Depth-First Iterative-Deepening: An Optimal Admissible Tree Search*

Richard E. Korf**

Department of Computer Science, Columbia University, New York, NY 10027, U.S.A.

ABSTRACT

The complexities of various search algorithms are considered in terms of time, space, and cost of solution path. It is known that breadth-first search requires too much space and depth-first search can use too much time and doesn't always find a cheapest path. A depth-first iterative-deepening algorithm is shown to be asymptotically optimal along all three dimensions for exponential tree searches. The algorithm has been used successfully in chess programs, has been effectively combined with bi-directional search, and has been applied to best-first heuristic search as well. This heuristic depth-first iterative-deepening algorithm is the only known algorithm that is capable of finding optimal solutions to randomly generated instances of the Fifteen Puzzle within practical resource limits.

1. Introduction

Search is ubiquitous in artificial intelligence. The performance of most AI systems is dominated by the complexity of a search algorithm in their inner loops. The standard algorithms, breadth-first and depth-first search, both have serious limitations, which are overcome by an algorithm called depth-first iterative-deepening. Unfortunately, current AI texts either fail to mention this algorithm [10, 11, 14], or refer to it only in the context of two-person game searches [1, 16]. The iterative-deepening algorithm, however, is completely general and can also be applied to uni-directional search, bi-directional search, and heuristic searches such as A*. The purposes of this article are to demonstrate the generality of depth-first iterative-deepening, to prove its optimality for exponential tree searches, and to remind practitioners in the field that it is the search technique of choice for many applications.

Depth-first iterative-deepening has no doubt been rediscovered many times

independently. The first use of the algorithm that is documented in the literature is in Slate and Atkin's Chess 4.5 program [15]. Berliner [2] has observed that breadth-first search is inferior to the iterative-deepening algorithm. Winston [16] shows that for two-person game searches where only terminal-node static evaluations are counted in the cost, the extra computation required by iterative-deepening is insignificant. Pearl [12] initially suggested the iterative-deepening extension of A*, and Berliner and Goetsch [3] have implemented such an algorithm concurrently with this work.

We will analyze several search algorithms along three dimensions: the amount of time they take, the amount of space they use, and the cost of the solution paths they find. The standard breadth-first and depth-first algorithms will be shown to be inferior to the depth-first iterative-deepening algorithm. We will prove that this algorithm is asymptotically optimal along all three dimensions for exponential tree searches. Since almost all heuristic tree searches have exponential complexity, this is a fairly general result.

We begin with the problem-space model of Newell and Simon [9]. A *problem space* consists of a set of states and a set of operators that are partial functions that map states into states. A *problem* is a problem space together with a particular initial state and a set of goal states. The task is to find a sequence of operators that will map the initial state to a goal state.

The complexity of a problem will be expressed in terms of two parameters: the branching factor of the problem space, and the depth of solution of the problem. The *node branching factor* (b) of a problem is defined as the number of new states that are generated by the application of a single operator to a given state, averaged over all states in the problem space. We will assume that the branching factor is constant throughout the problem space. The *depth* (d) of solution of a problem is the length of the shortest sequence of operators that map the initial state into a goal state. The time cost of a search algorithm in this model of computation is simply the number of states that are expanded. The reason for this choice is that we are interested in asymptotic complexity and we assume that the amount of time is proportional to the number of states expanded. Similarly, since we assume that the amount of space required is proportional to the number of states that are stored, the asymptotic space cost of an algorithm in this model will be the number of states that must be stored.

This work is focused on searches which produce optimal solutions. We recognize that for most applications, optimal solutions are not required and that their price is often prohibitive. There are occasions, however, when optimal solutions are needed. For example, in assessing the quality of non-optimal solutions, it is often enlightening to compare them to optimal solutions for the same problem instances.

2. Breadth-First Search

We begin our discussion with one of the simplest search algorithms, breadth-

first search. Breadth-first search expands all the states one step (or operator application) away from the initial state, then expands all states two steps from the initial state, then three steps, etc., until a goal state is reached. Since it always expands all nodes at a given depth before expanding any nodes at a greater depth, the first solution path found by breadth-first search will be one of shortest length. In the worst case, breadth-first search must generate all nodes up to depth d, or $b + b^2 + b^3 + \cdots + b^d$ which is $O(b^d)$. Note that on the average, half of the nodes at depth d must be examined, and therefore the average-case time complexity is also $O(b^d)$.

Since all the nodes at a given depth are stored in order to generate the nodes at the next depth, the minimum number of nodes that must be stored to search to depth d is b^{d-1}, which is $O(b^d)$. As with time, the average-case space complexity is roughly one-half of this, which is also $O(b^d)$. This space requirement of breadth-first search is its most critical drawback. As a practical matter, a breadth-first search of most problem spaces will exhaust the available memory long before an appreciable amount of time is used. The reason for this is that the typical ratio of memory to speed in modern computers is a million words of memory for each million instructions per second (MIPS) of processor speed. For example, if we can generate a million states per minute and require a word to store each state, memory will be exhausted in one minute.

3. Depth-First Search

Depth-first search avoids this memory limitation. It works by always generating a descendant of the most recently expanded node, until some depth cutoff is reached, and then backtracking to the next most recently expanded node and generating one of its descendants. Therefore, only the path of nodes from the initial node to the current node must be stored in order to execute the algorithm. If the depth cutoff is d, the space required by depth-first search is only $O(d)$.

Since depth-first search only stores the current path at any given point, it is bound to search all paths down to the cutoff depth. In order to analyze its time complexity, we must define a new parameter, called the *edge branching factor* (e), which is the average number of different operators which are applicable to a given state. For trees, the edge and node branching factors are equal, but for graphs in general the edge branching factor may exceed the node branching factor. For example, the graph in Fig. 1 has an edge branching factor of two, while its node branching factor is only one. Note that a breadth-first search of this graph takes only linear time while a depth-first search requires exponential time. In general, the time complexity of a depth-first search to depth d is $O(e^d)$. Since the space used by depth-first search grows only as the log of the time required, the algorithm is time-bound rather than space-bound in practice.

Another drawback, however, to depth-first search is the requirement for an arbitrary cutoff depth. If branches are not cut off and duplicates are not

FIG. 1. Graph with linear number of nodes but exponential number of paths.

checked for, the algorithm may not terminate. In general, the depth at which the first goal state appears is not known in advance and must be estimated. If the estimate is too low, the algorithm terminates without finding a solution. If the depth estimate is too high, then a large price in running time is paid relative to an optimal search, and the first solution found may not be an optimal one.

4. Depth-First Iterative-Deepening

A search algorithm which suffers neither the drawbacks of breadth-first nor depth-first search on trees is depth-first iterative-deepening (DFID). The algorithm works as follows: First, perform a depth-first search to depth one. Then, discarding the nodes generated in the first search, start over and do a depth-first search to level two. Next, start over again and do a depth-first search to depth three, etc., continuing this process until a goal state is reached.

Since DFID expands all nodes at a given depth before expanding any nodes at a greater depth, it is guaranteed to find a shortest-length solution. Also, since at any given time it is performing a depth-first search, and never searches deeper than depth d, the space it uses is $O(d)$.

The disadvantage of DFID is that it performs wasted computation prior to reaching the goal depth. In fact, at first glance it seems very inefficient. Below, however, we present an analysis of the running time of DFID that shows that this wasted computation does not affect the asymptotic growth of the run time for exponential tree searches. The intuitive reason is that almost all the work is done at the deepest level of the search. Unfortunately, DFID suffers the same drawback as depth-first search on arbitrary graphs, namely that it must explore all possible paths to a given depth.

Definition 4.1. *A brute-force search* is a search algorithm that uses no information other than the initial state, the operators of the space, and a test for a solution.

Theorem 4.2. *Depth-first iterative-deepening is asymptotically optimal among brute-force tree searches in terms of time, space, and length of solution.*

Proof. As mentioned above, since DFID generates all nodes at a given depth before expanding any nodes at a greater depth, it always finds a shortest path to the goal, or any other state for that matter. Hence, it is optimal in terms of solution length.

Next, we examine the running time of DFID on a tree. The nodes at depth d are generated once during the final iteration of the search. The nodes at depth $d - 1$ are generated twice, once during the final iteration at depth d, and once during the penultimate iteration at depth $d - 1$. Similarly, the nodes at depth $d - 2$ are generated three times, during iterations at depths d, $d - 1$, and $d - 2$, etc. Thus the total number of nodes generated in a depth-first iterative-deepening search to depth d is

$$b^d + 2b^{d-1} + 3b^{d-2} + \cdots + db.$$

Factoring out b^d gives

$$b^d(1 + 2b^{-1} + 3b^{-2} + \cdots + db^{1-d}).$$

Letting $x = 1/b$ yields

$$b^d(1 + 2x^1 + 3x^2 + \cdots + dx^{d-1}).$$

This is less than the infinite series

$$b^d(1 + 2x^1 + 3x^2 + 4x^3 + \cdots),$$

which converges to

$$b^d(1 - x)^{-2} \quad \text{for abs}(x) < 1.$$

Since $(1 - x)^{-2}$, or $(1 - 1/b)^{-2}$, is a constant that is independent of d, if $b > 1$ then the running time of depth-first iterative-deepening is $O(b^d)$.

To see that this is optimal, we present a simple adversary argument. The number of nodes at depth d is b^d. Assume that there exists an algorithm that examines less than b^d nodes. Then, there must exist at least one node at depth d which is not examined by this algorithm. Since we have no additional information, an adversary could place the only solution at this node and hence the proposed algorithm would fail. Hence, any brute-force algorithm must take at least cb^d time, for some constant c.

Finally, we consider the space used by DFID. Since DFID at any point is engaged in a depth-first search, it need only store a stack of nodes which represents the branch of the tree it is expanding. Since it finds a solution of optimal length, the maximum depth of this stack is d, and hence the maximum amount of space is $O(d)$.
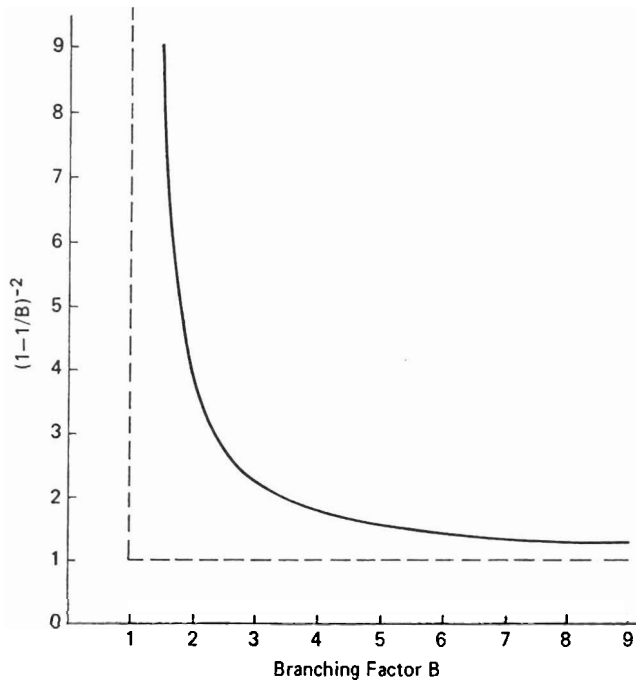
FIG. 2. Graph of branching factor vs. constant coefficient as search depth goes to infinity.

To show that this is optimal, we note that any algorithm which uses $f(n)$ time must use at least $k \log f(n)$ space for some constant k [7]. The reason is that the algorithm must proceed through $f(n)$ distinct states before looping or terminating, and hence must be able to store that many distinct states. Since storing $f(n)$ states requires $\log f(n)$ bits, and $\log b^d$ is $d \log b$, any brute-force algorithm must use kd space, for some constant k.   ☐

The value of the constant $(1 - 1/b)^{-2}$ gives an upper bound on how much computation is wasted in the lower levels of the search, since it is the limit of the constant coefficient as the search depth goes to infinity. Fig. 2 shows a graph of this constant versus the branching factor. As the branching factor increases, the constant quickly approaches one. For branching factors close to one, however, the value of the constant coefficient approaches infinity as the depth goes to infinity.

5. Bi-Directional Search

For those problems with a single goal state that is given explicitly and for which the operators have inverses, such as the Fifteen Puzzle, bi-directional search

[13] can be used. Bi-directional search trades space for time by searching forward from the initial state and backward from the goal state simultaneously, storing the states generated, until a common state is found on both search frontiers. Depth-first iterative-deepening can be applied to bi-directional search as follows: A single iteration consists of a depth-first search from one direction to depth k, storing only the states at depth k, and two depth-first searches from the other direction, one to depth k and one to depth $k + 1$, not storing states but simply matching against the stored states from the other direction. The search to depth $k + 1$ is necessary to find odd-length solutions. This is repeated for k from zero (to find solutions of length one) to $d/2$. Assuming that a hashing scheme is used to perform the matching in constant time per node, this algorithm will find an optimal solution of length d in time $O(b^{d/2})$ and space $O(b^{d/2})$. In experiments involving Rubik's Cube [8], which has an effective branching factor of 13.5, this algorithm was used to find solutions up to 11 moves long on a DEC VAX 11/780.

6. Heuristic Search

Depth-first iterative-deepening can also be combined with a best-first heuristic search such as A* [6]. The idea is that successive iterations correspond not to increasing depth of search, but rather to increasing values of the total cost of a path. For A*, this total cost is composed of the cost so far in reaching the node (g) plus the estimated cost of the path from the node to a goal state (h). Iterative-deepening-A* (IDA*) works as follows: At each iteration, perform a depth-first search, cutting off a branch when its total cost ($g + h$) exceeds a given threshold. This threshold starts at the estimate of the cost of the initial state, and increases for each iteration of the algorithm. At each iteration, the threshold used for the next iteration is the minimum cost of all values that exceeded the current threshold.

A well-known property of A* is that it always finds a cheapest solution path if the heuristic is *admissible*, or in other words never overestimates the actual cost to the goal [6]. This property also holds for iterative-deepening-A*. Furthermore, IDA* expands the same number of nodes, asymptotically, as A* in an exponential tree search.

The proofs of these results are much simpler and more intuitive if we restrict our attention to cost functions which are monotonically non-decreasing along any path in the problem space. Such a heuristic is called *monotone or consistent* [11]. Formally,

Definition 6.1. A cost function $f(n)$ is *monotone* if for all nodes n and $s(n)$, where $s(n)$ is a successor of n, $f(n) \leqslant f(s(n))$.

This restriction is not essential, and slightly more complex proofs will establish the same results without it. As a practical matter, however, almost all

reasonable cost functions are monotone [11]. In fact, using an idea proposed by Mérō [17], we can formally make this assumption without loss of generality, as shown in the following lemma.

Lemma 6.2. *For any admissible cost function f, we can construct a monotone admissible function f' which is at least as informed as f.*

Proof. We construct f' recursively from f as follows: if n is the initial state, then $f'(n) = f(n)$; otherwise, $f'(s(n)) = \max[f(s(n)), f'(n)]$. Clearly, f' is monotone since $f'(n) \leqslant f'(s(n))$. In order to show that f' is admissible, note that $f'(n)$ is equal to the maximum value of f applied to all the predecessors of n along the path back to the initial state. Since f is admissible, the maximum value of f along a path is a lower bound on the cost of that path, and hence a lower bound on the cost of n. Thus, f' does not violate admissibility. Furthermore, f' is at least as informed as f since for all n, $f'(n) \geqslant f(n)$ and hence $f'(n)$ is at least as accurate an estimate as $f(n)$. $\square$

Note that this lemma provides a simple and intuitive proof of the admissibility of A*. If we restrict our attention to cost functions which are monotone non-decreasing, and A* always expands the open node of least cost, it is clear that the first solution it finds will be one of least cost. Similarly, the result below follows just as easily.

Lemma 6.3. *Given an admissible monotone cost function, iterative-deepening-A* will find a solution of least cost if one exists.*

Proof. Since the initial cost cutoff of IDA* is the heuristic estimate of the cost of the initial state, and the heuristic never overestimates cost, the length of the shortest solution cannot be less than the initial cost cutoff. Furthermore, since the cost cutoff for each succeeding iteration is the minimum value which exceeded the previous cutoff, no paths can have a cost which lies in a gap between two successive cutoffs. Therefore, since IDA* always expands all nodes at a given cost before expanding any nodes at a greater cost, the first solution it finds will be a solution of least cost. $\square$

Not only does IDA* find a cheapest path to a solution and use far less space than A*, but it expands approximately the same number of nodes as A* in a tree search. Combining this fact with several recent results on the complexity and optimality of A* allows us to state and prove the following general result:

Theorem 6.4. *Given an admissible monotone heuristic with constant relative error, then iterative-deepening-A* is optimal in terms of solution cost, time, and space, over the class of admissible best-first searches on a tree.*

Proof. From Lemma 6.3, we know that IDA* produces a solution of optimal cost.

To determine the time used by IDA*, consider the final iteration, in other words the one which finds a solution. It must expand all descendents of the initial state with values greater than or equal to the initial cost estimate and less than the optimal solution cost, plus some number of nodes whose cost equals the optimal solution cost. If A* employs the tie-breaking rule of 'most recently generated', it must also expand these same nodes. Thus, the final iteration of IDA* expands the same set of nodes as A* under this tie-breaking rule. Furthermore, if the graph is a tree, each of these nodes will be expanded exactly once. IDA* must also expand nodes during the previous iterations as well. However, Pearl has shown that if the heuristic used by A* exhibits constant relative error, then the number of nodes generated by the algorithm increases exponentially with depth [11]. Thus, we can use an argument similar to the proof of Theorem 4.2 to show that the previous iterations of IDA* do not affect the asymptotic order of the total number of nodes [18]. Thus, IDA* expands the same number of nodes, asymptotically, as A*. Furthermore, a recent result of Dechter and Pearl [5] shows that A* is optimal, in terms of number of nodes expanded, over the class of admissible best-first searches with monotone heuristics. Therefore, IDA* is asymptotically optimal in terms of time for tree searches.

Since the number of nodes grows exponentially, we can again appeal to the argument in the proof of Theorem 4.2 to show that the space used by IDA* is also asymptotically optimal.   □

Is the assumption of constant relative error, i.e. that the error in the estimate grows at the same rate as the magnitude of the actual cost, valid for heuristics? Pearl observes that heuristics with better accuracy almost never occur in practice. For example, most physical measurements are subject to constant relative error [11]. Thus, we can conclude that heuristic depth-first iterative-deepening is asymptotically optimal for most best-first tree searches which occur in practice.

An additional benefit of IDA* over A* is that it is simpler to implement since there are no open or closed lists to be managed. A simple recursion performs the depth-first search inside an outer loop to handle the iterations.

As an empirical test of the practicality of the algorithm, both IDA* and A* were implemented for the Fifteen Puzzle. The implementations were in PASCAL and were run on a DEC 2060. The heuristic function used for both was the Manhattan distance heuristic: for each movable tile, the number of grid units between the current position of the tile and its goal position are computed, and these values are summed for all tiles. The two algorithms were tested against 100 randomly generated, solvable initial states. IDA* solved all instances with a median time of 30 CPU minutes, generating over 1.5 million nodes per minute. The average solution length was 53 moves and the maximum was 66 moves. A*

solved none of the instances since it ran out of space after about 30 000 nodes were stored. An additional observation is that even though IDA* generated more nodes than A*, it actually ran faster than A* on the same problem instances, due to less overhead per node. The data from this experiment are summarized in Table 1. These are the first published optimal solution lengths to randomly generated instances of the Fifteen Puzzle. Although the Fifteen Puzzle graph is not strictly a tree, the edge branching factor is only slightly greater than the node branching factor, and hence the iterative-deepening algorithm is still effective.

TABLE 1. Optimal solution lengths for 100 randomly generated Fifteen Puzzle instances using iterative-deepening-A* with Manhattan distance heuristic function

NUMBER	INITIAL STATE	ESTIMATE	ACTUAL	TOTAL NODES
1	14 13 15 7 11 12 9 5 6 0 2 1 4 8 10 3	41	57	276,361,933
2	13 5 4 10 9 12 8 14 2 3 7 1 0 15 11 6	43	55	15,300,442
3	14 7 8 2 13 11 10 4 9 12 5 0 3 6 1 15	41	59	565,994,203
4	5 12 10 7 15 11 14 0 8 2 1 13 3 4 9 6	42	56	62,643,179
5	4 7 14 13 10 3 9 12 11 5 6 15 1 2 8 0	42	56	11,020,325
6	14 7 1 9 12 3 6 15 8 11 2 5 10 0 4 13	36	52	32,201,660
7	2 11 15 5 13 4 6 7 12 8 10 1 9 3 14 0	30	52	387,138,094
8	12 11 15 3 8 0 4 2 6 13 9 5 14 1 10 7	32	50	39,118,937
9	3 14 9 11 5 4 8 2 13 12 6 7 10 1 15 0	32	46	1,650,696
10	13 11 8 9 0 15 7 10 4 3 6 14 5 12 2 1	43	59	198,758,703
11	5 9 13 14 6 3 7 12 10 8 4 0 15 2 11 1	43	57	150,346,072
12	14 1 9 6 4 8 12 5 7 2 3 0 10 11 13 15	35	45	546,344
13	3 6 5 2 10 0 15 14 1 4 13 12 9 8 11 7	36	46	11,861,705
14	7 6 8 1 11 5 14 10 3 4 9 13 15 2 0 12	41	59	1,369,596,778
15	13 11 4 12 1 8 9 15 6 5 14 2 7 3 10 0	44	62	543,598,067
16	1 3 2 5 10 9 15 6 8 14 13 11 12 4 7 0	24	42	17,984,051
17	15 14 0 4 11 1 6 13 7 5 8 9 3 2 10 12	46	66	607,399,560
18	6 0 14 12 1 15 9 10 11 4 7 2 8 3 5 13	43	55	23,711,067
19	7 11 8 3 14 0 6 15 1 4 13 9 5 12 2 10	36	46	1,280,495
20	6 12 11 3 13 7 9 15 2 14 8 10 4 1 5 0	36	52	17,954,870
21	12 8 14 6 11 4 7 0 5 1 10 15 3 13 9 2	34	54	257,064,810
22	14 3 9 1 15 8 4 5 11 7 10 13 0 2 12 6	41	59	750,746,755
23	10 9 3 11 0 13 2 14 5 6 4 7 8 15 1 12	33	49	15,971,319
24	7 3 14 13 4 1 10 8 5 12 9 11 2 15 6 0	34	54	42,693,209
25	11 4 2 7 1 0 10 15 6 9 14 8 3 13 5 12	32	52	100,734,844
26	5 7 3 12 15 13 14 8 0 10 9 6 1 4 2 11	40	58	226,668,645
27	14 1 8 15 2 6 0 3 9 12 10 13 4 7 5 11	33	53	306,123,421
28	13 14 6 12 4 5 1 0 9 3 10 2 15 11 8 7	36	52	5,934,442
29	9 8 0 2 15 1 4 14 3 10 7 5 11 13 6 12	38	54	117,076,111
30	12 15 2 6 1 14 4 8 5 3 7 0 10 13 9 11	35	47	2,196,593
31	12 8 15 13 1 0 5 4 6 3 2 11 9 7 14 10	38	50	2,351,811
32	14 10 9 4 13 6 5 8 2 12 7 0 1 3 11 15	43	59	661,041,936
33	14 3 5 15 11 6 13 9 0 10 2 12 4 1 7 8	42	60	480,637,867
34	6 11 7 8 13 2 5 4 1 10 3 9 14 0 12 15	36	52	20,671,552
35	1 6 12 14 3 2 15 8 4 5 13 9 0 7 11 10	39	55	47,506,056
36	12 6 0 4 7 3 15 1 13 9 8 11 2 14 5 10	36	52	59,802,602
37	8 1 7 12 11 0 10 5 9 15 6 13 14 2 3 4	40	58	280,078,791
38	7 15 8 2 13 6 3 12 11 0 4 10 9 5 1 14	41	53	24,492,852
39	9 0 4 10 1 14 15 3 12 6 5 7 11 13 8 2	35	49	19,355,806
40	11 5 1 14 4 12 10 0 2 7 13 3 9 15 6 8	36	54	63,276,188
41	8 13 10 9 11 3 15 6 0 1 2 14 12 5 4 7	36	54	51,501,544
42	4 5 7 2 9 14 12 13 0 3 6 11 8 1 15 10	30	42	877,823
43	11 15 14 13 1 9 10 4 3 6 2 12 7 5 8 0	48	64	41,124,767
44	12 9 0 6 8 3 5 14 2 4 11 7 10 1 15 13	32	50	95,733,125
45	3 14 9 7 12 15 0 4 1 8 5 6 11 10 2 13	39	51	6,158,733
46	8 4 6 1 14 12 2 15 13 10 9 5 3 7 0 11	35	49	22,119,320
47	6 10 1 14 15 8 3 5 13 0 2 7 4 9 11 12	35	47	1,411,294
48	8 11 4 6 7 3 10 9 2 12 15 13 0 1 5 14	39	49	1,905,023
49	10 0 2 4 5 1 6 12 11 13 9 7 15 3 14 8	33	59	1,809,933,698
50	12 5 13 11 2 10 0 9 7 8 4 3 14 6 15 1	39	53	63,036,422
51	10 2 8 4 15 0 1 14 11 13 3 6 9 7 5 12	44	56	26,622,863

TABLE 1. *Continued*

NUMBER	INITIAL STATE	ESTIMATE	ACTUAL	TOTAL NODES
52	10 8 0 12 3 7 6 2 1 14 4 11 15 13 9 5	38	56.	377,141,881
53	14 9 12 13 15 4 8 10 0 2 1 7 3 11 5 6	50	64	465,225,698
54	12 11 0 8 10 2 13 15 5 4 7 3 6 9 14 1	40	56	220,374,385
55	13 8 14 3 9 1 0 7 15 5 4 10 12 2 6 11	29	41	927,212
56	3 15 2 5 11 6 4 7 12 9 1 0 13 14 10 8	29	55	1,199,487,996
57	5 11 6 9 4 13 12 0 8 2 15 10 1 7 3 14	36	50	8,841,527
58	5 0 15 8 4 6 1 14 10 11 3 9 7 12 2 13	37	51	12,955,404
59	15 14 6 7 10 1 0 11 12 8 4 9 2 5 13 3	35	57	1,207,520,464
60	11 14 13 1 2 3 12 4 15 7 9 5 10 6 8 0	48	66	3,337,690,331
61	6 13 3 2 11 9 5 10 1 7 12 14 8 4 0 15	31	45	7,096,850
62	4 6 12 0 14 2 9 13 11 8 3 15 7 10 1 5	43	57	23,540,413
63	8 10 9 11 14 1 7 15 13 4 0 12 6 2 5 3	40	56	995,472,712
64	5 2 14 0 7 8 6 3 11 12 13 15 4 10 9 1	31	51	260,054,152
65	7 8 3 2 10 12 4 6 11 13 5 15 0 1 9 14	31	47	18,997,681
66	11 6 14 12 3 5 1 15 8 0 10 13 9 7 4 2	41	61	1,957,191,378
67	7 1 2 4 8 3 6 11 10 15 0 5 14 12 13 9	28	50	252,783,878
68	7 3 1 13 12 10 5 2 8 0 6 11 14 15 4 9	31	51	64,367,799
69	6 0 5 15 1 14 4 9 2 13 8 10 11 12 7 3	37	53	109,562,359
70	15 1 3 12 4 0 6 5 2 8 14 9 13 10 7 11	30	52	151,042,571
71	5 7 0 11 12 1 9 10 15 6 2 3 8 4 13 14	30	44	8,885,972
72	12 15 11 10 4 5 14 0 13 7 1 2 9 8 3 6	38	56	1,031,641,140
73	6 14 10 5 15 8 7 1 3 4 2 0 12 9 11 13	37	49	3,222,276
74	14 13 4 11 15 8 6 9 0 7 3 1 2 10 12 5	46	56	1,897,728
75	14 4 0 10 6 5 1 3 9 2 13 15 12 7 8 11	30	48	42,772,589
76	15 10 8 3 0 6 9 5 1 14 13 11 7 2 12 4	41	57	126,638,417
77	0 13 2 4 12 14 6 9 15 1 10 3 11 5 8 7	34	54	18,918,269
78	3 14 13 6 4 15 8 9 5 12 10 0 2 7 1 11	41	53	10,907,150
79	0 1 9 7 11 13 5 3 14 12 4 2 8 6 10 15	28	42	540,860
80	11 0 15 8 13 12 3 5 10 1 4 6 14 9 7 2	43	57	132,945,856
81	13 0 9 12 11 6 3 5 15 8 1 10 4 14 2 7	39	53	9,982,569
82	14 10 2 1 13 9 8 11 7 3 6 12 15 5 4 0	40	62	5,506,801,123
83	12 3 9 1 4 5 10 2 6 11 15 0 14 7 13 8	31	49	65,533,432
84	15 8 10 7 0 12 14 1 5 9 6 3 13 11 4 2	37	55	106,074,303
85	4 7 13 10 1 2 9 6 12 8 14 5 3 0 11 15	32	44	2,725,456
86	6 0 5 10 11 12 9 2 1 7 4 3 14 8 13 15	35	45	2,304,426
87	9 5 11 10 13 0 2 1 8 6 14 12 4 7 3 15	34	52	64,926,494
88	15 2 12 11 14 13 9 5 1 3 8 7 0 10 6 4	43	65	6,009,130,748
89	11 1 7 4 10 13 3 8 9 14 0 15 6 5 2 12	36	54	166,571,097
90	5 4 7 1 11 12 14 15 10 13 8 6 2 0 9 3	36	50	7,171,137
91	9 7 5 2 14 15 12 10 11 3 6 1 8 13 0 4	41	57	602,886,858
92	3 2 7 9 0 15 12 4 6 11 5 14 8 13 10 1	37	57	1,101,072,541
93	13 9 14 6 12 8 1 2 3 4 0 7 5 10 11 15	34	46	1,599,909
94	5 7 11 8 0 14 9 13 10 12 3 15 6 1 4 2	45	53	1,337,340
95	4 3 6 13 7 15 9 0 10 5 8 11 2 12 1 14	34	50	7,115,967
96	1 7 15 14 2 6 4 9 12 11 13 3 0 8 5 10	35	49	12,808,564
97	9 14 5 7 8 15 1 2 10 4 13 6 12 0 11 3	32	44	1,002,927
98	0 11 3 12 5 2 1 9 8 10 14 15 7 4 13 6	34	54	183,526,883
99	7 15 4 0 10 9 2 5 12 11 13 6 1 3 14 8	39	57	83,477,694
100	11 4 0 8 6 10 5 13 12 7 14 3 1 2 9 15	38	54	67,880,056

LEGEND

GOAL STATE

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

ESTIMATE   Initial heuristic estimate
ACTUAL   Length of optimal solution
TOTAL NODES Total number of states generated

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

7. Two-Person Games

In the discussion so far, we have assumed a single-agent search to find a solution to a problem, and have been concerned with minimizing time and space subject to a fixed solution depth and branching factor. However, a

two-person game such as chess with static evaluation and mini-max search is a somewhat different situation. In this case, we assume that accuracy of the static evaluation increases with increasing search depth, and hence we want to maximize search depth subject to fixed time and space constraints. Since depth-first iterative-deepening minimizes, at least asymptotically, time and space for any given search depth, it follows that it maximizes the depth of search possible for any fixed time and space restrictions as well.

Another reason that DFID is used in game programs is that the amount of time required to search the next deeper level in the tree is not known when the ply begins, and the search ply may have to be aborted due to time constraints. In this case, the complete search at the next shallower depth can be used to make the move.

Finally, the information from previous iterations of a DFID search can be used to order the nodes in the search tree so that alpha-beta cutoff is more efficient. In fact, the best move at a given iteration has been shown experimentally to terminate the next iteration in about 70% of cases. This improvement in ordering, which is critical to alpha-beta efficiency, is only possible with the use of iterative-deepening [4].

8. Conclusions

The standard algorithms for brute-force search have serious drawbacks. Breadth-first search uses too much space, and depth-first search in general uses too much time and is not guaranteed to find a shortest path to a solution. The depth-first iterative-deepening algorithm, however, is asymptotically optimal in terms of cost of solution, running time, and space required for brute-force tree searches. DFID can also be applied to bi-directional search, heuristic best-first search, and two-person game searches. Since almost all heuristic searches have exponential complexity, iterative-deepening-A* is an optimal admissible tree search in practice. For example, IDA* is the only known algorithm that can find optimal paths for randomly generated instances of the Fifteen Puzzle within practical time and space constraints.

REFERENCES

1. Barr, A. and Feigenbaum, E.A. (Eds.), *Handbook of Artificial Intelligence* (Kaufmann, Los Altos, CA, 1981).

2. Berliner, H., Search, Artificial Intelligence Syllabus, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1983.
3. Berliner, H. and Goetsch, G., A quantitative study of search methods and the effect of constraint satisfaction, Tech. Rept. CMU-CS-84-147, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1984.
4. Berliner, H., Personal communication, 1984.
5. Dechter, R. and Pearl, J., The optimality of A* revisited, in: *Proceedings of the National Conference on Artificial Intelligence*, Washington, DC (August, 1983) 95–99.
6. Hart, P.E., Nilsson, N.J. and Raphael, B., A formal basis for the heuristic determination of minimum cost paths, *IEEE Trans. Systems Sci. Cybernet.* 4(2) (1968) 100–107.
7. Hopcroft, J.E. and Ullman, J.D., *Introduction to Automata Theory, Languages, and Computation* (Addison-Wesley, Reading, MA, 1979).
8. Korf, R.E., *Learning to Solve Problems by Searching for Macro-Operators* (Pittman, London, 1985).
9. Newell, A. and Simon, H.A., *Human Problem Solving* (Prentice-Hall, Englewood Cliffs, NJ, 1972).
10. Nilsson, N.J., *Principles of Artificial Intelligence* (Tioga, Palo Alto, CA, 1980).
11. Pearl, J., *Heuristics* (Addison-Wesley, Reading, MA, 1984).
12. Pearl, J., Personal communication, 1984.
13. Pohl, I., Bi-directional search, in: B. Meltzer and D. Michie (Eds.), *Machine Intelligence 6* (American Elsevier, New York, 1971) 127–140.
14. Rich, E., *Artificial Intelligence* (McGraw-Hill, New York, 1983).
15. Slate, D.J. and Atkin, L.R., *CHESS 4.5 – The Northwestern University Chess Program* (Springer-Verlag, New York, 1977).
16. Winston, P.H., *Artificial Intelligence* (Addison-Wesley, Reading, MA, 1984).
17. Mérő, L., A heuristic search algorithm with modifiable estimate, *Artificial Intelligence* 23 (1984) 13–27.
18. Korf, R.E., Iterative-deepening-A*: an optimal admissible tree search, in: *Proceedings Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, 1985.