D-NODE RETARGETING IN BIDIRECTIONAL HEURISTIC SEARCH

George Politowski and Ira Pohl

Computer and Information Sciences
University of California, Santa Cruz, CA 95064

ABSTRACT

Although it is generally agreed that bidirectional heuristic search is potentially more efficient than unidirectional heuristic search, so far there have been no algorithms which realize this potential. The basic difficulty is that the two search trees (one rooted at the start, the other at the goal) do not meet in the middle. This results in essentially two unidirectional searches and poorer performance. In this paper we present an efficient algorithm for bidirectional heuristic search which overcomes this difficulty. We also compare this algorithm with de Champeaux's BHFFA [2,3] on the basis of search efficiency, solution quality, and computational cost.

I. INTRODUCTION

Searching for paths in very large graphs has been an important problem in AI research. Barr and Feigenbaum [1] gives an excellent overview of this area. In this paper we present a new algorithm for efficient bidirectional heuristic search. We demonstrate empirically that it is more efficient than other search methods, including previous bidirectional techniques [2,3,7,8].

The Heuristic Path Algorithm (HPA) [8] is a modified version of Dijkstra's algorithm [4]. The specification of the evaluation function used to order the nodes is $f = (1-w)*g + w*h$, where g is the length of the known path from the candidate node to the root of the search tree, h is the (heuristic) estimate of the distance (shortest path length) between the node and the goal, and w is a constant which adjusts the relative weights of the two terms. If w is zero, then HPA is equivalent to Dijkstra's algorithm. If w is less than or equal to one-half, and the heuristic estimate never exceeds the actual distance, and the edge costs in the graph are bounded below by some positive number, then HPA is still admissible (i.e. guaranteed to find the shortest path if any path exists) and considerably more efficient than breadth-first search. If w equals one, then the search is called pure heuristic search.

Frequently heuristics which satisfy the admissibility criterion are too weak to be of practical use. Also it is often the case that the length of the solution path is not of primary importance and finding any reasonable path is sufficient. In such cases it is generally desirable to set w greater than one-half in HPA, or to use a more accurate (but non-admissible) heuristic, or both. These choices trade off path quality for search efficiency.

It has been shown [9] that the efficiency of heuristic search may be improved if the search proceeds bidirectionally, i.e. if the search expands outwards from both the start and goal nodes until the searched areas overlap somewhere in between. Although this technique is guaranteed to improve non-heuristic breadth-first search in graphs of uniform density, it has so far not worked well with heuristic search because the expanded areas frequently do not meet 'in the middle.' In the worst case, bidirectional heuristic search performs worse than unidirectional heuristic search. Pohl [9] demonstrates this result for various models of error in tree spaces. Pohl [7] gives some data on bidirectional heuristic search using the 15-puzzle. The data was collected using a bidirectional version of the predecessor of HPA. This algorithm was first called the Very General Heuristic Algorithm (VGHA). Lawler [6] summarizes the potential efficiency of unidirectional and bidirectional search for both the heuristic and non-heuristic cases.

De Champeaux [2,3] describes a Bidirectional Heuristic Front-to-Front Algorithm (BHFFA) which is intended to remedy the 'meet in the middle' problem. Data is included from a set of sample problems corresponding to those of Pohl [7]. The data shows that BHFFA found shorter paths and expanded less nodes than Pohl's bidirectional algorithm. However, there are several problems with the data. One is that most of the problems are too easy to constitute a representative sample of the 15-puzzle state space, and this may bias the results. Another is that the overall computational cost of the BHFFA is not adequately measured, although it is of critical importance in evaluating or selecting a search algorithm. A third problem concerns admissibility. Although the algorithm as formally presented is admissible, the heuristics, weightings, termination condition, and pruning involved in the implemented version all violate admissibility. This makes it difficult to determine whether the results which were obtained are a product of the algorithm itself or of the particular implementation. It is also difficult to be sure that the results would hold in the context of admissible search. One additional problem is that no data is presented to support the claim that the search trees did in fact meet in the middle, although our own tests of BHFFA indicate this result.

In our current research we have attempted to avoid the pitfalls mentioned above. We have explicitly espoused non-admissible search and postponed all concerns about admissibility. We have conducted our tests on randomly generated (hard) problems. We have included data on how well the search trees met in the middle and on how costly the searches were. These precautions allow our data to be more easily interpreted and evaluated by other researchers.

II. ANALYSIS AND DESCRIPTION OF ALGORITHM

As stated above, the main problem in bidirectional heuristic search is to make the two partial paths meet in the middle. The problem with Pohl's bidirectional algorithm is that each search tree is 'aimed' at the root of the opposite tree. Pohl recognized this and compared the situation to two missiles 'independently aimed at each others base in the hope that they would collide.' [7, p. 108] What is needed is some way of aiming at the front (i.e. the leaves) of the opposite tree rather than at its root. There are two advantages to this. First, there is a better chance of meeting the opposite front if you are aiming at it. Second, for most heuristics the aim is better when the target is closer. However, aiming at a front

rather than a single node is somewhat troublesome since the heuristic function is only designed to estimate the distance between two nodes. One way to overcome this difficulty is to choose from each front a representative node which will be used as a target for nodes in the opposite tree. We call such nodes d-nodes, and in the following we discuss a simple scheme for choosing these nodes.

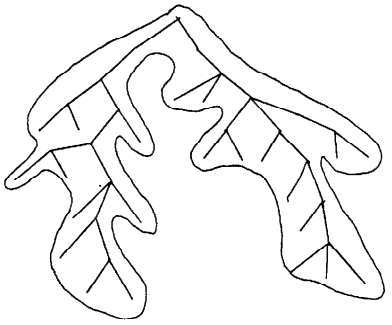

Figure 1

Consider a partially developed search tree, such as the one shown in Figure 1. The growth of the tree is guided by the heuristic function used in the search, and thus the whole tree is inclined, at least to some degree, towards the goal. This means that one can expect that on the average those nodes furthest from the root will also be closest to the goal. Based on the reasoning above, these nodes are the best candidates for the target to be aimed at from the opposite tree (not shown in figure). In particular, the very farthest node out from the root should be the one chosen. D-node selection based on this criterion costs only one comparison per node generated.

We incorporated this idea into a bidirectional version of HPA in the following fashion:

1. Let the root node be the initial d-node in each tree.
2. Advance the search n moves in either the forward or backward direction, aiming at the d-node in the opposite tree. At the same time, keep track of the furthest node out, i.e. the one with the highest g value.
3. After n moves, if the g value of the furthest node out is greater than the g value of the last d-node in this tree, then the furthest node out becomes the new d-node. Each time this occurs, all of the nodes in the opposite front should be re-aimed at the new d-node.
4. Repeat steps 2 and 3 in the opposite direction.

The above algorithm does not specify a value for n. Sufficient analysis may enable one to choose a good value based on other search parameters such as branching rate, quality of heuristic, etc. Otherwise, an empirical choice can be made on the basis of some sample problems. In our work good results were obtained with values of n ranging from 25 to 125. A value of 75 was eventually chosen for generating the data included in this paper.

It is instructive to consider what happens when n is too large or too small because it provides insight into the behavior of the d-node algorithm. A value of n which is too large will lead to performance similar to unidirectional search. This is not surprising since for a sufficiently large n a path will be found unidirectionally before any reversal occurs. A value of n which is too small will lead to poor performance in two respects. First, the runtime will be high because the overhead to re-aim the opposite tree is incurred too often. Second, the path quality will be lower (i.e. the paths will be longer). To understand the reason for this it is necessary to consider Figure 1 again. Note that if there are several major branches in the search tree and a new target is being chosen after each move, then it is possible that successive targets are not in the same branch. When this happens, it causes the opposite tree to be re-aimed at a node which is not near the previous target. If this happens very often, the result is a long 'zig-zag' path.

III. THE TESTS

The evaluation function used by the d-node search algorithm is the same as that used by HPA, namely $f=(1-w)*g+w*h$, except that h is now the heuristic estimate of the distance from a particular node to the d-node of the opposite tree. This is in contrast to Pohl's algorithm, where h estimates the distance to the root of the opposite tree, and to unidirectional heuristic search, where h estimates the distance to the goal. Our aim was to develop an algorithm which would perform well for a variety of heuristics and over a range of w values. With this in mind we decided to test our algorithm on a set of 50 problems with four different heuristics at three different w values.

The 15-puzzle was selected as a convenient and tractable problem domain. Appendix I shows the initial tile configurations for all 50 sample puzzles. Problems 1 through 10 are the same puzzles used by Pohl [7] and de Champeaux [2] for their tests. Problems 11 through 25 were generated by hand; included here are some systematic attempts at generating hard puzzles. Problems 26 through 50 were randomly generated by a program. The exponential nature of the problem space makes it highly probable that randomly generated puzzles will be relatively hard, i.e. their shortest solution paths will be relatively long with respect to the diameter of the state space.

The four functions used to compute h are listed below. These functions were originally developed by Doran and Michie [5], and they are the same functions as those used by Pohl and de Champeaux.

1. $h=P$
2. $h=P+20*R$
3. $h=S$
4. $h=S+20*R$

The three basic terms P, S, and R have the following definitions.

1. $P(a,b)=\sum_i p_i$ where p_i is the Manhattan distance between the position of tile i in a and in b.
2. $S(a,b)=\sum_i p_i^2 d_i^{0.5}$ where p_i is as above and d_i is the distance in a from tile i to the empty square.
3. $R(a,b)$ is the number of reversals in a with respect to b. A reversal means that for adjacent positions i and j, $a(i)=b(j)$ and $a(j)=b(i)$.

Finally, the w values which we used were 0.5, 0.75, and 1.0. This covers the entire 'interesting' range from $w = 0.5$, which will result in admissible search with a suitable heuristic, to $w = 1.0$, which is pure heuristic search.

IV. THE RESULTS

The results of our test of the d-node algorithm are shown in Table 1. For the purpose of comparison, we conducted identical tests on several other algorithms. These results are shown in Tables 2, 3, and 4 for unidirectional HPA, bidirectional HPA (Pohl's algorithm), and de Champeaux's BHFFA, respectively. For each algorithm, the set of 50 sample problems was run 12 times (once for each weighting of each heuristic); data was collected separately for each batch of problems. Listed below are the meanings of the code letters used in the tables. All of the averages

were computed on the basis of solved puzzles only. The search was terminated after 3000 moves if no solution was found.

S number of problems solved.
P average path length.
D average difference between the length of the partial path in the forward tree and the partial path in the backward tree. This is a measure of how well the paths met in the middle. (bidirectional only)
M average number of moves.
N average number of nodes generated.
T average CPU time in seconds.

TABLE 1	h = 1		h = 2		h = 3		h = 4	
w = 0.5	S	6	S	12	S	42	S	50
	P	32.3	P	42.0	P	104.3	P	95.0
	D	9.7	D	7.7	D	20.6	D	20.7
	M	784.0	M	1160.5	M	995.5	M	469.7
	N	1600.0	N	2347.3	N	2277.8	N	1103.4
	T	23.3	T	43.6	T	71.0	T	29.5
w = 0.75	S	41	S	50	S	47	S	50
	P	96.1	P	86.3	P	181.2	P	120.1
	D	20.3	D	17.9	D	28.0	D	19.1
	M	1170.0	M	701.6	M	1079.9	M	371.5
	N	2438.3	N	1466.9	N	2470.6	N	878.1
	T	37.5	T	25.3	T	93.6	T	23.5
w = 1.0	S	50	S	50	S	48	S	50
	P	280.4	P	149.5	P	298.6	P	155.6
	D	29.0	D	25.3	D	29.0	D	25.4
	M	909.6	M	415.6	M	1144.2	M	386.3
	N	1917.4	N	876.8	N	2650.0	N	918.5
	T	31.5	T	14.3	T	112.6	T	23.8

Table 1 - D-node Algorithm

TABLE 2	h = 1		h = 2		h = 3		h = 4	
w = 0.5	S	3	S	4	S	24	S	46
	P	19.3	P	23.5	P	69.3	P	81.3
	M	73.7	M	544.3	M	1373.8	M	827.4
	N	156.0	N	1076.8	N	3087.0	N	1917.1
	T	1.3	T	19.5	T	93.3	T	47.7
w = 0.75	S	17	S	35	S	22	S	50
	P	55.5	P	66.2	P	112.3	P	108.5
	M	1101.4	M	1231.4	M	1604.3	M	612.8
	N	2247.9	N	2517.0	N	3627.9	N	1434.0
	T	34.9	T	42.5	T	105.7	T	31.1
w = 1.0	S	36	S	49	S	34	S	50
	P	220.5	P	158.8	P	184.3	P	136.7
	M	1521.5	M	720.3	M	1595.6	M	544.4
	N	3167.2	N	1506.0	N	3629.2	N	1277.1
	T	34.0	T	19.2	T	107.3	T	23.1

Table 2 - Unidirectional HPA

The most significant result is that the d-node method dominates both previously published bidirectional techniques, regardless of heuristic or weighting. In comparison to de Champeaux's BHFFA, the d-node method is typically 10 to 20 times faster. This is chiefly because the front-to-front calculations required by BHFFA are computationally expensive, even though the number of nodes expanded is roughly comparable for both methods. In comparison to Pohl's bidirectional algorithm, the d-node method typically solves far more problems, and when solving the same problems it expands approximately half as many nodes. The time

TABLE 3	h = 1		h = 2		h = 3		h = 4	
w = 0.5	S	4	S	6	S	18	S	44
	P	24.5	P	30.7	P	69.3	P	80.1
	D	3.0	D	5.7	D	64.3	D	73.9
	M	533.3	M	1258.2	M	1031.4	M	948.4
	N	1052.5	N	2478.3	N	2353.2	N	2226.3
	T	14.3	T	42.0	T	43.9	T	43.6
w = 0.75	S	19	S	34	S	22	S	50
	P	60.1	P	66.6	P	97.0	P	99.4
	D	51.0	D	59.4	D	91.0	D	94.6
	M	1507.7	M	1277.8	M	1569.5	M	659.5
	N	3081.5	N	2614.8	N	3607.6	N	1569.8
	T	37.6	T	35.4	T	74.5	T	26.2
w = 1.0	S	34	S	50	S	31	S	50
	P	175.5	P	133.1	P	173.1	P	120.6
	D	169.5	D	128.3	D	168.1	D	116.3
	M	1703.3	M	834.6	M	1669.7	M	756.6
	N	3518.7	N	1741.6	N	3884.9	N	1795.2
	T	31.9	T	19.1	T	69.9	T	29.9

Table 3 - Bidirectional HPA

TABLE 4	h = 1		h = 2		h = 3		h = 4	
w = 0.5	S	26	S	44	S	21	S	50
	P	81.6	P	75.5	P	98.6	P	78.4
	D	4.8	D	4.4	D	9.0	D	7.0
	M	1045.3	M	861.0	M	996.8	M	346.7
	N	2198.0	N	1795.1	N	2213.1	N	773.1
	T	426.8	T	514.5	T	1020.3	T	411.6
w = 0.75	S	28	S	47	S	26	S	50
	P	135.4	P	125.3	P	172.5	P	91.3
	D	12.2	D	11.0	D	10.5	D	10.4
	M	985.0	M	825.5	M	1462.7	M	324.5
	N	2086.9	N	1750.6	N	3217.7	N	728.8
	T	405.9	T	500.9	T	1508.3	T	384.2
w = 1.0	S	43	S	50	S	33	S	50
	P	253.2	P	185.8	P	200.5	P	111.2
	D	26.3	D	20.0	D	13.2	D	13.7
	M	1311.6	M	868.4	M	1167.7	M	362.9
	N	2781.3	N	1849.5	N	2561.9	N	815.6
	T	543.3	T	522.9	T	1182.6	T	428.9

Table 4 - BHFFA

saving is not as dramatic because the overhead required by the d-node method is somewhat higher than it is in Pohl's algorithm. The results also show that the performance of unidirectional HPA is comparable to Pohl's bidirectional algorithm.

When comparing corresponding blocks in the tables it should be noted that S is the dominating statistic, i.e. if S differs greatly in two corresponding blocks, then the other data in the block are no longer directly comparable. This is because the block with the smaller S represents the solution of easier problems (i.e. those with shorter paths) which means that P is sure to be smaller in that block, and more than likely M, N and T as well. If two corresponding blocks have comparable values of S, then it is reasonable to compare the other statistics.

Another consideration concerning the data from BHFFA is that the algorithm requires pruning to restrict the size of the fronts. This has various effects on the search results, depending on which pruning technique is used. We restricted the front size to 50 nodes by pruning off those nodes with the lowest g values. Previous tests which we conducted indicate that this technique accounts for the high number of solutions in the blocks in Table 4 corresponding to heuristics 1 and 2 with $w = 0.5$.

V. FURTHER RESEARCH

Further investigation of the d-node algorithm is planned. Preliminary work by Politowski and Chapman on higher dimensional sliding block puzzles supports the current results. In the near future other combinatorial problems, such as the Rubik's cube will be similarly tested. Other areas to be worked on are path quality considerations (including admissibility) and a better formal model for understanding the performance of this algorithm.

ACKNOWLEDGEMENTS

Some of the ideas in this paper are based on discussion with and work of Brian Chapman, Dan Chenet, and Phil Levy.

APPENDIX I

GOAL:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	-
1:	1	2	3	4	5	6	7	8	13	15	14	11	10	9	12	-
2:	9	5	1	3	13	7	2	8	14	6	4	11	10	15	12	-
3:	6	2	4	7	5	15	11	8	10	1	3	12	13	9	14	-
4:	1	3	7	4	9	5	8	11	13	6	2	12	10	14	15	-
5:	2	5	6	4	9	1	15	7	14	13	3	8	10	12	11	-
6:	1	2	3	4	5	6	7	8	10	12	11	13	15	14	9	-
7:	1	4	2	3	6	5	8	11	14	9	12	15	10	13	7	-
8:	7	13	11	1	-	4	14	6	8	5	2	12	10	15	9	3
9:	15	11	14	12	7	10	13	9	8	4	6	5	3	2	1	-
10:	9	2	13	10	5	12	7	4	14	1	-	15	11	6	3	8
11:	15	5	7	2	13	10	11	4	12	9	-	8	1	14	3	6
12:	13	15	5	2	4	10	1	7	14	3	9	8	12	-	11	6
13:	12	14	4	6	11	5	13	2	15	3	9	1	10	8	7	-
14:	11	5	12	14	10	-	7	4	13	3	9	6	15	8	1	2
15:	14	4	1	6	9	10	13	11	2	-	12	8	3	15	7	5
16:	9	12	10	13	4	7	1	14	2	11	5	-	8	15	6	3
17:	15	5	12	8	11	4	14	1	9	2	13	6	3	7	-	10
18:	13	7	5	8	14	1	10	-	4	9	15	11	2	3	6	12
19:	-	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
20:	2	1	4	3	6	5	8	7	10	9	12	11	14	15	13	-
21:	15	14	13	12	11	10	9	8	-	7	6	5	4	3	2	1
22:	-	4	9	3	6	2	14	7	15	5	1	8	10	11	13	12
23:	6	13	9	1	3	11	7	-	12	15	5	14	10	4	2	8
24:	-	10	1	6	2	13	14	12	11	8	5	9	3	15	4	7
25:	5	6	7	8	9	10	11	12	13	14	15	-	1	2	4	3
26:	10	1	5	-	15	13	2	8	9	7	12	6	3	4	11	14
27:	4	1	13	7	10	11	5	6	-	8	3	9	14	15	2	12
28:	7	1	13	15	6	9	11	8	4	5	-	10	12	14	3	2
29:	4	-	1	15	13	3	9	11	7	10	12	8	5	14	6	2
30:	14	1	9	5	7	13	4	11	-	10	12	15	3	8	2	6
31:	7	-	1	10	12	11	9	8	5	6	3	14	2	13	15	4
32:	1	2	10	15	6	8	14	7	-	9	4	13	5	11	12	3
33:	10	1	2	4	13	8	15	-	3	14	7	6	9	11	12	5
34:	4	1	10	12	9	14	13	2	11	-	6	8	15	3	7	5
35:	13	1	2	15	3	8	14	11	4	12	-	7	9	10	6	5
36:	5	1	14	4	11	13	8	15	9	12	6	7	-	3	10	2
37:	-	3	7	10	5	11	13	12	2	15	1	6	8	14	4	9
38:	14	3	11	12	13	4	2	7	9	6	-	10	5	1	15	8
39:	7	2	3	15	-	14	8	13	11	1	9	10	4	12	6	5
40:	1	4	12	6	10	13	3	5	11	7	9	15	2	14	8	-
41:	15	3	-	5	14	6	13	7	10	8	1	11	4	9	12	2
42:	8	3	7	10	9	5	11	1	15	-	13	12	2	14	4	6
43:	12	4	8	5	9	1	13	7	10	11	-	6	15	14	3	2
44:	4	2	15	9	-	3	6	10	5	11	12	7	13	8	1	14
45:	15	4	8	14	10	-	2	9	13	12	1	11	3	7	5	6
46:	-	5	13	10	15	2	1	9	3	14	6	4	7	8	11	12
47:	10	5	6	-	9	3	12	14	13	1	4	7	11	8	2	15
48:	3	5	13	4	-	6	11	8	15	10	9	14	1	12	2	7
49:	13	5	6	9	10	-	15	3	7	8	4	1	14	12	2	11
50:	-	5	6	4	10	12	2	3	9	8	1	7	14	13	15	11

REFERENCES

1. Barr, A. and E. A. Feigenbaum, eds., The Handbook of Artificial Intelligence, (William Kaufmann, Inc., Los Altos, CA, 1981).

2. De Champeaux, D. and L. Sint, 'An Improved Bidirectional Heuristic Search Algorithm,' (Journal of the ACM, Vol. 24, No. 2, April 1977, pp. 177-191).

3. De Champeaux, D., 'Bidirectional Heuristic Search Again,' (Journal of the ACM, Vol. 30, No. 1, January 1983, pp. 22-32).

4. Dijkstra, E., 'A note on two problems in connection with graphs,' (Numerische Mathematik, Vol. 1, 1959, pp. 269-271).

5. Doran, J. and D. Michie, 'Experiments with the Graph Traverser program,' (Proceedings of the Royal Society A, Vol. 294, 1966, pp. 235-259).

6. Lawler, E. L., M. G. Luby and B. Parker, 'Finding Shortest Paths in Very Large Networks,' (unpublished, 1983).

7. Pohl, I., 'Bi-directional and Heuristic Search in Path Problems,' (SLAC Report 104, Stanford Univ., Stanford, CA, 1969).

8. Pohl, I., 'Bi-directional Search,' (Machine Intelligence, Vol. 6, 1971, pp. 127-140).

9. Pohl, I., 'Practical and Theoretical Considerations in Heuristic Search Algorithms,' (Machine Intelligence, Vol. 8, 1977, pp. 55-72).