From: AAAI Technical Report SS-99-07. Compilation copyright © 1999, AAAI (www.aaai.org). All rights reserved.

Truncated and Anytime Depth-First Branch and Bound: A Case Study on the Asymmetric Traveling Salesman Problem*

Weixiong Zhang
Information Sciences Institute and Computer Science Department
University of Southern California
4676 Admiralty Way, Marina del Rey, CA 90292
Email: zhang@isi.edu

Abstract

One of the most applied search algorithms for finding optimal solutions in practice is depth-first branch and bound (DFBnB), and the most popular method for approximate solutions is local search. Besides the fact that DFBnB is typically applied to obtain optimal solutions, it can also be used to find approximate solutions, and can also run as an anytime algorithm. In this paper, we study DFBnB used as approximation and anytime algorithms. We compare DFBnB against a local search algorithm, which is the best approximation algorithm we found, on the asymmetric Traveling Salesman Problem (ATSP), an important NP-complete problem. Our experimental results on the ATSP, up to 1,000 cities, show that DFBnB significantly outperforms the local search algorithm on various ATSP structures, finding better solutions much earlier than the local search; and the quality of approximate ATSP tours from a truncated DFBnB is typically several times better than that from the local search. Our study demonstrates that DFBnB is high-performance anytime and approximation algorithms for solving large problems.

1 Introduction

Depth-first branch and bound (DFBnB) [4; 14] and local search [16; 11; 9; 10] are the two most applied search algorithms for solving combinatorial optimization problems, such as planning and scheduling, in artificial intelligence and operations research. DFBnB is typically the algorithm for finding optimal solutions in practice, due to its linear-space requirement. Local search, on the other hand, is a method for high-quality approximate solutions, as it has been shown to be effective and efficient on many combinatorial optimization problems, such as the symmetric Traveling Salesman Problem [16; 10] and constraint satisfiability [22].

*This research was supported in part by NSF Grant IRI-9619554.

Besides the fact that DFBnB is an efficient algorithm for finding optimal solutions, it is also in fact an anytime algorithm. An anytime algorithm [6] can provide a solution at any time during its execution, and is able to improve the quality of the current best solution with more computation. In contrast to the importance of anytime problem solving and the effort of developing new anytime algorithms [1; 2; 3], DFBnB has not been studied as an approximation algorithm or an anytime algorithm.

Furthermore, although it has been suggested that DFBnB can be tailored to algorithms for finding approximate solutions [8], it is not clear, however, if such an approximation method is a choice of algorithm for a particular application. In order to select an approximation algorithm based on DFBnB, one needs to show that it is superior to local search, as local search is usually the best anytime and approximation algorithm for many combinatorial optimization problems.

Motivated by the above observations, we study DFBnB as anytime and approximation algorithms. To this end, we compare DFBnB against local search on the asymmetric Traveling Salesman Problem (ATSP) [15]. The local search algorithm of [11] is used in this study, which is the best approximation algorithm we found in the literature. We choose the ATSP due to the following reasons. First, the problem is an important one in the NP-complete class [21] and has many practical applications. Many difficult combinatorial optimization problems, such as vehicle routing, workshop scheduling and computer wiring, can be formulated and solved as the ATSP [15]. Secondly, a very accurate heuristic cost function, the assignment problem [21], is available for the ATSP, which is able to significantly increase the efficiency of a state-space search algorithm. Thirdly, despite its importance, little work has been done on the ATSP, which is inproportional to that on the symmetric TSP (see [10] for an excellent survey). It will be very helpful if information regarding which algorithm should be used for a particular type of ATSP problems is available to guide algorithm selection in practice.

The paper is organized as follows. In Section 2, we describe the ATSP, DFBnB, and local search. In Section 3, we experimentally compare DFBnB with the local search algorithm on various ATSP problem structures. In Section 4, we discuss the factors that lead to the strong performance of DFBnB. We conclude in Section 5.

2 The Problem and Algorithms

2.1 The problem

Given n cities, $\{1, 2, 3, \cdots, n\}$, and a matrix $(c_{i,j})$ that defines costs between pairs of cities, the Traveling Salesman Problem (TSP) is to find a minimum-cost tour that visits each city exactly once and returns to the starting city. When the cost matrix is asymmetric, i.e., the cost from city i to city j is not necessarily equal to the cost from j to i, the problem is the asymmetric TSP (ATSP).

2.2 DFBnB and truncated DFBnB

The *assignment problem* (AP) [21] is the most accurate cost function for the ATSP, and is solvable in $O(n^3)$ time [21]. It is to assign to each city i another city j, with $c_{i,j}$ as the cost of this assignment, such that the total cost of all assignments is minimized. The AP is a relaxation of the ATSP, since the assignments need not form a complete tour, allowing collections of disjoint subtours. Therefore, the solution cost of the AP provides a lower bound on the cost of the ATSP, which is an assignment of each city to its successor on the tour. If the AP solution happens to be a complete tour, it is also the solution to the ATSP.

Branch and bound (BnB) [4; 14] solves an ATSP as a state-space search. It takes the original problem as the root problem and repeats the following two steps. First, solve the AP for the current problem. If the AP solution is not a complete tour, decompose the problem into subproblems by subtour elimination. Specifically, select a subtour from the AP solution, and generate subproblems by excluding some edges from the assignments, so as to eliminate the subtour. Next, select as the current problem a new subproblem that has been generated but not yet expanded. This process continues until there are no unexpanded problems, or all unexpanded problems have costs greater than or equal to the cost of the best complete tour found so far. There are many heuristics of subtour elimination for decomposing a problem [4]. In our experimental study, we use the scheme proposed by Carpaneto and Toth [5], which generates no duplicate subproblems to make the total number of subproblems generated as small as possible, and the resulting search space is a tree with no duplicate node.

Depth-first branch and bound (DFBnB) is a special BnB that explores nodes or subproblems in a depth-first order. DFBnB uses a global upper bound α on the cost of an optimal goal. Starting at the root node, DFBnB always selects a most recently generated node to examine next. Whenever a new leaf node, a node on which the AP solution is a complete ATSP tour, is reached and the cost of the node, the cost of the corresponding AP, is less than the current upper bound α, α is revised to the cost of this new leaf. Whenever a node is selected for expansion whose cost is greater than or equal to α, it is pruned, because node costs are non-decreasing along a path from the root, and all descendents of a node must have costs at least as great as that of their ancestors. In order to find an optimal goal node quickly, the newly generated child nodes should be searched in an increasing order of their costs. This is called *node ordering.* Node ordering is cheap to perform and usually provides significant improvement to search efficiency. Throughout this paper, when we refer to DFBnB, we mean DFBnB with node ordering.

DFBnB can be used as an anytime algorithm, as it encounters many leaf nodes during the search, which may improve the best solution found so far over the time. Furthermore, DFBnB can be stopped at any time during its execution. This is an extension to the M-Cut strategy suggested in [8], which terminates the execution of BnB when a fixed total computation has been consumed. We call DFBnB with an early termination *truncated DFBnB.* Among all possible stopping points, of particular interest is the one where the first leaf node is reached. This special truncated DFBnB is a greedy search in a search space, which always chooses to explore next the minimum-cost child node among all children of a node until it reaches a leaf node. Without confusion, we call DFBnB with this special stopping criteria *truncated DFBnB* in the rest of this paper.

2.3 Local search

Local search is based on a fundamental concept called *neighborhood structure.* If two TSP tours differ by λ edges, and one can be changed into the other by swapping the corresponding different edges, we say that one is a λ-change neighbor of the other. A neighborhood structure is established by defining the legitimate changes. Within a neighborhood, a tour is a local optimum if its cost is not greater than the costs of its neighboring tours. Given a neighborhood structure, local search algorithm moves from one tour to another by a sequence of legitimate changes that reduce the tour cost, until a local optimum is reached.

The local search algorithm for the ATSP [11] follows that for the symmetric TSP [16], and uses *primary changes* described as follows. For any two complete tours π and π', let $X \in \pi$ and $Y \in \pi'$ be disjoint sets of arcs such that $\pi' = (\pi - X) + Y$, where the arcs in X are *deleted* arcs, and that in Y are *added* arcs. If $|X| = |Y| = \lambda$, then π' is a λ-change of π. For example, Figure 1(a) shows a tour π (in solid curves and dotted links), and a 3-change π' of π (in solid curves and solid links), where $X = \{x_1, x_2, x_3\}$ and $Y = \{y_1, y_2, y_3\}$. For any two tours π and π', we can define a directed graph $G(\pi, \pi')$ as follows. The nodes of $G(\pi, \pi')$ correspond to deleted arcs in X. If $x_i = (k, l) \in X$, $x_j = (p, q) \in X$ and $(k, q) \in Y$, then there is a directed arc (x_i, x_j) in $G(\pi, \pi')$. In general, $G(\pi, \pi')$ is a collection of disjoint cycles. π' is called a primary change of π if $G(\pi, \pi')$ consists of a single directed cycle. For example, for π and π' in Figure 1(a), $G(\pi, \pi')$ is a single cycle, as shown in Figure 1(b), and thus π' is a primary 3-change of π. It can be shown that there is no primary λ-change for

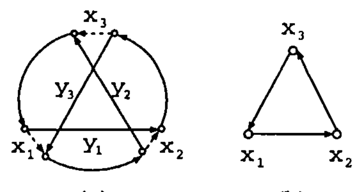

Figure 1: Primary 3-change.

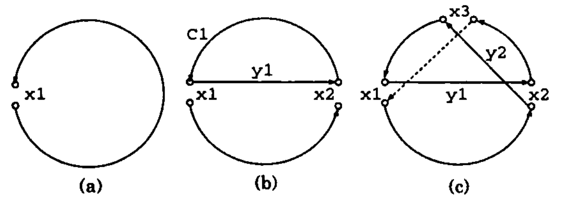

Figure 2: Sequential search of primary changes.

any even number λ for the ATSP [11]. Therefore, the simplest legitimate change is a 3-change.

To reduce computation, searching for primary changes is restricted by the following *sequential* process. To gradually construct a primary change of a tour π, we first remove an arc x_1 of π, resulting in a directed path (Figure 2(a)). We then add a new arc y_1, which automatically determines an arc x_2 that needs to be deleted and creates a cycle C_1 (Figure 2(b)). We can immediately break the cycle C_1 by adding a new arc y_2 as shown in Figure 2(c). This essentially ends up with a directed path, as we started with in Figure 2(a). In general, we call two subsequent pairs of deleted and added arcs, $< x_i, y_i >$ and $< x_{i+1}, y_{i+1} >$ for $i = 1, 3, 5, \cdots$, *a pair of steps*, if y_{i+1} subsequently breaks the cycle produced by y_i. The sequential process proceeds in pairs of steps, or searches for a primary change by a sequence of cycle creations, each immediately followed by a breaking of the cycle. Obviously, a path can be closed by linking its two end cities, resulting in a complete tour, e.g. Figure 2(c).

Another useful change is the *quad* change, as shown in Figure 3. It is not primary, but seems to substantially enrich the neighborhood structure [11].

The local search algorithm of [11] uses all the primary changes, obtained sequentially, plus quad changes, and proceeds as follows in seeking a local optimum. Starting with a tour, it first searches for improving primary $2k + 1$ changes, making k as large as possible. To reduce computation, the algorithm requires that it has a favorable change at every pair of steps. In other words, in searching for a better $2k + 1$ primary change, the algorithm has already retained a $2k - 1$ primary change that improves the starting tour. The algorithm also presorts the intercity distances to facilitate the search of favorite primary changes [11]. The algorithm repeatedly searches for primary changes, until none exists. It then repeatedly improves the tour by quad changes, until no improving quad changes are available. It then starts to search for primary changes again, and the process repeats. If the tour cannot be improved by either primary changes or quad changes, it is a local optimum. The process can be repeated on different initial tours, until the total computation is used up.

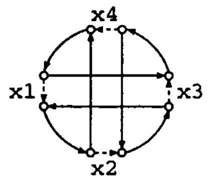

Figure 3: Quad change.

3 Experimental Study and Results

3.1 Problem structures

In our experimental study, we considered the ATSP of various sizes, ranging from a few dozens to 1,000 cities, and various ATSP structures. We used the following different cost matrix $(c_{i,j})$ structures. (a) Random matrices with $c_{i,j}$ from $\{0, 1, 2, \cdots, r\}$, where $r = 2^{16} - 1$. We use a large intercity cost range r because there exists an average-case complexity transition for BnB. Specifically, the ATSP is relatively easy to solve when r is small, with respect to the number of cities n; while relatively difficult when r is large [23; 24]. The problem instances thus generated are difficult as $r \gg n$. (b) Matrices with the triangle inequality, $c_{i,j} \leq c_{i,k} + c_{k,j}$, for all $i, j, k \in (c_{i,j})$. We first generated random matrices as in (a), and then used a closure algorithm to enforce the triangle inequality. (c) Matrices satisfying the constraint $c_{i,j} \leq i \times j$. To generate $(c_{i,j})$, we chose $c_{i,j}$ independently and uniformly from $\{0, 1, \cdots, i \times j\}$. This problem structure is known to be difficult for the methods using the AP cost function [19]. (d) Matrices from some actual problems encountered in industry [18]. (e) Constructed ATSPs that are difficult for the local search algorithm [20].

3.2 Tour construction heuristics

Local search requires an initial tour to start. Several polynomial-time tour construction heuristics can be used to find a complete tour. These include Nearest Neighbor [7; 9], Nearest Insertion [7; 9], Greedy algorithm [7; 9], and Karp's patching algorithm [12]. Due to space limitation, the interested reader is referred to these references for the details of these methods.

To find the best tour construction heuristic for the local search algorithm, we experimentally compared these tour construction heuristics using the first three cost matrices discussed in Section 3.1, with the number of cities n ranging from 100 to 1,000, in 100 city increments. Due to space limitation, we only show the results on the ATSP with random cost matrices in Table 1. The results with other cost matrices have similar characteristics. In Table 1, *error* is the relative error of tour costs compared

number of cities		100	200	300	400	500	600	700	800	900	1000
error (%)	patch	8.488	6.830	5.662	4.768	4.559	4.159	3.895	3.752	3.518	3.401
	greedy	165.798	205.193	227.963	244.651	257.838	268.843	278.615	287.454	293.217	312.450
	n-n	187.932	230.233	253.579	268.842	285.616	293.249	304.759	311.974	318.762	332.502
	n-i	260.588	399.134	505.199	594.314	673.109	742.327	809.458	871.207	927.579	987.342
time (sec.)	patch	0.009	0.051	0.134	0.278	0.485	0.769	1.139	1.595	2.125	2.437
	greedy	0.029	0.182	0.514	1.083	1.918	3.050	4.463	6.156	8.120	9.872
	n-n	0.000	0.001	0.003	0.006	0.009	0.014	0.019	0.026	0.033	0.051
	n-i	0.004	0.022	0.058	0.118	0.206	0.323	0.466	0.637	0.817	1.178

Table 1: Performance of tour construction heuristics on ATSP with random cost matrices.

to the AP lower bounds, and *time* is the CPU time on a Sun Ultra Sparc 2 workstation. All results are averages of 1,000 trials each. Table 1 shows that nearest neighbor (labeled as n-n in the table) runs in the minimum average time, followed by nearest insertion (labeled as n-i), patching and Greedy algorithms. Patching has the best tour quality, followed by Greedy, nearest neighbor, and nearest insertion. One important observation is that the average tour costs from the patching algorithm decrease when the problem size increases, while the average tour costs from the other three algorithms increase. Furthermore, the quality of the tours from patching algorithm is at least an order of magnitude smaller than those from the other methods on all problem structures that we considered. The superiority of patching algorithm is in part due to the accuracy of the AP lower-bound function, as discussed in Section 4.

The best tour construction algorithm, Karp's patching, was selected to generate an initial tour for the local search in our comparison of DFBnB and local search.

3.3 Truncated DFBnB versus local search

We compared truncated DFBnB with the local search for finding approximate solutions. Table 2 summarizes the results on the ATSP with the first three cost matrices, i.e., random matrices, matrices with triangle inequality, and matrices with $c_{i,j} \leq i \times j$. Table 2 shows the tour qualities and CPU times of both algorithms, averaged over 100 instances. The tour quality is expressed as the error of tour costs relative to the AP lower bounds, and CPU time is on a Sun Ultra Sparc 2 workstation. The relative error of truncated DFBnB is less than 1.8% for 100-city, and decreases to less than 0.21% for 1,000-city instances on all three types of cost matrices. Across all problem structures and sizes we examined, the average tour quality of truncated DFBnB is better than that of local search, and the average execution time of truncated DFBnB is less than the average time of local search. Table 2 also shows the percentage of instances on which truncated DFBnB finds better tours than local search, as well as the percentage of instances on which local search is better than truncated DFBnB (labeled as better). It shows that on a majority of problem instances, truncated DFBnB outperforms local search.

We also compared truncated DFBnB with the local search on two particular problems from industry [18], a 23-city and a 43-city ATSPs. These two problems have AP costs 85 and 43, but optimal ATSP costs 259 and 5620, respectively. On the 23-city problem, truncated DFBnB finds an optimal tour of cost 259, while the local search stops at a non-optimal tour of cost 262. DFBnB using AP cost function is not efficient on the 43-city problem. To this problem, although the fast algorithm in [19] finds an approximate tour of cost 5625 almost immediately, it cannot optimally solve this problem [18]. For this problem, our implementation of DFBnB with Carpaneto and Toth's decomposition rules does not find an optimal tour after generating 30 million subproblems, and BnB using best-first strategy runs out of memory on all machines available to us. For this particular ATSP, truncated DFBnB finds a tour of cost 5623 in 1 second of CPU time by expanding 54 subproblems and generating 173 subproblems. This result is slightly better than the tour of cost 5625 found in [19] and a tour of cost 5627 from the local search, also with 1 second of CPU time.

Both DFBnB and local search algorithms were also tested on constructed ATSPs that are difficult for the local search algorithm [20]. For an ATSP with $n = 6k$ cities, there is an optimal tour with cost 0, and $(k-1)!$ local optimums that have arbitrarily large costs. Not surprisingly, the local search algorithm stops at a local minimum which is not a global optimal, while truncated DFBnB finds the optimal tour by expanding only the original problem and generating four subproblems.

3.4 Anytime DFBnB versus local search

To better understand how DFBnB and the local search algorithm perform on the ATSP, we compare the quality of tours that they find over the time of their execution, i.e., we consider the anytime performance of these algorithms. It turns out that DFBnB has better anytime performance than the local search. Figure 4 shows the result on the 300-city ATSP with random cost matrices, averaged over 100 instances. The horizontal axis of Figure 4 is the CPU time, in a logarithmic scale, and the vertical axis is the average error of tour costs relative to the optimal tour costs. Both algorithms start with a complete tour generated by Karp's patching algorithm. Since local search typically finishes earlier than DFBnB, which finds an optimal tour at the end, we restart the local search on initial tours generated by the other tour construction heuristics, Greedy algorithm, nearest neighbor and

number of cities			100	200	300	400	500	600	700	800	900	1000
random matrices	error (%)	*local search*	6.326	5.198	4.396	3.842	3.616	3.489	2.997	2.958	2.703	2.792
		T DFBnB	1.719	0.947	0.632	0.531	0.372	0.274	0.261	0.244	0.197	0.208
	time (sec.)	*local search*	0.031	0.145	0.398	0.798	1.385	2.203	3.253	4.399	5.843	7.993
		T DFBnB	0.021	0.113	0.329	0.674	1.239	1.825	2.794	3.887	5.710	7.236
	better (%)	*local search*	3	0	0	1	0	0	0	0	0	0
		T DFBnB	95	98	100	99	100	100	100	100	100	100
matrices with triangle inequality	error (%)	*local search*	0.990	0.666	0.471	0.324	0.341	0.286	0.237	0.255	0.236	0.255
		T DFBnB	0.589	0.355	0.205	0.150	0.131	0.117	0.087	0.103	0.082	0.082
	time (sec.)	*local search*	0.033	0.158	0.437	0.889	1.489	2.529	3.534	4.871	6.479	7.982
		T DFBnB	0.023	0.116	0.311	0.645	1.086	1.817	2.448	3.928	5.277	7.093
	better (%)	*local search*	9	3	4	6	3	5	3	3	2	1
		T DFBnB	65	62	61	54	61	61	54	66	61	70
matrices with constraints $c_{i,j} \leq i \times j$	error (%)	*local search*	3.220	2.285	2.073	2.138	1.636	1.543	1.493	1.288	1.264	1.250
		T DFBnB	1.660	0.836	0.622	0.467	0.372	0.257	0.253	0.215	0.202	0.194
	time (sec.)	*local search*	0.052	0.273	0.800	1.738	3.115	5.213	7.711	10.435	13.788	18.109
		T DFBnB	0.035	0.201	0.601	1.133	2.023	3.254	4.945	6.905	8.807	11.241
	better (%)	*local search*	13	3	3	1	3	2	0	0	2	1
		T DFBnB	79	96	95	97	97	97	100	99	98	99

Table 2: Truncated DFBnB vs. local search.

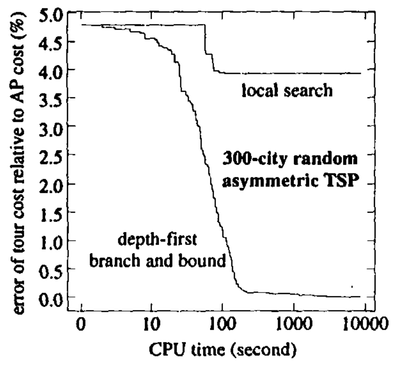

Figure 4: DFBnB vs. local search on 300-city ATSP.

nearest insertion, until the local search uses at least the same amount of time as used by DFBnB. However, the local search generally fails to improve the best tour found using these additional initial tours, since these tour construction heuristics are not very effective, as we have seen in Section 3.2.

Figure 4 shows that DFBnB significantly outperforms the local search on the ATSP with random cost matrices. Similar results have also been observed on other problem sizes and structures. DFBnB typically finds better tours earlier than the local search. This is partially because the AP can be computed in $O(n^2)$ time on an interior node of the search tree [17], rather than in $O(n^3)$ time on the root node. Based on our experiments, the AP on an interior node can be computed in roughly one tenth, one twentieth, and one seventeenth of the CPU time required for the AP of the initial ATSP with random cost matrices, matrices with triangle inequality, and matrices with $c_{i,j} \leq i \times j$, respectively. This also helps to understand why truncated DFBnB can quickly reach a leaf node, as discussed in the previous section. Thanks to the superior performance of truncated DFBnB, DFBnB can obtain a high-quality tour very early in its execution, which can further help to restrict the search to the areas where better solutions can be found.

The poor performance of the local search algorithm for the ATSP indicates that its neighborhood structure may be very restricted, comparing to that for the symmetric TSP of [16]. For the ATSP, there is no primary changes with an even number of edges. Furthermore, As discussed in Section 2.3, the local search searches increasingly better primary changes, which may substantially curtail the effectiveness of the algorithm. As pointed out in [11], however, using increasingly better primary changes is necessary to reduce the computation cost. In addition, to find favorable pairs of steps, edges need to be sorted, which prevents the local search to quickly improve the initial tour, as shown in Figure 4.

4 Discussions

The superior performance of DFBnB and truncated DFBnB on the ATSP is primarily due to two factors. The first one is the lower-bound cost function based on the assignment problem (AP). As observed in previous research [4] and in our own experiments, this cost function gives a superb estimation on the actual cost of the corresponding ATSP tour. In our experiments on the random ATSP with the elements of cost matrices independently and uniformly chosen from $\{0, 1, 2, \cdots, 2^{16} - 1\}$, the cost of the AP is 99.090% of the ATSP cost on average for 100-city instances, 99.816% for 500-city instances, and 99.916% for 1000-city instances. A good lower-bound cost function can usually give rise to a strong branch-and-bound algorithm. This has also been observed on number partitioning using branch and bound with Karmarkar-Karp heuristics [13].

The second factor that leads to the superb performance of DFBnB is that the search space under BnB subtour elimination with the decomposition rules of [5] is a tree without duplicate nodes. A search tree without duplicates is typically very compacted, giving rise to a small total number of nodes in the space. In addition, a tree structure can also significantly speed up DFBnB, since no duplicate or cycle detection, which consumes a substantial amount of computation, is required.

5 Conclusions

Depth-first branch and bound (DFBnB) is not only a general technique for *optimally* solving difficult NP-complete combinatorial optimization problems, but can also be adapted to efficient anytime and approximation algorithms. In this paper, we studied DFBnB and truncated DFBnB, a DFBnB with an early termination, on the asymmetric Traveling Salesman Problem (ATSP) of various structures and sizes. Specifically, we experimentally compared DFBnB and truncated DFBnB against the local search algorithm of [11], which is the best approximation algorithm for the ATSP so far. Our experimental results showed that DFBnB outperforms the local search algorithm, finding better ATSP tours significantly earlier than the local search. The results also showed that truncated DFBnB is superior to the local search algorithm for finding approximate ATSP tours.

The contribution of this work is twofold. First, to the specific problem of the ATSP, it provides a thorough comparison of DFBnB and local search, showing that DFBnB and truncated DFBnB are the choices of algorithms for solving the ATSP in practice. Secondly, beyond the specific problem of the ATSP, this work shows that DFBnB, a systematic approach, is also well suited for anytime problem solving and approximate computation. To our knowledge, this is the only work so far which studies the anytime performance of DFBnB, and is also the only work that demonstrated that DFBnB can compete with and outperform a local search algorithm in the categories of anytime and approximation algorithms.

Acknowledgment

Thanks to David Johnson and Richard Korf for many discussions related to this work, and to Donald Miller for providing two real-world problem instances used in this paper.

References

[1] *AAAI Spring Symp. on Limited Rationality*, Stanford, CA, 1989. AAAI.

[2] *AAAI Fall Symposium on Rational Agency*, Cambridge, MA, 1995. AAAI.

[3] *IJCAI-95 Workshop on Anytime Algorithms and Deliberation Scheduling*, Montreal, Canada, 1995.

[4] E. Balas and P. Toth. Branch and bound methods. In *The Traveling Salesman Problem*, pages 361–401. John Wiley & Sons, 1985.

[5] G. Carpaneto and P. Toth. Some new branching and bounding criteria for the asymmetric TSP. *Management Science*, 26:736–743, 1980.

[6] T. Dean and M. Boddy. An analysis of time-dependent planning. In *Proc. AAAI-88*, pages 49–54, St. Paul, MN, Aug. 1988.

[7] A. Frieze, G. Galbiati, and F. Maffioli. On the worst-case performance of some algorithms for the asymmetric TSP. *Network*, 12:23–39, 1982.

[8] T. Ibaraki, S. Muro, T. Murakami, and T. Hasegawa. Using branch-and-bound algorithms to obtain suboptimal solutions. *Zeitchrift für Operations Research*, 27:177–202, 1983.

[9] D. S. Johnson. Local optimization and the TSP. In *Proc. 17th Intern. Colloquium on Automata, Languages and Programming*, pages 446–461, 1990.

[10] D. S. Johnson and L. A. McGeoch. The traveling salesman problem: a case study. In E. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. John Wiley & Sons, West Sussex, England, 1997.

[11] P. C. Kanellakis and C. H. Papadimitriou. Local search for the asymmetric traveling salesman problem. *Operations Research*, 28:1086–1099, 1980.

[12] R. M. Karp. A patching algorithm for the nonsymmetric traveling-salesman problem. *SIAM Journal on Computing*, 8:561–573, 1979.

[13] R. E. Korf. A complete anytime algorithm for number partitioning. *Artificial Intelligence*, 105:133–155, 1998.

[14] V. Kumar. Search branch-and-bound. In *Encyclopedia of Artificial Intelligence*, pages 1468–1472. Wiley-Interscience, 2 edition, 1992.

[15] E. L. Lawler, J. K. Lenstra, A. H. G Rinnooy Kan, and D. B. Shmoys. *The Traveling Salesman Problem*. John Wiley & Sons, Essex, 1985.

[16] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21:498–516, 1973.

[17] S. Martello and P. Toth. Linear assignment problems. *Annals of Discrete Math.*, 31:259–282, 1987.

[18] D. L. Miller. personal communications, 1992.

[19] D. L. Miller and J. F. Pekny. Exact solution of large asymmetric TSP. *Science*, 251:754–761, 1991.

[20] C. H. Papadimitriou and K. Steiglitz. Some examples of difficult traveling salesman problems. *Operations Research*, 26:434–443, 1978.

[21] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Englewood Cliffs, NJ, 1982.

[22] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proc. AAAI-92*, pages 440–446, San Jose, CA, 1992.

[23] W. Zhang and R. E. Korf. Performance of linear-space search algorithms. *Artificial Intelligence*, 79:241–292, 1995.

[24] W. Zhang and R. E. Korf. A study of complexity transitions on the asymmetric Traveling Salesman Problem. *Artificial Intelligence*, 81:223–239, 1996.