INDIAN INSTITUTE OF MANAGEMENT CALCUTTA

WORKING PAPER SERIES

WPS No. 678/ August 2011

A Tale of Two Searches: Bidirectional Search Algorithm that Meets in the Middle

by

Ambuj Mahanti
Professor, Indian Institute of Management Calcutta, Joka, Kolkata 700104

Samir K. Sadhukhan
Senior System Analyst, Indian Institute of Management Calcutta, Joka, Kolkata 700104

&

Supriyo Ghosh
Infosys Ltd. Manikonda Village, Lingampally Ranga Reddy District, Hyderabad 500 032

A Tale of Two Searches:

Bidirectional Search Algorithm that Meets in the Middle

Ambuj Mahanti
Management Information System Group
Indian Institute of Management Calcutta
Joka, D. H. Road, KolKata-700104

Samir K. Sadhukhan
Computer Centre
Indian Institute of Management Calcutta
Joka, D. H. Road, KolKata-700104

Supriyo Ghosh
Infosys Ltd.
Manikonda Village, Lingampally
Ranga Reddy District, Hyderabad 500 032.

A Tale of Two Searches:

Bidirectional Search Algorithm that Meets in the Middle

List of notation

G	Implicit Graph
s	Start node
t	Goal node
m, n, p, q, r, n_1, n_2…	Nodes in G
d	Direction of the current search; d=1 implies forward search from s to t, d=2 implies backward search from t to s
(m,n)	Directed arc from node m to node n in G
c(m,n)	Cost of arc (m,n) = the cost of reverse arc (n,m)
δ	Small positive number
$g_d^*(n)$	Cost of a minimal cost path from s to n if d=1, or from n to t if d=2
$h_d^*(n)$	Cost of a minimal cost path from n to t if d=1, or from s to n if d=2
$h^*(s)$	Cost of a minimal cost solution path in G
$g_d(n)$	Estimate of $g_d^*(n)$
$h_d(n)$	Estimate of $h_d^*(n)$
Γ(n)	Operators at node n
P	Directed path
c(P,m,n)	Cost of a directed path P from node m to node n
$FE_d(P,n)$	Forward Error on a path P from s to n if d=1, or from n to t if d=2

$BE_d(P,n)$	Backward Error on a path P from s to n if d=1, or from n to t if d=2
$TE_d(P,n)$	Total Error on a path P from s to n if d=1, or from n to t if d=2; equals $FE_d(P,n) + BE_d(P,n)$
$FE_d^*(n)$ d=2	$FE_d(P,n)$ when P is a minimal-cost path from s to n if d=1, or from n to t if
$BE_d^*(n)$ d=2	$BE_d(P,n)$ when P is a minimal-cost path from s to n if d=1, or from n to t if
$TE_d^*(n)$	$TE_d(P,n)$ when P is a minimal-cost path from s to n if d=1, or from n to t if d=2; equals $FE_d^*(n) + BE_d^*(n)$
$FE_d(n)$	Estimate of $FE_d^*(n)$
$BE_d(n)$	Estimate of $BE_d^*(n)$
$TE_d(n)$	Estimate of $TE_d^*(n)$

1. Introduction

The Sliding Tiles problem (also known as n-puzzle) is quite popular as a problem testbed in AI. This problem is basically a game on a square grid, containing several tiles numbered consecutively from 1 to n and one blank space, such that n+1 is a square number. The blank space allows the movement of an adjacent tile to the blank position, thereby swapping the positions of the blank space and the tile. The objective of the n-puzzle is to arrive at a desired tile configuration (the end or goal state) starting from a given configuration (the start state), in a minimum number of moves.

Minimization problems in discrete domains such as the n-puzzle are often modeled as directed graphs. Each graph G contains a specified start node (s), a specified goal node (t), and a cost function mapping each arc <m,n> into a nonnegative cost c(m,n). Modeled this way, the objective function reduces to finding a least-costly path from s to t. Occasionally, a non-negative heuristic function h(n) is defined on the nodes of the graph, with h(n) being an estimate of h*(n), the cost of a minimal-cost path from n to t. Used appropriately, the heuristic can cut down the combinatorial explosion of the search space and guide the search along better (more promising) solution paths. Heuristic estimates are used in the evaluation functions of classical AI search algorithms such as A* (1), IDA* (2), etc.

Algorithm A* finds a minimal-cost path from s to t by searching the graph in a best-first manner. It creates two lists, one for the nodes which are yet to be expanded (the OPEN list) and another for nodes which have already been expanded (the CLOSED list). For each node *n* in OPEN and CLOSED, A* maintains three values: g(n) which is the cost of a currently known minimal cost path from s to n; h(n), which is an estimate of a minimal-cost path from n to t; and the total path-cost f(n) = g(n) + h(n) which is the estimate of cost of minimal-cost path from s to t constrained to go through n. Initially, OPEN is set to {s} with g(s) =0 (i.e f(s) = h(s)), while CLOSED is set to Null. In each iteration, A* removes from OPEN a least-costly node n (giving priority to a goal node), expands it if it not a goal node and generates all its children, say n_1, n_2, .. n_k. While the expanded node n is sent to CLOSED, each of the children n_i are evaluated using a cost function $f(n_i) = g(n_i) + h(n_i)$, where $g(n_i)$ is the cost of reaching n_i via n, i.e. $g(n_i) = g(n) + c(n,n_i)$ (recall that $c(n,n_i)$ is the cost of the arc (n,n_i)). After evaluating each n_i, if it is found that the child did not exist in OPEN or CLOSED earlier (or existed with a higher g-value), it is put in OPEN; in the latter case, its g-value is updated to the lower g-value available now. A* terminates when the goal node is selected from OPEN, outputting f(t) as the solution cost. It has been well-established in the literature that under the assumption of

admissible heuristics (h(n) <= h*(n) for all nodes n in G), A* terminates with an optimal solution, h*(s).

In domains such as the n-puzzle problem, several heuristics have been proposed, each capturing the domain information to a different degree. Some popular heuristics are: number of tiles out of place, the Manhattan distance, etc.

A* is an admissible search algorithm, but it takes up too much memory (due to the maintenance of OPEN and CLOSED lists) which can be prohibitive in many domains such as 15 and 24-puzzle. To overcome this memory requirement of A* while also outputting an optimal solution, algorithm IDA* has been proposed (2). IDA* works iteratively, each iteration being a depth-first search starting from node s. In each iteration, a branch of the search tree is cut off when its total cost (f = g+h) exceeds a particular threshold. Initially the threshold is set to the total cost estimate of start node s, h(s). Then in each iteration IDA* sets the threshold for the next iteration equal to the minimum of all node costs which exceeded the current threshold. Operating iteratively in this depth-first manner, IDA* outputs the optimal solution cost if the heuristic function is admissible. Note that the memory requirement is vastly reduced when compared to A*, as in each iteration IDA* needs to maintain only one path from s to the current node n.

IDA* has been extensively studied in the heuristic search literature and it has been found efficient to solve problems such as the 15-puzzle, for which the first successful execution of an algorithm was reported in (2).

However, on larger instances of the n-puzzle, such as the 24-puzzle, neither A* nor IDA* have been successful – A* due to its larger memory requirement, and IDA* due to its longer time of processing. For such domains the Bidirectional Search is supposed to be more successful. We briefly introduce this variant below. The rest of the paper will contain detailed description of different Bidirectional search approaches, as well as our version of the same.

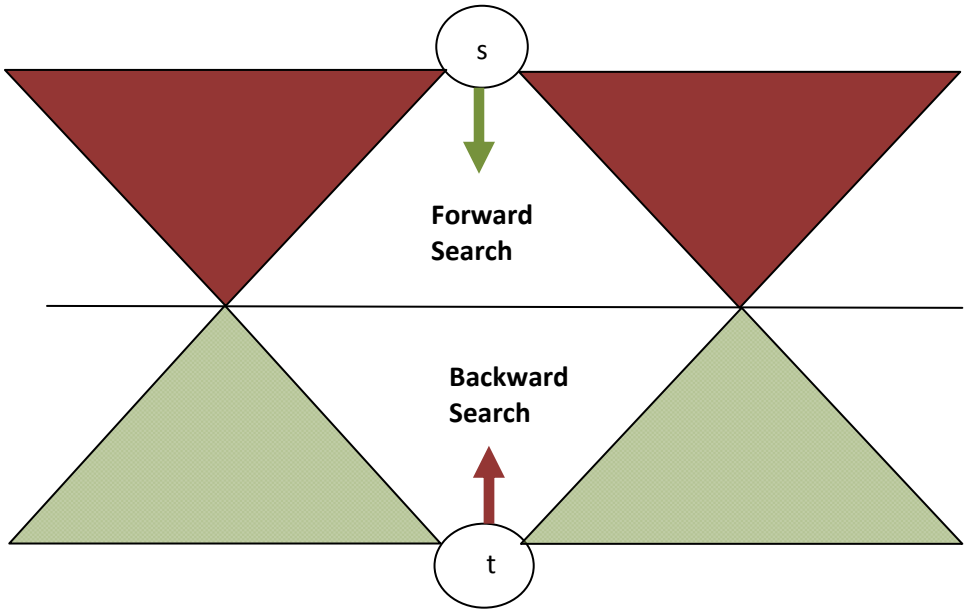

Figure 1. The concept of Bidirectional Search. Ideally, the two search frontiers (Forward and Backward) should meet around the line in the center, thereby eliminating the search spaces shown in color.

The concept of bidirectional search is relatively simple. A bidirectional search consists of two searches, one from *s* to *t* and the other from *t* to *s*. The two searches are conducted alternately and when they meet under some suitable condition, they together would have found a minimal-cost path from s to t (see Figure 1.)

The idea behind bidirectional search is that, an A*-like search essentially unfolds in the form of a tree, and two half-trees, taken together, may be smaller than a full tree - thereby resulting in reduced space and time requirement of the algorithm. Although the premise of bidirectional search is quite intuitive, it suffers from the problem that the two searches may not meet in the middle, but at either end of the graph (near s or near t.) In that situation, instead of having two half-trees which reduce the time-space complexity, we will have two nearly full trees thereby increasing the time and space requirement. This problem which is also known as the “Missile Metaphor”, has vexed researchers for the last few decades.

In this paper, we take a closer look at bidirectional search. We start with a brief survey of past approaches to the bidirectional search – particularly algorithms such as BHPA, BS*, BIDA* etc. Then we develop a more efficient bidirectional algorithm by exploiting particular search characteristics. This is done by developing an "error-function" on the search path as a surrogate of the evaluation function f. By defining the error function suitably, we show how it helps to control the search more tightly and converges the Forward and Backward searches always in the middle. Our theoretical results prove the admissibility and complexity of the algorithm. Traces of the algorithm on n-puzzle illustrate how it works.

This paper is organized as follows: The problem definition is presented in section 2. Section 3 contains the literature survey. In section 4 we present our new bidirectional search algorithm called MSG* (named after Mahanti Sadhukhan and Ghosh). In section 5 we present a new approach to heuristic search using a novel technique of error minimization. Section 6 contains the experimental results. We conclude the paper in section 7.

2. Problem Definition

Now we introduce some of the basic concepts and notation, which will be used throughout the paper.

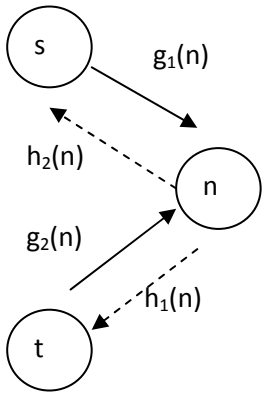

Figure 2. Search Direction and notations

Figure 2 shows the search directions: d=1 indicates search in Forward direction (from s to t) and d=2 indicates search in Backward direction (from t to s). Then $g_1(n)$ and $h_1(n)$ which refer to the Forward search (d=1) have the usual connotations as g(n) and h(n). $g_2(n)$ and h2(n) pertain to the Backward search (d=2) and have analogous connotations in a backward sense: $g_2(n)$ is the

currently-known least cost of a path from t to n, while $h_2(n)$ is the heuristic estimate of a path from n from t. In the figure, dotted arrows indicate heuristics for unexplored paths in either direction.

Admissible heuristics: The heuristic function is said to be admissible, if for both d=1 and d=2,

$$h_d(n) \le h_d^*(n) \forall n \in G$$

we have:

Clearly, we have, for d = 1 or d = 2:

$$h_d^*(n) = g_{3-d}^*(n) \le g_{3-d}(n)$$

Clearly, we have $g_d(n) >= g_d^*(n)$ and $h_d^*(n) = g_{3-d}^*(n)$, where d can be 1 (for forward search) or 2 (for backward search).

Assumptions

1. All operators are reversible. The operator in forward direction (i.e. the arc given in the implicit graph) is used for searching in the forward direction only. The reversed operators are used in the backward search only. (Note: Whenever we speak of a directed path in the search process, it means a path consisting either of arcs or of reverse arcs, but never a combination of the two.) A reverse arc or reverse operator has the same cost as that of the corresponding arc or operator. We denote this common arc-cost with only single link between nodes m and n as c(m,n), where $c(m,n) \ge \delta > 0$.
2. The graph must contain exactly one goal node, denoted by t.
3. The goal node must be explicitly specified.
4. There is a path from the start node s to the goal node t with finite cost.
5. We place a mild restriction on the heuristic distribution, as follows: $h_1(s) = h_2(t)$. This assumption, which is quite realistic, is critical to prove the theoretical properties of our algorithm.

3. Literature Survey

Most bi-directional search algorithms contain two sets of *OPEN* and *CLOSED* nodes: $OPEN_F$ and $CLOSED_F$ for search in the *Forward* direction, and $OPEN_B$ and $CLOSED_B$ for search in the *Backward* direction. The algorithm starts with the *Forward* direction, putting s in $CLOSED_F$, its successors in $OPEN_F$ and computing their heuristic values and evaluation functions. After the first *Forward* iteration, it does the first *Backward* iteration, using *t*, $OPEN_B$ and $CLOSED_B$, and proceeding in a reverse-A* like manner, generating parent nodes instead of child nodes from the graph. In the iterations that follow, the search alternates between *Forward* and *Backward* directions; however the alternation is not strict. A common strategy employed by most bidirectional search algorithms is to search in the direction which has the smaller size of *OPEN*; thus if the current search direction is *Forward* and $|OPEN_F| < |OPEN_B|$ at the end of this iteration, then the next search direction will still be *Forward* – it will not reverse. Similar rule holds for the *Backward* pass. Ensuring that we always search from the direction of smaller *OPEN* seems quite intuitive and may help reduce the width of the search graph.

As discussed in (3), there are mainly two types of bidirectional searches: where the evaluation function measures the distance from the selected node to a goal node (known as front-to-end evaluation), and where the distance is measured from the selected node to any node in the opposite OPEN (known as front-to-front evaluation.)

Front-to-end Evaluations

Front-to-end evaluation was employed in the first bidirectional search algorithm, *BHPA* (4), through two functions, f_F and f_B (for forward and backward passes, respectively). Per their definition, $f_F = g_F + h_F$, where g_F and h_F are same as *g* and *h* in A* (1), and $f_B = g_B + h_B$, where $g_B(n)$ is the currently-known shortest distance *from t to n*, and $h_B(n)$ is an estimate of the shortest distance *from n to s*. The *termination condition* of *BHPA* is dependent on a_{min}, the currently-known cost of a complete path from *s* to *t*. Initially a_{min} is set to *infinity*, indicating that no *s-t* path has been found yet; then in each iteration, if the selection of a node *n* from OPEN results in the discovery of a complete *s-t* path through *n* (indicated by $n \in CLOSED_F \cap CLOSED_B$), a_{min} is set to the new pathcost $g_F(n) + g_B(n)$ if the new pathcost is less than the existing a_{min} *value*. Doing so in every iteration ensures that a_{min} is set to the minimum cost of all complete *s-t* paths found so far. Further, if

$$a_{min} <= \max \{min_{x \in OPEN_F} (f_F(x), min_{y \in OPEN_B} (f_B(y)\}$$

then the nodes in $OPEN_F$ and $OPEN_B$ cannot further improve (reduce) the value of a_{min}, implying the current a_{min} is the cost of an *optimal s-t* path, and BHPA terminates outputting a_{min} as

solution cost. The admissibility of algorithm *BHPA* under a monotone heuristic is proved in Pohl's paper. Although admissible, Pohl has pointed out that the *Forward* and *Backward* searches may meet only near *t* (or *s)*, thus making the bidirectional search no more efficient than unidirectional. This has resulted in the quest for a bidirectional search algorithm where the two search frontiers meet mostly in the middle.

However, it has been pointed out in (3) that the problem of two search frontiers not meeting each other, as originally mentioned by Pohl, is a misconception; the actual problem is that the two frontiers pass through each other before a complete path is detected. This issue has been tackled in algorithm *BS** (5) via a set of techniques, such as *trimming (*removing nodes with $f >= a_{min}$*)*, *screening (*inserting new nodes in *OPEN* only if $f < a_{min}$*)*, and *nipping* and *pruning* (putting the selected node into CLOSED *without* expansion, if it is already CLOSED in the opposite search tree, and also removing its OPEN children from the opposite search tree.) With these optimizations, and using front-to-end evaluation functions as in BHPA, BS* is admissible under the monotone restriction, and has some performance improvement compared to BHPA (though compared with A* it is at par or even at a slight disadvantage.)

Front-to-front Evaluations

Sensing the difficulty of meeting in the middle, algorithm BHFFA was presented (6) using a pair of more powerful heuristics, $h_{F,}$ and h_B which use *front-to-front* evaluation. To compute h_F , BHFFA first estimates the cost from node *n* to *t* in two parts*:* cost of reaching from *n* to some node in $OPEN_B$ (an estimate), plus the cost from the same node in $OPEN_B$ to *t*. h_F is then set to the minimum of all such sums, i.e. $h_F(n) = min_{y \in OPEN_B}\{h(n,y) + g_B(y)\}$, where $h(x,y)$is a nonnegative estimator of the distance between *x* and *y* with $h(x,y) = h(y,x)$. In the Backward search, h_B is defined similarly with respect to *s* and $OPEN_F$. Algorithm *BHFAA* terminates when the node selected from one OPEN list also belongs to the other OPEN.

Even though BHFFA was originally claimed to be admissible, in a subsequent paper (7) De Champeaux clarified the claim to be wrong; to overcome this issue, an updated algorithm BHFFA2 was presented and proved to be admissible. BHFFA2 uses two loops, the first one being a "Find a Path" loop to find any {s-t} path and thereby determine an upper bound on the solution cost. As soon as an {s-t} path is found, control switches to the other loop, "Find Best Path" loop, and stays there till an optimal path is found.

BHFFA and BHFFA2 are known as wave-shaping algorithms as they try to force the opposing wavefronts to converge in the middle of the search space. The issue in front-to-front algorithms is that the time taken to compute heuristic functions is too high, since now we are not

computing the heuristic just with respect to a goal, but with respect to the entire OPEN set in the opposite tree.

An intermediate method of bidirectional search, known as perimeter search, has been proposed in (8). In this method, first a one-time breadth-first search around the goal node is conducted, creating a perimeter *P* around the goal node to a pre-determined fixed *depth d*[1]. Then the algorithm proceeds unidirectionally from *s* in search of this perimeter, using the heuristic function $h_p(n) = min_{m \in P}[h(n,m) + h^*(m,t)]$, where $h(n,m)$ is as above and $h^*(m,t)$ is the minimum cost of a path from *m* to *t*. The forward search from *s* can be conducted either like A*, when the algorithm is called PS*, or like IDA* (2), when it is called IDPS*.

There is an interesting variant of perimeter search, which uses IDA* and hence known as BIDA* (9). BIDA* is more efficient compared to IDPS* on Fifteen Puzzle and works as follows. After the perimeter P has been generated, it starts IDA* from s, using the threshold value at each iteration to filter the nodes in the perimeter *P*; only those nodes in *P* having f-values less than or equal to the current threshold *T* are retained in a "virtual" perimeter $P_d(n,T)$ (*n* being the currently expanded node.) As IDA* proceeds from a node *n* to its successor $\hat{n}$ along a path, the virtual perimeter gradually shrinks ($P_d(\hat{n},T) \subseteq P_d(n,T)$), eventually becoming null, and is used to decide the cutoff of that path instead of checking $f(n) > T$. This technique obviates the need to compute f-values repeatedly against each node in the perimeter *P*, and places BIDA* at a considerable performance advantage over IDPS* and other algorithms. However as reported in experimental studies, the advantage is domain-dependent; it has been reported that BIDA* has strong advantage on Fifteen Puzzle relative to IDA*((9), (3)), but not on the Maze Problem relative to A* (3).

In yet another variation of Perimeter Search, a best-first search (A*) is conducted from one end and a linear-space search (IDA*) from the other, giving the name of the algorithm BAI* (3). The A* search creates the perimeter around the goal, and the IDA* then searches for this perimeter instead of the goal. An advantage of this approach is that IDA* can utilize the higher threshold generated by A*, thereby terminating in fewer iterations. This is the approach followed if memory is limited; however if memory is sufficient, then they conduct A* from the other end too (instead of IDA*), leading to the algorithm being named BAA. BAA has further variants of its own, based on the incorporation of several "error" functions – such as

[1] A variation of this algorithm creates the perimeter as a pre-determined fixed *cost* to the goal using a reverse A* search from *t*. As mentioned in (8), this method is no more efficient than the fixed depth method.

$Mindiff_1 = Min_{n \in CLOSED_B}\{g_B(n) - h_F(n)\}$, which is just a constant error value to be added to h_F during Forward Search (algorithm Add-BAA).

Other approaches to bidirectional search include choosing a representative node from each front as a target for the opposite search (d-node retargeting, (10)); conducing bidirectional search till the search trees meet for the first time and then switching to unidirectional mode in the direction which has higher minimum *f*-value in OPEN (11); and an iterative-deepening approach to BHFFA (12).

Other notable work related to bidirectional search include an algorithm employs a much more efficient front-to-front evaluation method but is inadmissible (13), and the Divide-and Conquer Bidirectional Search (14) which reduces the space complexity of the algorithm by storing only the OPEN lists and not the CLOSED ones.

4. Algorithm Meet-At-The-Middle – Dual Threshold MSG*

(As implemented in program)

Global variables: next_backward_threshold, next_forward_threshold, solution_found

Procedure Main()

Variable: threshold

1. next_backward_threshold = $h_2(t)$;
   next_forward_threshold = $h_1(s)$;
   solution_found=0;

```
while (!solution_found)
 {
   threshold = next_backward_threshold;
   next_backward_threshold = create_backward_frontier(threshold);


   if (!solution_found) {

     threshold = next_forward_threshold;

      next_forward_threshold = forward_search(threshold);
    }
 }
```

Procedure create_backward_frontier($Thresh_B$)

Do a dfs under $Thresh_B$:

If *s* is encountered, set *solution_found* = 1 and return;

Else create $OPEN_B$ as a hash table with every distinct node *n,* such that *n* is the first node on a path from *t* to *n* with $f(n) > Thresh_B$;

Eliminate duplicate nodes by keeping the one with least path cost. Calculate *f(n)* as $c(P,s,n) + h_1(n) + c(P,s,n) – h_2(n)$

Set *next_backward_threshold = min {f(n) | n ε* $OPEN_B$*}*.

Return next_backward_threshold.

Procedure forward_search($Thresh_f$)

Set next_forward_threshold = Infinity.

For every path P from s, do a dfs with threshold $Thresh_f$. (If Goal node is detected, exit with solution cost.) Let n be the first node on P such that $f(n) > Thresh_f$.

If q be the parent of n on P:

if q ε $OPEN_B$

Set solution_found = 1; exit with $g_1(q) + g_2(q)$ as the solution cost ;

else

set next_forward_threshold = min {f(n), next_forward_threshold}.

5. A New Approach to Heuristic Search using Error Minimization

$FE_1(P,n) = h_1(n) + c(P,s,n) – h_1(s)$

$BE_1(P,n) = c(P,s,n) – h_2(n)$

$FE_2(P,n) = c(P,t,n) + h_2(n) – h_2(t)$

$BE_2(P,n) = c(P,t,n) – h_1(n)$

$TE_1(P,n) = FE_1(P,n) + BE_1(P,n)$

$TE_2(P,n) = FE_2(P,n) + TE_2(P,n)$

Theorem-1: Let P_1 and P_2 be two solution paths in G. Let n_1 be a node on P_1 and n_2 be a node on P_2. Then $TE_1(P_1,n_1) + TE_2(P_1,n_1) > TE_1(P_2,n_2) + TE_2(P_2,n_2)$ iff $c(P_1,s,t) > c(P_2,s,t)$.

Proof: Let us assume that $c(P_1,s,t) > c(P_2,s,t)$. Now, by substituting $TE_1(P,n)$ and $TE_2(P,n)$ by their values, we get

$TE_1(P_1,n_1) + TE_2(P_1,n_1)$

$= FE_1(P_1,n_1) + BE_1(P_1,n_1) + FE_2(P_1,n_1) + BE_2(P_1,n_1)$

$= h_1(n_1) - \{h_1(s) - c(P_1,s,n_1)\} + c(P_1,s,n_1) - h_2(n_1)$

$+ h_2(n_1) - \{h_2(t) - c(P_1,t,n_1)\} + c(P_1,t,n_1) - h_1(n_1)$

$= 2\ \{c(P_1,s,n_1) + c(P_1,t,n_1)\} - \{h_1(s) + h_2(t)\}$

$= 2\ c(P_1,s,t) - 2\ h_1(s)$ ---------- (1)

Similarly,

$TE_1(P_2,n_2) + TE_2(P_2,n_2)$

$= FE_1(P_2,n_2) + BE_1(P_2,n_2) + FE_2(P_2,n_2) + BE_2(P_2,n_2)$

$= h_1(n_2) - \{h_1(s) - c(P_2,s,n_2)\} + c(P_2,s,n_2) - h_2(n_2)$

$+ h_2(n_2) - \{h_2(t) - c(P_2,t,n_2)\} + c(P_2,t,n_2) - h_1(n_2)$

$= 2\ \{c(P_2,s,n_2) + c(P_2,t,n_2)\} - \{h_1(s) + h_2(t)\}$

$= 2\ c(P_2,s,t) - 2\ h_1(s)$ ---------- (2)

Thus (1) – (2) yields

$TE_1(P_1,n_1) + TE_2(P_1,n_1) - TE_1(P_2,n_2) - TE_2(P_2,n_2)$

$= 2\ \{ c(P_1,s,t) - c(P_2, s,t)\}$

> 0, by assumption.

Conversely, let us assume that

$TE_1(P_1,n_1) + TE_2(P_1,n_1) > TE_1(P_2,n_2) + TE_2(P_2,n_2)$

i.e. $2\ \{c(P_1,s,n_1) + c(P_1,t,n_1)\} - 2\ h_1(s) > 2 * \{c(P_2,s,n_2) + c(P_2,t,n_2)\} - 2\ h_1(s)$

i.e. $c(P_1,s,n_1) + c(P_1,t,n_1) > c(P_2,s,n_2) + c(P_2,t,n_2)$

i.e. $c(P_1,s,t) > c(P_2,s,t)$.

□

Theorem-2: Let P be a solution path in G and n_1 and n_2 be two nodes on P such that n_2 is the child of n_1 on P. Then, if the heuristic function h_d is monotone, $TE_1(P,n_2) \geq TE_1(P,n_1)$.

Proof: $TE_1(P,n_2) - TE_1(P,n_1)$

$= c(P,s,n_2) + h_1(n_2) - h_1(s) + c(P,s,n_2) - h_2(n_2)$

$- c(P,s,n_1) - h_1(n_1) + h_1(s) - c(P,s,n_1) + h_2(n_1)$

$= 2\ \{c(P,s,n_2) - c(P,s,n_1)\} + \{h_1(n_2) - h_2(n_1)\} + \{h_2(n_1) - h_2(n_2)\}$

$= 2\ c(n_1,n_2) + \{h_1(n_2) - h_2(n_1)\} + \{h_2(n_1) - h_2(n_2)\}$

$= \{h_1(n_2) - h_1(n_1) + c(n_1,n_2)\} + \{h_2(n_1) - h_2(n_2) + c(n_1,n_2)\}$

$= \geq 0 + \geq 0$, as the heuristic is monotone.

Hence $TE_1(P,n_2) \geq TE_1(P,n_1)$.

□

Theorem-3a: Let P be a path below s in G. Let n_1 and n_2 be two nodes on P such that n_2 is a child of n_1 on P. Then, if the heuristic function h_d is monotone, TE1(P,n_2) >= TE1(P,n_1).

Proof: Clear.

□

Theorem-3b: Let P be a path below t in G. Let n_1 and n_2 be two nodes on P such that n_1 is a child of n_2 on P. Then, if the heuristic function h_d is monotone, $TE_2(P, n_1) >= TE_2 (P,n_2)$.

Proof: Clear.

□

Theorem-4: At any instant prior to the termination of the algorithm MSG*, during any forward and backward search, if n is the node selected for expansion,

$$TE_d (n) <= TE^*_d(n) <= O_{min\ d}$$

Proof: Clear.

□

Assumptions:

1) h(n,m) = h(m,n). Thus h(s,t) = h(t,s) or h1(s) = h2(t)
2) c(n,m) = c(m,n)

(1) and (2) imply # of forward thresholds = # of backward thresholds

During Forward Search, let P be one optimal path. Let ni1, ni2,…,nik be all the nodes on P where h1(nij), I <=j <= k, increases in value (instead of decreasing.) Clearly then, along path P, these k nodes define the forward error thresholds. Because of (1) and (2), similarly there will be k backward error thresholds along path P. If these forward and backward error thresholds are non-overlapping, then in total there will be 2K thresholds. In general, along any optimal path P, there will be total T thresholds, where K <= T <= 2K.

Theorem-5: MSG* will meet at middle and find optimal solution for both T even and T odd.

Proof: Omitted.

□

Assumptions:

(1)Heuristic is monotone

(2) Algorithm MSG* works in alternate directions in successive iterations

Lemma-1: At any instant prior to the termination of MSG*, both $OPEN_F$ and $OPEN_B$ contain the forward and backward leading nodes from any optimal path P.

Proof: Let P be an optimal path. Initially s and t from P will belong to $OPEN_F$ and $OPEN_B$ respectively. We may assume that the statement is true up to iteration i, i may be an iteration in either forward or backward direction. We will show that the lemma is true for iteration i+1. Also at iteration i, we call the leading node of optimal path as n_i*.

Case I: In iteration i, n_i* is not expanded.

Then n_{i+1}* = n_i*, meaning the same node remains as leading node of P in the next iteration in that particular direction (F or B).

Case II: n_i* is expanded at iteration i.

Let n_{i+1}* be the child of n_i* on P. Then:

Case II(a): n_{i+1}* is a new node. Then it becomes a leading node at iteration i+1.

Case II(b): n_{i+1}* belongs to CLOSED. Proof that n_{i+1}* belongs to another optimal path P'.

Then consider leading node of P' which is at OPEN at instant i. This becomes leading node of P at instant i+1.

Case II(c): n_{i+1}* belongs to OPEN through:

Case II(c)(i): an optimal path. Then n_{i+1}* will also be the leading node of P with same g-value.

Case II(c)(ii): a suboptimal path. Then n_{i+1}* will have revision of g-value and will become leading node of P.

□

Theorem-6: When the algorithm MSG* finds a match at node x (i.e. x has been selected from OPEN in one direction and x also belongs to OPEN in the other direction) and $L_{min} \leq O_{min}$, x must be on an optimal path.

Proof:

Let P an optimal path in G. Let n* be the leading node of P in FS and m* be the leading node of P in BS.

Since leading node belongs to OPEN, we have

$O_{minF} \leq TE_1(n^*)$, $O_{minB} \leq TE_2(m^*)$. Let the searches meet at node x. Then at termination of the algorithm,

$TE_1(n^*) + TE_2(m^*) \geq O_{min} \geq L_{min}$ (given) $= TE_1(x) + TE_2(x)$, which is valid only if x is on an optimal path.

□

6. Experimental Results

We now show the distribution of forward and backward thresholds for three problem instances namely, Problem Nos. 12, 47 and 48 in Table Nos. 1, 2, and 3. Table-4 contains detailed data on the working of unidirectional IDA* and the newly proposed bidirectional search technique.

g_1	h_1	h_2	g_2	$g_1+h_1-h_2$	$g_2+h_2-h_1$
0	35	0	45	35	10
1	34	1	44	34	11
2	33	2	43	33	12
3	32	3	42	32	**13**
4	**33**	4	41	**33**	12
5	32	5	40	32	13
6	31	6	39	31	14
7	30	7	38	30	15
8	29	8	37	29	16
9	28	9	36	28	**17**
10	**29**	**8**	35	**31**	**14**
11	28	**7**	34	**32**	13
12	27	8	33	31	14
13	26	9	32	30	15
14	25	10	31	29	16
15	24	11	30	28	**17**
16	**25**	12	29	**29**	16
17	24	13	28	28	**17**
18	**25**	14	27	**29**	16
19	24	15	26	28	17
20	23	16	25	27	18
21	22	17	24	26	19
22	21	18	23	25	20
23	20	19	22	24	21
24	19	20	21	23	22
25	18	21	20	22	23
26	17	22	19	21	**24**
27	16	**21**	18	**22**	23
28	15	22	17	21	24
29	14	23	16	20	25
30	13	24	15	19	**26**
31	**14**	**23**	14	**22**	23
32	13	24	13	21	24
33	12	25	12	20	25
34	11	26	11	19	26
35	10	27	10	18	27
36	9	28	9	17	28
37	8	29	8	16	29
38	7	30	7	15	30
39	6	31	6	14	31
40	5	32	5	13	32
41	4	33	4	12	33
42	3	34	3	11	34
43	2	35	2	10	35
44	1	36	1	9	**36**
45	0	**35**	0	**10**	35

Table-1: Showing data for Problem No. 12

g_1	h_1	h_2	g_2	$g_1+h_1-h_2$	$g_2+h_2-h_1$
0	35	0	47	35	12
1	34	1	46	34	13
2	33	2	45	33	**14**
3	**34**	3	44	**34**	13
4	33	4	43	33	14
5	32	5	42	32	15
6	31	6	41	31	16
7	30	7	40	30	17
8	29	8	39	29	18
9	28	9	38	28	19
10	27	10	37	27	20
11	26	11	36	26	**21**
12	**27**	12	35	**27**	20
13	26	13	34	26	21
14	25	14	33	25	**22**
15	**26**	**13**	32	**28**	**19**
16	25	**12**	31	**29**	18
17	24	13	30	28	**19**
18	**25**	14	29	**29**	18
19	24	15	28	28	19
20	23	16	27	27	20
21	22	17	26	26	21
22	21	18	25	25	22
23	20	19	24	24	23
24	19	20	23	23	24
25	18	21	22	22	25
26	17	22	21	21	26
27	16	23	20	20	27
28	15	24	19	19	28
29	14	25	18	18	**29**
30	**15**	**24**	17	**21**	26
31	14	25	16	20	27
32	13	26	15	19	28
33	12	27	14	18	**29**
34	**13**	28	13	**19**	28
35	12	29	12	18	29
36	11	30	11	17	30
37	10	31	10	16	31
38	9	32	9	15	32
39	8	33	8	14	33
40	7	34	7	13	34
41	6	35	6	12	35
42	5	36	5	11	**36**
43	4	**35**	4	**12**	35
44	3	36	3	11	36
45	2	37	2	10	**37**
46	1	**36**	1	**11**	**36**
47	0	**35**	0	**12**	35

Table-2: Showing data for Problem No. 47

g_1	h_1	h_2	g_2	$g_1+h_1-h_2$	$g_2+h_2-h_1$
0	39	0	49	39	**10**
1	**40**	1	48	**40**	9
2	39	2	47	39	10
3	38	3	46	38	11
4	37	4	45	37	12
5	36	5	44	36	13
6	35	6	43	35	14
7	34	7	42	34	15
8	33	8	41	33	16
9	32	9	40	32	**17**
10	**33**	10	39	**33**	16
11	32	11	38	32	17
12	31	12	37	31	18
13	30	13	36	30	**19**
14	**31**	14	35	**31**	18
15	30	15	34	30	19
16	29	16	33	29	20
17	28	17	32	28	21
18	27	18	31	27	**22**
19	26	**17**	30	**28**	21
20	25	18	29	27	22
21	24	18	28	27	22
22	23	20	27	25	24
23	22	21	26	24	25
24	21	22	25	23	26
25	20	23	24	22	27
26	19	24	23	21	28
27	18	25	22	20	29
28	17	26	21	19	30
29	16	27	20	18	31
30	15	28	19	17	**32**
31	**16**	**27**	18	**20**	29
32	15	28	17	19	30
33	14	29	16	18	31
34	13	30	15	17	32
35	12	31	14	16	33
36	11	32	13	15	**34**
37	10	**31**	12	**16**	33
38	9	32	11	15	**34**
39	**10**	33	10	**16**	33
40	9	34	9	15	34
41	8	35	8	14	**35**
42	7	**34**	7	**15**	34
43	6	35	6	14	35
44	5	36	5	13	36
45	4	37	4	12	37
46	3	38	3	11	38
47	2	39	2	10	**39**
48	1	**38**	1	**11**	38
49	0	39	0	10	39

Table-3: Showing data for Problem No. 48

Pno	BTh	HTable	BackPass	FwdPass	Total	IDA*	Imprv	h-val	Opt	CPU
1	7	201132	665061	16140424	16805485	276361933	16.44	41	57	30.00
2	5	703468	2782240	2177237	4959477	15300442	3.09	43	55	10.00
3	8	3055024	15539341	21008424	36547765	565994203	15.49	41	59	86.00
4	6	383710	1486864	5408966	6895830	62643179	9.08	42	56	14.00
5	6	421209	1585590	1585192	3170782	11020325	3.48	42	56	7.00
6	7	98621	335650	2282543	2618193	32201660	12.30	36	52	5.00
7	10	528562	2209638	9039013	11248651	387138094	34.42	30	52	23.00
8	8	214267	762163	3267753	4029916	39118937	9.71	32	50	7.00
9	6	93290	360167	295035	655202	1650696	2.52	32	46	2.00
10	7	932383	3621225	8829163	12450388	198758703	15.96	43	59	26.00
11	6	242108	925926	9265885	10191811	150346072	14.75	43	57	20.00
12	4	13438	37797	143370	181167	546344	3.02	35	45	2.00
13	4	71304	252153	1822609	2074762	11861705	5.72	36	46	3.00
14	8	1341230	5934093	36314595	42248688	1369596778	32.42	41	59	84.00
15	8	7326349	36083156	21325891	57409047	543598067	9.47	44	62	210.00
16	8	163389	662331	1734376	2396707	17984051	7.50	24	42	10.00
17	9	2809350	11862075	26615351	38477426	607399560	15.79	46	66	80.00
18	5	145428	487242	2722444	3209686	23711067	7.39	43	55	8.00
19	4	7830	21731	337332	359063	1280495	3.57	36	46	1.00
20	7	630851	2534038	1866615	4400653	17954870	4.08	36	52	10.00
21	9	635224	2803392	9320012	12123404	257064810	21.20	34	54	25.00
22	8	1028194	3678935	18757405	22436340	750746755	33.46	41	59	47.00
23	7	212937	845504	1836250	2681754	15971319	5.96	33	49	6.00
24	9	841007	3194647	2792616	5987263	42693209	7.13	34	54	13.00
25	9	501572	2056026	5261191	7317217	100734844	13.77	32	52	15.00
26	8	1497079	6914580	12753424	19668004	226668645	11.52	40	58	39.00
27	9	799114	3566072	11926202	15492274	306123421	19.76	33	53	29.00

Pno	*BTh*	*HTable*	*BackPass*	*FwdPass*	*Total*	*IDA**	*Imprv*	*h-val*	*Opt*	*CPU*
28	7	103829	351729	744800	1096529	5934442	5.41	36	52	3.00
29	7	439094	1835847	6011908	7847755	117076111	14.92	38	54	16.00
30	5	34989	110789	443384	554173	2196593	3.96	35	47	2.00
31	5	18395	55322	408125	463447	2351811	5.07	38	50	1.00
32	7	3665125	16714650	26575412	43290062	661041936	15.27	43	59	105.00
33	8	2125911	9709309	16434860	26144169	480637867	18.38	42	60	56.00
34	7	528287	2087381	1633201	3720582	20671552	5.56	36	52	8.00
35	7	1009283	4340836	4518596	8859432	47506056	5.36	39	55	18.00
36	7	298725	1158203	4275956	5434159	59802602	11.00	36	52	10.00
37	8	625293	2664890	11259900	13924790	280078791	20.11	40	58	25.00
38	5	44532	143261	2801568	2944829	24492852	8.32	41	53	6.00
39	6	341856	1488319	2222306	3710625	19355806	5.22	35	49	7.00
40	8	597149	2407238	4018758	6425996	63276188	9.85	36	54	13.00
41	8	379578	1400077	3090963	4491040	51501544	11.47	36	54	8.00
42	5	21919	70813	195168	265981	877823	3.30	30	42	1.00
43	7	985886	3382647	3861866	7244513	41124767	5.68	48	64	18.00
44	8	445724	1869522	6025995	7895517	95733125	12.12	32	50	15.00
45	5	70663	246670	1041812	1288482	6158733	4.78	39	51	3.00
46	6	90553	286254	2303442	2589696	22119320	8.54	35	49	5.00
47	5	25169	85305	325956	411261	1411294	3.43	35	47	1.00
48	4	45681	135863	398468	534331	1905023	3.57	39	49	1.00
49	12	5831003	36617867	32037125	68654992	1809933698	26.36	33	59	242.00
50	6	186062	674175	5143738	5817913	63036422	10.83	39	53	11.00
51	5	487236	1921378	3339133	5260511	26622863	5.06	44	56	9.00
52	8	719058	3045723	13040367	16086090	377141881	23.45	38	56	29.00

Pno	BTh	HTable	BackPass	FwdPass	Total	IDA*	Imprv	h-val	Opt	CPU
53	6	1642114	6248260	29839431	36087691	465225698	12.89	50	64	64.00
54	7	1079724	4446693	11702724	16149417	220374385	13.65	40	56	33.00
55	5	6454	18543	189752	208295	927212	4.45	29	41	1.00
56	12	2419166	12522987	23932839	36455826	1199487996	32.90	29	55	78.00
57	6	184256	735640	884759	1620399	8841527	5.46	36	50	4.00
58	6	61377	186955	1296121	1483076	12955404	8.74	37	51	3.00
59	10	2474181	12184952	28737374	40922326	1207520464	29.51	35	57	76.00
60	8	7218881	32145822	73717585	105863407	3337690331	31.53	48	66	246.00
61	6	99004	355769	1051980	1407749	7096850	5.04	31	45	5.00
62	6	720326	3082844	2208331	5291175	23540413	4.45	43	57	11.00
63	7	878869	3780370	28800165	32580535	995472712	30.55	40	56	55.00
64	9	684088	2955025	9280787	12235812	260054152	21.25	31	51	24.00
65	7	97757	342600	1719152	2061752	18997681	9.21	31	47	4.00
66	9	1982024	8427787	47308972	55736759	1957191378	35.11	41	61	105.00
67	10	746588	3280177	10460866	13741043	252783878	18.40	28	50	28.00
68	9	348816	1439661	4778834	6218495	64367799	10.35	31	51	12.00
69	7	900081	4785592	6099223	10884815	109562359	10.07	37	53	22.00
70	10	576116	2397684	5395603	7793287	151042571	19.38	30	52	16.00
71	6	73821	271041	1247909	1518950	8885972	5.85	30	44	3.00
72	8	839927	3479968	29299284	32779252	1031641140	31.47	38	56	56.00
73	5	40990	138320	334579	472899	3222276	6.81	37	49	1.00
74	4	23327	68880	496306	565186	1897728	3.36	46	56	2.00
75	8	497318	2035823	3091389	5127212	42772589	8.34	30	48	9.00
76	7	882266	3881765	6124821	10006586	126638417	12.66	41	57	21.00
77	9	776237	3248532	1358480	4607012	18918269	4.11	34	54	12.00
78	5	77150	245470	1326581	1572051	10907150	6.94	41	53	3.00

Pno	BTh	HTable	BackPass	FwdPass	Total	IDA*	Imprv	h-val	Opt	CPU
79	6	31872	96617	146270	242887	540860	2.23	28	42	2.00
80	6	615895	2369719	7219175	9588894	132945856	13.86	43	57	17.0
81	6	39760	120769	1193495	1314264	9982569	7.60	39	53	2.00
82	10	6071770	30557880	91579379	122137259	5506801123	45.09	40	62	279.0
83	8	192812	703360	3831857	4535217	65533432	14.45	31	49	10.00
84	8	457948	1678539	7319872	8998411	106074303	11.79	37	55	16.00
85	5	53659	193865	467397	661262	2725456	4.12	32	44	1.00
86	4	99213	391938	460840	852778	2304426	2.70	35	45	2.00
87	8	591092	2557289	3877647	6434936	64926494	10.09	34	52	12.00
88	10	17948104	95521198	84280802	179802000	6009130748	33.42	43	65	827.0
89	7	661622	3012393	8303480	11315873	166571097	14.72	36	54	27.00
90	6	120263	407354	1069855	1477209	7171137	4.85	36	50	3.00
91	7	303839	1036159	22448731	23484890	602886858	25.67	41	57	37.00
92	9	1553975	6635901	28252407	34888308	1101072541	31.56	37	57	60.00
93	5	48021	174480	309286	483766	1599909	3.31	34	46	2.00
94	3	45075	156875	291483	448358	1337340	2.98	45	53	1.00
95	7	145675	547011	856576	1403587	7115967	5.07	34	50	3.00
96	6	191900	714282	1313533	2027815	12808564	6.32	35	49	4.00
97	5	64569	204198	238016	442214	1002927	2.27	32	44	2.00
98	9	1499198	7459573	8060138	15519711	183526883	11.83	34	54	31.00
99	8	1125482	5146510	5168507	10315017	83477694	8.09	39	57	22.00
100	7	819191	3656733	4473008	8129741	67880056	8.35	38	54	18.00

Table-4: Performance of IDA* and the newly proposed Bidirectional Search

7. Conclusion

In this paper we have proposed a new search technique for bidirectional heuristic search. We have assumed that the heuristic is monotone. In this preliminary study we have shown how a bidirectional version of IDA* can be used so that the two opposite search frontiers always meet at the middle and an optimal path is obtained. We have obtained an average speed up of 12.3837 times compared to the unidirectional IDA* in terms of total node generations on the select 100 problem instances of 15-puzzle as presented in Korf (2). The heuristic used was Manhattan distance. This experiment was conducted on a Wipro Netpower 10220: Itanium Processor 64 BIT, Single CPU, 8GB RAM, 73GB*3 HDD, RAID 5. We are in the process of designing more efficient version of the proposed algorithm.

Acknowledgement: This work has been supported by the Institute's CMDS grant under Work Order No. 019/RP: STACA/3392/2008-09 (the project started in 2009.)

References

1. **Nilsson, Nils J.** *Principles of Artificial Intelligence.* s.l. : Morgan Kaufmann, 1982.

2. *Depth-first iterative deepening: An optimal admissible tree search.* **Korf, R.** 1, 1985, Artificial Intelligence, Vol. 27, pp. 97-109.

3. *Bidirectional Heuristic Search Reconsidered.* **Kaindl, H. and Kainz, G.** s.l. : AI Access Foundation and Morgan Kaufmann Publishers, December 1997, Journal of Artificial Intelligence Research, pp. 283-317.

4. *Bi-directional Search.* **Pohl, Ira.** s.l. : Edinburgh University Press, 1971. Machine Intelligence. Vol. 6, pp. 127 -140.

5. *BS*: An Admissible Bidirectional Staged Heuristic Search Algorithm.* **Kwa, J.** s.l. : Elsevier Science Publishers B.V. (North Holland), 1989, Artificial Intelligence, pp. 95-109.

6. *An Improved Bidirectional Search Algorithm.* **Champeaux, D. and Sint, L.** 2, 1977, Journal of the Association for Computing Machinery, Vol. 24, pp. 177-191.

7. *Bidirectional Heuristic Search Again.* **Champeaux, D.** 1983, Journal of the Association for Computing Machinery, pp. 22-32.

8. *Perimeter Search.* **Dillenburg, J. and Nelson, P.** s.l. : Elsevier Science B.V., 1994, Artificial Intelligence (Research Note), Vol. 65, pp. 165-178.

9. *BIDA*: an improved perimeter search algorithm.* **Manzini, G.** 2, 1995, Artificial Intelligence, Vol. 75, pp. 347-360.

10. *D-node retargeting in bidirectional heuristic search.* **Politowski, G. and Pohl, I.** 1984. AAAI-84. pp. 274-277.

11. *Switching from bidirectional to unidirectional search.* **Kaindl, H, Kainz, G, Steiner, R., Auer, A. and Radda, K.** 1999. IJCAI. pp. 1178 - 1183.

12. *A new approach of iterative deepening bi-directional heuristic front-to-front algorithm (IDBHFFA).* **Shamsul Arefin, K. and Saha, A.** 02, 2010, International Journal of Electrical & Computer Sciences, Vol. 10, pp. 13-21.

13. *Bidirectional Heuristic Search with Limited Resources.* **Ghosh, S. and Mahanti, A.** 1991, Information Processing Letters, Vol. 40, pp. 335-340.

14. *Divide-and-Conquer Bidirectional Search: First Results.* **Korf, R.E.** 1999. Proc. of the Sixteenth International Joint Conference on Artificial Intelligence. pp. 1184-1191.