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 500032.

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)
G	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)
*(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 $FE_d(P,n) + BE_d(P,n)$
$FE_d*(n)$	$FE_d(P,n)$ when P is a minimal-cost path from s to n if d=1, or from n to t if d=2
$BE_d*(n)$	$BE_d(P,n)$ when P is a minimal-cost path from s to n if d=1, or from n to t if d=2
$TE_d*(n)$	$TE_d(P,n)$ 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 function 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 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.

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 illu 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, we have:

$$h_d(n) \; d h_d^*(n) \quad n \bullet G$$

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

$$h_d^*(n) \quad g_{3\ d}^*(n) \; d g_{3\ d}(n)$$

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 the common arc-cost with only single link between nodes m and n as c(m,n), where c(m,n) G> 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

lgm he hs js hw ps to a gl, tu hw ps to he he OPEN ts in he ps te

An dte th fo dl eb, tw as pn eb, hs e pd in (8). In hs ph, fs a e tm ed fs eb and he gl ed is pd eg a pn P and he gl ed to a p th ld depth d[1]. The he agm pd

, bw is ja a btn e xae to e aed
to h_F igl Fmd Seah (atgn Ad BAA).

Oe aphs to idl eah bd g a pte ed fm eah
fo as a atg fo te p eah (d ed teg (10)); ig idl
eah t te eah te tem fo te fs tem ad te tgy to idl eh in
te ed bw as g im f xe in OPEN (11); ad an tae ig
aph to BHFFA (12).

Oe tae bw te to idl eah bd an atgn gn a bm em
fe fo to fo wah th tu is aish (13), ad te Did ad Cq
Bidl Sah (14) bw ed te pe gn fo te atgn y g y te OPEN
ls ad to te CLOSED e

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=h2(t);
      next_forward_threshold=h1(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

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

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

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

$=$ t 0 + t 0, as the heuristic is monotone.

Hence $TE_1(P,n_2)$ t $TE_1(P,n_1)$.

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

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'

g_1 h_1 h_2 g_2 $g_1+h_1-h_2$

g_1	h_1	h_2	g_2	$g_1+h_1-h_2$	$g_2+h_2-h_1$

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

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

9. BIDA*: an improved perimeter search algorithm. Manzini, G. 2, 1995, Aftal Ilegg Vb 75, p 347 360.

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

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

12. A new approach of iterative deepening bi directional heuristic front to front algorithm (IDBHFFA). Shamsul Arefin, K. and Saha, G.