Depth-First vs. Best-First Search: New Results *

Weixiong Zhang and *Richard E. Korf*
Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90024
zhang@cs.ucla.edu, korf@cs.ucla.edu

ABSTRACT

Best-first search (BFS) expands the fewest nodes among all admissible algorithms using the same cost function, but typically requires exponential space. Depth-first search needs space only linear in the maximum search depth, but expands more nodes than BFS. Using a random tree, we analytically show that the expected number of nodes expanded by depth-first branch-and-bound (DFBnB) is no more than $O(d \cdot N)$, where d is the goal depth and N is the expected number of nodes expanded by BFS. We also show that DFBnB is asymptotically optimal when BFS runs in exponential time. We then consider how to select a linear-space search algorithm, from among DFBnB, iterative-deepening (ID) and recursive best first search (RBFS). Our experimental results indicate that DFBnB is preferable on problems that can be represented by bounded-depth trees and require exponential computation; and RBFS should be applied to problems that cannot be represented by bounded-depth trees, or problems that can be solved in polynomial time.

1 Introduction and Overview

A major factor affecting the applicability of a search algorithm is its memory requirement. If a problem is small, and the available memory is large enough, then best-first search (BFS) may be used. BFS maintains a partially expanded search graph, and expands a minimum-cost frontier node at each cycle until an optimal goal node is chosen for expansion. One important property of BFS is that it expands the minimum number of nodes among all admissible algorithms using the same cost function [2]. However, it typically requires exponential space, making it impractical for most applications.

Practical algorithms use space that is only linear in the maximum search depth. Linear-space algorithms include depth-first branch-and-bound (DFBnB), iterative-deepening [4], and recursive best-first search [6, 7]. DFBnB starts with an upper bound on the cost of an optimal goal, and then searches the entire state space in a depth-first fashion. Whenever a new solution is found whose cost is less 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 is greater than or equal to the current upper bound, it is pruned. DFBnB expands more nodes than BFS. In particular, when the cost function is monotonic, in the sense that the cost of a child is always greater than or equal to the cost of its parent, DFBnB may expand nodes whose costs are greater than the optimal goal cost, none of which are explored by BFS.

To avoid expanding nodes that are not visited by BFS, iterative-deepening (ID) [4] may be adopted. It runs a series of depth-first iterations, each bounded by a cost threshold. In each iteration, a branch is eliminated when the cost of a node on that path exceeds the cost threshold for that iteration. When the cost function is not monotonic, however, ID may not expand newly visited nodes in best-first order.

In this case, recursive best-first search (RBFS) [6, 7] may be applied, which always expands unexplored nodes in best-first order, using only linear space, (cf. [6, 7] for details.) Another advantage of RBFS over ID is that the former expands fewer nodes than the latter, up to tie-breaking, when the cost function is monotonic. Both ID and RBFS suffer from the overhead of expanding many nodes more than once.

Some of these algorithms have been compared before. Wah and Yu [14] argued that DFBnB is comparable to BFS if the cost function is very accurate or very inaccurate. Using an abstract model in which the number of nodes with a given cost grows geometrically, Vempaty *et al* [13] compared BFS, DFBnB and ID.

*This research was supported by NSF Grant No. IRI-9119825, a grant from Rockwell International, and a GTE fellowship.

Their results are based on the solution density, the ratio of the number of goal nodes to the total number of nodes with the same cost as the goal nodes, and the heuristic branching factor, the ratio of the number of nodes with a given cost to the number of nodes of the next smaller cost. They concluded that: (a) DFBnB is preferable when the solution density grows faster than the heuristic branching factor; (b) ID is preferable when the heuristic branching factor is high and the solution density is low; (c) BFS is useful only when both the solution density and the heuristic branching factor are very low, provided that sufficient memory is available. Using a random tree, in which edges have random costs, and the cost of a node is the sum of the costs of the edges on the path from the root to the node, Karp and Pearl[3], and McDiarmid and Provan [10, 11] showed that BFS expands either an exponential or quadratic number of nodes in the search depth, depending on certain properties. On a random tree with uniform branching factor and discrete edge costs, we argued in [15] that DFBnB also runs in polynomial time when BFS runs in quadratic time. Kumar originally observed that ID performs poorly on the traveling salesman problem (TSP), compared to DFBnB [13] (cf. Section 4.2 as well).

Although DFBnB is very useful for problems such as the TSP, it is not known how many more nodes DFBnB expands than BFS on average. Using a random tree, we analytically show that the expected number of nodes expanded by DFBnB is no more than $O(d \cdot N)$, where d is the goal depth and N is the expected number of nodes expanded by BFS (Section 2). We compare BFS, DFBnB, ID and RBFS under the tree model, and demonstrate that DFBnB runs faster than BFS in some cases, even if the former expands more nodes than the latter (Section 3). The purpose is to provide a guideline for selecting algorithms for given problems. Finally, we consider how to choose linear-space algorithms for two applications, lookahead search on sliding-tile puzzles, and the asymmetric TSP (Section 4). Our results in Sections 2 and 3 are included in [16].

2 Analytic Results: DFBnB vs. BFS

Search in a state space is a general model for problem solving. While a graph with cycles is the most general model of a state space, depth-first search explores a state space *tree*, at the cost of regenerating the same nodes arrived at via different paths. This is a fundamental difference between linear-space algorithms, which cannot detect duplicate nodes in general, and exponential-space algorithms, which can. Associated with a state space is a cost function that estimates the cost of a node. Alternatively, a cost can be associated with an edge, representing the incremental change to a node cost when the corresponding operator is applied. A node cost is then computed as the sum of the edge costs on the path from the root to the node, or the sum of the cost of its parent node and the cost of the edge from the parent to the child. Therefore, we introduce the following tree model, which is suitable for any combinatorial problem with a monotonic cost function.

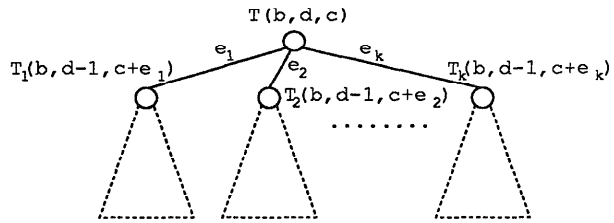

Figure 1: Recursive structure of a random tree

Definition 2.1 *A* random tree $T(b,d,c)$ *is a tree with depth d, root cost c, and independent and identically distributed random branching factors with mean b. Edge costs are independently drawn from a non-negative probability distribution. The cost of a non-root node is the sum of the edge costs from the root to that node, plus the root cost c. An* optimal goal node *is a node of minimum cost at depth d.*

Lemma 2.1 *Let $N_B(b,d)$ be the expected number of nodes expanded by BFS, and $N_D(b,d,\alpha)$ the expected number of nodes expanded by DFBnB with initial upper bound α, on $T(b,d,0)$. As $d \to \infty$,*

$$N_D(b,d,\infty) \leq (b-1)\sum_{i=1}^{d-1} N_B(b,i) + (d-1)$$

Proof: As shown in Figure 1, the root of $T(b,d,c)$ has k children, $n_1, n_2, ..., n_k$, where k is a random variable with mean b. Let e_i be the edge cost from the root of $T(b,d,c)$ to the root of its i-th subtree $T_i(b,d-1,c+e_i)$, for $i = 1,2,...,k$. The children of the root are generated all at once and sorted in nondecreasing order of their costs. Thus $e_1 \leq e_2 \leq ... \leq e_k$, arranged from left to right in Figure 1. For convenience of discussion, let $N_D(b,d,c,\alpha)$ be the expected number of nodes expanded by DFBnB on $T(b,d,c)$ with initial upper bound α. We first make the following two observations.

First, subtracting the root cost from all nodes and the upper bound has no affect on the search. Therefore, the number of nodes expanded by DFBnB on $T(b,d,c)$ with initial upper bound α is equal to those expanded on $T(b,d,0)$ with initial upper bound $\alpha - c$. That is

$$\left.\begin{array}{l} N_D(b,d,c,\alpha) = N_D(b,d,0,\alpha-c) \\ N_D(b,d,c,\infty) = N_D(b,d,0,\infty) \end{array}\right\} \quad (1)$$

Secondly, because a larger initial upper bound causes at least as many nodes to be expanded as a smaller upper bound, the number of nodes expanded by DFBnB on $T(b,d,c)$ with initial upper bound α is no less than the number expanded with initial upper bound $\alpha' \leq \alpha$. That is

$$N_D(b,d,c,\alpha') \leq N_D(b,d,c,\alpha), \quad \text{for } \alpha' \leq \alpha \quad (2)$$

Now consider DFBnB on $T(b,d,0)$. It first searches the subtree $T_1(b,d-1,e_1)$ (see Figure 1), expanding $N_D(b,d-1,e_1,\infty)$ expected number of nodes. Let ρ be the minimum goal cost of $T_1(b,d-1,0)$. Then the minimum goal cost of $T_1(b,d-1,e_1)$ is $\rho+e_1$, which is the upper bound after searching $T_1(b,d-1,e_1)$. After $T_1(b,d-1,e_1)$ is searched, subtree $T_2(b,d-1,e_2)$ will be explored if its root cost e_2 is less than the current upper bound $\rho+e_1$, and the expected number of nodes expanded is $N_D(b,d-1,e_2,\rho+e_1)$, which is also an upper bound on the expected number of nodes expanded in $T_i(b,d-1,e_i)$, for $i=3,4,...,k$. This is because the upper bound can only decrease after searching $T_2(b,d-1,e_2)$ and the edge cost e_i can only increase as i increases, both of which cause fewer nodes to be expanded. Since the root of $T(b,d,0)$ has b expected number of children, we write

$$N_D(b,d,0,\infty) \le N_D(b,d-1,e_1,\infty)+ (b-1)N_D(b,d-1,e_2,\rho+e_1)+1$$

where the 1 is for the expansion of the root of $T(b,d,0)$. By (1), we have

$$N_D(b,d,0,\infty) \le N_D(b,d-1,0,\infty)+ (b-1)N_D(b,d-1,0,\rho+e_1-e_2)+1$$

Since $\rho+e_1-e_2 \le \rho$ for $e_1 \le e_2$, by (2), we write

$$N_D(b,d,0,\infty) \le N_D(b,d-1,0,\infty)+ (b-1)N_D(b,d-1,0,\rho)+1 \qquad (3)$$

Now consider $N_D(b,d-1,0,\rho)$, the expected number of nodes expanded by DFBnB on $T(b,d-1,0)$ with initial upper bound ρ. If $T(b,d-1,0)$ is searched by BFS, it will return the optimal goal node whose expected cost is ρ, and expand $N_B(b,d-1)$ nodes on average. When $T(b,d-1,0)$ is searched by DFBnB with upper bound ρ, only those nodes whose costs are strictly less than ρ will be expanded, which also must be expanded by BFS. We thus have

$$N_D(b,d-1,0,\rho) \le N_B(b,d-1) \qquad (4)$$

Substituting (4) into (3), we then write

$$\begin{aligned} N_D(b,d,0,\infty) &\le N_D(b,d-1,0,\infty)+ (b-1)N_B(b,d-1)+1 \\ &\le N_D(b,d-2,0,\infty)+ (b-1)(N_B(b,d-1)+N_B(b,d-2))+2 \\ &\le ... \\ &\le N_D(b,0,0,\infty)+ (b-1)\sum_{i=1}^{d-1} N_B(b,i)+(d-1) \qquad (5) \end{aligned}$$

This proves the lemma since $N_D(b,0,0,\infty)=0$. □

Theorem 2.1 *$N_D(b,d,\infty) < O(d \cdot N_B(b,d-1))$, where N_D and N_B are defined in Lemma 2.1.*

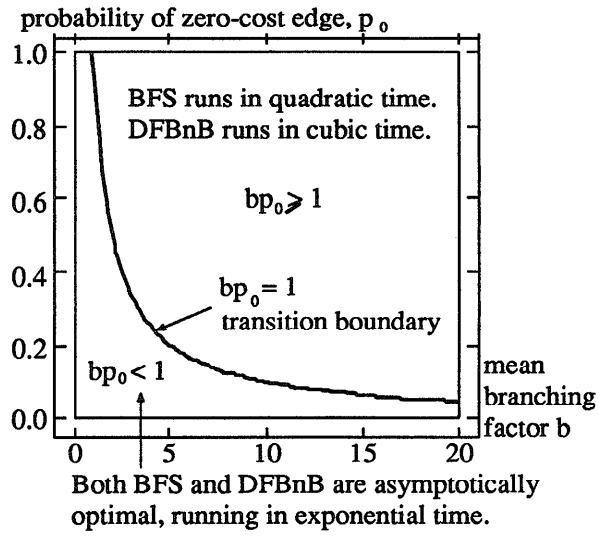

Figure 2: Complexity regions of tree search.

Proof. It directly follows Lemma 2.1 and the fact that $\sum_{i=1}^{d-1} N_B(b,i) < (d-1)N_B(b,d-1)$, since $N_B(b,i) < N_B(b,d-1)$ for all $i < d-1$. □

McDiarmid and Provan [10, 11] showed that if p_0 is the probability of a zero-cost edge, then the average complexity of BFS on $T(b,d,c)$ is determined by bp_0, the expected number of children of a node whose costs are the same as their parents. In particular, they proved that as $d \to \infty$, and conditional on the tree not becoming extinct, the expected number of nodes expanded by BFS is: (a) $O(\beta^d)$ when $bp_0 < 1$, where $1 < \beta < b$ is a constant; (b) $O(d^2)$ when $bp_0 = 1$, and (c) $O(d)$ when $bp_0 > 1$.

Theorem 2.2 *The expected number of nodes expanded by DFBnB on $T(b,d,0)$, as $d \to \infty$, conditional on $T(b,d,0)$ being infinite, is: (a) $O(\beta^d)$, when $bp_0 < 1$, where $1 < \beta < b$ is a constant; (b) $O(d^3)$ when $bp_0 = 1$, and (c) $O(d^2)$ when $bp_0 > 1$, where p_0 is the probability of a zero-cost edge.*

Proof. To use McDiarmid and Provan's result on BFS [10, 11], we have to consider the asymptotic case when $d \to \infty$. Generally, searching a deep tree is more difficult than searching a shallow one. In particular, $N_B(b,i) < N_B(b,2i)$, for all integers i. Therefore, by Lemma 2.1 and McDiarmid and Provan's result, when $bp_0 < 1$ and $d \to \infty$,

$$\begin{aligned} N_D(b,d,0) &< 2(b-1)\sum_{i=\lfloor d/2 \rfloor}^{d-1} N_B(b,i)+(d-1) \\ &= 2(b-1)\sum_{i=\lfloor d/2 \rfloor}^{d-1} O(\beta^i)+(d-1) = O(\beta^d) \end{aligned}$$

The other two cases directly follow from Theorem 2.1 and McDiarmid and Provan's results on BFS. □

This theorem significantly extends and tightens our previous result in [15], which stated that the average complexity of DFBnB is $O(d^{m+1})$ on a random tree with constant branching factor, discrete edge costs $\{0, 1, 2, ..., m\}$ and $bp_0 \geq 1$. Theorem 2.2 indicates that DFBnB is asymptotically optimal as the depth of the tree grows to infinity when $bp_0 < 1$, since it expands the same order of nodes as BFS in this case, and BFS is optimal [2]. In addition, this theorem shows that, similar to BFS, the average complexity of DFBnB experiences a transition as the expected number of same-cost children bp_0 of a node changes. Specifically, it decreases from exponential ($bp_0 < 1$) to polynomial ($bp_0 \geq 1$) with a transition boundary at $bp_0 = 1$. These results are summarized in Figure 2.

3 Experimental Results

3.1 Comparison of Nodes Expanded

We now experimentally compare BFS, DFBnB, ID and RBFS on random trees. We used random trees with uniform branching factor, and two edge cost distributions. In the first case, edge costs were uniformly selected from $\{0, 1, 2, 3, 4\}$. In the second case, zero edge costs were chosen with probability $p_0 = 1/5$, and non-zero edge costs were uniformly chosen from $\{1, 2, 3, ...2^{16} - 1\}$. The comparison of these algorithms on trees with continuous edge cost distributions is similar to that with the second edge cost distribution and $bp_0 < 1$, because a continuous distribution has $p_0 = 0$, and thus $bp_0 < 1$. We chose three branching factors to present the results: $b = 2$ for an exponential complexity case, $b = 5$ for the transition case ($bp_0 = 1$), and $b = 10$ for an easy problem. The algorithms were run to different depths, each with 100 random trials. The results are shown in Fig. 3. The curves labeled by BFS, DFBnB, ID, and RBFS are the average numbers of nodes expanded by BFS, DFBnB, ID, and RBFS, respectively. The upper bound on DFBnB is based on Lemma 2.1.

The experimental results are consistent with the analytical results: BFS expands the fewest nodes among all algorithms, RBFS is superior to ID, and DFBnB is asymptotically optimal when $bp_0 < 1$ and tree depth grows to infinity. When $bp_0 > 1$ (Fig. 3(c) and 3(f)), ID and RBFS are comparable to BFS. Moreover, when $bp_0 \geq 1$ (Fig. 3(b), 3(c), 3(e) and 3(f)), DFBnB is worse than both ID and RBFS. In these cases, the overhead of DFBnB, the number of nodes expanded whose costs are greater than the optimal goal cost, is larger than the re-expansion overheads of ID and RBFS. When $bp_0 < 1$ (Fig. 3(a) and 3(d)), however, DFBnB outperforms both ID and RBFS. In addition, when $bp_0 < 1$ and the edge costs are discrete (Fig. 3(a)), the DFBnB, ID and RBFS curves are parallel to the BFS curve for large search depth d. Thus DFBnB, ID and RBFS are asymptotically optimal, and this confirms our analysis of ID and RBFS in [16]. However, when $bp_0 < 1$ and edge costs are chosen from a large range (Fig. 3(d)), the slopes of the ID and RBFS curves are nearly twice the slope of the BFS curve, in contrast to the DFBnB curve that has the same slope as the BFS curve. This confirms our analytical result that ID expands $O(N^2)$ nodes on average when edge costs are continuous, where N is the expected number of nodes expanded by BFS [16]. This also indicates that in this case, RBFS has the same unfavorable asymptotical complexity as ID.

In summary, for problems that can be formulated as a tree with a bounded depth, and require exponential computation ($bp_0 < 1$), DFBnB should be used, and for easy problems ($bp_0 \geq 1$), RBFS should be adopted.

3.2 Comparison of Running Times

Although BFS expands fewer nodes than a linear-space algorithm, the former may run slower than the latter. Fig. 4(a) shows one example where the running time of BFS increases faster than that of DFBnB: a random binary tree in which zero edge costs were chosen with probability $p_0 = 1/5$, and non-zero edge costs were uniformly chosen from $\{1, 2, 3, ...2^{16} - 1\}$. The reason is the following. The running time of DFBnB is proportional to the total number of nodes generated, since nodes can be generated and processed in constant time. The other linear-space algorithms also have this feature. The time of BFS to process a node, however, increases as the logarithm of the total number of nodes generated. To see this, consider the time per node expansion in BFS as a function of search depth. BFS has to use a priority queue to keep all nodes generated but not expanded yet, which is exponential in the search depth, say γ^d, when $bp_0 < 1$. To expand a node, BFS first has to select the node with the minimum cost from the priority queue, and then insert all newly generated nodes into the queue. If a heap is used, to insert one node or delete the root of the heap takes time logarithmic in the total number of nodes in the heap, which is $\ln(\gamma^d) = O(d)$. This means that BFS takes time linear in the search depth to expand a node. Fig. 4(b) illustrates the average time per node expansion for both BFS and DFBnB in this case. Therefore, for some problems, BFS is not only unapplicable because of its exponential space requirement, but also suffers from increasing time per node expansion.

4 Comparison on Real Problems

4.1 Lookahead Search on Sliding-Tile Puzzles

A square sliding-tile puzzle consists of a $k \times k$ frame holding $k^2 - 1$ distinct movable tiles, and a blank space. Any tiles that are horizontally or vertically adjacent to the blank may move into the blank position. Examples of sliding-tile puzzles include the 3×3 Eight Puzzle, the

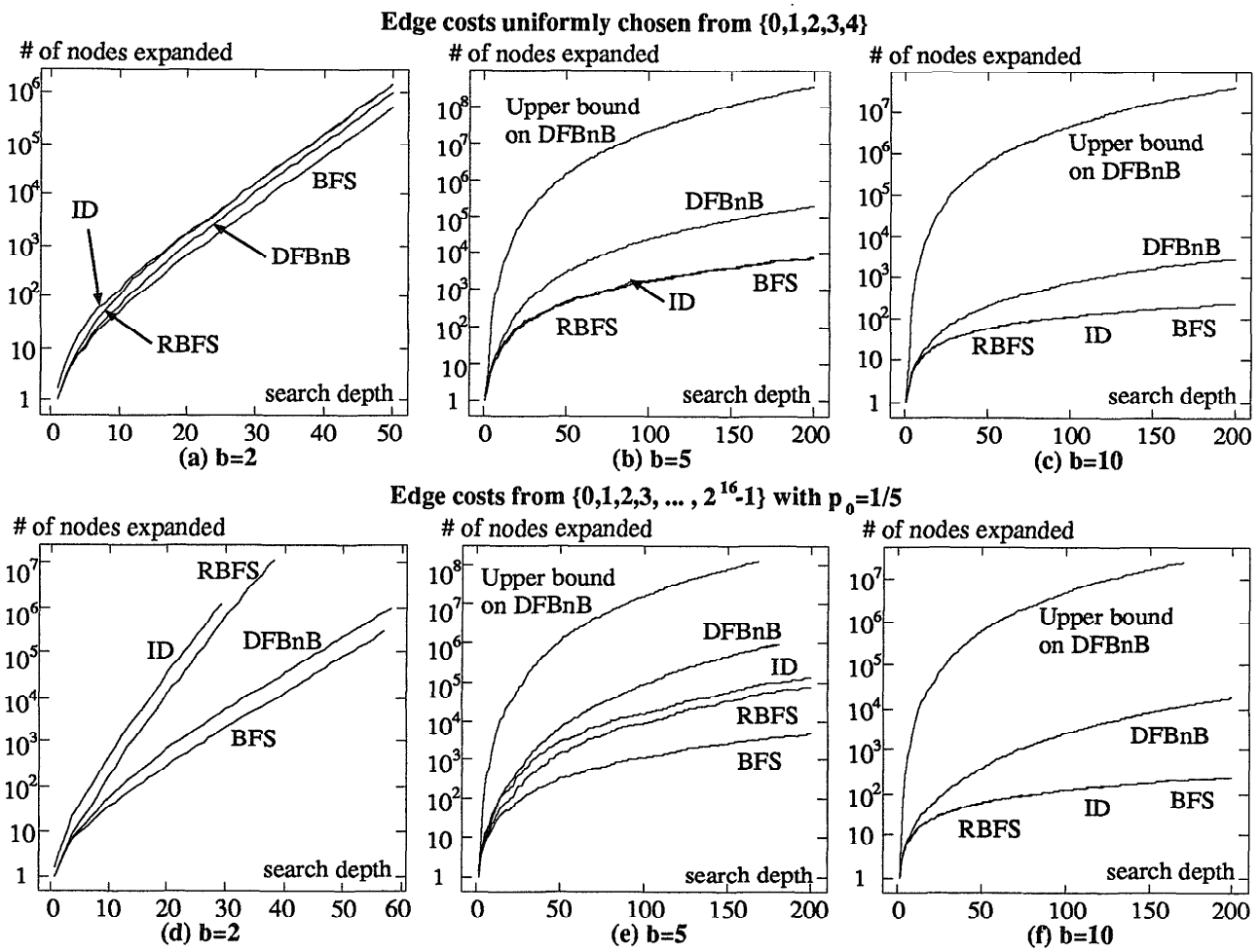

Figure 3: Average number of nodes expanded.

4×4 Fifteen Puzzle, the 5×5 Twenty-four Puzzle, and the 10×10 Ninety-nine Puzzle. A common cost function for sliding-tile puzzles is $f(n) = g(n) + h(n)$, where $g(n)$ is the number of moves from the initial state to node n, and $h(n)$ is the Manhattan distance from node n to the goal state, which is the sum of the number of moves along the grid of all tiles to their goal positions. Given an initial and a goal state of a sliding-tile puzzle, we are asked to find a sequence of moves that maps the initial state into the final state. To find such a sequence with minimum number of moves is NP-complete for arbitrary size puzzles [12].

In real-time settings, however, we have to make a move with limited computation. One approach to this problem, called *fixed-depth lookahead search*, is to search from the current state to a fixed depth, and then make a move toward a minimum cost frontier node at that depth. This process is then repeated for each move until a goal is reached [5].

Our experiments show that ID is slightly worse than RBFS for lookahead search, as expected. Figure 5 compares DFBnB and RBFS. The horizontal axis is the lookahead depth, and the vertical axis is the number of nodes expanded, averaged over 200 initial states. The results show that DFBnB performs better than RBFS on small puzzles, while RBFS is superior to DFBnB on large ones. The reason is briefly explained as follows. Moving a tile either increases or decreases its Manhattan distance h by one. Since every move increases the g value by one, the cost function $f = g + h$ either increases by two or stays the same. The probability that the cost of a child state is equal to the cost of its parent is approximately 0.5 initially, *i.e.* $p_0 \approx 0.5$. In addition, the average branching factors b of the Eight, Fifteen, Twenty-four, and Ninety-nine Puzzles are approximately 1.732, 2.130, 2.368, and 2.790, respectively, *i.e.* b grows with the puzzle size. Thus, bp_0 increases with the puzzle size as well, and lookahead search is

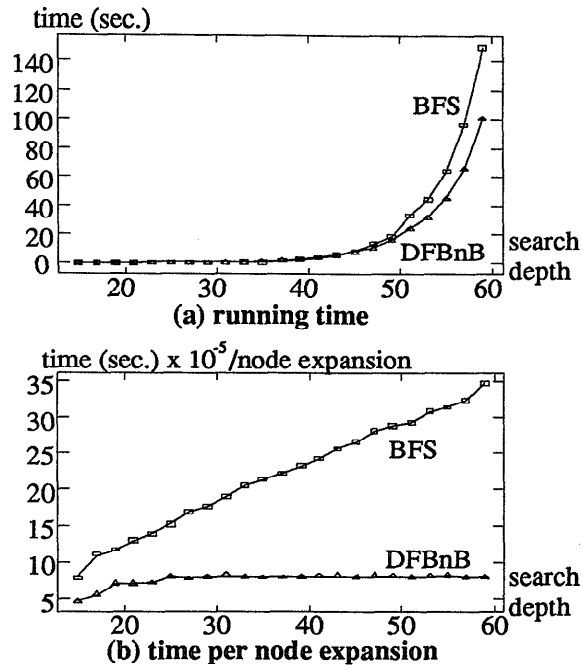

Figure 4: Running time and time per node expansion.

easier on large puzzles by Theorem 2.2. As shown in Section 3, DFBnB will do better with smaller branching factors.

Unfortunately, the problem of finding a shortest solution path cannot be represented by a bounded-depth tree, since the solution length is unknown in advance. Without cutoff bounds, DFBnB cannot be applied in these cases. This limits the applicability of DFBnB, and distinguishes DFBnB from BFS, ID and RBFS. In these cases, RBFS is the algorithm of choice, since ID is worse than RBFS, as verified by our experiments.

4.2 The Asymmetric TSP

Given n cities $\{1, 2, ..., n\}$ and a cost matrix $(c_{i,j})$ that defines a cost between each pair of cities, the *traveling salesman problem* (TSP) is to find a minimum-cost tour that visits each city once and returns to the starting city. When the cost from city i to city j is not necessarily equal to that from city j to city i, the problem is the *asymmetric TSP* (ATSP). Many NP-complete combinatorial problems can be formulated as ATSPs, such as vehicle routing, no-wait workshop scheduling, computer wiring, *etc.* [8].

The most efficient approach known for optimally solving the ATSP is subtour elimination [1], with the solution to the assignment problem as a lower-bound function. Given a cost matrix $(c_{i,j})$, the *assignment problem* (AP) [9] is to assign to each city i another city j, with $c_{i,j}$ as the cost of this assignment, such that the

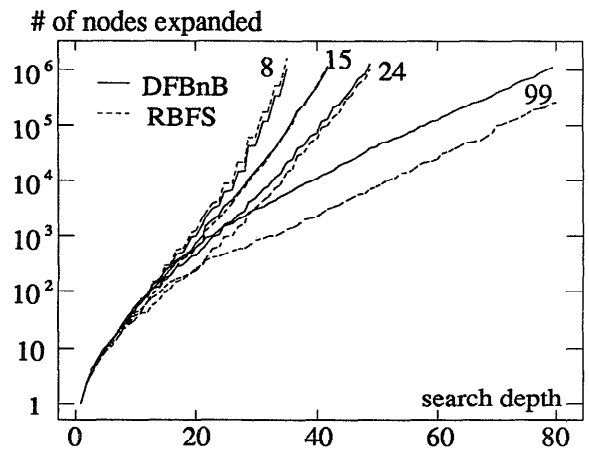

Figure 5: Lookahead search on sliding-tile puzzles.

total cost of all assignments is minimized. The AP is a generalization of the ATSP with the requirement of a single complete tour removed, allowing collections of subtours, and is solvable in $O(n^3)$ time [9]. Subtour elimination first solves the AP for the n cities. If the solution is not a tour, it then expands the problem into subproblems by breaking a subtour (cf. [1] for details), and searches the space of subproblems. It repeatedly checks the AP solutions of subproblems and expands them if they are not complete tours, until an optimal tour is found. The space of subproblems can be represented by a tree with maximum depth less than n^2.

We ran DFBnB, ID and RBFS on the ATSP with the elements of cost matrices independently and uniformly chosen from $\{0, 1, 2, 3, ..., r\}$, where r is an integer. Figures 6(a) and 6(b) show our results on 100-city and 300-city ATSPs. The horizontal axes are the range of intercity costs r, and the vertical axes are the numbers of tree nodes generated, averaged over 500 trials for each data point. When the cost range r is small or large, relative to the number of cities n, the ATSP is easy or difficult, respectively [15, 17]. Figure 6 shows that ID cannot compete with RBFS and DFBnB, especially for difficult ATSPs when r is large. RBFS does poorly on difficult ATSPs, since in this case the node costs in the search tree are unique [15, 17], which causes significant node regeneration overhead.

5 Conclusions

We first studied the relationship between the average number of nodes expanded by depth-first branch-and-bound (DFBnB), and best-first search (BFS). In particular, we showed analytically that DFBnB expands no more than $O(d \cdot N)$ nodes on average for finding a minimum cost node at depth d of a random tree, where N is the average number of nodes expanded by BFS on

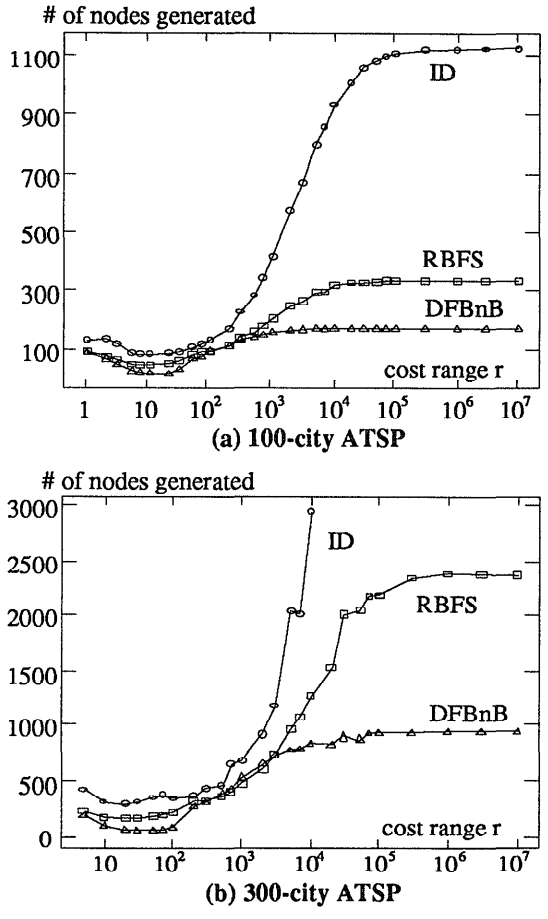

Figure 6: Performance on the ATSPs.

the same tree. We also proved that DFBnB is asymptotically optimal when BFS runs in exponential time. We then considered how to select a linear-space algorithm, from among DFBnB, iterative-deepening (ID) and recursive best-first search (RBFS). We also showed that DFBnB runs faster than BFS in some cases, even if the former expands more nodes than the latter. Our results on random trees and two real problems, lookahead search on sliding-tile puzzles and the asymmetric traveling salesman problem, show that (a) DFBnB is preferable on problems that can be formulated by bounded-depth trees and require exponential computation; (b) RBFS should be applied to problems that cannot be represented by bounded-depth trees, or problems that can be solved in polynomial time.

References

[1] Balas, E., and P. Toth, "Branch and bound methods," *The Traveling Salesman Problems,* E.L. Lawler, *et al.* (eds.) John Wiley and Sons, 1985, pp.361-401.

[2] Dechter, R., and J. Pearl, "Generalized best-first search strategies and the optimality of A*," *JACM,* **32** (1985) 505-36.

[3] Karp, R.M., and J. Pearl, "Searching for an optimal path in a tree with random cost," *Artificial Intelligence,* **21** (1983) 99-117.

[4] Korf, R.E., "Depth-first iterative-deepening: An optimal admissible tree search," *Artificial Intelligence,* **27** (1985) 97-109.

[5] Korf, R.E., "Real-time heuristic search," *Artificial Intelligence,* **42** (1990) 189-211.

[6] Korf, R.E., "Linear-space best-first search: Summary of results," *Proc. 10-th National Conf. on AI, AAAI-92,* San Jose, CA, July, 1992, pp.533-8.

[7] Korf, R.E., "Linear-space best-first search," *Artificial Intelligence,* to appear.

[8] Lawler, E.L., *et al.*, *The Traveling Salesman Problems,* John Wiley and Sons, 1985.

[9] Martello, S., and P. Toth, "Linear assignment problems," *Annals of Discrete Mathematics,* **31** (1987) 259-82.

[10] McDiarmid, C.J.H., "Probabilistic analysis of tree search," *Disorder in Physical Systems,* G.R. Gummett and D.J.A. Welsh (eds), Oxford Science Pub., 1990, pp.249-60,

[11] McDiarmid, C.J.H., and G.M.A. Provan, "An expected-cost analysis of backtracking and non-backtracking algorithms," *Proc. 12-th Intern. Joint Conf. on AI, IJCAI-91,* Sydney, Australia, Aug. 1991, pp.172-7.

[12] Ratner, D., and M. Warmuth, "Finding a shortest solution for the NxN extension of the 15-puzzle is intractable," *Proc. 5-th National Conf. on AI, AAAI-86,*, Philadelphia, PA, 1986.

[13] Vempaty, N.R., V. Kumar, and R.E. Korf, "Depth-first vs best-first search," *Proc. 9-th National Conf. on AI, AAAI-91,*, CA, July, 1991, pp.434-40.

[14] Wah, B.W., and C.F. Yu, "Stochastic modeling of branch-and-bound algorithms with best-first search," *IEEE Trans. on Software Engineering,* **11** (1985) 922-34.

[15] Zhang, W., and R.E. Korf, "An average-case analysis of branch-and-bound with applications: Summary of results," *Proc. 10-th National Conf. on AI, AAAI-92,*, San Jose, CA, July, 1992, pp.545-50.

[16] Zhang, W., and R.E. Korf, "Performance of linear-space branch-and-bound algorithms," submitted to *Artificial Intelligence,* 1992.

[17] Zhang, W., and R.E. Korf, "On the asymmetric traveling salesman problem under subtour elimination and local search," submitted, March, 1993.