Proceedings CPAIOR'02

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

Sharlee Climer and Weixiong Zhang
Department of Computer Science
Campus Box 1045
Washington University
One Brookings Drive
St. Louis, MO USA 63130-4899
email: `sclimer@cs.wustl.edu`, `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 Carpaneto's pre-processor for several different classes of ATSP instances and reveal dramatic performance improvements. Our experiment also demonstrates the utility of searching for fat arcs.

*The research was supported in part by NSF grants IIS-0196057 and EIA-0113618, by fellowships from NDSEG and Olin Foundation, and in part by DARPA Cooperative Agreements F30602-00-2-0531 and F33615-01-C-1897.

1 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 [10] and correlating the size of the backbone with problem hardness and phase transitions [8, 11, 14].

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. In this paper, we demonstrate the utility of discovering fat, as well as backbone, variables.

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. With concise proofs in hand, 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 approximation 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.

A number of techniques for reducing graph problems have been developed in the past. 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. In section 3, we formalize limit-crossing, prove its validity, and present its properties. Armed with concise tools, we are able to exploit limit-crossing and reduce ATSP instances in section 4. The results of our experiment are presented in section 5. In section 6 we discuss characteristics of the limit-crossing method that we have discovered in the course of our work. Finally, in the last section, we conclude and plan for the future.

2 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). We conclude this section with examples illustrating the broad range of methods that are loosely related to limit-crossing.

Our first example is a hypothetical algorithm described by Papadimitriou in [9], 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 [7], 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 [2]. 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 section 5. 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 Method (AP) method to the graph. (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 are two examples of techniques that differ from our use of limit-crossing, yet embody the fundamental limit-crossing concept.

Our first example is the previously mentioned depth-first branch-and-bound search method used to find optimal or approximate solutions for the ATSP [2, 7, 13]. We use this algorithm to optimally solve ATSP instances in section 5. 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 pruning criteria is similar to limit-crossing. However, the nodes of the search tree, not arcs in the original ATSP graph, are eliminated.

Our final example is a propagation method used to reduce job-shop scheduling problem instances. Job-shop scheduling is the NP-hard problem of optimizing the scheduling of a set of jobs on a set of machines such that the chain of operations for each job is performed without interruption. A number of variants of this reduction method have been introduced as *global operations* [1] and *shaving* techniques [6, 12]. Essentially, these variants assume that a job is started within a reduced portion of its actual time-window and propagate this restriction throughout the system, looking for an infeasibility. If an infeasibility occurs, the original time-window can be reduced. This method constrains the original problem by excluding possible starting times for a job. If these exclusions lead to limit-crossing (infeasibility) then it is realized that the job must be started during one of the times in the excluded set.

3 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 there exists at least one set of arcs O_G for graph G and $\sum_{(i,j) \in O_G} c_{ij} = F(G)$.

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

Theorem 3.1 *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. $\square$

Theorem 3.1 is quite intuitive. If we exclude a set of arcs and find that now the least possible value is greater than the optimal solution of the original graph, then the optimal solution of the new graph must be greater than the optimal solution of the original graph. Since the value of the optimal solution has increased, we realize that at least one of the excluded arcs is necessary to maintain the original optimal solution.

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

Corollary 3.2 *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 3.3 *If $LB(G \backslash S) > UB(G)$, then there exists an $s \in S$ such that $s \in L_G$.*

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

$$LB(G \backslash S) > LB(G). \tag{3}$$

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

This results in a contradiction, as from (3), we have $LB(G \backslash S) > LB(G)$. Therefore, there exists an $s \in S$ such that $s \in L_G$. $\square$

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.

4 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. In section 4.2 the algorithm will be summarized and its complexity considered.

4.1 Mathematical Foundation

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:

$$TS(G) = min \left(\sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \right) \tag{4}$$

subject to the following constraints:

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

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

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

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

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.

The following theorem provides us with a tool for finding arcs that cannot be part of any optimal tour of G. Essentially, we force the inclusion of arc (i,j) and see if our new lower bound (with (i,j) included) exceeds our original upper bound. In the theorem, we force the inclusion of arc (i,j) by excluding all other arcs with tail i and head j.

Theorem 4.1 *Assume $(i,j) \in A$ and $S = \{(k,l) \mid (k = i$ or $l = j)$ and $(k,l) \neq (i,j)\}$. If $LTS(G \backslash S) > UTS(G)$, then (i,j) cannot be an arc in any optimal tour T_G associated with $TS(G)$.*

Proof From Theorem 3.1, every optimal tour T_G associated with $TS(G)$ contains at least one arc $s \in S$. However, due to constraints (5) and (6) from above, (i,j) cannot be an element of any optimal tour that also contains an element of S. Therefore, (i,j) cannot be an arc in any optimal tour associated with 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 4.2 *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 3.1 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.

4.2 ATSP Reduction Algorithm

We have developed an ATSP pre-processing tool which we call the *limitcrosser*. 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 [13]. This algorithm provides a good approximation very quickly [4]. The Helsgaun algorithm yields a tighter upper bound for some problem instance types, however, the run time for the Helsgaun method can be substantially longer [4].

Next, we consider each vertex i and attempt to eliminate a set of arcs with i as their tail. We use theorem 4.2 to accomplish this. From theorem 3.3 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 the limits cross, 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.

In our implementation, we have a predetermined threshold, τ, which is used to abort the search for limit-crossing for a vertex. If the cost of the newest member of S exceeds τ, we abort and move on to the next vertex. The user can vary the value of τ according to their preferences. This value reflects a time/performance trade-off. Our algorithm is depicted in Table 1.

Algorithm 1 ATSP pre-processing using limit-crossing.

```
for (each city in the graph) do
  while (|AP| ≤ upper bound) do
    find the arc in AP leaving the city;
    set the cost of this arc to ∞;
    find new AP;
  end while
  remove the outgoing arcs that were not set to ∞;
  reset the costs of the arcs that were set to ∞;
end for
```

The time complexity of finding the upper bound using the Zhang algorithm is $O(n^4)$, where n is the number of cities [13]. 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 [5]. 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 the limitcrosser with the AP-based pre-processor that was developed by Carpaneto, Dell'Amico, and Toth and described in section 2. Although the AP-based pre-processor identifies backbone and fat arcs using a limit-crossing procedure, it is fundamentally different from our method. First of all, the AP technique is tied to the AP lower bound, whereas the limitcrosser 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 [4]. Second, the AP algorithm considers each arc, one at a time. Our procedure is more powerful as we are free to chose clusters of arcs. Finally, the AP pre-processor can only identify backbone arcs by the process of elimination. Only

n	% rel. error	AP-based pre-processor % back bone found	% fat found	# AP calls	time (sec)	Limitcrosser % back bone found	% fat found	# AP calls	time (sec)	time ratio
100	1.72	14.21	97.15	1954	0.16	38.95	98.50	1681	0.14	1.14
200	0.69	18.18	98.71	8222	2.08	46.27	99.36	6280	1.30	1.60
300	0.48	15.68	99.12	32,394	16.7	44.43	99.59	23,007	7.90	2.11
400	0.40	14.71	99.30	99,884	85.3	41.68	99.65	76,619	46.5	1.83
500	0.26	17.71	99.50	590,625	854	46.32	99.76	357,670	240	3.56
600	0.17	22.42	99.63	668,205	1348	58.01	99.84	438,232	538	2.51
700	0.16	21.87	99.67	2,046,917	5039	57.76	99.85	810,393	550	9.17

Table 1: Comparisons of the pre-processors when finding all optimal solutions for random ATSP instances.

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.

5 Experimental Results

In this section, we first compare the limitcrosser pre-processor with the AP-based pre-processor that is described in section 2 for various sizes of random problem instances. Then we compare the two pre-processors for a variety of ATSP classes. Finally, we demonstrate the utility of discovering fat arcs by experimenting with a single class of 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 [2], instead of this general method. Therefore, the CPU times shown in our tables could be reduced.

Table 1 presents the results of finding all of the optimal solutions for random ATSP graphs using the two pre-processors. We used a depth-first branch-and-bound method (similar to the method discussed in section 2) to find all of the optimal solutions for the reduced graphs. We varied the number of cities n and averaged 50 trials for n up to 400, 25 trials for n equal to 500 and 600, and 10 trials for n equal to 700. 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 [15].

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. 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.

We use the Zhang algorithm to compute the upper bound for the AP-based 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 AP 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 AP-based pre-processor were a subset of those found using the limitcrosser. This is not surprising as both methods use the same upper-bound function and variations of the AP solution for the lower-bound function. As shown in table 1, the limitcrosser 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 AP pre-processor are found using the limitcrosser. Although the limitcrosser 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 time to find all of the optimal solutions is less when the limitcrosser is used. Furthermore, the time savings increase with larger problem sizes. The AP method required an average of 84 minutes to solve the 700-city instances while the limitcrosser solved the instances in about 9 minutes on average.

In our second 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 [3, 4] and correspond to real-world applications of the ATSP.

The first class is `amat`. These cost matrices are simply random asymmetric values. `tmat` cost matrices are the same as `amat`, except the costs are forced to obey the triangle inequality. `smat` matrices are random symmetric cost matrices. Similarly, `tsmat` matrices are random symmetric matrices in which the triangle inequality is obeyed. `rtilt` and `stilt` are tilted drilling machine instances using two different norms. The `crane` matrices correspond to the stacker crane problem. Scheduling for the read head on a computer disk is modeled by the `disk` class. The `coins` matrices model route planning for collecting coins from pay phones. The last instance type is `super`, modeling the problem of finding approximate shortest common superstrings.

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 fifty trials for each class.

Table 2 shows the effectiveness of each of the pre-processors. 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 limitcrosser found more than twice as many backbone arcs as the AP-based pre-processor. An interesting result 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.)

class	% rel. error	AP pre-processor backbone arcs found	fat arcs found	Limitcrosser backbone arcs found	fat arcs found	% more backbone found	% more fat found
tmat	0.12	44.52	74,365	89.76	79,287	102	6.6
amat	0.48	37.02	88,532	104.92	88,949	183	0.5
disk	0.99	6.76	84,537	15.34	85,470	127	1.1
super	1.31	0.00	3	0.00	24	-	700
crane	2.72	0.00	57,484	0.00	61,688	-	7.3
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	45.50	0.00	42,242	0.00	42,734	-	1.2

Table 2: Comparisons of the pre-processors for various classes of ATSP instances ($n = 300$).

generator parameter	% rel. error	No pre-processor # of AP calls	time (sec)	Limitcrosser % fat found	# of AP calls	time (sec)	time ratio
100	54.54	5,565,833	103	30.73	4,350,211	95	1.08
1000	55.72	7,358,722	104	29.18	4,346,198	100	1.04
10,000	55.41	10,992,340	155	29.75	4,801,047	108	1.44
100,000	56.54	11,258,885	161	29.05	5,223,506	119	1.35
1,000,000	56.54	17,514,078	347	29.04	4,995,078	114	3.04

Table 3: Comparisons of finding all optimal solutions for `disk` problem instances with and without limit-crossing.

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 45.50%. 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.

Finally, we demonstrate the utility of fat discovery by testing 30-city `disk` instances for various generator parameters and averaging over 10 trials. As shown in table 3 about 30% of the fat arcs are discovered during pre-processing. However, none of the backbone arcs were identified. In all cases, the time and the number of AP calls required to find all of the optimal solutions is less when the limitcrosser is used, despite the fact that only 30% of the fat arcs were discovered. When the parameter is set to 1,000,000, limit-crossing cuts the time to less than one-third of the time required without pre-processing. Since no backbone arcs were discovered, this test demonstrates the benefit of identifying fat arcs.

6 Discussion

In the course of our work, we have discovered some interesting characteristics of the limit-crossing method. A brief discussion of these follows.

When using limit-crossing, we are only able to search for arcs that are part of the backbone or part of the fat. For problems with many optimal solutions, there may be many arcs that don't fall into either category, thus limiting the amount of reduction possible.

Theorem 3.3 demonstrates another characteristic. It is necessary for set S (the set of arcs that have been temporarily excluded from the graph) to include at least one arc in the lower-bound solution to make limit-crossing possible. Thus, we can prove that an arc in the lower-bound solution is part of the backbone, however, we are unable to prove that it is part of the fat, even if it isn't part of any optimal solution. Furthermore, we are unable to identify an arc that is not in the lower-bound solution as part of the backbone. However, alternate lower-bound algorithms may yield different arcs in their solution sets.

An important characteristic of limit-crossing is its strong dependency on the quality of the upper- and lower-bound functions. When these functions return values that are close to the optimal solution, limit-crossing may occur with a small number of arcs in set S. However, when these functions are far apart, the size of S that causes limit-crossing may be quite large. In these cases, the amount of reduction may be small. It is important to note that this dependency does not always exist, as demonstrated by the `smat` class (see section 5).

While developing our ATSP pre-processing tool, we were surprised by the simplicity of the tool that was necessary. At first we planned to have two more steps in our algorithm and then loop through the entire procedure until no new backbone or fat arcs are discovered. The third step is similar to the second step, however, this step considers each vertex, j, one at a time and looks for limit-crossing while excluding the arcs that are returned from the AP function with head j. In the fourth step, we consider each arc (i, j) that is still left in the graph. We then try to cross the limits by deriving the AP value with all other arcs with tail i and head j excluded. If a crossing occurs, we know from Theorem 4.1 that (i, j) must be a fat arc and can be excluded.

In our limited testing, we found that none of these additional steps led to the discovery of any new fat or backbone arcs. Since they reduced the speed of the program, we only used the first two steps.

7 Conclusions and Future Work

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 established the advantage of fully exploiting this concept by demonstrating its utility and power in discovering backbone and fat variables. Our experimental results comparing the limitcrosser pre-processor with Carpaneto's AP-based pre-processor attest the power of limit-crossing. In many cases, instances that were pre-processed with limit-crossing required less than half the time to solve than the same instances that were pre-processed using the AP-based method. The most dramatic performance improvement witnessed was for the 700-city random instances. These instances were solved in less than one-ninth the time. 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 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.

References

[1] J. Carlier and E. Pinson. Adjustment of heads and tails for the job-shop problem. *European Journal of Operational Research*, 78:146–161, 1994.

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

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

[4] D. S. Johnson, G. Gutin, L. A. McGeoch, A. Yeo, W. Zhang, and A. Zverovich. 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.

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

[6] P. Martin and D. B. Shmoys. A new approach to computing optimal schedules for the job-shop scheduling problem. *Proceedings of the 5th International IPCO Conference*, pages 389–403, 1996.

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

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

[9] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, New York, NY, 1995.

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

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

[12] P. Torres and P. Lopez. Overview and possible extensions of shaving techniques for job-shop problems. In *CP-AI-OR'2000*, pages 181–186, Paderborn, Allemagne, March 2000.

[13] W. Zhang. 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*, pages 160–166, 1993.

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

[15] W. Zhang. Phase transitions, backbones, measurement accuracy, and phase-aware problem solving: The ATSP as a case study. *Proc. of CP-AI-OR 02*, 2002.