Searching for Backbones and Fat: A Limit-Crossing Approach with Applications

Sharlee Climer and **Weixiong Zhang**
Computer Science Department
Washington University
St. Louis, MO 63130
Email: {sclimer,zhang}@cs.wustl.edu

Abstract

Backbone variables are the elements that are common to all optimal solutions of a problem instance. We call variables that are absent from every optimal solution *fat* variables. Identification of backbone and fat variables is a valuable asset when attempting to solve complex problems. In this paper, we demonstrate a method for identifying backbones and fat. Our method is based on an intuitive concept, which we refer to as *limit crossing*. Limit crossing occurs when we force the lower bound of a graph problem to exceed the upper bound by applying the lower-bound function to a constrained version of the graph. A desirable feature of this procedure is that it uses approximation functions to derive exact information about optimal solutions. In this paper, we prove the validity of the limit-crossing concept as well as other related properties. Then we exploit limit crossing and devise a pre-processing tool for discovering backbone and fat arcs for various instances of the Asymmetric Traveling Salesman Problem (ATSP). Our experimental results demonstrate the power of the limit-crossing method. We compare our pre-processor with the Carpaneto, Dell'Amico, and Toth pre-processor for several different classes of ATSP instances and reveal dramatic performance improvements.

Introduction

The *backbone* of a problem instance is referred to as the set of variables that are common to all optimal solutions for the given instance. These variables are critically constrained as the elimination of any one of them will negate any possibility of finding any optimal solution. Currently, there is a significant amount of research activity in finding backbone variables (Schneider *et al.* 1996) and correlating the size of the backbone with problem hardness and phase transitions (Monasson *et al.* 1999; Slaney & Walsh 2001; Zhang 2001; 2002).

In this paper, we develop and demonstrate a method for searching for backbone variables as well as variables of the opposite type - those that are not part of any optimal solution. We will call this set of variables the *fat* of the problem instance, as we would like to trim away these variables and derive a sparser problem instance that still retains all of the variables that are part of any optimal solution. When fat variables are found, we permanently delete them so as to reduce the size of the problem to be solved.

Our technique for discovering backbones and fat is a product of a general graph reduction technique which we call *limit crossing*. In this paper, we formulate the concept of limit crossing formally and prove its validity along with other related properties. Then we exploit limit crossing and devise a mechanism for discovering backbone and fat arcs for various instances of the Asymmetric Traveling Salesman Problem (ATSP). (The ATSP is the NP-hard problem of finding a minimum-cost complete tour for a set of cities in which the cost from city i to city j may not be the same as the cost from city j to city i.)

Limit crossing is a general procedure that is based on the concept of inclusion and exclusion principle from the field of combinatorics. It can be used on graph problems that have lower-bound and upper-bound functions. The essence of the concept is to make the lower-bound value exceed the upper-bound value by applying the lower-bound function to a constrained version of the graph. The branch-and-bound method was built on top of this concept.

To use limit crossing, we first find an upper bound for a problem instance by finding a feasible, though not necessarily optimal, solution. Then we exclude a set of variables from the graph and apply a lower-bound function to the new graph. If this lower bound is greater than the original upper bound, we realize that all optimal solutions for this problem instance must include at least one of the variables from the excluded set. A benefit of limit crossing is that we can use approximation techniques to extract exact information about optimal solutions.

We use limit crossing to identify backbone and fat variables. These identifications allow us to reduce the problem graph and solve the instance more quickly. Reducing the problem size is of great value for problems with high complexities, such as the ATSP. The next section describes examples of some existing graph reduction techniques. Then we formalize limit crossing, prove its validity, and present its properties. In the following section, we use limit crossing to reduce ATSP instances. The results of our experiment are then presented. Finally, in the last section, we conclude and plan for the future.

Previous and Related Work

The limit-crossing method is based on an intuitive idea, and as such, has appeared in a variety of forms in previous work. In this section we briefly describe several examples of related work in the realm of the Traveling Salesman Problem (TSP).

Our first example is a hypothetical algorithm described by Papadimitriou in (Papadimitriou 1995), where multiple decision problems are solved in order to construct an optimal solution. In this algorithm, the cost C of an optimal tour is found for a TSP instance by solving a series of decision problems in a binary search of the possible tour cost values. Then the arcs that comprise this optimal tour are discovered by considering each arc (i, j) in the entire graph one at a time. The cost of (i, j) is set to $C + 1$ and it is determined if the modified graph has a tour with a cost of C or less. If so, then this arc is not a necessary part of the optimal solution and its cost may be left equal to $C+1$. If there is no tour with a cost less than or equal to C, then (i, j) is part of the optimal tour and its cost must be returned to the original value. After all of the arcs are considered, the ones with values less than $C + 1$ comprise the optimal tour. In this case, the graph is completely reduced. The only arcs that remain are part of the optimal solution. (This algorithm assumes that there is only one optimal solution.) It is an example of the limit-crossing concept in which the excluded set of arcs contains only one element and the upper- and lower-bound functions are exact. However, the implementation of this hypothetical algorithm would not be practical.

Following is a brief description of three examples of pre-processing tools for graph reduction, or *sparsification*, techniques that have been developed to aid the solution of ATSP problem instances.

In (Miller & Pekny 1991), Miller and Pekny present a modified version of a popular branch-and-bound method for solving large ATSP instances. In a pre-processing step, all of the arcs with cost greater than a specified threshold are eliminated from the graph. After an optimal solution is found for the sparse graph, it is checked for optimality for the original dense graph. If it is not optimal, the threshold is increased and the entire process is repeated. This procedure continues until an optimal solution for the original graph is found. Although the entire process may have to be repeated a number of times, the total time for searching several sparse graphs for optimal solutions may be less than the time required to search a dense graph only once.

Both of the other sparsification examples are due to Carpaneto, Dell'Amico, and Toth (Carpaneto, Dell'Amico, & Toth 1995). In their paper, two variations of a reduction procedure are presented as pre-processing tools for the previously mentioned branch-and-bound method for optimally solving large ATSP instances. The reduction procedure is essentially a limit-crossing technique, however, only one of the variants yields exact information. This variant first finds an upper bound for the problem instance. We use this variant for comparisons with our pre-processor in the Experimental Results section. The other variant calculates an "artificial" upper bound by multiplying the lower-bound value by a predetermined value α. For both variants, the lower bound is found by applying the Assignment Problem (AP) method to the graph (Martello & Toth 1987). (The AP method involves finding a minimum-cost matching on a bipartite graph constructed by including all of the arcs and two nodes for each city where one node is used for the tail of all its outgoing arcs, and one is used for the head of all its incoming arcs.)

In both variations of the algorithm a *reduced cost* is calculated for each arc (i, j) by subtracting the value of its *dual variables* (derived from the AP) from the original cost of the arc. From sensitivity analysis in linear programming, the reduced cost is a lower bound on the increase of the value of the AP solution if the arc were forced to be included in the solution. Thus if the reduced cost of (i, j) is greater than or equal to the difference between the upper and lower bounds, inclusion of (i, j) in a solution will necessarily lead to a tour cost that is at least as large as the upper bound that is already derived. Therefore, the arc is eliminated from the graph.

This algorithm searches for limit crossing when a single arc (i, j) is forced to be included. It is important to observe that forcing the inclusion of an arc (i, j) is the same as forcing the exclusion of all other arcs with tail i or head j. Therefore, this algorithm is an example of limit crossing.

If there is only one optimal solution for the problem instance, this sparsification is exact for the first variant. (If there is more than one optimal solution, some arcs from optimal solutions may be excluded.) When an artificial upper bound is used, the procedure may need to be repeated a number of times, as in the Miller and Pekny algorithm. After the branch-and-bound search is completed, the solution is compared with the artificial upper bound. If it is greater than the artificial upper bound, then α is increased and the entire procedure is repeated. The program terminates when the solution for the sparse graph is less than or equal to the artificial upper bound.

The essential idea of limit crossing is very general and appears in a great number of loosely related methods. Following is an example of a technique that differs from our use of limit crossing, yet it embodies the fundamental limit crossing concept.

Our example is the previously mentioned depth-first branch-and-bound search method used to find optimal or approximate solutions for the ATSP (Carpaneto, Dell'Amico, & Toth 1995; Miller & Pekny 1991; Zhang 1993). We use this algorithm to optimally solve ATSP instances in the Experimental Results section. The AP method is used for the lower-bound function and a current upper bound is maintained throughout the search. For each node in the search tree, there are two corresponding sets of arcs. One set of arcs is forced to be included in a constrained AP solution, the other set is forced to be excluded. The sets are empty for the root node. Furthermore, the sets for each child node are supersets of their parent's sets. Whenever the value of the AP solution is greater than or equal to the upper bound, it is clear that the given sets of included / excluded arcs cannot lead to a solution that is better than the current upper bound. Since the descendents of this node will have supersets of these sets, they cannot yield better solutions either. Therefore, the node is pruned. This depth-first search uses limit crossing on the search space, as opposed to our use on the problem graph, and it eliminates nodes in the search tree, rather than arcs in the original problem graph.

Limit Crossing Formalized

In this section, we prove the validity of limit crossing and present two related properties. Let us consider a directed graph $G = (V, A)$ with vertex set $V = \{1, \ldots, n\}$, arc set $A = \{(i, j) \mid i, j = 1, \ldots, n\}$, and cost matrix $c_{n\times n}$ such that, for all $i, j \in V$, c_{ij} is the cost associated with arc (i, j) and $c_{ij} \geq 0$. Assume that we can exclude an arc (i, j) from G by setting c_{ij} to ∞, and that this exclusion has no effect on other values in the cost matrix.

Many graph problems can be defined as minimizing (or maximizing) functions subject to specific constraints. Without loss of generality, let us consider a minimizing function F operating on graph G as follows:

$$F(G) = min \left(\sum_{i\in V} \sum_{j\in V} c_{ij} x_{ij} \right) \tag{1}$$

$$x_{ij} \in \{0, 1\},\ i, j \in V$$

subject to constraints $\{C_1, \ldots, C_m\}$.

Let O_G be a set of arcs representing an optimal solution of G *i.e.*, $O_G = \{(i, j) \mid x_{ij} = 1\}$. Note that if $F(G)$ exists (if the constraints are satisfiable) then there exists at least one set of arcs O_G for graph G and $\sum_{(i,j)\in O_G} c_{ij} = F(G)$.

Theorem 1 *Let LB and UB, respectively, be any lower-bound and upper-bound functions, such that $LB(G) \leq F(G) \leq UB(G)$ for all directed graphs G for which $F(G)$ exists. If $S \subset A$, and $LB(G\backslash S) > UB(G)$, then every optimal solution O_G associated with $F(G)$ contains at least one arc s, such that $s \in S$.*

Proof By definition $F(G\backslash S) \geq LB(G\backslash S)$, by assumption $LB(G\backslash S) > UB(G)$, and by definition $UB(G) \geq F(G)$. Therefore,

$$F(G\backslash S) > F(G). \tag{2}$$

Assume there exists a set of arcs O_G associated with $F(G)$ such that $O_G \cap S = \emptyset$. Since O_G and S are disjoint, O_G is a feasible solution for $G\backslash S$ and its associated cost is $\sum_{(i,j)\in O_G} c_{ij} = F(G)$. Now, since O_G is a feasible solution for $G\backslash S$ and $F(G\backslash S)$ is a minimizing function of $G\backslash S$, $F(G\backslash S) \leq F(G)$.

This results in a contradiction, as from (2), we have $F(G\backslash S) > F(G)$. Therefore, every optimal solution O_G associated with $F(G)$ contains at least one arc that is an element of S. □

If S is composed of only one arc, then that arc is essential for all optimal solutions. This gives us the following corollary:

Corollary 2 *If $a \in A$ and $LB(G\backslash\{a\}) > UB(G)$, then a is part of the backbone of $F(G)$.*

Assume our lower-bound function LB is the same minimizing function as (1), with one or more of the constraints $C_1, \ldots, C_m$ relaxed. (A minimizing function with one or more constraints relaxed will necessarily yield values less than or equal to the values computed by the same minimizing function with all the constraints enforced.)

Let L_G be the set of arcs associated with this lower-bound function acting on G. Therefore, $LB(G) = \sum_{(i,j)\in L_G} c_{ij}$.

Theorem 3 *If $LB(G\backslash S) > UB(G)$, then there exists an $s \in S$ such that $s \in L_G$.*

The proof of this theorem is similar to the proof of theorem 1. This theorem provides information that is useful when selecting arcs to be included in set S. In order to make it possible for limits to cross, S must include at least one arc from the set associated with the lower-bound function acting on G.

Pushing Traveling Salesmen Across the Limits

In this section we apply the limit-crossing method to the ATSP. First we develop the mathematical tools we will use. Then our algorithm will be summarized and its complexity considered.

Given directed graph $G = (V, A)$ with vertex set $V = \{1, \ldots, n\}$, arc set $A = \{(i, j) \mid i, j = 1, \ldots, n\}$, and cost matrix $c_{n\times n}$ such that $c_{ij} \geq 0$ and $c_{ii} = \infty$, the ATSP can be defined as a constrained minimizing function TS as follows:

$$TS(G) = min \left(\sum_{i\in V} \sum_{j\in V} c_{ij} x_{ij} \right) \tag{3}$$

subject to the following constraints:

$$x_{ij} \in \{0, 1\},\ i, j \in V \tag{4}$$

$$\sum_{i\in V} x_{ij} = 1,\ j \in V \tag{5}$$

$$\sum_{j\in V} x_{ij} = 1,\ i \in V \tag{6}$$

$$\sum_{i\in W} \sum_{j\in W} x_{ij} \leq |W| - 1,\ \forall\, W \subset V,\ W \neq \emptyset \tag{7}$$

Let T_G be the set of arcs representing an optimal tour of G: $T_G = \{(i, j) \mid x_{ij} = 1\}$.

Assume there exists a lower-bound function LTS, such that $LTS(G) \leq TS(G)$ and an upper-bound function UTS, such that $UTS(G) \geq TS(G)$ for all directed graphs G.

Assume $i \in V$, $V_s \subset V$, and $S = \{(i, k) \mid k \in V_s, k \neq i\}$. That is, S is a subset of the arcs for which i is the tail.

Theorem 4 *If $LTS(G\backslash S) > UTS(G)$, then no arc (i, j) such that $j \notin V_s$ can be part of any optimal solution for graph G.*

Proof Since $LTS(G\backslash S) > UTS(G)$, it follows from theorem 1 that every optimal solution for graph G contains at least one arc $s \in S$. However, constraint (6) dictates that there is precisely one arc with tail i in every optimal solution. Therefore, arcs (i, j) such that $j \notin V_s$ cannot be a part of any optimal solution. □

Similarly, for $j \in V$, $V_s \subset V$, and $S = \{(k, j) \mid k \in V_s, k \neq j\}$; $LTS(G\backslash S) > UTS(G)$ entails that there is no arc $(k, j) \mid k \notin V_s$ that can be part of any optimal solution for graph G.

These theorems provide us with tools for finding backbone and fat arcs and allow us to reduce the size of ATSP graphs. We directly apply these tools in our algorithm as described below.

Our algorithm is comprised of two steps: finding an upper bound for the original graph, then eliminating sets of arcs with common tail vertices. We are currently using the Assignment Problem method for finding the lower bound for our problem instances. Let $|AP|$ equal the cost of the Assignment Problem solution, and AP equal the set of arcs corresponding to this solution.

First, we find an upper bound value using the Zhang algorithm (Zhang 1993). This algorithm provides a good approximation very quickly (Johnson *et al.* in print). Next, we consider each vertex i and attempt to eliminate a set of arcs with i as their tail. We use theorem 4 to accomplish this. From theorem 3 we know that an arc from the AP solution must be in S for there to be any possibility of the limits crossing, so the first arc selected is $(i, j) \in AP$. Note there is precisely one arc that is in AP with tail i. We exclude this arc from G and find $|AP|$ for $G - (i, j)$. If this value is greater than the original upper bound, we have found an arc that is part of the backbone for the problem instance. If not, we add another arc to S and try again. The next arc to be excluded is the arc (i, k) that is in the AP solution for $G - (i, j)$. If the $|AP|$ for this graph exceeds the upper bound, then we know that no other arc with tail i can be in any optimal solution, so we exclude those arcs permanently, keeping only (i, j) and (i, k). If the limits do not cross, we continue to exclude arcs returned by the AP solutions and checking for crossings. Whenever the crossing occurs, we retain the arcs in S and permanently eliminate the arcs that have tail i and are not in S. The eliminated arcs are part of the fat of the problem instance.

The time complexity of finding the upper bound using the Zhang algorithm is $O(n^4)$, where n is the number of cities (Zhang 1993). In the second step, an AP solution is derived for each arc in set S. The first of the AP calls requires $O(n^3)$ time, but this call is made during the Zhang procedure in the previous step. The rest require $O(n^2)$ time (Martello & Toth 1987). Therefore, the overall time complexity for the second step is $O(kn^2)$, where k is the number of arcs kept in the graph after applying limit crossing.

In the next section, we compare our limit-crossing pre-processor with the pre-processor that was developed by Carpaneto, Dell'Amico, and Toth (CDT) and described previously. Although the CDT pre-processor identifies backbone and fat arcs using a limit-crossing procedure, it is fundamentally different from our method. First of all, the CDT technique is tied to the AP lower bound, whereas the limit-crossing technique can be used with any lower-bound function. This is advantageous as the AP lower bound is not very tight for many classes of ATSP graphs (Johnson *et al.* in print). Second, the CDT algorithm considers each arc, one at a time. Our procedure is more powerful as we are free to choose clusters of arcs. Finally, the CDT pre-processor can only identify backbone arcs by the process of elimination. Only fat arcs can be directly discovered. If all but one of the arcs with a common head or tail are identified as fat arcs, then a backbone arc is found. Our method works in the opposite direction. We consider a set of arcs with a common tail and check to see if we can identify a backbone arc immediately. If we can't identify a backbone arc in the set, we search for the minimum subset of these arcs that we can identify as containing all of the arcs from the original set that are not fat arcs. Although these pre-processors possess some similarities, our method derives more power by fully exploiting the limit-crossing concept.

Experimental Results

In this section, we first compare the limit-crossing pre-processor with the CDT pre-processor that was described previously for a variety of ATSP classes. Then we compare the two pre-processors for various sizes of random problem instances. In our tests, we used a general AP algorithm. The time requirement for pre-processed graphs could be reduced by substituting an AP algorithm that is particularly suited for sparse graphs (Carpaneto, Dell'Amico, & Toth 1995), instead of this general method. Therefore, the CPU times shown in our tables could be reduced.

We use the Zhang algorithm to compute the upper bound for the CDT pre-processor so that both pre-processors employ the same upper-bound function. Furthermore, since we are searching for backbones and fat, we adjusted the CDT pre-processor to only exclude arcs in which the lower bound exceeded the upper bound; thus retaining arcs for which the limits are equal.

In our tests, the backbone and fat arcs that were identified using the CDT pre-processor were a subset of those found using limit crossing. This is not surprising as both methods use the same upper-bound function and variations of the AP solution for the lower-bound function.

In our first set of tests, we compare the numbers of backbone and fat arcs found by both pre-processors for ten different classes of problem instances. These classes are described in detail in (Cirasella *et al.* 2001; Johnson *et al.* in print) and correspond to real-world applications of the ATSP.

For our tests, we used 300 cities with a generator parameter of 300. (Larger values for the generator parameter generally correspond to a larger range of values for the arc costs, however, the classes `super` and `coins` are not dependent on the generator parameter). We computed the average values for 100 trials for each class.

Table 1 shows the effectiveness of each of the pre-processors. The table also shows the relative error, or *duality gap*, for the upper- and lower-bound functions. This error is equal to the difference of the two bounds divided by the lower-bound value. The relative errors for the classes fall into two well-defined clusters. The classes `amat`, `tmat`, `disk`, `super`, and `crane` have relative errors that are less than 3%. All of the other classes have relative errors in the range of 22% to 46%.

For the `amat`, `tmat`, and `disk` classes, the limit-crossing pre-processor found more than twice as many backbone arcs as the CDT pre-processor. An interesting result

class	% rel. error	CDT pre-processor: backbone arcs found	CDT pre-processor: fat arcs found	Limit crossing: backbone arcs found	Limit crossing: fat arcs found	% more bb found	% more fat found
tmat	0.06	46.20	78,326	93.02	82,538	101	5.4
amat	0.47	39.26	88,562	104.88	88,959	167	0.4
disk	0.85	5.33	85,140	14.49	86,032	172	1.0
super	1.25	0.00	37	0.00	144	-	289
crane	2.78	0.00	56,762	0.00	60,999	-	7.5
coins	22.53	0.00	0	0.00	0	-	-
tsmat	27.25	0.00	0	0.00	0	-	-
stilt	32.72	0.00	0	0.00	0	-	-
rtilt	35.55	0.00	0	0.00	0	-	-
smat	45.81	0.00	41,978	0.00	42,474	-	1.2

Table 1: Comparisons of the pre-processors for various classes of ATSP instances ($n = 300$).

is that the `super` class has a small relative error, however, pre-processing is not nearly as effective for this class as it is for the other classes with small relative errors. We observed that `super` problem instances tend to have a large number of optimal solutions. (We found an average of 58.4 solutions for 100-city instances.)

The other five classes form a cluster with large relative errors. Four of these classes have poor results, as can be expected. Surprisingly, a large number of fat arcs were found for the `smat` instances despite a relative error of 45.81%. These graphs contain 89,700 arcs and nearly half of these arcs were identified as fat. It is interesting to note that for `tsmat` - the variant of `smat` that obeys the triangle inequality - not one fat arc was discovered.

In our second set of tests, we pre-processed random ATSP graphs and then found all the optimal solutions using each of the two methods. We used a depth-first branch-and-bound method (similar to the method discussed previously) to find all of the optimal solutions for the reduced graphs. We varied the number of cities n and ran 100 trials for each value of n. Furthermore, the arc cost values range uniformly from zero to n for each case. These values result in a large number of solutions, and finding all of these solutions is a computationally expensive task (Zhang 2002).

Table 2 presents the results of these tests. Note that the relative errors are extremely small as our upper- and lower-bound functions are exceptionally tight for these random instances. Furthermore, the relative error decreases with increasing n because the AP function becomes more accurate as n increases.

As shown in table 2, the limit-crossing method finds two to three times more backbone arcs than the AP-based method. Furthermore, roughly half of the fat arcs that are overlooked by the CDT pre-processor are found using limit crossing. Although the limit-crossing method requires more AP calls for the pre-processing step, the total number of AP calls is still less, due to the increased sparsification. Note that the average time divided by the number of AP calls tends to be larger for the CDT method than for the limit-crossing method. This is probably due to the fact that a number of AP calls are made in the pre-processing stage for the limit-crossing method, and other computational costs in this stage are very small.

In these tests, the time to find all of the optimal solutions is less when the limit-crossing pre-processor is used. Furthermore, the time savings tend to increase with larger problem sizes. For the cases with $n \geq 500$, the limit-crossing method required less than one-third the time required by the CDT method. After more than five days of running, the CDT tests with $n = 700$ have not finished. For the first 50 trials, the median time was 559 seconds (yielding a median time ratio of about 3.8).

Conclusions and Future Directions

In this paper, we developed a technique for identifying backbone and fat variables. We also formalized limit crossing, an intuitive concept that uses approximation techniques to derive exact information about backbone and fat variables and, consequently, about optimal solutions. Although limit crossing has been previously utilized in a variety of forms, we exploit this concept and demonstrate its utility and power for discovering backbone and fat variables. The identification of backbone and fat variables allows us to reduce the problem graph and optimally solve the instance in less time. Our experimental results comparing the limit-crossing pre-processor with the Carpaneto, Dell'Amico, and Toth (CDT) pre-processor attest the power of limit crossing. Whenever either method identified backbone or fat arcs, limit crossing consistently found more fat arcs than CDT and it consistently found two to three times as many backbone arcs. In many cases, instances that were pre-processed with limit crossing required less than one-third the time to solve than the same instances that were pre-processed using the CDT method. Furthermore, the benefits of limit crossing appear to increase with increasing problem size for the class of random asymmetric matrices.

For the immediate future, we plan to experiment with approximation functions for the ATSP and study interesting structural properties of ATSP graphs. These properties include number of cities, range of cost values, degree of symmetry, degree of obeyance of the triangle inequality, number of solutions, size of backbone, size of fat, and accuracy of approximation functions. We are interested in the interplay

n	% rel. error	CDT pre-processor					Limit-crossing pre-processor					median time ratio
		% bb found	% fat found	# AP calls	mean time	median time	% bb found	% fat found	# AP calls	mean time	median time	
100	1.55	16.32	97.28	1737	0.15	0.10	42.34	98.61	1516	0.12	0.08	1.3
200	0.74	16.11	98.68	9188	2.44	1.37	43.67	99.35	6828	1.38	0.76	1.8
300	0.47	17.02	99.16	33,604	18.5	10.0	45.46	99.60	24,070	8.99	3.56	2.8
400	0.37	17.22	99.33	138,105	130	31.7	46.91	99.67	99,337	50.5	15.9	2.0
500	0.28	16.94	99.48	431,377	625	95.6	47.21	99.75	259,152	199	31.8	3.0
600	0.21	19.17	99.60	640,220	1266	398	51.58	99.81	442,695	609	119	3.3
700	0.19	-	-	-	-	-	48.98	99.83	1,293,702	2541	148	-

Table 2: Comparisons of the pre-processors when finding all optimal solutions for random ATSP instances.

of these properties and the effectiveness of limit crossing.

Our long-term plans include studying other graph problems with high complexities and attempting to derive limit-crossing tools that may be useful to identify backbone and fat variables within their domains.

Acknowledgments

This research was funded in part by NSF Grants IIS-0196057 and IET-0111386, NDSEG and Olin Fellowships, and in part by DARPA Cooperative Agreements F30602-00-2-0531 and F33615-01-C-1897. We are grateful for useful suggestions provided by the anonymous referees.

References

Carpaneto, G.; Dell'Amico, M.; and Toth, P. 1995. Exact solution of large-scale, asymmetric Traveling Salesman Problems. *ACM Trans. on Mathematical Software* 21:394–409.

Cirasella, J.; Johnson, D.; McGeoch, L.; and Zhang, W. 2001. The asymmetric traveling salesman problem: Algorithms, instance generators, and tests. In *Proc. of the 3rd Workshop on Algorithm Engineering and Experiments*.

Johnson, D. S.; Gutin, G.; McGeoch, L. A.; Yeo, A.; Zhang, W.; and Zverovich, A. in print. Experimental analysis of heuristics for the ATSP. To appear in The Traveling Salesman Problem and its Variations, G. Gutin and A. Punnen, Editors, Kluwer Academic Publishers.

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

Miller, D. L., and Pekny, J. F. 1991. Exact solution of large asymmetric traveling salesman problems. *Science* 251:754–761.

Monasson, R.; Zecchina, R.; Kirkpatrick, S.; Selman, B.; and Troyansky, L. 1999. Determining computational complexity from characteristic 'phase transitions'. *Nature* 400:133–137.

Papadimitriou, C. H. 1995. *Computational Complexity*. New York, NY: Addison-Wesley.

Schneider, J.; Froschhammer, C.; Morgenstern, I.; Husslein, T.; and Singer, J. M. 1996. Searching for backbones - an efficient parallel algorithm for the traveling salesman problem. *Comput. Phys. Commun.* 96:173–188.

Slaney, J., and Walsh, T. 2001. Backbones in optimization and approximation. In *IJCAI-01*.

Zhang, W. 1993. Truncated branch-and-bound: A case study on the Asymmetric Traveling Salesman Problem. In *Proc. of AAAI 1993 Spring Symp. on AI and NP-Hard Problems*, 160–166.

Zhang, W. 2001. Phase transitions and backbones of 3-sat and maximum 3-sat. In *7th Intern. Conf. on Principles and Practice of Constraint Programming (CP-2001)*, 153–167.

Zhang, W. 2002. Phase transitions, backbones, measurement accuracy, and phase-aware approximation: The ATSP as a case study. *Proc. of CP-AI-OR 02* 345–357.