Depth-First vs Best-First Search

Nageshwara Rao Vempaty
Dept. of Computer Sciences,
Univ. of Central Florida,
Orlando, FL - 32792.

Vipin Kumar*
Computer Science Dept.,
Univ. of Minnesota,
Minneapolis, MN - 55455.

Richard E. Korf†
Dept. of Computer Science,
Univ. of California,
Los Angeles, CA - 90024

Abstract

We present a comparison of three well known heuristic search algorithms: best-first search (BFS) , iterative-deepening (ID), and depth-first branch-and-bound (DFBB). We develop a model to analyze the time and space complexity of these three algorithms in terms of the *heuristic branching factor* and *solution density*. Our analysis identifies the types of problems on which each of the search algorithms performs better than the other two. These analytical results are validated through experiments on different problems. We also present a new algorithm, DFS*, which is a hybrid of iterative deepening and depth-first branch-and-bound, and show that it outperforms the other three algorithms on some problems.

Introduction

Heuristic search algorithms are used to solve a wide variety of combinatorial optimization problems. Three important algorithms are: (i) best-first search (BFS); (ii) iterative-deepening (ID)[Korf, 1985]; and (iii) depth-first branch-and-bound (DFBB)[Lawler and Woods, 1966; Kumar, 1987]. The problem is to find a path of least cost from an initial node to a goal node, in an implicitly specified state-space tree, for which a consistent admissible cost function is available.

Best-first search (BFS) is a generic algorithm that expands nodes in non-decreasing order of cost. Different cost functions $f(n)$ give rise to different variants. For example, if $f(n) = depth(n)$, then best-first search becomes breadth-first search. If $f(n) = g(n)$, where $g(n)$ is the cost of the path from the root to node n, then best-first search becomes Dijkstra's single-source shortest-path algorithm[Dijkstra, 1959]. If $f(n) = g(n) + h(n)$, where $h(n)$ is the heuristic estimate of the cost of the path from node n to a goal, then best-first search becomes A*[Hart *et al.*, 1968].

Given a consistent, non-overestimating cost function, best-first search expands the minimum number of nodes necessary to find an optimal solution, up to tie-breaking among nodes whose cost equals the goal cost [Dechter and Pearl, 1985]. The storage requirement of BFS, however, is linear in the number of nodes expanded. As a result, even for moderately difficult instances of many problems, BFS runs out of memory very quickly . For example, for the 15-puzzle problem, A* runs out of memory within a few minutes of run time on a SUN 3/50 workstation with 4 Megabytes of memory.

Iterative deepening (ID)[Korf, 1985] was designed to remedy this problem. It is based on depth-first search, which only maintains the current path from the root to the current node, and hence uses space that is only linear in the search depth. ID performs a series of depth-first searches, in which a branch is pruned when the cost of a node on that path exceeds a cost threshold for that iteration. The cost threshold for the first iteration is the cost of the root node, and the threshold for each succeeding iteration is the minimum cost value that exceeded the threshold on the previous iteration. The algorithm terminates when a goal is found whose cost does not exceed the current threshold. Since the cost bound used in each iteration is a lower bound on actual cost, the first solution chosen for expansion is optimal. Special cases of iterative deepening include depth-first iterative-deepening (DFID), where $f(n) = depth(n)$, and iterative-deepening-A* (IDA*), where $f(n) = g(n) + h(n)$.

Clearly, ID expands more nodes than BFS, since all the nodes expanded in one iteration are also expanded in all following iterations. Define the *heuristic branching factor (b)* of a problem to be the ratio of the number of nodes of a given cost to the number of nodes with the next smaller cost. For example, if cost is simply depth, then the heuristic branching factor is the well-known brute-force branching factor. If the heuristic branching factor is greater than one, meaning that the tree grows exponentially with cost, then IDA* generates asymp-

*This research was supported by Army Research Office grant # 28408-MA-SDI to the University of Minnesota and by the Army High Performance Computing Research Center at the University of Minnesota.

†This research was supported by an NSF Presidential Young Investigator Award, and a grant from Rockwell International.

totically the same number of nodes as A*[Korf, 1985].

The problem occurs when b is less than one or very close to one. In the former case, where the size of the problem space does not grow exponentially with cost, ID generates asymptotically more nodes than BFS. In fact, in the worst case, where every node has a unique cost value, ID generates $O(M^2)$ nodes where M is the number of nodes generated by BFS[Patrick *et al.*, 1991]. If b is greater than but close to one, while asymptotically optimal, ID will be very inefficient in practice, compared to BFS. This occurs in problems such as the Traveling Salesperson Problem (TSP) and VLSI floorplan optimization. For example, on a small instance of the TSP which could be solved by A^* in a few minutes, IDA* ran for several days without finding a solution. Ideally, we would like an algorithm with both low space and time requirements.

Depth-first branch-and-bound (DFBB)[Lawler and Woods, 1966] is a potential candidate. DFBB starts with an upper bound on the cost of an optimal solution, and then searches the entire space in a depth-first fashion. Whenever a new solution is found whose cost is lower than the best one found so far, the upper bound is revised to the cost of this new solution. Whenever a partial solution is encountered whose cost equals or exceeds the current bound, it is eliminated. Note that DFBB and ID are complementary to each other, in that DFBB starts with an upper bound, and ID starts with an lower bound. Both can expand more nodes than BFS. ID performs repeated expansion of nodes, while DFBB expands each node exactly once, but expands nodes costlier than the optimal solution cost. Since the node selection strategy in both DFBB and ID is depth-first, both have low memory requirements and a much faster node expansion rate compared with A^*.

There are two main reasons why the time needed by BFS to expand a node is much larger than that of depth-first search algorithms such as ID and DFBB. First, each time a node is expanded by BFS, a priority queue has to be accessed to remove the node and to insert its descendants, multiplying the node expansion time by a logarithmic factor. Second, in the depth-first algorithms, successor nodes can often be constructed by making simple changes in the current parent node, and the parent can be reconstructed by simply undoing those changes while backtracking. This optimization is not directly implementable in BFS. For example, in the $N \times N$ sliding tile puzzles, such as the Fifteen Puzzle, the time taken to expand a node for ID and DFBB is $O(1)$ while it is $O(N^2)$ for BFS, just to make a copy of the state.

Given these three algorithms, we address two questions in this paper: 1) What are the characteristics of problems for which one of the algorithms is better than the others?, and 2) Are there additional algorithms that are memory efficient and may be better for some classes of problems?

We show that for problems with high solution densities, DFBB asymptotically expands the same number of nodes as BFS, and outperforms ID. For problems with low solution densities, ID beats DFBB. Finally, when both the solution density and the heuristic branching factor are low, both DFBB and ID perform poorly. For this type of problem, we propose a hybrid of the two algorithms, DFS*, and demonstrate its effectiveness on a natural problem.

We experimentally verified our analysis on three different problems: the Fifteen Puzzle, Traveling Salesperson Problem (TSP) and solving mazes. We implemented IDA*, DFBB and A^* algorithms to solve each of these problems. Our experimental investigation showed that the Fifteen Puzzle has a low solution density, but a high heuristic branching factor. Conversely, TSP has high solution density and low heuristic branching factor. Comparison of run times of the algorithms shows that ID is superior on the Fifteen Puzzle, and DFBB is superior on TSP. BFS is poor on both problems because of high memory requirements. The maze problem has both low heuristic branching factor and low solution density. Hence, this problem is unfavorable to both ID and DFBB algorithms, and the hybrid algorithm, DFS*, outperforms them on this problem. Thus our experimental results support the theoretical analysis.

Analysis of Search Algorithms

Assumptions and Definitions

We restrict our analysis to state-space trees. For each node n, $f(n)$ denotes a lower bound on the cost of solutions that include node n. The cost function f is monotonic, in the sense that $f(n) \leq f(m)$ if m is a successor of n. Let N be the number of different values taken by f over all the nodes of the tree. Let V_i denote the set of nodes whose cost is the ith-smallest of the f-values. Thus $V_0, V_1, \cdots V_{N-1}$ is a sequence of sets of nodes in increasing order of cost. Clearly, the children of a node in V_i can only occur in those V_j for which $j \geq i$. V_0 contains the start node. We assume that the sequence of sizes $|V_i|$ of the node sets is a geometric progression with ratio b, the heuristic branching factor[Korf, 1988]. If we assume that V_0 is a singleton set, then $|V_i| = b^i$. We assume that $b > 1$.[1]

Let V_d be the set of nodes that contains the optimal solution(s). Hence, there are no solutions in V_i for $i < d$. Furthermore, we assume that each element of V_d is a solution with probability ρ_0, and in successive V_{d+i}'s, the probability of a node being a solution is ρ_i. Thus the sequence of ρ_i's is a measure of the density of solutions in successive search frontiers. We assume that the solutions follow a Bernoulli distribution among the elements of each V_i for $i \geq d$. Therefore, the average number of nodes expanded in V_{d+i} before

[1] If $b \leq 1$, then ID would perform quite poorly, and one would choose between DFBB and A^*.

a solution is found from this set is $\frac{1}{\rho_i}$. Since each V_{d+i} has at least one solution, $\rho_i > \frac{1}{b^{d+i}}$. For simplicity of presentation, we make the assumption that all the algorithms expand all the nodes in V_d, in order to find the optimal solution. Similar results can also be obtained under either of the following alternate assumptions: (i) all algorithms expand exactly one node in V_d (i.e., the first node searched in V_d is a solution); (ii) the average number of nodes expanded by all algorithms in V_d is $\frac{1}{\rho_0}$.

Analysis of Best-First Search

Best-first search expands all the nodes in V_i for $i \leq d$. Let M denote the number of nodes expanded by BFS. We have

$$M = \sum_{i=0}^{i=d} |V_i|$$

$$= \sum_{i=0}^{i=d} b^i$$

$$M = \frac{b^{d+1} - 1}{b - 1} \; for \; b > 1 \qquad (1)$$

The above formula denotes the 'mandatory nodes' in our search tree. These are the nodes that have to be expanded by any algorithm that finds an optimal solution.

Analysis of Iterative Deepening

Iterative deepening reexpands nodes in prior iterations during later iterations, but does not expand any nodes costlier than the optimal solution. Let DI denote the average number of nodes expanded by ID. The algorithm starts with an initial bound equal to the cost of the root.

$$DI = \sum_{i=0}^{i=d} \sum_{j=0}^{j=i} |V_j|$$

The inner sum adds the nodes expanded in a given iteration, while the outer sum adds up the different iterations.

$$DI = \sum_{i=0}^{i=d} \frac{b^{i+1} - 1}{b - 1} \; since \; b > 1$$

$$= \frac{b}{b-1} \frac{b^{d+1} - 1}{b - 1} - \frac{d}{b-1}$$

$$DI \approx \frac{b}{b-1} M \qquad (2)$$

A similar result was proved in [Korf, 1985; Stickel and Tyson, 1985]. It is clear from this equation that when $b > 1$, ID expands asymptotically the same number of nodes as BFS, and that when b is close to one, ID expands many more nodes than BFS.

Analysis of Depth-First Branch-and-Bound

Depth-first branch and bound starts with an upper bound on the cost of the optimal solution, and decreases it whenever a better solution is found. Eventually, the bound equals the cost of the optimal solution and then only mandatory nodes are expanded. While DFBB may expand nodes costlier than the optimal solution, it never expands a node more than once. Let DB denote the number of nodes expanded by DFBB. These nodes fall into two disjoint categories - (i) those which are costlier than the optimal solution(s) and hence lie in V_i for $i > d$ and (ii) those which are not costlier than the optimal solution(s) and lie in V_j for $j \leq d$. The average number of nodes in the second category is M. The initial bound with which DFBB starts is quite important. It is generated by using a fast approximation algorithm. In problems like floor plan optimization and planar TSP, the initial bound is often within twice the cost of the optimal solution. We assume that the initial bound gives the cost of nodes at level $kd+1$, where d is the level containing the optimal solution(s). (Note that k need not be an integer, and is a measure of accuracy of the approximation algorithm used.) Hence nodes in the first category belong to V_j for $d < j \leq kd$. Each of these V_j's contain b^j nodes, out of which approximately $b^j \rho_{j-d}$ are solutions. Let B_i denote the average number of nodes expanded by DFBB from V_{d+i} for $0 \leq i \leq kd$. We have $B_0 = |V_d|$, and

$$for \; 1 \leq i \leq kd, \quad B_i = \min\{bB_{i-1}, \frac{1}{\rho_i}\}$$

This says that either $\frac{1}{\rho_i}$ nodes are expanded from V_{d+i} before a solution is found at this level, or a solution is found earlier at level V_{d+i-1} itself. In either case, no more nodes are expanded from level V_{d+i}. Hence,

$$B_i \leq \frac{1}{\rho_i}$$

Using this, we can derive the following -

$$DB \leq M + \sum_{i=1}^{i=kd} B_i$$

$$DB \leq M + \sum_{i=1}^{i=kd} \frac{1}{\rho_i} \qquad (3)$$

Clearly, the behavior of DFBB depends on the sequence of ρ_i's. It is always the case that $M \leq DB$. Let ρ denote the harmonic mean of the sequence $\rho_1, \rho_2 \cdots \rho_{kd}$. Thus the sum $\sum_{i=1}^{i=kd} \frac{1}{\rho_i}$ is equal to $\frac{kd}{\rho}$.[2]

[2]Harmonic mean $\rho = \frac{kd}{\sum_{i=1}^{i=kd} \frac{1}{\rho_i}}$. For ρ to be well defined, we need to have $0 < \rho_i \leq 1.0$.

We are interested in sequences for which DB is close to M. From equation 3, it is clear that a sufficient condition for $DB \leq 2M$ is $\frac{kd}{M} \leq \rho$. The harmonic mean, ρ, is a measure of the overall density of solutions and strongly influences the running time of DFBB.

An interesting sequence of ρ_i's is the case for which the number of solutions increase exponentially in successive levels, as do nodes. Consider the case where V_d has $s(= b^d\rho_0)$ solutions, and in every successive level, the number of solutions increase by a factor of s. This is the case for which $\rho_i = MIN(\frac{(s)^{i+1}}{b^{d+i}}, 1.0)$. For $s > 1$, the reader can verify that

$$B_i = \frac{1}{\rho_i}$$

For this type of problem, we have

$$DB = M + \frac{b^{d+1}(1 - (\frac{b}{s})^{kd})}{s(s-b)}) \qquad (4)$$

From 4, we can see that DB is close to M when $s \geq 2b$. In this case DB is not very sensitive to k or d. When s decreases from $2b$ to b, DB gradually increases and also becomes more sensitive to kd. For $s \leq b$, DB is much larger than M unless kd is very small. Thus DFBB performs well when the number of solutions grows more than twice as fast as the number of nodes. It performs poorly when the number of solutions at successive cost levels grows slower than the number of nodes.

Comparison of the Algorithms

The space complexity of BFS is $O(M)$, while for the depth-first strategies it is $O(d)$. Using equations 1, 2, 3 and 4, we can analyze the relative time complexities of each of the algorithms.

As pointed out earlier, node expansion time in the depth-first search algorithms, ID and DFBB, is usually much smaller than that for best-first search. Let r be the ratio of the node expansion time for best-first search compared to depth-first search. Typical values of r in our experiments range from 1 to 10. For any particular value of r, we can find combinations of b and ρ for which one of the algorithms dominates the other two, in terms of time.

1. *DFBB vs BFS.* DFBB runs faster than BFS when $DB \leq rM$. For small values of r (such as 2), this will be true when the number of solutions grows at least twice as fast as the number of nodes. BFS runs faster than DFBB when the number of solutions grows slower than the number of nodes. Note that BFS is impractical when M exceeds the available memory.
2. *BFS vs ID.* ID runs faster than BFS when $DI \leq rM$. This will be true roughly when $\frac{b}{b-1} \leq r$. Otherwise BFS will run faster than ID. Again BFS may still be impractical to use due to memory limits.
3. *ID vs DFBB.* ID runs faster than DFBB, when $b > 2$ and $s < b$. DFBB runs faster than ID when $b < 2$ and $s > 2b$. When $b < 2$ and $s < b$, both algorithms will perform poorly. For other cases, there is a transition between the two algorithms.

To summarize the results of the above analysis -

- DFBB is preferable when the increase in solution density is larger than the heuristic branching factor.
- ID is preferable when the heuristic branching factor is high and density of solutions is low.
- BFS is useful only when both density of solutions and heuristic branching factor are very very low.

Experimental Results.

We chose three problems to experimentally validate our results - the Fifteen Puzzle, Traveling Salesperson Problem (TSP), and maze search. They have different heuristic branching factors and solution densities.

The Fifteen Puzzle is a classical search example. It consists of a 4×4 square with 15 numbered square tiles and a blank position. The legal operators are to slide any tile horizontally or vertically adjacent to the blank position into the blank position. The task is to map an arbitrary initial configuration into a particular goal configuration, using a minimum number of moves. A common heuristic function for this problem is called Manhattan Distance: it is computed by determining the number of grid units each tile is away from its goal position, and summing these values over all tiles. IDA*, using the Manhattan Distance heuristic, is capable of finding optimal solutions to randomly generated Fifteen Puzzle problem instances within practical resource limits[Korf, 1985]. A* is completely impractical due to the memory required, up to six billion nodes in some cases. In addition, IDA* runs faster than A*, due to reduced overhead per node generation, even though it generates more nodes.

We compared IDA* and DFBB, on the ten easiest problems from [Korf, 1985], based on nodes generated by IDA*. For the initial bound in DFBB, we used twice the Manhattan Distance of the initial state. Table 1 shows that DFBB generates many times more nodes than IDA*. Their running times per node generation are roughly equal. The average heuristic branching factor of the Fifteen Puzzle is about 6, which is relatively high. The solution density is quite low, and actually decreases slightly as we go deeper into the search space. This explains why IDA* performs very well, while DFBB and A^* perform poorly.

The Traveling Saleperson Problem (TSP) is to find a shortest tour among a set of cities, ending where it started. Each city needs to be visited exactly once in the tour. We compared all three algorithms on the euclidean TSP. Between 10 and 15 cities were randomly located within a square, 2^{15} units on a side, since our random number generator produced 16 bit random numbers. A partial contiguous tour was extended

by adding cities to its end city, ordered by the nearest neighbor heuristic. The minimum spanning tree of the remaining cities, plus the two end cities of the current partial tour, was used as the heuristic evaluation function. Each data point in table 2 is an average over 100 randomly generated problem instances. The first column gives the number of cities. The second column gives the cost of an optimal tour and third column gives the number of mandatory nodes, or the number of nodes generated by A*. The fourth and fifth columns give the number of nodes generated by DFBB and IDA*, respectively. No data is given for IDA* for 13 through 15 cities, since it took too long to generate. Finally, the last column gives the ratio of the number of nodes generated by IDA* to the number of nodes generated by DFBB. The data demonstrates that DFBB is quite effective on this problem, generating only 10 to 20% more nodes than necessary. This is due to the high solution density, since at a depth equal to the number of cities, every node is a solution. The data also shows that IDA* performs very poorly on this problem, generating hundreds of times more nodes than DFBB. This is due to the low heuristic branching factor, since there are relatively few ties among nodes with the same cost value. Similar results were observed for the Floorplan Optimization Problem, using the best known heuristic functions in [Wimer *et al.*, 1988].

A new search algorithm : DFS*

Our discussion so far suggests that DFBB and ID are complementary to each other. ID starts with a lower bound on cost, and increases it until it is equal to the optimal cost. DFBB starts with a upper bound on cost, and decreases it until it is equal to the optimal cost. Since ID conservatively increases bounds, it does not expand any nodes costlier than the optimal solution, but it may repeat work if the heuristic branching factor is low. DFBB does not repeat work, but expands nodes costlier than the optimal solution. Such wasteful node expansion is high when the initial bound it starts with is much higher than the final cost, and if the solution density is low.

This suggests a hybrid algorithm, which we call DFS* to suggest a depth-first algorithm that is admissible. DFS* initially behaves like iterative deepening, but increases the cost bounds more liberally than necessary, to minimize repeated node expansions [Korf, 1988]. When a solution is found that is not known to be optimal, DFS* then switches over to the DFBB algorithm. The DFBB phase starts with the cost of this solution as its initial bound and continues searching, reducing the upper bound as better solutions are found. Also, if the cost bound selected in any iteration of the ID phase is greater than an alternate upper-bound, which may be available by other means, then DFS* switches over to the DFBB algorithm. A very similar algorithm, called MIDA*, was independently discovered by Benjamin Wah[Wah, 1991].

DFS* is a depth-first search strategy and it finds optimal solutions given non-overestimating heuristics. DFS* may be useful on certain problems where both DFBB and ID perform poorly. For example when both the heuristic branching factor and solution density are low ($b < 2$ and $s < 2b$), DFS* can perform well provided reasonable increments in bounds can be found.

Define B as the ratio between the number of nodes first generated by successive iterations of ID. If we set successive thresholds to the minimum costs that exceeded the previous iteration, then $B = b$, the heuristic branching factor. By manipulating the threshold increments in DFS*, we can change the value of B. Too low a value of B results in too much repeated work in early iterations. Too high a value of B results in too much extra work in the final iteration generating nodes with higher costs than the optimal solution cost. What value of B produces optimal performance, relative to BFS, in the worst case?

Let d be the first cost level that contains an optimal solution. In the worst case for DFS*, BFS will not expand any nodes at level d, but all nodes at level $d-1$. The number of such nodes is approximately $B^d/(B-1)$ Similarly, in the worst case, DFS* will expand all nodes at level d. Thus DFS* expands approximately $B^d * (B^2/(B-1)^2)$. The ratio of the nodes expanded by DFS* to the nodes expanded by BFS is $B^2/(B-1)$. Taking the derivative of this function with respect to B gives us $B(B-2)/(B-1)^2$. Setting this derivative equal to zero and solving for B gives us B=2. In other words, to optimize the ratio of the nodes generated by DFS* to BFS in the worst case, we'd like B to be 2. If we substitute $B = 2$ back into $B^2/(B-1)$, we get a ratio of 4. In other words, if $B = 2$, then in the worst case, the ratio of DFS* to BFS will be only 4. This analysis was motivated by the formulation of the problem presented in [Wah, 1991].

To achieve this value of B, the approximate increment in cost can be estimated by sampling the distribution of nodes across a cost range during an iteration, as follows. We divide the total cost range between 0 and maxcost into several parts, and associate a counter with each range. Each counter keeps track of the number of nodes generated in the corresponding cost range. Any time a node is generated and its cost computed, the appropriate counter is incremented. This data can be used to find a cost increment as close as possible to the desired increase in the number of nodes expanded.

A much simpler, though less effective heuristic, would be to increment successive thresholds to the maximum value that exceeded the previous threshold. This guarantees a value of B that is at least as large as the brute-force branching factor.

To evaluate DFS* empirically, we considered the problem of finding the shortest path between two points in a maze. This problem models the task of navigation in the presence of obstacles. We imple-

mented IDA*, DFBB, A^* and DFS*, and tested them on 120×90 mazes, all of which were drawn randomly by the Xwindows demo package Xmaze. Figure shows an example of a maze. The manhattan distance heuristic was used to guide the search. For this problem the heuristic branching factor is typically low, as is the solution density. The starting nodes were close to centers of the mazes, and a series of experiments were performed, each with the goal node being farther away from the start node. When the goal node is not too far away, the boundary walls are not encountered often during the search, minimizing boundary effects. Table 3, summarizes the number of nodes expanded by each algorithm, averaged over 1000 randomly generated problem instances. In these experiments, the cost bound for DFS* was doubled after each iteration. DFS* outperformed the other depth-first algorithms, as predicted by our analysis, and performed close to A* on these mazes. The space requirements of A^* are very high; it requires 1 MByte of memory for handling a 200×200 maze.

Conclusions

We analyzed and compared three important heuristic search algorithms, DFBB, ID and BFS, and identified their domain of effectiveness in terms of heuristic branching factor and solution density. DFBB is the best when solution density is high. ID is the best when heuristic branching factor is high. Since both of them use a depth-first search strategy, they overcome the memory limitations of BFS and hence can solve larger problems. We also identified a natural relation between them and presented a new hybrid depth-first search algorithm DFS*, that is suitable when both heuristic branching factor and solution density are low. We experimentally demonstrated these results on three natural problems.

References

Dechter, R. and Pearl, J. 1985. Generalized best-first search strategies and the optimality of a*. *Journal of the Association for Computing Machinery* Vol. 32, No. 3:505–536.

Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik* Vol. 1:269–271.

Hart, R.E.; Nilsson, N.J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* Vol. 4, No. 2:100–107.

Korf, Richard E. 1985. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence* 27:97–109.

Korf, Richard 1988. Optimal path finding algorithms. In Kanal, Laveen and Kumar, Vipin, editors 1988, *Search in Artificial Intelligence*. Springer-Verlag, New York.

Kumar, Vipin 1987. Branch-and-bound search. In Shapiro, Stuart C., editor 1987, *Encyclopaedia of Artificial Intelligence: Vol 2.* John Wiley and Sons, Inc., New York. 1000–1004.

Lawler, E. L. and Woods, D. 1966. Branch-and-bound methods: A survey. *Operations Research* 14.

Patrick, B.G.; Almulla, M.; and Newborn, M.M. 1991. An upper bound on the complexity of iterative-deepening-a*. *Annals of Mathematics and Artificial Intelligence* To Appear.

Stickel, M.E. and Tyson, W.M. 1985. An analysis of consecutively bounded depth-first search with applications in automated deduction. In *IJCAI.* 1073–1075.

Wah, Benjamin W. 1991. Mida*: An ida* search with dynamic control. Technical report, Coordinated Science Laboratory, University of Illinois, Urbana, Ill.

Wimer, S.; Koren, I.; and Cederbaum, I. 1988. Optimal aspect rations of building blocks in vlsi. In *25th ACM/IEEE Design Automation Conference.* 66–72.

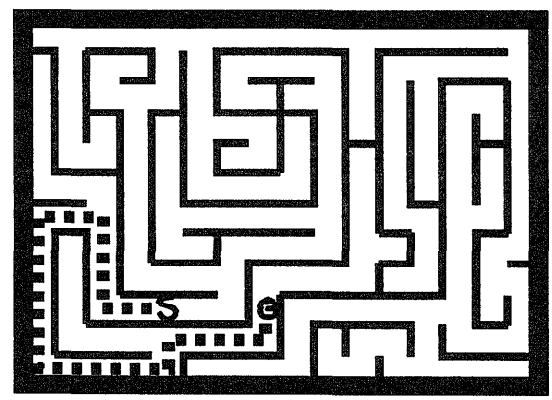

Figure 1: Example of a maze. S is the starting point. G is the goal. The path $\cdots$ is the shortest solution.

Prob No	*Sol. Cost h**	*ID Nodes*	*DFBB Nodes*	Ratio DFBB/ID
79	42	540860	27895080	52
12	45	546344	8812886	16
42	42	877823	37744947	43
55	41	927212	7424666	8
97	44	1002927	13833342	14
19	46	1280495	146316234	114
94	53	1337340	391836062	293
47	47	1411294	91036843	65
93	46	1599909	11343626	7
9	46	1650696	19976208	12

Table 1: Experimental results on 15-puzzle.

Number of cities	*Optimal Sol. Cost*	*A* Nodes expanded*	*DFBB Nodes expanded*	*ID nodes expanded*	*ID/DFBB ratio*
10	93421	1408	1552	325575	210
11	97493	2843	3126	1952350	625
12	100511	5007	5576	5084812	912
13	103834	6806	8163	NA	-
14	107524	16849	19133	NA	-
15	111084	45211	49833	NA	-

Table 2: Experimental results on the Traveling Salesperson Problem. Each row shows the average value over 100 runs. The entries indicated NA mean that the experiment was abandoned because it takes too long.

Sol. Cost Range	*DFBB Nodes expanded*	*ID Nodes expanded*	*DFS* Nodes expanded*	*A* nodes expanded*
5 – 15	4535	23	29	15
15 – 50	4571	322	112	55
50 – 100	4992	1915	290	143
100 – 200	4817	8972	624	314
200 – 500	5413	58585	1901	905

Table 3: Experimental results for finding optimal routes in mazes. The data for each cost range was obtained by averaging over 1000 randomly generated mazes.