, search is discontinued at each frontier as soon as a node is open in both directions, therefore avoiding *nipping* and *prunning* operations. Once the common line of encounter has been completely defined, the second phase resorts to unidirectional search to guarantee that an optimal solution is found. Therefore, the switch to unidirectional search is delayed until both frontiers have had enough time to clearly define their midline of encounter.

3. A new approach: Two phase bidirectional heuristic search algorithm

This section sketches 2PBS*, a new search algorithm based on two different and consecutive phases. The first phase is bidirectional, while the second one is unidirectional. The algorithm and its related notation are described below. Finally, admissibility is proved for 2PBS*.

3.1. General overview

The key idea of 2PBS* is to improve efficiency exploiting two main ideas: performing bidirectional search until a common line of encounter is defined halfway between both searches, and avoiding overlap between the search frontiers. Unlike previous bidirectional algorithms, 2PBS* explicitly prevents any search to invade the open area of the other one. Therefore, the new algorithm discontinues search at any open node n as soon as it is found in both search trees. Such nodes are removed from the open sets, and added to a common frontier list. Their eventual expansion is postponed to the second (unidirectional) phase.

Bidirectional search (phase 1) terminates as soon as one of the open lists becomes empty. At that point, the common frontier list defines the line of encounter between both searches. A number of (possibly suboptimal) solutions have already been found at the end of phase 1. Let L_{min} be the cost of the best solution found so far.

The second search phase resorts to unidirectional search to guarantee that an optimal solution is returned. The search direction that caused phase 1 to terminate (i.e. the one that run out of open nodes) is selected for unidirectional search. Its open list is fed with those nodes from the common frontier list that could possibly lead to a solution with cost smaller than L_{min}.

Let us consider the search direction that caused the first phase to terminate. If an admissible search was conducted in this direction during the first phase (e.g. like A*), then one of the paths that reaches the common frontier must be part of the optimal solution. The second search phase restarts search from this direction in order to guarantee that an optimal solution is finally returned.

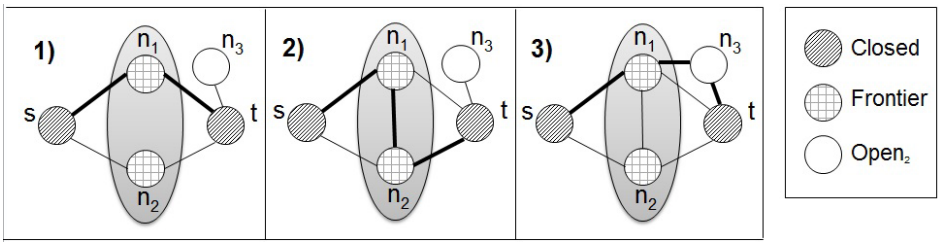

Figure 2. Termination possibilites in 2PBS* phase 2: 1) The solution $s - n_1 - t$ was found in phase 1. 2) Solution $s - n_1 - n_2 - t$ is found after reopening n_1, n_2. 3) Solution $s - n_1 - n_3 - t$ is found through an open node n_3.

There are several scenarios where this optimal path can be found. These can be easily illustrated through a simple example. Let us consider a simple graph like the one shown in Figure 2. Let us assume that search 1 denotes search from s to t, and search 2 from t to s. In the first iteration, search 1 expands node s, adding n_1 and n_2 to its open list. In the second iteration, search 2 expands t. Since $n1$ and $n2$ are already open in search 1, they are removed from both searches and added to the common frontier. On the other hand, n_3 remains open in search 2. The termination condition for the first phase is now satisfied, since search 1 has run out of open nodes. Figure 2, illustrates the three possible conceptual termination scenarios for the second phase of 2PBS*:

1. The optimal solution path is one of those already found, as shown in Figure 2 (1).
2. When the common frontier nodes are reinserted in search 1, an optimal path can be found that goes from s *through the frontier* to t. Figure 2 (2) illustrates the scenario in our example, where the optimal path goes from n_1 to n_2.
3. When the common frontier nodes are reinserted in search 1, an optimal path can be found that goes from s to the frontier, and then to t through an open node of search 2 (see Figure 2 (3)).

3.2. Notation

The following notation is used in the rest of this paper.

s, t	Start node and goal node, respectively.
d	Current search direction index; when search is in forward direction $(s \rightarrow t)$ d=1 and d=2 when in backward $(t \rightarrow s)$.
d'	Index of opposite direction to the current search direction; 3-d.
$successors(n)$	Successors of node n.
$c_d(m, n)$	Positive cost of the arc from m to n if $d = 1$, or from n to m if $d = 2$.
$g_d^*(n)$	Optimal path cost from s to n if $d = 1$, or from n to t if $d = 2$.
$h_d^*(n)$	Optimal path cost from n to t if $d = 1$, or from s to n if $d = 2$.
$f_d^*(n)$	$g_d^*(n) + h_d^*(n)$; Optimal path cost from s to t constrained to contain n.
$g_d(n), h_d(n)$	Estimates of $g_d^*(n)$ and $h_d^*(n)$, respectively.
$f_d(n)$	$g_d(n) + h_d(n)$;
L_{min}	Cost of the least costly complete path found so far linking s to t.
λ	cost of the optimal path from s to t.
$TREE_d$	Search tree used in direction d.

$OPEN_d$	The set of open nodes in $TREE_d$.
$\|OPEN_d\|$	Number of nodes in $OPEN_d$.
$closed_d(n)$	Label to indicate if a node n is closed in search d.
$FRONTIER$	Nodes belonging to $TREE_1$ and $TREE_2$ which will be transferred to the second phase of the algorithm.
$p_d(n)$	Parent of node n in $TREE_d$.
$MeetingN$	Node where $TREE_1$ met $TREE_2$ and yielded the best complete path found so far.

3.3. Pseudocode

TreeUpdate(TREE, OPEN, n2, n, g, f, d)
  if $n2 \notin TREE$ **then** /* new node */
    Add $n2$ to $OPEN$ with a pointer to n.
    $g_d(n2) \leftarrow g$; $f_d(n2) \leftarrow f$; $p_d(n2) \leftarrow n$
  elseif $g < g_d(n2)$ **then** /* better path to n2 */
    $g_d(n2) \leftarrow g$; $f_d(n2) \leftarrow f$; $p_d(n2) \leftarrow n$
  endif
endTreeUpdate

UpdateFrontier (FRONTIER, OPEN', n2, n, g, g2, f, d)
  if $\neg closed_{d'}(n2)$ **then** /* bouncing */
    if $n2 \notin FRONTIER$ **then**
      Add $n2$ to $FRONTIER$
      $g_d(n2) \leftarrow g$; $f_d(n2) \leftarrow f$; $p_d(n2) \leftarrow n$
      if $n2 \in$ OPEN' **then** $\rightarrow$ Remove $n2$ from OPEN'
    elseif $g < g2$ /* better path to the frontier node */
      $g_d(n2) \leftarrow g$; $f_d(n2) \leftarrow f$; $p_d(n2) \leftarrow n$
    endif
  endif
endUpdateFrontier

UpdateLmin (TREE', g_1, g_2, L_{min}, MeetingN, n2)
  if ($n2 \in$ TREE') $\wedge (g_1 + g_2 < L_{min})$ **then**
    $L_{min} \leftarrow g_1 + g_2$
    $MeetingN \leftarrow n2$
    Remove from $OPEN_1$ and $OPEN_2$ those
    nodes with *f-values* $\geq L_{min}$ /* trimming */
  endif
endUpdateLmin

1. **INITIALIZATION**:
   $L_{min} \leftarrow \infty$; $g_1(s) \leftarrow g_2(t) \leftarrow \emptyset$
   $f_1(s) \leftarrow f_2(t) \leftarrow h_1(s)$; $MeetingN \leftarrow nil$
   $OPEN_1 \leftarrow \{s\}$; $OPEN_2 \leftarrow \{t\}$; $FRONTIER \leftarrow \emptyset$

2. **CHECK PHASE 1 TERMINATION**:
   If $(OPEN_1 = \emptyset \vee OPEN_2 = \emptyset)$, then **Go to Search Phase 2**.

3. **CHOOSE SEARCH DIRECTION**: Select the search direction index d according to the cardinality criterion until the first encounter, and then according to the greater f-value criterion in $OPEN_d$.

4. **SEARCH - PHASE 1** (Defining the encounter frontier):
   $n =$ best node from $OPEN_d$
   Remove n to $OPEN_d$ and label as $closed_d$
   foreach $n2 \in$ *successors*(n) **do**
   $g \leftarrow g_d(n) + c_d(n, n2)$; $f \leftarrow g + h_d(n2)$
   if $f < L_{min}$ **then** /* screening */
   if $n2 \in TREE_{d'}$ **then**
   UpdateFrontier $(FRONTIER, OPEN_{d'}, n2, g, g_d(n2))$
   else
   TreeUpdate $(TREE_d, OPEN_d, n2, n, g, f, d)$
   endif
   UpdateLmin $(TREE_{d'}, g_1(n2), g_2(n2), L_{min}, MeetingN, n2)$
   endforeach
   Go to 2 (Check phase 1 termination);

5. **SEARCH - PHASE 2** (Exploring through the shared frontier):
   if $Open_1 = \emptyset$ **then** $d \leftarrow 1$ **else** $d \leftarrow 2$
   foreach $n \in FRONTIER$ **do**
   if $f_d(n) < L_{min}$ **then** Add n to $OPEN_d$
   endforeach
   repeat until $Open_d = \emptyset$
   $n =$ best node from $OPEN_d$
   Remove n to $OPEN_d$ and label as $closed_d$
   foreach $n2 \in$ *successors*(n) **do**
   $g \leftarrow g_d(n) + c_d(n, n2)$; $f \leftarrow g + h_d(n2)$
   if $f < L_{min} \wedge (n2 \notin TREE_{d'} \vee \neg closed_{d'}(n2))$ **then**
   TreeUpdate(TREE, OPEN, n2, n, g, f, d)
   endif
   UpdateLmin $(TREE_{d'}, g_1(n2), g_2(n2), L_{min}, MeetingN, n2)$
   endforeach
   endrepeat
   if $L_{min} = \infty$ **then** no path exists
   else the solution path with cost L_{min} is:
   $(s, ..., p_1(MeetingN), MeetingN, p_2(MeetingN), ..., t)$.
   endif
   end.

3.3.1. Pseudocode description

The pseudocode os 2PBS* is divided into five steps. The first one is an initialization of the variables needed for the search. In particular, $OPEN_1$ is initialized with s, and $OPEN_2$ with t.

Bidirectional search spans steps 2 to 4. Step 2 checks for the termination condition of bidirectional search and, in such case, starts unidirectional search (step 5). Step 3 determines search direction. If no solution has been found yet, the algorithm follows a

cardinality criterion (i.e. the search with the smallest open set is selected). Otherwise, a maximin criterion is followed, i.e. the minimum *f-value* of each open list is calculated, and the search with the largest minimum is selected.

Step 4 describes the core of the bidirectional search phase. The active search direction d selects the best alternative n from its $OPEN_d$ set, removing the node from $OPEN_d$ and labeling it as $closed_d$. Each successor n_2 of n is discarded if its *f-value* is not smaller than (L_{min}). When n_2 belongs to $TREE_{d'}$, it is added to the $FRONTIER$ unless it is labeled as $closed_{d'}$ (this occurs only when path $n_2 \rightarrow n$ was discarded in d'). Supposing that n_2 does not belong to $TREE_{d'}$ the usual A^* search methodology is followed, adding n_2 to $TREE_d$, when n_2 is a new node, or improving its *g-value* in $TREE_d$ if a shorter path has been found. If a new complete path $s \rightarrow t$ is found and the cost of the complete path $(s \rightarrow n_2) + (n_2 \rightarrow t)$ is less than L_{min}, then L_{min} is updated.

Step 5 implements the unidirectional search phase. First, a search direction is selected. This will be the one with empty $OPEN_d$. All nodes n from $FRONTIER$ such that $f_d(n) < L_{min}$ will be re-added to $OPEN_d$. From here on, search is performed in the traditional A^* fashion, pruning unpromising nodes until $OPEN_d$ becomes empty. A solution path of cost L_{min} is returned.

3.4. Properties

A sketch of the formal proof of the admissibility of 2PBS* is presented below. Our first step is to show that 2PBS* will terminate when a path exists.

Lemma 3.1. 2PBS* terminates for finite graphs.
Proof: Follows a similar line of argument as in [13].

Lemma 3.2. Effort-saving techniques applied in 2PBS* do not discard nodes from any optimal path P.
Proof: See [2] pag. 103.

Lemma 3.3. *Bouncing*, presented in function *UpdateFrontier*, ocurrs when a node n_2 is generated in $TREE_d$ and n_2 was already closed in $TREE_{d'}$. Let us consider the node n from which $TREE_d$ accesses n_2. The node n is obviously a successor of n_2 in $TREE_{d'}$ and it had to be generated and eliminated by trimming or screening, but as shown in [2] there is no optimal path through arc $n- > n_2$ and it is safe to eliminate n_2.

Corollary 3.4. Let s_d be s if $d = 1$, otherwise t. Let $s_{d'}$ be t if $d = 1$, otherwise s. Let us assume an optimal path $P = (s_d, ..., n_i, n_j, ...s_{d'})$. When the first phase has terminated and $OPEN_d = \emptyset$, a path from s_d to n_i has been generated in $TREE_d$ and $n_i \in FRONTIER$, as well as path from $s_{d'}$ to n_j has been generated in $TREE_{d'}$ and $n_j \in (FRONTIER \cup OPEN_{d'})$.

Theorem 3.5. 2PBS* will terminate with the optimal path. **Proof:** Since 2PBS* behaves like A* in the second phase and the search reduction techniques which we have already demonstrated never prune a node along the optimal path, 2PBS* will terminate with the optimal path.

	Node expansions	Running time
A*	100	100
BS*	35.74	35.10
Switch-A*	36.57	35.07
2PBS*	33.39	29.42

Table 1. Average performance over 94 instances from the 15-Puzzle problem suite (results relative to A* in %)

	Node expansions	Running time
BS*	100	100
Switch-A*	104.21	104.90
2PBS*	89.58	79.14

Table 2. Average performance over 99 instances from the 15-Puzzle problem suite (results relative to BS* in %)

4. EMPIRICAL EVALUATION

The performance of the new algorithm (2PBS*) has been compared to those of A*, BS*, and Switch-A* over two standard different domains. The first one is the problem suite of 15-puzzle problems introduced by Korf [14] using the Manhattan-distance heuristic. In this domain search fronts meet rather quickly and bidirectional algorithms frequently report advantages over A* search. The second domain involves pathfinding problems in a set of game maps available from the Hierarchical Open Graph (HOG) library[2]. This set comprises 120 maps obtained from the fantasy popular roleplaying game *Baldur's gate II: Shadows of Amn* by *Bioware Inc.* scaled up to 512×512 grids. This is a standard benchmark for discrete state spaces with impassable obstacles. Experiments considered 8-neighborhood grids and the octile-distance heuristic. We took 75 of the 120 maps and considered 93160 problem instances with randomly generated start and goal locations. In this kind of domain bidirectional search frequently performs worse than A*, and search fronts frequently meet at the later stages of search.

The 15-puzzle problem suite was solved on a 2,6GHz AMD Opteron Processor with 64 Gbytes of RAM. Algorithm 2PBS* was able to solve the full 100-problem suite, BS* and Switch-A* solved 99 problem instances, and A* only 94 problem instances with the available memory and our lisp implementation in LispWorks 6.0.1. Table 1 summarizes the average performance of bidirectional search relative to A*, and Table 2 the average performance of Switch-A* and 2PBS* relative to BS*. Figure 3 shows the time performance of the bidirectional search algorithms over the different problem instances, ordered on increasing time for BS*.

Average results of A*, Switch-A*, and 2PBS* against BS* for the path finding problems are displayed in Table 3.

[2]http://code.google.com/p/hog2/

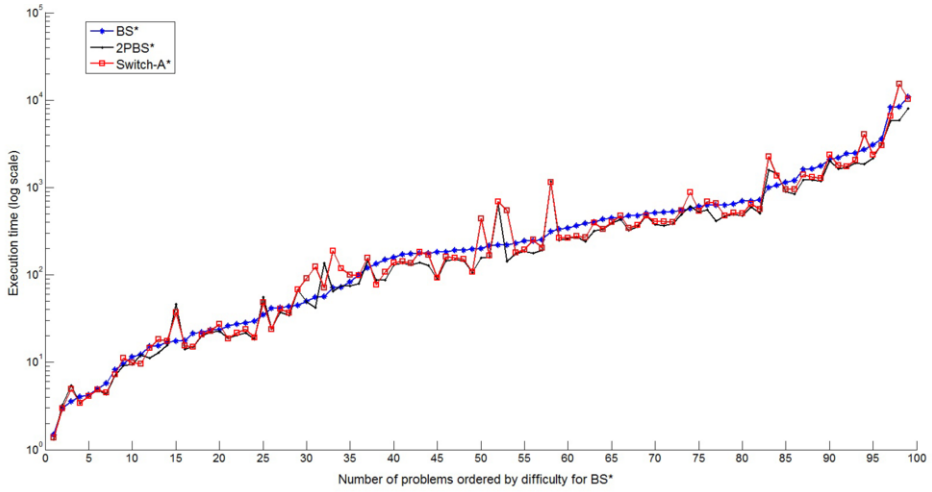

Figure 3. Time requirements for bidirectional search algorithms for the 99 problem instances of the 15-puzzle (ordered by increasing time for BS*).

	Iterations	Running time
BS*	100	100
A*	91.22	71.77
Switch-A*	99.98	100.18
2PBS*	99.99	81.58

Table 3. Average performance on the pathfinding problems (results relative to BS* in %)

5. DISCUSSION

As expected, all bidirectional search algorithms outperform A* in the 15-puzzle problem suite. Two additional important results can be pointed out. In the first place, 2PBS* achieves an improvement of over 20% in time performance over BS*. At the same time, Switch-A* was found to perform worse than BS*, contrary to previous reported results over a limited set of 56 problem instances, which reported a modest improvement of around 6% of Switch-A* over BS* [3].

Figure 3 reveals the heuristic nature of the different effort-saving techniques of bidirectional search algorithms. The vertical axis is shown in logarithmic scale in this figure. While many problems are either difficult or easy for all algorithms, occasionally a given problem can be oddly easy or difficult for a given algorithm when compared to the others. At the same time, it is obvious that good or bad performance on the most difficult problems can have an important impact on average results.

Figure 4 provides a deeper insight on the results obtained by BS*, Switch-A* and 2PBS*. This shows the time performance of the algorithms Switch-A* and BS* relative to 2PBS*, averaged over an increasing number of problem instances from the complete problem suite (from just 1 up to 99 problems, as indicated in the horizontal axis). Problems were ordered by difficulty for 2PBS* choosing first the easier problems. This figure reveals how reports on sets of the easier problems can influence average results, depending on the relative performance of BS* over these same difficult problems. The

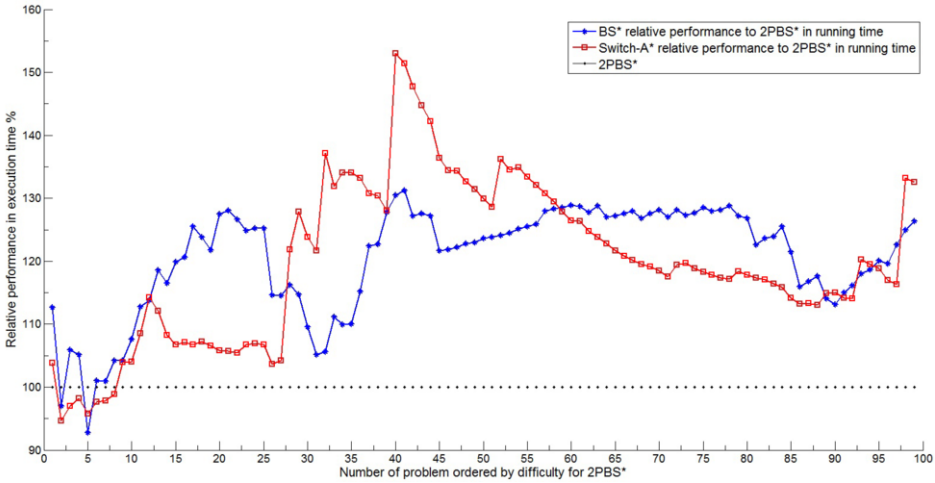

Figure 4. Performance of Switch-A* and BS* to 2PBS* averaged for an increasing number of 15-puzzle problem instances (ordered by difficulty for 2PBS*).

availability of large amounts of memory makes it possible for the first time to provide more precise comparisons over almost the entire problem set (99 problem instances).

Regarding the path finding problem set, the performance of the algorithms (in explored nodes and time) was found to depend largely on the tie-breaking rule. In this domain, many states achieve the same $f(n)$ value in the open sets at the same time. It is generally acknowledged that breaking ties in favor of nodes with larger $g(n)$ value is better for A*. The algorithms were run with this policy, as well as with an arbitrary order of tie-breaking. The results were that BS* and A* perform 3.86% and 11.34% worse with the arbitrary order respectively. However, Switch-A* and 2PBS* perform 2.8% and 20.56% better with the arbitrary order respectively. The results shown in Table 3 display the best results for each algorithm.

The performance of BS* is (as could be expected) clearly worse than that of A*. A* performs less expansions, and these seem to be performed even more efficiently. In this domain Switch-A* performs slightly worse than BS*. The impact of switching from bidirectional to unidirectional search is smaller in this domain, since the first solution is typically found after 84.5% of the running time. The best time performance among bidirectional search algorithms is clearly achieved by 2PBS*, though still worse than A*.

6. CONCLUSIONS AND FUTURE WORK

This paper introduces 2PBS*, a new best-first bidirectional heuristic search algorithm. The algorithm is guided by the general principles of letting search effort to be equally distributed between both fronts, and preventing these from overlapping. This is achieved using a two-phase scheme that avoids classical operations like nipping and prunning. The first phase exploits bidirectional search, and allows search fronts to define a common line of encounter. The second phase turns to unidirectional search in order to guarantee an optimal solution is returned.

The new approach is evaluated using a domain where bidirectional search has traditionally obtained good results (15-puzzle) and another where results are generally not competitive over unidirectional search (path finding). Results show that the new algorithm outperforms previous best-first front-to-end bidirectional search algorithms (BS* and Switch-A*). The experimental evaluation also reveals that partial evaluations over the standard 15-puzzle problem set can introduce important bias in the results. Also, the choice of an adequate tie-breaking policy in path-finding can have a significant impact in performnace results.

This paper has focused on a novel application of effort-saving techniques in bidirectional search. A different avenue of research focuses on the opportunity offered by these algorithms to dynamically improve the values of the heuristic evaluation function $f(n) = g(n) + h(n)$ used in each of the search fronts [4]. This can be done exploiting the difference between the values of the heuristic estimates $h(n)$ in one direction, and the actual cost values $g(n)$ of paths explored in the opposite direction. The general idea has been applied to improve the performance of different bidirectional search algorithms, like Max-switch-A* [3], Max-BS* or BiMax-BS$^*_F$. [5]. The evaluation of this general technique in 2PBS* is an interesting subject for future work.

References

[1] I. Pohl, Bi-directional search, Machine Intelligence 6, 127-140, (1969).

[2] James B.H. Kwa, BS*: An admissible bidirectional staged heuristic search algorithm, Artif. Intell., 38, 95-109, (February 1989).

[3] Hermann Kaindl, Gerhard Kainz, Roland Steiner, Andreas Auer, and Klaus Radda, Switching from bidirectional to unidirectional search, in Proceedings of the 16th international joint conference on Artificial Intelligence - Volume 2, pp. 1178-1183, San Francisco, CA, USA, (1999). Morgan Kaufmann Publishers Inc.

[4] Hermann Kaindl and Gerhard Kainz, Bidirectional heuristic search re-considered, J. Artif. Int. Res., 7, 283-317, (December 1997).

[5] Andreas Auer and Hermann Kaindl, A Case Study of Revisiting Best-First vs. Depth-First Search, in ECAI, eds., Ramon Lopez de Mantaras and Lorenza Saitta, pp. 141-145. IOS Press, (2004).

[6] Dennis de Champeaux and Lenie Sint, An Improved Bidirectional Heuristic Search Algorithm., J. ACM, 177-191, (1977).

[7] Dennis de Champeaux, Bidirectional Heuristic Search Again, J. ACM, 22-32, (1983).

[8] George Politowski and Ira Pohl, D-Node Retargeting in Bidirectional Heuristic Search., in AAAI–84, pp. 274-277, (1984).

[9] John F. Dillenburg and Peter C. Nelson, Perimeter Search, Artif. Intell, 65(1), 165-178, (1994).

[10] Giovanni Manzini, Artificial Intelligence BIDA*: an improved perimeter search algorithm, Artificial Intelligence, 75, (1995).

[11] Ariel Felner, Carsten Moldenhauer, Nathan R. Sturtevant, and Jonathan Schaeffer, Single-Frontier Bidirectional Search, in AAAI, (2010).

[12] Carsten Moldenhauer, Ariel Felner, Nathan R. Sturtevant, and Jonathan Schaeffer, Single-Frontier Bidirectional Search., in SOCS10, pp. -1–1, (2010).

[13] Judea Pearl, Heuristics - intelligent search strategies for computer problem solving, Addison-Wesley series in artificial intelligence, Addison-Wesley, 1984.

[14] Richard E. Korf, Depth-First Iterative-Deepening: An Optimal Admissible Tree Search, Artificial Intelligence, 27, 97-109, (1985).