Chapter 5

Tolerance-based algorithms for the traveling salesman problem

Diptesh Ghosh*
P&QM Area, IIM Ahmedabad, India.

Boris Goldengorin
Faculty of Economic Sciences, University of Groningen, The Netherlands.

Gregory Gutin
Department of Computer Science, Royal Holloway University of London, Egham, Surrey TW20 OEX, UK and Department of Computer Science, University of Haifa, Israel.

Gerold Jäger
Computer Science Institute, University of Halle-Wittenberg, Germany.

Abstract
Most research on algorithms for combinatorial optimization use the costs of the elements in the ground set for making decisions about the solutions that the algorithms would output. For traveling salesman problems, this implies that algorithms generally use arc lengths to decide on whether an arc is included in a partial solution or not. In this paper we study the effect of using element tolerances for making these decisions. We choose the traveling salesman problem as a model combinatorial optimization problem and propose several greedy algorithms for it based on tolerances. We report extensive computational experiments on benchmark instances that clearly demonstrate that our tolerance-based algorithms outperform their weight-based counterpart. This indicates that the potential for using tolerance-based algorithms for various optimization problems is high and motivates further investigation of the approach.

Key Words: Traveling salesman problems, greedy algorithms, arc tolerances

5.1. Introduction

In this paper we propose several algorithms for the traveling salesman problem (TSP). In a TSP instance of size \( n \), we are given a weighted complete digraph \( D = (V, A, C) \) where \( V \) is the set of \( n \) vertices, \( A \) the set of arcs

*Corresponding author: diptesh@iimahd.ernet.in
between vertices in $V$, and $C = [c(i,j)]$ is the $n \times n$-matrix of arc weights, and we are required to find a Hamilton cycle (called a tour) such that the sum of the weights of the arcs in the tour is as small as possible. A TSP instance is called a symmetric TSP (STSP) instance if for each pair of vertices $i$ and $j$, $c(i,j) = c(j,i)$; and an asymmetric TSP (ATSP) instance otherwise. Also, a TSP instance is defined by the weight matrix $C$.

Most algorithms for solving the TSP make use of the arc weights to decide whether or not to include an arc in the solution that they finally output. For example, the weight-based greedy algorithm and its variations are popular heuristics to produce initial tours for local search and other improvement heuristics (see, e.g., [Gamboa et al. (2006)]). However, as pointed out in [Goldengorin et al. (2004)] and [Turkensteen (2005)], arc tolerances are better indicators than arc weights for generating good tours. An arc tolerance (see e.g., [Goldengorin et al. (2006)], [Goldengorin and Sierksma (2003)], [Libura (1991)]) is the maximum amount by which the weight of the arc that is in (not in) an optimal tour can be increased (respectively, decreased) while keeping other arc weights unchanged for the tour to remain optimal. Among currently known algorithms for the TSP, only Helsgaun's version of the Lin-Kernighan heuristic for the STSP (see [Helsgaun (2000)]) explicitly applies tolerances in algorithm design. Implicit applications of tolerances in algorithm design are found in the Vogel's method for the Transportation Problem and in the MAX-REGRET heuristic for solving the Three-Index Assignment Problem (see [Balas and Saltzman (1991)]).

To the best of our knowledge the concept of tolerances has not been applied to the design of greedy algorithms for the TSP prior to this paper. Our aim is to motivate research on the use of tolerances for decision making in fast TSP heuristics. The algorithms that we propose may therefore not be the best of breed, but they demonstrate the superiority of tolerance-based algorithms over their arc-weights counterparts. Our results thus indicate a high potential of tolerance-based algorithms for various optimization problems and motivate further investigation of the approach.

In the next section, we develop concepts that will help us to describe the algorithms that we introduce for the TSP in Section 5.3. Our tolerance-based greedy algorithms are described in Section 5.3. We report computational experience with our greedy algorithms in Section 5.4. We conclude the paper in Section 5.5 with a summary of our main contributions and suggestions for future research.
5.2. Some relevant concepts

5.2.1. The Relaxed Assignment Problem

The Assignment Problem (AP) is a well-known relaxation of the TSP, which is used more often for the ATSP than for the STSP. Let $D = (V, A, C)$ be a bipartite digraph with bipartition $V = V_1 \cup V_2$, $|V_1| = |V_2| = n$, and such that $A = V_1 \times V_2$. The AP is defined as the problem to find $n$ arcs $(s_i, t_i)$, $1 \leq i \leq n$ of minimum total weight such that $s_i \neq s_j$ and $t_i \neq t_j$ for every $1 \leq i \neq j \leq n$, i.e., the AP is the problem to find a minimum weight perfect matching. Notice that if $V_1$ and $V_2$ are two copies of the vertex set of an TSP instance, where the arc weights of the bipartite directed graph correspond to the arc weights of the TSP and the arc weight of a vertex and its copy is set to $\infty$, then the AP solution can be interpreted as a collection of cycles (called subtours) for the instance.

An integer programming formulation of the AP on an ATSP instance defined on a complete digraph $G = (V, A, C)$ (where $|V| = n$, and $C = [c(i, j)]$) using variables $x_{ij}$, $i, j \in V$ such that $x_{ij} = 1$ when $(i, j)$ is included in the solution and 0 otherwise, is given below.

$$\text{Minimize } \sum_{i=1}^{n} \sum_{j=1}^{n} c(i, j)x_{ij}$$

Subject to

$$\sum_{j=1}^{n} x_{ij} = 1 \quad i \in \{1, \ldots, n\} \quad (5.1)$$

$$\sum_{i=1}^{n} x_{ij} = 1 \quad j \in \{1, \ldots, n\} \quad (5.2)$$

$$x_{ij} \in \{0, 1\} \quad i, j \in \{1, \ldots, n\}$$

The Relaxed Assignment Problem (RAP) is a relaxation of the AP in which constraint set (5.2) is removed from the earlier formulation. Thus, instead of an one-to-one matching in case of the AP, in the RAP the first copy of $V$ maps into the second copy of $V$. Note that a solution to the RAP may not consist exclusively of cycles.

5.2.2. Determining tolerances for AP and RAP

Extending the informal definition of tolerances in the introductory section, the upper (lower) tolerance of an arc that is included in (respectively, excluded from) an optimal solution to the AP is the maximum amount by
which the weight of the arc can be increased (respectively, reduced) while keeping other arc weights unchanged, such that the current optimal solution to the AP remains optimal. Tolerances for arcs of the RAP can be defined analogously.

Computing arc tolerances for the AP involves revising the arc weight to a suitably high value if the arc is a part of the optimal solution, and a suitably low value if it is not (see [Goldengorin et al. (2006)]), and re-solving the AP. The AP can be solved in $O(n^3)$ time, and using a shortest path based approach, all arc tolerances can also be computed in $O(n^3)$ time (see [Volgenant (2006)])..

Computing arc tolerances for the RAP, on the other hand, is a more tractable problem. An optimal solution to the RAP can be characterized as a collection of arcs, one from each vertex in the graph, such that the weight of the arc is the smallest among those of all arcs from that vertex. Therefore, for each arc that belongs to an optimal solution to the RAP, its upper tolerance is the excess of the weight of the second smallest weight out-arc from the same vertex over the weight of that arc. If the arc is not in an optimal solution, then its lower tolerance would be the excess of the weight of the arc over that of the smallest weight out-arc from the same vertex. Obtaining all tolerances therefore requires finding the weights of the two least weight entries in each row of the cost matrix, and then performing a simple subtraction operation once for each arc. Both jobs can be achieved in $O(n^2)$ time so that the overall complexity of determining all arc tolerances for the RAP is $O(n^2)$ time.

5.2.3. The contraction procedure and a greedy algorithm

The (path) contraction procedure (see, e.g., [Glover et al. (2001)]) is a method of updating a digraph once a directed path is removed from it and replaced by a single vertex. Consider a digraph $D = (V, A, C)$ with $C = [c(i, j)]$ and a directed path $P = v_1 v_2 \cdots v_k$ in it. The contraction procedure for marking the path (and replacing it by a vertex $p$) replaces $D$ by a digraph $D_p$. The vertex set of $D_p$ is $V_p = V \cup \{p\} - \{v_1, \ldots, v_k\}$. The arc set of $D_p$ includes all arcs $(i, j)$ from $D$ where $i, j \notin P$. In addition for all vertices $i$ in $V_p$ except $p$, it introduces and includes arcs $(i, p)$ with weight $c(i, v_1)$ and $(p, i)$ with weight $c(v_k, i)$ (where $c$, $i$, $v_1$, and $v_k$ are defined for digraph $D$). In this paper we only need a special case of the path contraction procedure, namely contracting only a single arc (say $a$) from a digraph $D$. We use a shorthand notation $CP(a, D)$ for this procedure.
Tolerance-based algorithms for the traveling salesman problem

Given the contraction procedure, a generic greedy algorithm can be defined as follows:

**A generic greedy algorithm**

**Input:** A weighted complete digraph \( D = (V, A, C) \).

**Output:** A tour \( T \).

**Step 1:** \( G \leftarrow D, T \leftarrow \emptyset \).

**Step 2:** While \( G \) consists of at least three vertices, using a suitable myopic procedure, choose an arc (say \( a = (u, v) \)), that does not create a cycle, to include in the tour.

(For example, if arcs (1,2) and (2,3) are already contracted, the contraction of (3,1) would create a cycle.)

Set \( T \leftarrow T \cup \{a\} \), \( G \leftarrow CP(a, G) \).

**Step 3:** Set \( T \leftarrow T \cup \{(v_1, v_2), (v_2, v_1)\} \) and output \( T \).

This algorithm is generic since the myopic arc selection procedure used in Step 2 has not been defined. Typically greedy algorithms employ myopic procedures based on arc weights, choosing the least weight arc as the one to contract. Therefore, as a benchmark for tolerance-based algorithms that we present in the next section, we define the following variant.

**W-GREEDY algorithm:** At each iteration of the generic greedy algorithm, in Step 2, the myopic procedure chooses the least weight arc. This arc is chosen for contraction (i.e., inclusion in the tour).

### 5.3. Tolerance-based greedy algorithms

Since exploratory computations (see, e.g., [Turkensteen (2005)]) show that, given an optimal AP solution to an TSP instance, the ‘probability’ of the arc with the largest upper tolerance for the AP solution being in an optimal TSP solution is much higher than the ‘probability’ of the smallest weight arc being in an optimal TSP solution, it is interesting to create myopic procedures for the generic greedy algorithm developed in Section 5.2.3 which use tolerances instead of arc weights to choose arcs. In this section, we introduce the following three variants of such myopic procedures, leading to three greedy algorithms.

**R-R-GREEDY algorithm:** At Step 2 of each iteration of the generic greedy algorithm, the myopic procedure generates an optimal RAP solution on the digraph. Then the upper tolerances of each arc
included in the solution are generated. The arc in the optimal RAP solution with the largest upper tolerance is chosen for contraction (i.e., inclusion in the tour).

**A-R-GREEDY algorithm:** At each iteration, the myopic procedure generates an optimal AP solution and an optimal RAP solution on the digraph. For each arc in the AP solution and in the RAP solution, the upper tolerance (w.r.t. the RAP) is computed, and for each arc in the AP solution but not in the RAP solution, the lower tolerance (w.r.t. the RAP) is computed, and multiplied with $-1$. The values thus obtained are sorted, and the arc with the largest value is chosen for contraction.

The relaxation of constraint set (5.2) in the formulation of AP to generate RAP was arbitrary. One could easily come up with another relaxation of the AP (let us call it RAP1) in which constraint set (5.1) is relaxed instead of the set (5.2). The third algorithm implements a myopic procedure that uses both the RAP and RAP1 relaxations.

**A-RC-GREEDY algorithm:** Optimal solutions are generated for AP as well as for RAP and RAP1. The myopic procedure described in the A-R-GREEDY algorithm is carried out twice, once with the optimal solutions to AP and RAP, and the second time with the optimal solutions to AP and RAP1. In the second case, the tolerances are computed with respect to the RAP1 relaxation. Of the two candidates that emerge from the two procedures, the one which has a larger value is chosen for contraction.

Note that for A-R-GREEDY and A-RC-GREEDY we only approximate the upper tolerances for the AP. The reason is, that in practice, solving an AP in $O(n^3)$ time by the Hungarian algorithm and then computing approximate tolerances in $O(n^2)$ time is much faster than using the Hungarian algorithm and then Volgenant’s method (see [Volgenant (2006)]) for exactly computing the tolerances in $O(n^3)$, even though both methods have an overall $O(n^3)$ time complexity.

The greedy algorithms described above can be speeded up considerably using book-keeping techniques. For example, in R-R-GREEDY, if in an iteration, the end vertex of the contracted arc does not contain a smallest or a second smallest weight arc from any of the vertices, then in the next iteration, both the RAP solution and the upper tolerances remain unchanged. Even otherwise, the changes in the RAP solution and upper tolerance at
the next iteration involve only those vertices from which the smallest or second smallest weight arcs were directed to the end vertex of the contracted arc in the previous iteration. Furthermore, in the A-R-GREEDY and A-RC-GREEDY algorithms, if the arc contracted does not belong to a subtour with two arcs only, the optimal AP solution before and after the contraction operation differ only by the arc contracted.

Our extensive computational experiments with the W-GREEDY and R-R-GREEDY applied to a wide set of the AP instances with \( n \geq 100 \) (see [Dell’Amico and Toth (2000)] for a description of the instances) show that the quality of R-R-GREEDY solutions is at least 10 times better than the quality of W-GREEDY solutions and these results are further supported by domination analysis. The domination number of a heuristic \( H \) for a combinatorial optimization problem \( P \) is the maximum number of solutions that are not better than the solution found by \( H \) for any instance of size \( n \). The domination number of W-GREEDY for the AP equals 1 [Gutin and Yeo (2005)], i.e., for every \( n \) there are instances of AP for which W-GREEDY finds the unique worst solution. It can be shown that the domination number of R-R-GREEDY for the AP is exponential.

In the next section, we compare the three tolerance-based greedy algorithms introduced in this section with each other on benchmark TSP instances using the W-GREEDY algorithm to calibrate the algorithms. Since the performance of the W-GREEDY algorithm has been compared with other well-known algorithms for the TSP (see, e.g., [Glover et al. (2001)]), the next section also provides an indirect comparison of the three algorithms proposed here with those algorithms.

5.4. Computational experience

The four greedy algorithms mentioned in the paper were implemented in order to observe their performance on benchmark instances of the TSP. The implementations were done in C under Linux on a GenuineIntel Intel® Xeon™ 3.2GHz machine with 4 GB RAM. In our implementations we use the Jonker and Volgenant’s (see [Jonker and Volgenant (1987)]) code for solving the AP.

Out of the four algorithms, only the W-GREEDY algorithm is known in the literature (see the GR algorithm in [Glover et al. (2001)] and [Gutin et al. (2002b)]). Therefore, we report our computational results using W-GREEDY as a base. Assume that for a particular TSP instance, W-GREEDY finds a tour of length \( L_W \) and in \( T_W \) time, while another al-
algorithm $A$ takes execution time $T_A$, and finds a tour of length $L_A$. Then for that instance we define the solution quality parameter $q_A$ and time parameter $\tau_A$ for $A$ as

$$q_A = \frac{L_A - L^*}{L_W - L^*} \times 100 \quad \tau_A = \frac{T_A \times 100}{T_W}$$

Clearly, the smaller the values of $q_A$ and $\tau_A$ the better the algorithm. We tested the algorithms on nine classes of instances. Classes 1 through 7 were taken from [Glover et al. (2001)], Class 8 is the class of GYZ instances introduced in [Gutin et al. (2002b)] for which the domination number of the W-GREEDY algorithm for the ATSP is 1 (see Theorem 2.1 in [Gutin et al. (2002b)]) and Class 9 is the amalgamation of several classes of instances from [Johnson et al. (2002)]. The nine classes are described below.

**Class 1:** All asymmetric instances from TSPLIB [Reinelt (1991)] (26 instances).

**Class 2:** All symmetric instances from TSPLIB [Reinelt (1991)] with less than 3000 vertices (99 instances).

**Class 3:** Asymmetric instances with $c(i,j)$ randomly and uniformly chosen from $\{0, 1, \cdots, 100000\}$ for $i \neq j$. 10 instances are generated for dimensions 100, 200, $\cdots$, 1000 and three instances for dimensions 1100, 1200, $\cdots$, 3000 (160 instances).

**Class 4:** Asymmetric instances with $c(i,j)$ randomly and uniformly chosen from $\{0, 1, \cdots, i \cdot j\}$ for $i \neq j$. 10 instances are generated for dimensions 100, 200, $\cdots$, 1000 and three instances for dimensions 1100, 1200, $\cdots$, 3000 (160 instances).

**Class 5:** Symmetric instances with $c(i,j)$ randomly and uniformly chosen from $\{0, 1, \cdots, 100000\}$ for $i < j$. 10 instances are generated for dimensions 100, 200, $\cdots$, 1000 and three instances for dimensions 1100, 1200, $\cdots$, 3000 (160 instances).

**Class 6:** Symmetric instances with $c(i,j)$ randomly and uniformly chosen from $\{0, 1, \cdots, i \cdot j\}$ for $i < j$. 10 instances are generated for dimensions 100, 200, $\cdots$, 1000 and three instances for dimensions 1100, 1200, $\cdots$, 3000 (160 instances).

**Class 7:** Sloped plane instances with given $x_i, x_j, y_i, y_j$ randomly and uniformly chosen from $\{0, 1, \cdots, i \cdot j\}$ for $i \neq j$ and $c(i,j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} - \max\{0, y_i - y_j\} + 2 \cdot \max\{0, y_j - y_i\}$ for $i \neq j$. 10 instances are generated for dimensions 100, 200, $\cdots$, 1000 and three instances for dimensions 1100, 1200, $\cdots$, 3000 (160 instances).
Class 8: GYZ instances (see Theorem 2.1 in [Gutin et al. (2002b)]) in which the arc weights $c(i, j)$ are defined as

$$c(i, j) = \begin{cases} n^3 & \text{for } i = n, j = 1; \\ in & \text{for } j = i + 1, i = 1, 2, \ldots, n - 1; \\ n^2 - 1 & \text{for } i = 3, 4, \ldots, n - 1; j = 1; \\ n \min\{i, j\} + 1 & \text{otherwise.} \end{cases}$$

One instance is generated for each $n = 5, 10, \ldots, 1000$ (200 instances).

Class 9: There are 12 problem generators from Johnson et al. [Johnson et al. (2002)], called \text{tmat}, \text{amat}, \text{shop}, \text{disc}, \text{super}, \text{crane}, \text{coin}, \text{stilt}, \text{rtilt}, \text{rect}, \text{smat}, \text{and} \text{tsmat}. Each of these generators yields 24 instances, 10 of dimensions 100, 10 of dimension 316, three of dimension 1000, and one of dimension 3162 (288 instances).

Note that for Classes 1 and 2 we use as $L^*$ the known optima (see [Reinelt (1991)]), for the symmetric and almost-symmetric Classes 3, 4, 7, and 8 the AP lower bound and for the asymmetric Classes 5, 6, and 9 the HK (Held-Karp) lower bound ([Held and Karp (1970)]).

It is clear from Table 5.1 that the usual weight-based greedy algorithm is comprehensively outperformed by tolerance-based greedy algorithms in terms of solution quality, although it takes much less execution time than two of the tolerance-based algorithms. It is also clear that A-RC-GREEDY, and to a lesser extent, A-R-GREEDY are greedy algorithms of choice if one desires good-quality solutions. Even the extremely simplistic R-R-GREEDY algorithm generates better quality solutions for all except two classes (Classes 1 and 2) in nearly the same time. This fact is seen most starkly in Classes 4 and 7.

An interesting observation is that AP relaxation based algorithms require very long execution times on average for instances in Classes 7 and 9. For instances in these classes, experiments show that the optimal solutions to the AP relaxation for the digraphs in several iterations have many cycles of length 2, and the arc to be contracted usually comes from one of these cycles. Consequently, in the next step of the algorithm, the AP relaxation needs to be solved again, and the tolerances recalculated, thus leading to long execution times (refer to the discussion on book-keeping techniques in Section 5.3).
### Table 5.1. Performance of tolerance-based algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Problem Class</th>
<th>q values</th>
<th></th>
<th>r values</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>mean</td>
<td>std. dev.</td>
<td>mean</td>
<td>std. dev.</td>
</tr>
<tr>
<td>R-R-GREEDY</td>
<td>1</td>
<td>92.02</td>
<td>7.70</td>
<td>101.24</td>
<td>32.19</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>103.80</td>
<td>14.99</td>
<td>63.76</td>
<td>48.59</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>48.33</td>
<td>4.92</td>
<td>32.51</td>
<td>22.70</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>9.30</td>
<td>2.28</td>
<td>34.07</td>
<td>20.85</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>53.65</td>
<td>4.39</td>
<td>35.28</td>
<td>23.56</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>12.08</td>
<td>2.74</td>
<td>36.66</td>
<td>21.21</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>9.25</td>
<td>3.32</td>
<td>43.25</td>
<td>24.45</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>33.23</td>
<td>0.01</td>
<td>153.68</td>
<td>41.46</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>80.61</td>
<td>24.58</td>
<td>76.09</td>
<td>27.06</td>
</tr>
<tr>
<td>A-R-GREEDY</td>
<td>1</td>
<td>87.56</td>
<td>7.07</td>
<td>100.24</td>
<td>26.61</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>91.16</td>
<td>12.15</td>
<td>275.99</td>
<td>152.25</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>31.02</td>
<td>5.11</td>
<td>37.68</td>
<td>23.06</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>7.75</td>
<td>1.85</td>
<td>41.13</td>
<td>20.98</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>37.17</td>
<td>4.86</td>
<td>609.72</td>
<td>234.95</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>10.52</td>
<td>2.43</td>
<td>1189.16</td>
<td>502.16</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>6.31</td>
<td>3.30</td>
<td>3150.58</td>
<td>1259.05</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>16.75</td>
<td>0.01</td>
<td>149.70</td>
<td>39.96</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>69.17</td>
<td>26.07</td>
<td>373.18</td>
<td>543.07</td>
</tr>
<tr>
<td>A-RC-GREEDY</td>
<td>1</td>
<td>84.48</td>
<td>9.21</td>
<td>120.21</td>
<td>32.02</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>89.67</td>
<td>12.91</td>
<td>292.76</td>
<td>159.29</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>27.55</td>
<td>4.45</td>
<td>52.16</td>
<td>22.25</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>7.92</td>
<td>1.88</td>
<td>56.90</td>
<td>19.88</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>33.44</td>
<td>4.53</td>
<td>497.98</td>
<td>178.01</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>11.38</td>
<td>2.62</td>
<td>746.48</td>
<td>295.36</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>6.19</td>
<td>3.19</td>
<td>3396.10</td>
<td>1289.19</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>16.75</td>
<td>0.01</td>
<td>174.78</td>
<td>69.03</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>67.39</td>
<td>26.84</td>
<td>391.89</td>
<td>565.70</td>
</tr>
</tbody>
</table>

### 5.5. Summary and Future Research Directions

In this paper, we examine in detail the idea of using arc tolerances instead of arc weights as a basis for making algorithmic decisions on whether or not to include an arc in an optimal solution. Such methods have only been studied in passing in the literature (see [Helsgaun (2000)]) and deserve more attention. In order to evaluate the usefulness of the concept, three tolerance-based greedy algorithms are proposed (see Section 5.3) for the traveling salesman problem. Two of these (A-R-GREEDY and A-RC-GREEDY) are based on an AP relaxation of the original problem, while the third one (R-R-GREEDY) is based on a new relaxation of the AP relax-
With the purpose of investigating the usefulness of the relaxed AP (RAP), we made extensive computational experiments with our R-R-GREEDY heuristic applied to the AP (not reported here in detail due to the space limitation). The computational results for the TSP show that the R-R-GREEDY outperforms a weight-based greedy (W-GREEDY) in quality at least 10 times on average, while for AP the corresponding domination numbers for R-R-GREEDY and W-GREEDY are $2^{n-1}$ and 1, respectively.

Our experiments show that the quality of solutions produced by tolerance-based greedy algorithms are overall significantly better than those found by the arc weight-based greedy algorithm. Unfortunately, A-R-GREEDY and A-RC-GREEDY are often slower than W-GREEDY, but R-R-GREEDY, being superior to W-GREEDY in quality, is nearly as fast as W-GREEDY. Overall, the simplest tolerance-based greedy, R-R-GREEDY, is the best algorithm for solving the STSP, while the A-RC-GREEDY algorithm could be suggested for the ATSP.

It is worth mentioning that the construction heuristics in [Glover et al. (2001)] (see from Table 1 in [Glover et al. (2001)]) have the following average excesses (taken over seven families of instances) over the length of an optimal tour or lower bound: GR= 580.35%, RI= 710.95%, KSP= 135.08%, GKS= 98.09%, RPC= 102.02%, COP= 23.01%. Computational experiments reported in [Goldengorin and Jäger (2005)] for our algorithms give R-R-GREEDY= 67.14%, A-R-GREEDY=34.75%, and A-RC-GREEDY= 29.19%. We know that the domination number of ATSP-R-R-GREEDY is $2(n−3)!$ and it would be interesting to find a non-trivial lower bound.

Another question is whether a 1-tree-based relaxation of the traveling salesman problem would generate tolerance-based greedy algorithms that are better for the STSP. Also it would be interesting to replicate the success of tolerance-based algorithms on the TSP to other combinatorial optimization problems.

References


