A New Approach of Iterative Deepening Bi-Directional Heuristic Front-to-Front Algorithm (IDBHFFA)

First A. Kazi Shamsul Arefin, and Second B. Aloke Kumar Saha

***Abstract*— Artificial Intelligence (AI) is a subject that studies techniques for making computers exhibit intelligent behavior. Searching still remains one of the problem in AI. Bi-directional search is performed by searching simultaneously in forward direction from the initial node and in backward direction from the goal node. Bi-directional heuristic search algorithms need less time and space than their unidirectional versions. Bi-directional Heuristic Front to Front Algorithm (BHFFA) is one of the Bi-directional heuristic search algorithm. However, it has some disadvantages. It needs to store many unnecessary nodes prior to termination. Moreover, in large problem spaces the computational overhead for the selection of the next node to be expanded increases significantly. This paper presents a modification to the BHFFA called Iterative Deepening Bi-directional Heuristic Front-to-Front Algorithm (IDBHFFA) that has been analyzed and implemented using the 8-puzzle problem. The proposed algorithm performs BHFFA in a number of iterations. For each iteration, two thresholds are maintained, one for each search frontier. In each iteration, some non-promising nodes on a particular search frontier having cost estimates greater than the corresponding threshold value are pruned. The process continues with successive iterations with the thresholds increasing with each iteration. The proposed algorithm will return optimal solutions with an admissible heuristic function. It can minimize the computational time and memory space requirement of BHFFA considerably.**

***Index Term*— Interative Deepining Bi-directional Heuristic Front-to-Front Algorithm (IDBHFFA), Bi-directional Heuristic Front-to-Front Algorithm (BHFFA), Bi-directional Depth-First Iterative Deepening (DFID) Search, Bi-directional Heuristic Path Algorithm (BHPA).**

1. INTRODUCTION

When a problem has an explicitly given single goal state and all node generation operators have inverses, Bi-directional search can be used [6]. Bi-directional search is performed by searching forward from the initial node and backward from the goal node simultaneously. To do so, the program must store the nodes generated on both search frontiers until a common node is found. With some modifications, brute-force [3,7] and heuristic [3,4,7] search methods may be used to perform Bi-directional search. As for example, to perform Bi-directional Depth-First Iterative Deepening (BDDFID) search [2] to a depth of k, the search is made from one direction and the nodes at depth k are stored. At the same time, a search to a depth of k and k+1 is made from the other direction and all nodes generated are matched against the nodes stored from the other side. The search to depth k+1 is needed to account for odd-length paths. This process is repeated for lengths k=0 to d/2 from both directions.

The situation is however more complex in case of bi-directional heuristic search. If the heuristic function used by the bi-directional process is slightly inaccurate, the search frontiers may pass each other without intersecting. In such a case, the bi-directional search process may expand twice as many nodes as would the Unidirectional one. Important bi-directional heuristic search algorithms are Pohl's Bi-directional Heuristic Path Algorithm (BHPA) [5,6], Bi-directional Heuristic Front-to-Front Algorithm (BHFFA) [1] etc.

BHFFA is an important bi-directional heuristic search technique. In BHFFA, the heuristic function is calculated in a different manner. This paper presents a modification to BHFFA. We call this new algorithm Iterative Deepening Bi-directional Heuristic Front-to-Front Algorithm (IDBHFFA). The proposed algorithm performs BHFFA in a number of iterations and can minimize the time and memory space requirement. In each iteration, some non-promising nodes on a particular search frontier having cost estimates greater than the corresponding threshold value are pruned. Like BHFFA, the proposed algorithm will return optimal solutions with an admissible [3,4] heuristic function.

- *F.A. Kazi Shamsul Arefin is with the Department of Computer Science and Engineering, University of Asia Pacific (www.uap-bd.edu), Dhanmondi, Dhaka-1209, Bangladesh. E-mail: arefin@uap-bd.edu.*
- *S.B. Aloke Kumar Saha is with the Department of Computer Science and Engineering, University of Asia Pacific (www.uap-bd.edu), Dhanmondi, Dhaka-1209, Bangladesh. E-mail: aloke@uap-bd.edu.*

Date of Submission: March 10th, 2010.

2. PROCEDURE OF BI-DIRECTIONAL SEARCH

2.1 Algorithm of Bi-directional Search

When a problem has an explicitly given single goal state and all node generation operators have inverses, bi-directional search can be used [6]. Bi-directional search is performed by searching forward from the initial node and backward from the goal node simultaneously. To do so, the program must store the nodes generated on both search frontiers until a common node is found. With some modifications, all tree of the brute-force search methods may be used to perform bi-directional search. The bi-directional version of Depth-First Iterative Deepening (DFID) is presented here.

To perform bi-directional DFID search to a depth of k, the

search is made from one direction and the nodes at depth k are stored. At the same time, a search to a depth of k and k+1 is made from the other direction and all nodes generated are matched against the nodes stored from the other side. These nodes need not be stored. The search to depth k+1 is needed to account for odd-length paths. This process is repeated for lengths k=0 to d/2 from both directions.

2.1.1 Algorithm: Bi-directional DFID

1. Set SEARCH_DEPTH = 0.
2. Conduct a DFS from the start state s to a depth of SEARCH_DEPTH and store all the nodes generated.
3. For DEPTH = SEARCH_DEPTH to SEARCH_DEPTH + 1 do:

Conduct a DFS from the goal state g to a depth of DEPTH. Match the nodes generated with the nodes stored from the other side. If a common node is found, return success and exit.

4. Increment SEARCH_DEPTH by one and go to step 2.

The bi-directional DFID will always return an optimal solution. When the node matching is done in constant time per node, its time and space complexities are both O(bd/2) [8]. The bi-directional process trades space for time and it usually generates many fewer nodes than does the unidirectional process.

2.1.2 Bi-directional Heuristic Front-to-Front Algorithm (BHFFA)

The situation is more complex, however when comparing bi-directional and unidirectional heuristic searches. If the heuristic functions used by the bi-directional process are given slightly inaccurate, the search frontiers may pass each other without intersecting. In such a case, the bi-directional search process may expand twice as many nodes as would the unidirectional one.

Before going in to a precise description of the BHFFA, we give an intuitive sketch. Suppose we have the situation shown in figure 2.1, where S and T are the sets of closed nodes and S` and T` are the two fronts of the open nodes, and we decide to expand a node from S`. In Pohl's bi-directional heuristic path algorithm(BHPA) a node was chosen in S` which had a minimum value of the function g s + h s, where g s was the current minimum to the start node s and h s was an estimator of the distance from the node to the goal node t. The h s we use is different. For a node n, hs(n) is a minimum of H` (n,y) + g t (y) , where y ranges over all nodes in the opposite front T`; g t is like g but with respect to the goal node t, and H` is an estimator of the shortest distance between pairs of nodes.

The disadvantages of this algorithm with respect to the BHPA [3] must be immediately clear, since the calculation of the function h s in this algorithm is much more complicated than the calculation of the distance to t in the BHPA. In order to describe the BHFFA, we have to give the following notation and definitions, in which we follow as closely as possible the terminology of [6].

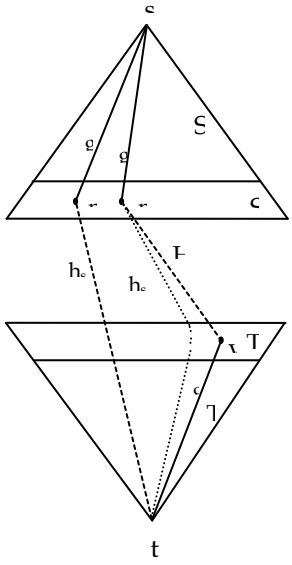

Fig. 2.1 Diagram showing which distances are estimated by heuristic functions in Pohl's and in BHFFA.

2.1.3 Notation and definitions:

s : Start node.
t : Goal node.
S : Collection of nodes reached from s which were expanded.
T : Collection of nodes reached from t which were expanded.
S` : Collection of nodes which are not in S but are direct successors of nodes in S.
T` : Collection of nodes which are not in T but are direct successors of nodes in T.
H(x,y) : Minimum distance between node x and node y.
H`(x,y) : Non-negative estimator of the distance between x and y with H` (x ,y)= H` (y, x).
g s(y) : Minimum distance between s and y for $y \in S \cup S`$, with path in $S \cup S`$, along the current pointers.
g t(y) : Minimum distance between t and y for $y \in T \cup T`$, with path in $T \cup T`$, along the current pointers.
h s(n) : min $y \in T`$ (H` (n,y) + g t(y)).
h t(n) : min $y \in S`$ (H` (n,y) + g s(y)).
f s(x) : g s(x) + h s(x).
f t(x) : g t(x) + h t (x).
Γ(x) : Finite set of nodes obtainable by applicable operators on x.
Γ -1(x): The inverse operator of Γ (x).
l(n,x) : Non-negative edge length between n and x.

2.1.4 Algorithm: BHFFA

1. Place x in S` and t in T` , S:=T:=∅.
2. If $S` \cup T` = \emptyset$ then stop without a solution; Otherwise decide to go forward, step 3, or backward, step 10.
3. Select n in S` with f s(n) = min $y \in S`$ f s (y); Remove n from S` and put n in S; Let descendants (n):= Γ(n) (if more than one can be selected try to find a node n for which $n \in T`$ holds).
4. If $n \in T`$ then stop with a solution path.
5. If descendants (n) = ∅, then go to step 2.
6. Select x from descendants (n) and remove it.

7. If $x \in S`$ then (if $g_s(n) + l(n,x) < g_s(x)$ then (redirect the pointer to n);
go to step 5).
8. If $x \in S$ then (if $g_s(n) + l(n,x) < g_s(x)$ then (redirect the pointer to n;
remove x from S;
and put x in S`;
go to step 5).
9. Put x in S`, provide a pointer to n, and go to step 5.
10. Do steps 3 through 9 with (s, S, S`, Γ) replaced by (t, T, T`, Γ -1).

In step 2 nothing is said about the decision to go forward or backward. As investigated by Pohl, the most promising procedure is to count the number of nodes in S` and T` and to select the front which has the fewest (but at least one).

Unlike the case with the unidirectional algorithm, the h value cannot be stored at a node since this value changes dynamically as a consequence of mutations in the opposite front.

2.2 Some Theorems about BHFFA

We give some theorems and proofs about the BHFFA which parallel the theorems and proofs about the unidirectional A* algorithm of [8].

Theorem 1. *If $H`(x,y) \leq H(x,y)$ and if all edge labels are not less than some positive δ, then the BHFFA halts with a shortest path between s and t (provided there is one).*

Proof. As in the unidirectional case, we first prove a lemma.

Lemma 1 : If $H`(x,y) \leq H(x,y)$, then, for every iteration of the BHFFA and for every optimal path P from s to t, there exist open nodes $n \in S`$, $m \in T`$, on P with $f_s(n) \leq H(s,t)$ and $f_t(m) \leq H(s,t)$.

Proof. Let n be the first node P, counted from s, with $n \in S`$ and m be the first node on P, counted from t, with $m \in T`$ (they exist because otherwise all nodes on P would be closed and the BHFFA had already halted with the solution path P).

$f_s(n) = g_s(n) + h_s(n)$,
$= g_s(n) + H`(n,y) + g_t(y)$ for some $y \in T`$,
$\leq g_s(n) + H`(n,m) + g_t(m)$ by definition of h_s,
$\leq g_s(n) + H(n,m) + g_t(m)$,
$= H(s,t)$ since we are on an optimal path.

$f_t(m) \leq H(s,t)$ is proved in the same way.

Now suppose *Theorem 1* doesn't hold. Then we have three cases :

2.2.1 BHFFA doesn't halt:

Let, P is an optimal path from s to t. According to Lemma 1 there always exists an open node n in $S` \cup T`$ on P with $f_s(n)$ or $f_t(n) \leq H(s,t)$. Therefore the nodes expanded must have an f-value less than or equal to H(s,t). Consequently their g-values are less than or equal to H(s,t). Thus the BHFFA only expands nodes at most $H(s,t)/\delta$ steps away from s or t and this is a finite number. Let M_s and M_t be the sets of all nodes number of successors and as the maximum number of steps any node is away from s or t finite both M_s and M_t can only contain a finite number of nodes, and so $M = M_s \cup M_t$ is of finite size. Let the number of nodes in M be γ. Let p_m be the number of different paths from s to m if $m \in M_s$, and from t to m if $m \in M_t$, and let p be the maximum over all p_m. Then p is the maximum number of different times a node can be reopened. After $p.\gamma$ iterations of the BHFFA ,all nodes of M are permanently closed. So $S` \cup T` = \varnothing$ and the BHFFA halts, which produces a contradiction.

2.2.2 BHFFA halts without a solution path:

We have just proved that the BHFFA eventually halts and it can only do so for two reasons. It has found a solution path or $S \cup T`$ is empty.
Lemma 2, however, prohibits that $S \cup T` = \varnothing$.

2.2.3 BHFFA halts without a shortest path:

Just before ending with a node m, there would, by Lemma 1, be a node n in S` with $f_s(n) \leq H(s,t) \leq f_s(m) + g_t(m)$, and thus n would be chosen for expansion instead of m.

The next theorem proved in [8,10] for the unidirectional algorithm is the optimality theorem, which states that if two heuristics H` and H`* are related by $H`^*(n,t) < H`(n,t) \leq H(n,t)$ for all n, and if $H`(n,t) + H(n,x) \leq H`(x,t)$ for all n and x (this property is called the consistency property) , then every node expanded by H` will also be expanded by H`*. The analogue of this theorem for the BHFFA doesn't hold.

2.3 Analysis

A first order comparison of these algorithms can be done by investigating how they behave in worst-case situations. For a certain search space we give formulas for the number of expanded nodes for the unidirectional and bi-directional Pohl algorithms and the BHFFA, assuming that the heuristic function used gives a maximum error between relative bounds.

Let the search space be an undirected graph containing a countable collection of nodes; two nodes, the start and goal nodes, have m edges (m>1), and there is a path of length K between them. From all other nodes emanate m + 1 edges, there are no cycles; and all the edge lengths are one. So all nodes except the start and the goal nodes have m + 1 successors, of which one is the direct ancestor. Since for the unidirectional and bi-directional Pohl algorithms and the BHFFA, the direct ancestor will be found in the set of closed nodes and will consequently be ignored (in this space there is only one g-value possible, so that cannot be improved), we only consider the remaining m successors. From a unidirectional point of view , the space is a tree with branching rate m since the algorithm will not look beyond the goal node.

Suppose that for each node x on the solution path (s.p), nodes off the s.p. are expanded till some depth n, and that this n is a function of the real distance R to the goal node in the unidirectional case, to the start respectively goal node in the case of Pohl's bi-directional algorithm, and to the opposite front in the BHFFA case: n = u(R). At depth 1 off the s.p. there are m - 1 nodes, at depth i there are $m^{i-1}(m-1)$ nodes, and so the total number of nodes in one such side trees of depth u(R) is given by

$$\sum_{i=1}^{u(R)-1}$$

$$V_R = \sum_{i=0} (m^i (m-1)) = (m-1)(m^{u(R)} - 1) / (m-1) = m^{u(R)} - 1.$$

If we denote by f_A the total number of nodes erroneously expanded by algorithm A, we get the following results:

2.3.1 Unidirectional

R is the distance to the goal node, going from s to t on the s.p.; R decreased in steps of 1 from K till 1 and so

$$\int_{unidirectional} = \sum_{R=1}^{K} V_R.$$

2.3.2 Bidirectional

For some node lying on the s.p. and expanded by the forward algorithm, R is the distance to the goal node, and this distance decreases in steps of 1 from K for the start node to K / 2 +1 for the intersection node. So the forward algorithm which are ever generated from s to t respectively. As ever node has only a finite backward algorithm (for convenience we assume for all the bi-directional cases that K is even and stick alternation: slight changes are required when K is odd), and so

$$\int_{Bi\text{-}directional} = 2 \sum_{R-K/2+1}^{K} V_R.$$

2.3.3 BHFFA

First s is expanded, then t. Suppose a is the successor of s lying on the s.p. and b is the predecessor of t on the s.p. Then for the side trees of s, R = R(s,b) = K - 1, and so the depth of the side tree hanging off s is u(K - 1). The same goes for the tree hanging off t. When a and b are expanded after the side trees at s and t are completed, the distances from a and b to the opposite front are K -3, etc. So in this case R is decreasing in steps of 2 from K - 1 until 1 is reached; therefore the total number of nodes in all the side trees is given by

$$\int_{BHFFA} = 2 \sum_{R=1}^{K/2} V_{2R-1}.$$

These results hold independently of the form of u(R).

It is reasonable to assume that the smaller the real distance, the more accurate (or the less erroneous) will be the estimated distance. Thus we expect that the depth of the side tree hanging off a node on the s.p. will becomes smaller as R become smaller. In that case u(R) is a monotonically decreasing function.

2.3.4 An Example of BHFFA

We consider the 8-puzzle problem to analyze the BHFFA. Figure 2.2(a) and 2.2(b) show search trees where s is the start node and t is the goal node. Here each node is numbered. First we consider one open list (S`.s) and one close list (S.s) for the searching of Forward direction and one open list (T`.t) and one close list(T.t) for the searching of Backward direction. According to BHFFA, at first every close list of both directions are empty and s is the first member of S`.s and t is the first member of T`.t . Then s is shifted from S`.s to S.s and the successors of s (Node: 1,2,3) are generated to S`.s ; t is shifted from T`.t to T.t and the successors of t (Node : 4,5,6,7) are generated to T`.t .

Next we have to find out the heuristic estimate of every node of S`.s (Node : 1,2,3) with respect to every node of T`.t (Node : 4,5,6,7) . In this case the heuristic estimate of node 2 of S`.s is minimum. So node 2 is shifted from S`.s to S.s and generates the successors of node 2 (Node: 8, 9, 10) to S`.s. Then we have to find out the heuristic estimate of every nodes

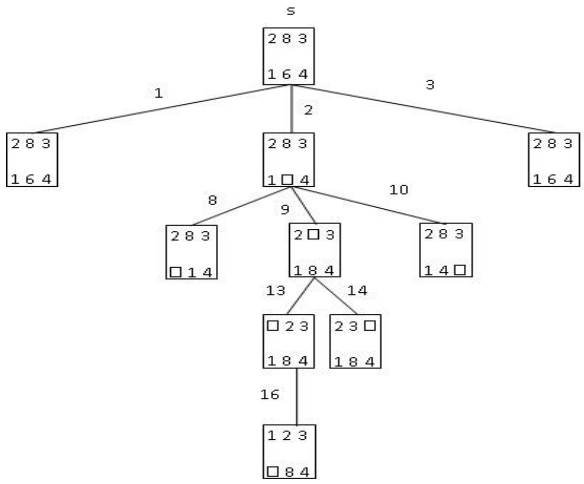

Fig. 2.2. (a). Search tree generated by BHFFA in Forward direction.

of T`.t (Node: 4,5,6,7) with respect to every nodes of S`.s (Node : 1,3,8,9,10) . In this case the heuristic estimate of node 6 is minimum. So node 6 is shifted from T`.t to T.t and the successors of node 6 are generated to T`.t.

This process will continue until a common node will present both of S.s and T.t . In this case node 13 of S.s and node 11 of T.t are common. Because the elements of both node are common. Nodes in S.s are s,2,9 and 13 . Nodes in T.t are t,6 and 11 .So the solution path is s, 2, 9, 13(or 11), 6 to t .In this searching process 18 nodes were generated.

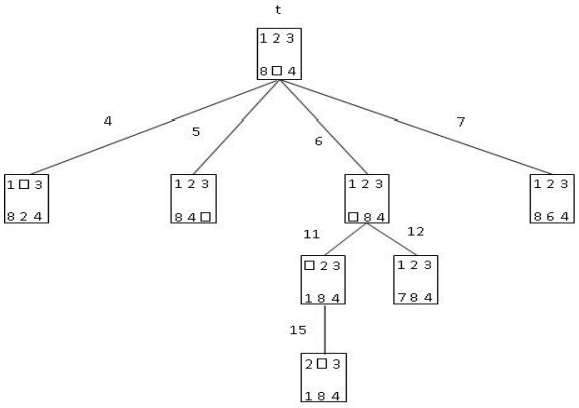

Fig. 2.2. (b). Search tree generated by BHFFA in Backward direction.

3. Iterative Deepening Bi-Directional Heuristic Front-to-Front Algorithm (IDBHFFA): A New Approach

3.1 Introduction

The BHFFA has some disadvantages. All nodes generated by BHFFA, i.e., all nodes on two *open* sets are required to be saved prior to termination and these include many unnecessary nodes. With large *problem* spaces the memory can even become exhausted. Moreover, since the *h*-value of a node is not stored, the *h*-values of all open nodes are calculated whenever a node is expanded and calculation of the *h*-value of a node requires checking all nodes on opposite *open* set. As the size of the problem instance increases, the computational overhead for the selection of the next node to be expanded by BHFFA becomes significant.

Some modifications were proposed [1] to implement the algorithm which can reduce these disadvantages. If the program does not find a solution path after expanding 1000 nodes it is terminated. The number of *open* nodes in a front is restricted to some maximum *m*, which is given to the program as an input parameter but must be less than 100. This is achieved by deleting the worst node of a front whenever the number of nodes exceeds *m*. This pruning operation is done to reduce the overhead in calculating $h(n)=\min(H(n, y)+g'(y))$. But these modifications result in loss of completeness and admissibility.

If the BHFFA can be performed with several iterations where each iteration corresponds to a threshold, some non-promising nodes generated can be pruned on each iteration and the space requirement can also be reduced to a great extent. Keeping this in view, an improved algorithm is proposed in this section.

3.2 Working Principle

The IDBHFFA consists of several iterations. Two thresholds T1 and T2 must be kept : one for search in each direction. Like BHFFA, two open and two closed sets are used, one for each search frontier. Initially, OPEN1 and OPEN2 contain the start state and the goal state respectively and CLOSED1 and CLOSED2 are empty. Moreover, the initial values for T1 and T2 are taken as the heuristic estimate of the minimum distance between the start and goal states.

In each iteration, nodes are expanded in alternating sequence. First, a node is expanded in the forward direction, then the next node is expanded in the backward direction, then another in the forward direction, and so on. In the forward direction, at any point, the node with the lowest f-value (where f(n) is the same as used by BHFFA) from OPEN1 is selected for expansion and is placed on CLOSED1. Whenever a node is expanded it is matched with the nodes generated from the other side, i.e., the nodes on OPEN2. If a common node is found, the search terminates with a solution path. Each successor of the expanded node, whose f-value does not exceed T1, is placed in OPEN1. That is, whenever a node is expanded, each of its successors whose f-value exceeds T1 would be pruned and would not be placed into OPEN1. Moreover, when a node is expanded, if all of its possible successors are generated without being pruned, it is marked. If any of its successors is pruned, it is kept unmarked. This is done to avoid excessive regeneration of nodes.

Then another node is expanded in the backward direction in the same manner with OPEN1, CLOSED1, OPEN2, and T1 replaced by OPEN2, CLOSED2, OPEN1, and T2 respectively. When one of the open sets becomes empty, search from the corresponding direction stops and nodes are expanded only from the opposite direction. When the other open set also becomes empty, the thresholds T1 and T2 are updated to begin the next iteration. The new values of T1 and T2 are taken as the minimum of the f1 and f2 values those exceeded T1 and T2 respectively.

At the start of the second iteration, each unmarked node in both the closed sets will be checked first to generate its pruned successors in the manner described above. If an unmarked node generates all its remaining successors without being cut-off, then this node is marked. After expanding all the unmarked nodes, the original process begins and nodes from the open sets are taken for expansion again. The second iteration continues until the open sets are empty or a solution path is found.

The process continues in this manner with successive iterations. The threshold increases with each iteration. At any iteration the threshold is taken as the minimum of the f-values of the pruned nodes that exceed the previous threshold. This process can continue until a goal node is found or some time-bound is reached. Instead of calculating the h-values of all the nodes in the open sets when a node is to expanded, as in BHFFA, IDBHFFA calculates the h-value of a node whenever it is generated as a successor of another node, i.e., whenever it is placed in one of the open sets. Hence h-value of a node is static, whereas in BHFFA the h-value changes dynamically.

3.3 The Modified Algorithm

Let, S Start node
G Goal node
OPEN1 Collection of nodes generated in the forward direction
OPEN2 Collection of nodes generated in the backward direction
CLOSED1 Collection of nodes expanded in the forward direction

CLOSED2 Collection of nodes expanded in the backward direction

H(x, y) Non-negative heuristic estimate of the minimum distance between nodes x and y with H(x, y)=H(y, x).

g1(y) Minimum distance between S and y for y∈OPEN1

g2(y) Minimum distance between G and y for y∈OPEN2

h1(n) min{ H(n, y)+g2(y) | y∈OPEN2 }

h2(n) min{ H(n, y)+g1(y) | y∈OPEN1 }

f1(n) g1(n) + h1(n)

Evaluation function of any node n for search in the forward direction, i.e., the cost estimate of the optimal path from S to G constrained to go through n

f2(n) g2(n) + h2(n)

Evaluation function of any node n for search in the backward direction, i.e., the cost estimate of the optimal path from G to S constrained to go through n

Mark(n) identifies any node n either as marked or unmarked

T1 Threshold for any iteration for search in forward direction

T2 Threshold for any iteration for search in backward direction

3.3.1 Algorithm IDBHFFA

1. Set T1:=T2:=H(S, G).
2. Set OPEN1:={S}, OPEN2:={G}, CLOSED1:=∅, CLOSED2:=∅.
3. Compute f1(S) and f2(G).
4. Repeat steps 5–11 until a solution is found or some time-bound is reached.
5. Set Temp1:=Temp2:=0.
6. Repeat steps 7–8 until OPEN1∪OPEN2 =∅.
7. If OPEN1≠∅ then

(a) Select n in OPEN1 with f1(n) = min{f1(y) | y∈OPEN1}; remove n from OPEN1 and place it on CLOSED1. In case of a tie, select the most recently generated node.

(b) If n∈OPEN2 then stop with a solution path.

(c) Call Expand(n, OPEN1, CLOSED1, f1(n), T1, Temp1, Mark(n)).

8. If OPEN2≠∅ then

(a) Select n in OPEN2 with f2(n) = min{f2(y) | y∈OPEN2}; remove n from OPEN2 and place it on CLOSED2. In case of a tie, select the most recently generated node.

(b) If n∈OPEN1 then stop with a solution path.

(c) Call Expand(n, OPEN2, CLOSED2, f2(n), T2, Temp2, Mark(n)).

9. Set T1:=Temp1, T2:=Temp2.
10. For ∀n∈{x | x∈CLOSED1 and Mark(x)=0}, Call Expand(n, OPEN1, CLOSED1, f1(n), T1, Temp1, Mark(n)).
11. For ∀n∈{x | x∈CLOSED2 and Mark(x)=0}, Call Expand(n, OPEN2, CLOSED2, f2(n), T2, Temp2, Mark(n)).

Procedure Expand(n, OPEN, CLOSED, f, T, Temp, Mark)

1. Expand n and set Mark:=1. For each successor n' of n execute steps 2–4.
2. Compute f(n').
3. If f(n')<=T then

(a) If n'∉(OPEN∪CLOSED) then add n' to OPEN and provide pointer from n' to n.

If n'∈OPEN then call the old node n".

If f(n')< f(n") then

Set f(n"):= f(n') and redirect the pointer of n" to n.

If n'∈CLOSED then call the old node n".

If f(n')< f(n") then

Move it back to OPEN and redirect the pointer of n" to n.

If f(n')>T then

If Temp=0 or f(n')<Temp then

Set Temp:= f(n') and Mark:=0.

The Expand( ) procedure is used to expand any node n. It will place each successor of n, whose f-value does not exceed the threshold T into one of the open sets. Each of the successors of n whose f-value exceeds T would be pruned and would not be placed into the open set. Moreover, if any of the successors of n is pruned, it is unmarked by making Mark equal to zero.

3.4 Some Analytical Results

If a non-overestimating admissible heuristic function is used, IDBHFFA will always return an optimal solution. At each iteration, two searches are performed from each direction. In each direction, as each node is expanded, it is checked to see whether it has been generated from the opposite direction. If this node has already been generated from the opposite direction, the search terminates. Since, in both directions the paths to this node are of shortest length (because a best-first search is performed with an admissible evaluation function), the total solution will be a shortest path solution. Again, the initial thresholds are equal to the heuristic estimate of the minimum distance between the start and goal states and if the heuristic is admissible, the initial thresholds must be less than or equal to the length of the shortest path solution. Furthermore, in any frontier, all nodes at a given threshold will always be expanded before expanding any node at the next threshold and no node having a cost estimate less than the solution length will be pruned.

Calculation of the *h*-value of a node in BHFFA, as well as in IDBHFFA, requires checking all nodes on opposite *open* set. But in BHFFA, the *h*-value is not stored as it changes dynamically. Whenever a node is to expanded, the *h*-values of all the nodes in the corresponding *open* set is calculated. Whereas, in IDBHFFA, whenever a node is generated as a successor of another node and placed in one of the *open* sets, the *h*-value of the node is calculated statically. As a result of this, a huge number of computations are saved. Although some unmarked nodes in the *closed* sets are required to generate some of their remaining successors, this affect can be ignored considering the computational time saved. If some nodes get unmarked in one iteration, they usually become marked in the next iteration and need not be expanded again. Furthermore, since the *open* sets contain many fewer nodes, the overhead in selecting the next node to be expanded is reduced to a great extent.

The IDBHFFA will need smaller amount of space than the BHFFA algorithm. Since in each iteration, IDBHFFA will prune nodes from both the search frontiers whose costs exceed the respective threshold value, the maximum space used will be much lower than that used by BHFFA. At the final iteration, there would be a lot of nodes whose costs exceed the respective threshold, and IDBHFFA would not have to keep them prior to

termination, which BHFFA would have to.

4. Experimental Results

In the following table 4.1, 4.2, 4.3, we present the list of number of movements and explored nodes of BHFFA (see Table 4.1 and Fig. 4.1) and IDBHFFA (see Table 4.2 and Fig. 4.2) for different initial and goal states of the 8 puzzles. From the table it is clear that always IDBHFFA needs less movements and number of nodes than BHFFA.

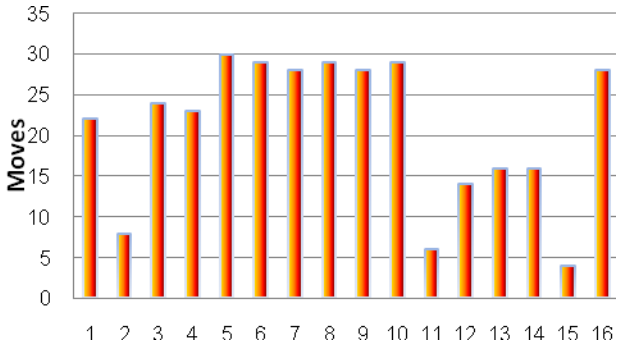

Fig. 4.1. Moves of BHFFA.

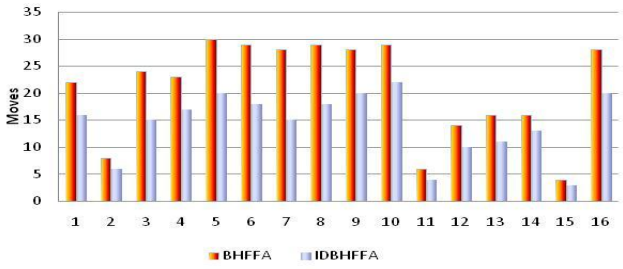

Fig. 4.3. Comparisons between BHFFA and IDBHFFA. From our all experimental results, it has been proved that IDBHFFA needs less movement to reach its goal than BHFFA.

TABLE 4.1
MOVES FOR BHFFA

Initial State	Goal State	BHFFA
456123078*	123804765**	22 moves (explored 451 nodes)
123645087	123804765	8 moves (explored 12 nodes)
741306852	123804765	24 moves (explored 1578 nodes)
306741852	123804765	23 moves (explored 1131 nodes)
567408321	123804765	30 moves (explored 1602 nodes)
567480321	123804765	29 moves (explored 2631 nodes)
56748321	123804765	28 moves (explored 3507 nodes)
506748321	123804765	29 moves (explored 3911 nodes)
567483210	123804765	28 moves (explored 1326 nodes)
567483201	123804765	29 moves (explored 6771 nodes)
134805726	123804765	6 moves (explored 8 nodes)
231708654	123804765	14 moves (explored 79 nodes)
231804765	123804765	16 moves (explored 355 nodes)
123804765	123804765	16 moves (explored 291 nodes)
283104765	123804765	4 moves (explored 5 nodes)
876105234	123804765	28 moves (explored 18514 nodes)

Initial state 456123078 means

4	5	6
1	2	3
0	7	8

**Goal state 123804765 means*

1	2	3
8	0	4
7	6	5

TABLE 4.2
MOVES FOR IDBHFFA

Initial State	Goal State	IDBHFFA
456123078*	123804765**	16 moves (explored 357 nodes)
123645087	123804765	6 moves (explored 10 nodes)
741306852	123804765	15 moves (explored 953 nodes)
306741852	123804765	17 moves (explored 759 nodes)
567408321	123804765	20 moves (explored 1004 nodes)
567480321	123804765	18 moves (explored 1235 nodes)
56748321	123804765	15 moves (explored 1735 nodes)
506748321	123804765	18 moves (explored 1532 nodes)
567483210	123804765	20 moves (explored 735 nodes)
567483201	123804765	22 moves (explored 4521 nodes)
134805726	123804765	4 moves (explored 6 nodes)
231708654	123804765	10 moves (explored 54 nodes)
231804765	123804765	11 moves (explored 250 nodes)
123804765	123804765	13 moves (explored 152 nodes)
283104765	123804765	3 moves (explored 4 nodes)
876105234	123804765	20 moves (explored 987 nodes)

1	2	3
8	0	4
7	6	5

Fig. 4.4. We may have different Initial States. However,123804765 is the Goal state.

Fig. 4.5. We analyzed from 16 (Sixteen) different Initial States between BHFFA and IDBHFFA. We can see that our proposed IDBHFFA is able to solve in Less move than BHFFA.

TABLE 4.3
LESS MOVES FOR IDBHFFA

Initial State	BHFFA Moves	IDBHFFA Moves	Less Moves in IDBHFFA
456123078	22	16	6
123645087	8	6	2
741306852	24	15	9
306741852	23	17	6
567408321	30	20	10
567480321	29	18	11
56748321	28	15	13
506748321	29	18	11
567483210	28	20	8
567483201	29	22	7
134805726	6	4	2
231708654	14	10	4
231804765	16	11	5
123804765	16	13	3
283104765	4	3	1
876105234	28	20	8

1	2	3
8	0	4
7	6	5

Fig. 4.4. We may have different Initial States. However,123804765 is the Goal state.

5. CONCLUSION

Bi-directional search can be used in problem spaces where it is equally possible to search from the start state to the goal state as the other way round and the goal can be explicitly represented as a single state. Bi-directional Heuristic Front-to-Front Algorithm (BHFFA), presented by Champeaux and Sint, is an important bi-directional heuristic search method. In this paper, the attempt is to explore some disadvantages of this algorithm. Consequently, a

modification to this algorithm has been proposed called Iterative Deepening Bi-directional Heuristic Front-to-Front Algorithm (IDBHFFA). The proposed algorithm performs BHFFA in a number of iterations. IDBHFFA will return an optimal solution with an admissible heuristic function. It will save a large number of computations used by BHFFA and will also considerably reduce the overhead in selecting the next node to be expanded. The memory requirement of IDBHFFA will be lower than that of BHFFA.

ACKNOWLEDGMENT

We wish to thank Mr. Shantanu Chakraborty, Md. Anawarul Islam, Sk. Md. Munir Hassan and Mohammad Abdullah Junayeed. This work was partially supported by their undergraduate thesis work.

REFERENCES

[1] Champeaux, D. and Sint, L. "An Improved Bi-directional Heuristic Search Algorithm," Journal of the ACM, 24(2), 1991.

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

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

[4] Pearl, J. "HEURISTICS: Intelligent Search Strategies for Computer Problem Solving," Addison-Wesley, Reading, 1984.

[5] Pohl, I. "Bi-directional heuristic search in path problems," Ph.D Thesis, Department of Computer Science, Stanford University, 1969.

[6] Pohl, I. "Bi-directional Search Machine Intelligence," 6(pp. 127-140), 1971.

[7] Rich, E. and Knight, K. Artificial Intelligence. McGraw-Hill, New York, 1991.

[8] Nilsson, N. J. "Problem-solving methods in Artificial Intelligence," McGraw-Hill, New York.

[9] Korf, R. "Real-time Heuristic search. Artifitial Intelligence," 42, 1990.

[10] Ishida, T. "Real-time bidirectional search: coordinated problem solving inuncertain situations," *http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=506412*, ISSN: 0162-8828, Jun 1996, Volume: 18, Issue: 6.

First A. Kazi Shamsul Arefin has been serving as a Lecturer and Convener of Research and Publication Club with the Department of Computer Science and Engineering (CSE). He is also an Editor of Journal of Computer and Information Technology (JCIT), ISSN 2078-5828. He joined in UAP in October 2009 after completion his M.Sc. Engg. degree. He stood 1st and 2nd position in M.Sc. Engg. and B.Sc. Engg. degree respectively. He was engaged in research activities throughout his undergraduate years and has already published 6 (Six) research papers in different international conferences. Moreover, Mr. Arefin was also an employer of Warid Telecom International Limited. Warid Telecom takes pride in being backed by the Abu Dhabi Group (The Dhabi Group is a multinational company based in the UAE), one of the largest groups in the Middle East and in Pakistan. He worked its Bangladesh operation for 2(two) years long as a team leader of Software Developing unit.

Second B. Aloke Kumar Saha is Head of Computer Science and Engineering Department of University of Asia Pacific (UAP), Dhaka, Bangladesh. He usually teaches courses on Digital Logic Design, Numerical Methods, Data Structures, Discrete Mathematics, Computer Graphics and Basic Electrical Engineering. His current research interests are Algorithm, Artificial Intelligence and Software Development. For more than 13 (Thirteen) years, he is also working with the undergraduate students of UAP, as a part of their thesis works, on the software development and implementation, Bi-Directional Heuristic Search Algorithm etc.