Computers & Operations Research 39 (2012) 291–298

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

Lower tolerance-based Branch and Bound algorithms for the ATSP

Remco Germs [a], Boris Goldengorin [a,c], Marcel Turkensteen [b,*]

[a] Faculty of Economics and Business, University of Groningen, P.O. Box 800, 9700 AV Groningen, The Netherlands
[b] CORAL, Department of Business Studies, Aarhus School of Business and Social Sciences, Aarhus University, Fuglesangs Alle 4, 8210 Aarhus V, Denmark
[c] Department of Applied Mathematics and Informatics, The Higher School of Economics, B. Pecherskaya, 25/12, Nizhny Novgorod 603155, Russia

ARTICLE INFO

Available online 8 April 2011

Keywords:
Branch and Bound
Asymmetric traveling salesman problem
Tolerances

ABSTRACT

In this paper, we develop a new tolerance-based Branch and Bound algorithm for solving NP-hard problems. In particular, we consider the asymmetric traveling salesman problem (ATSP), an NP-hard problem with large practical relevance. The main algorithmic contribution is our lower bounding strategy that uses the expected costs of including arcs in the solution to the assignment problem relaxation of the ATSP, the so-called lower tolerance values. The computation of the lower bound requires the calculation of a large set of lower tolerances. We apply and adapt a finding from [23] that makes it possible to compute all lower tolerance values efficiently. Computational results show that our Branch and Bound algorithm exhibits very good performance in comparison with state-of-the-art algorithms, in particular for difficult clustered ATSP instances.

1. Introduction

A combinatorial optimization problem (COP) is a problem with a ground set of elements, from which a certain combination has to be selected. COPs are common in routing, location and network applications. For some COPs, algorithms exist that are able to find optimal solutions in polynomial time. For the class of *NP-hard problems*, on the other hand, the quality of a solution can be determined in polynomial time, but it may be necessary to search through an exponential number of solutions [10].

A basic approach to solving NP-hard COPs to optimality is to construct a new, polynomially solvable problem, called the *relaxation*. Two ways to form a relaxation are (1) the formulation of the LP relaxation of the problem, where the relaxation can be improved by adding the so-called cuts [19]; (2) the releasing of constraints defining structural properties of the general NP-hard problem in order to obtain a polynomially solvable COP.

One of the solution approaches that uses relaxation solutions to the original problem is Branch and Bound (BnB). BnB methods start by solving a relaxation of the original problem. If the relaxation solution is feasible to the original problem, this solution is optimal. Otherwise, there exists a structure in the relaxation solution that is infeasible to the original problem, which we call an *infeasibility*. In order to remove the infeasibility, the solution space is divided into smaller problems, the *subproblems*, which are likely to contain fewer infeasibilities. This process continues until the original problem is solved, i.e., each subproblem is either solved or fathomed.

The asymmetric traveling salesman problem (ATSP) is a well-known NP-hard COP. It has numerous applications to different problems in logistics and management (see e.g. [1]). A feasible solution is a combination of connections, or *arcs*, that form a tour through n locations. Each location must be visited exactly once. A commonly used relaxation of the ATSP is the *assignment problem* (*AP*). This is the problem of finding a minimum cost assignment of n workers to n jobs, where every worker has to be assigned to exactly one job. The AP can be represented in a graph, where the vertices denote the workers and the distance of each connection ij denotes the cost of assigning worker i to job j. In the graph, the connections belonging to a feasible AP solution may form a single tour, which is a feasible ATSP solution, but more likely, they will form a set of disjoint subcycles. Thus, the ATSP property that a single tour is obtained, is relaxed. Since the AP can be solved in $O(n^3)$ time, it is a polynomially solvable relaxation of the ATSP. Note that if an optimal AP solution is feasible for the corresponding ATSP instance, i.e., with the same set of arcs and the same cost matrix C, it is also an optimal ATSP solution.

BnB is a commonly used and effective class of solution methods for solving the ATSP to optimality; see [9]. The BnB methods CDT [5] and FTV [9] use the AP as a relaxation. The *Concorde* [3] and FT b&c [9] algorithms are based on the linear programming (LP) relaxation and belong to the BnB variant called branch-and-cut (BnC). It is found in [9] that LP-based algorithms perform rather

* Corresponding author.
E-mail addresses: r.germs@rug.nl (R. Germs), b.goldengorin@rug.nl, bgoldengorin@hse.ru (B. Goldengorin), matu@asb.dk (M. Turkensteen).
URLS: http://www.rug.nl/staff/r.germs (R. Germs), http://www.rug.nl/staff/b.goldengorin (B. Goldengorin), http://www.asb.dk/staff/bs/matu.aspx (M. Turkensteen).

0305-0548/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2011.04.003

well on instances that are relatively difficult, whereas BnB methods are fast for large instances. A recent effective algorithm that is not of type BnB is suggested by Climer and Zhang [8], namely the *iterated cut-and-solve* algorithm (abbreviated to CZ).

The BnB algorithms for the ATSP introduced in [11,22] use the *upper tolerance values* of arcs in the corresponding AP instance to determine which arcs should be excluded. The upper tolerance value of an arc is, roughly speaking, the change in the value of the optimal AP solution when an arc from the current solution is forbidden. Branching on upper tolerances is similar to strong branching from [2]. We call BnB algorithms based on upper tolerances *UBnB*. The UBnB algorithm from [22] performs very well in comparison with CDT.

In this paper, we enhance this approach by incorporating a new type of information into the BnB search process, namely *lower tolerances*. Lower tolerances correspond to the additional costs of a solution with a connecting arc [12]. They can be used to compute the costs of including arcs and therefore, to connect two subcycles.

Intuitively, connecting subcycles may appear to be unattractive. There are up to $n(n-2)$ arcs outside of an optimal AP solution, compared to only n candidate arcs inside it. An evaluation of the candidate inclusions of elements appears to take much longer than the evaluation of the candidate exclusions. In particular, the computation of each upper and lower tolerance value takes $O(n^2)$, and there are up to $n(n-2)$ arcs outside of an optimal AP solution, computing all lower tolerances appears to be costing $O(n^4)$ time.

However, it may be effective to compute the costs of connecting subcycles with lower tolerances. A recent finding in [23] reduces the computational complexity of all lower tolerances to $O(n^3)$ time, which is the same complexity as for the computation of the upper tolerances of all arcs in an optimal AP solution. Similar results are known for other polynomially solvable COPs, such as the minimum spanning tree problem [15] and the shortest path problem [20]. In Section 3, we improve the result from [23] by narrowing it to subsets of interesting candidate arcs for inclusion and their lower tolerances. It is likely that the inclusion of an arc decreases the infeasibility of an AP solution with respect to the ATSP more than the exclusion of an arc. A structural ATSP property that is violated by many AP solutions is namely *connectivity*. This property is more likely to be satisfied by a connecting than by a subcycle-breaking step.

We use the advantages of lower tolerances to strengthen the lower bounds in BnB. For a minimization problem such as the ATSP, a *lower bound of a subproblem* is a lower estimate of the value of a best solution to a subproblem. The higher the lower bound of subproblems is, the larger the part of the search tree that can be excluded from further search. Section 3 introduces a lower bound that uses the estimates of connecting subcycles with lower tolerances. Subsequently, we determine the complexity and the accuracy of the lower bounds, both analytically and experimentally. We prove that these lower bounds are at least as tight, but often tighter than the upper tolerance-based lower bounds from [22]. The BnB algorithms that apply lower tolerance-based lower bounds are called lower tolerance-based BnB (LBnB) algorithms.

In our computational experiments, we compare different types of lower bounds based on connecting and breaking subcycles in an optimal AP solution, including the lower bound from [6]. The options are compared on their solution times and their search tree sizes. Next, we compare the solution times of LBnB with UBnB and with the freely available codes CDT and Concorde.

The paper is organized as follows. In the next section, we introduce the necessary terminology and notation. The new lower tolerance-based lower bounds are derived in Section 3, and a brief overview of all compared algorithms is presented in Section 4. Computational experiments and conclusions follow in Sections 5 and 6, respectively.

2. Terminology and notation

In this section, we formulate upper and lower tolerances for graph theoretical minimization problems in general. We formulate the AP and ATSP as graph theoretic problems and derive properties of the AP upper and lower tolerances.

Let $G=(V,E,C)$ be a complete simple weighted digraph with the set of vertices $V=\{1,\ldots,n\}$, arc set E and nonnegative weights $C=[C(e)]$ for each $e\in E$. The solutions to the AP and ATSP can be presented as sets of *cycles*. A cycle in a graph is a connected set of arcs of the graph such that it is possible to move from vertex to vertex along the arcs in the cycle, encountering all vertices exactly once, and finishing at the initial vertex. If a cycle contains all vertices of the graph, it is called a Hamiltonian cycle; otherwise, we use the term *subcycle* or *subtour*.

In order to formally define tolerances, the ATSP and AP are considered within the framework of the following combinatorial minimization problem; see [11,13]. $(\mathcal{E},C,\mathcal{D},f_C)$ is the problem of finding

$$S^*\in \operatorname{argmin}\{f_C(S)|S\in\mathcal{D}\},$$

where $C:\mathcal{E}\to\Re$ is the given *instance* of the problem with a *ground set* $\mathcal{E}$ satisfying $|\mathcal{E}|=m(m\geq 1),\mathcal{D}\subseteq 2^{\mathcal{E}}$ is the *set of feasible solutions*, and $f_C:2^{\mathcal{E}}\to\Re$ is the *objective function* of the problem. By $\mathcal{D}^*=\operatorname{argmin}\{f_C(S)|S\in\mathcal{D}\}$ the set of optimal solutions is denoted. It is assumed that $\mathcal{D}^*\neq\emptyset$, and that $S\neq\emptyset$ for some $S\in\mathcal{D}$. Let $g\in\mathcal{E}$, and $\alpha\geq 0$. By $C_{\alpha,g}:\mathcal{E}\to\Re$ we denote the instance defined as $C_{\alpha,g}(e)=C(e)$ for each $e\in\mathcal{E}\backslash\{g\}$, and $C_{\alpha,g}(g)=C(g)+\alpha$. Take any $S^*\in\mathcal{D}^*$. The *upper tolerance*, $u_{S^*}(e)$, of e with respect to S^* is defined as

$$u_{S^*}(e)=\max\{\alpha\geq 0:S^*\in \operatorname{argmin}\{f_{C_{\alpha,e}}(S):S\in\mathcal{D}\}\},$$

and the *lower tolerance*, $l_{S^*}(e)$, with respect to S^* as

$$l_{S^*}(e)=\max\{\alpha\geq 0:S^*\in \operatorname{argmin}\{f_{C_{-\alpha,e}}(S):S\in\mathcal{D}\}\}.$$

So $u_{S^*}(e)$ is the maximal increase of $C(e)$ under which S^* stays optimal, and $l_{S^*}(e)$ is the maximal decrease of $C(e)$ under which S^* stays optimal, provided that the data remain unchanged.

We will use the following notation. Let $e\in\mathcal{E}$. Then $\mathcal{D}_+(e)=\{S\in\mathcal{D}:e\in S\}$, and $\mathcal{D}_-(e)=\{S\in\mathcal{D}:e\notin S\}$. Clearly, $\mathcal{D}=\mathcal{D}_-(e)\cup\mathcal{D}_+(e)$ and $\mathcal{D}_-(e)\cap\mathcal{D}_+(e)=\emptyset$ for all $e\in\mathcal{E}$. Similarly, $\mathcal{D}^*_+(e)$ and $\mathcal{D}^*_-(e)$ are the sets of optimal solutions containing e and not containing e, respectively.

We define the AP (resp., ATSP) on a complete simple weighted digraph $G=(V,E,C)$ with the set of vertices $V=\{1,\ldots,n\}$, arc set $E=\mathcal{E}$, weights $C=[c(e)]$ for each $e\in E$; $\mathcal{A}$ is the set feasible AP solutions (resp., $\mathcal{D}=\mathcal{H}$ is the set of all Hamilton cycles). It then holds that:

1. Each feasible AP solution $A\in\mathcal{A};A=\bigcup_{i=1}^{k}K_i$ contains $k\geq 1$ disjoint subcycles K_i;
2. Let $K(A)=\{K_1,\ldots,K_k\}$ be the set of subcycles in an AP solution A. The vertex set V is covered by the disjoint vertex sets $V(K_i)$ of each subcycle K_i, i.e., $V=\bigcup_{i=1}^{k}V(K_i)$ and $V(K_i)\cap V(K_j)=\emptyset$ for $i\neq j$.
3. $f_C(A)=\sum_{e\in A}c(e)$ for each $A\in\mathcal{A}$.
4. $f_C(H)=\sum_{e\in H}c(e)$ for each $H\in\mathcal{H}$.

Instances of the AP and the ATSP are called *corresponding* if they are defined on the same graph $G=(V,E,C)$. It then holds that $\mathcal{H}\subseteq\mathcal{A}$ and $f_C(A^*)\leq f_C(H^*)$.

In [11,12], it is shown that for the AP the finite upper and lower tolerance values are independent of the chosen optimal solution A^*; therefore, we write $u(e)$ and $l(e)$ instead of $u_{A^*}(e)$ and

$l_{A^*}(e)$, respectively. The upper and lower tolerance values can be computed as follows: for each $e \in A^*$, $u(e) = f_C(A^*_-(e)) - f_C(A^*)$, and for each $e \notin A^*$, $l(e) = f_C(A^*_+(e)) - f_C(A^*)$.

3. Lower tolerance-based lower bounds for the ATSP

In BnB, subproblems can be fathomed if their lower bound values are higher than the value of the current upper bound, usually the value of the current best solution. A tighter lower bound is likely to lead to the fathoming of a larger number of subproblems. In this section, we construct lower bounds for the ATSP based on lower tolerances. Firstly, we identify the key lower and upper tolerance values, called the *bottleneck tolerances*, and show that these values can be used to form a lower bound. The term bottleneck tolerances is used, as it refers to an arc with the minimum upper or lower tolerance value in a selected set of arcs. Next, we compare the time complexity and the accuracy of the upper and lower bottleneck tolerance-based lower bounds. We prove that, for the ATSP, the lower bounds based on bottleneck lower tolerances are tighter than their upper tolerance-based counterpart, and that the computational complexities of both types of lower bounds are the same.

An optimal AP solution A^* is a collection of disjoint subcycles, but every feasible solution H to the ATSP should be a single (Hamiltonian) cycle. Our lower bounds compute the additional costs of adding arcs between subcycles in order to obtain an ATSP solution, i.e., connecting subcycles.

The idea of connecting subcycles in an AP optimal solution with the purpose of improving the AP-based lower bound has been used by Christofides in [6], where the Christofides lower bound (CLB) gives an estimate by adding the reduced costs of the arcs between subcycles. The reduced cost matrix is formed by subtracting values from each row and column in such a way that the arcs in each optimal AP solution have zero costs. An iterative procedure solves a sequence of adjusted AP instances obtained after each iteration. The adjusted AP instance is, roughly speaking, created by contracting each disjoint subcycle into a new vertex until the adjusted AP solution consists of a Hamiltonian cycle. The computation of the CLB from an AP solution requires $0.143n^3$ computations on average (see [6], p. 272).

The CLB is based on inclusion of arcs that simultaneously connect all subcycles. The number of the arcs is exactly equal to the number of connected subcycles. Contrary to this, we present a lower bound based on the so-called bottleneck lower tolerance value of a single arc. This does not mean that we only consider the inclusion of a single arc: a transformation of one feasible AP solution into another one is done by changing at least two arcs.

Lower tolerance-based lower bounds are computed as follows. Suppose that the optimal solution A^* to the AP relaxation consists of $k(>1)$ subcycles, say, $A^* = \bigcup_{i=1}^{k} K_i$. Then A^* is infeasible to the ATSP and any Hamiltonian cycle must have exactly two arcs incident (one directed *inward* and one directed *outward*), with at least one node in a subcycle K_i. We call these incident arcs the *connecting arcs* of a subcycle. Fig. 1 illustrates graphically for an 8-city ATSP that any subcycle of an infeasible AP solution has exactly two connecting arcs.

To obtain a feasible ATSP solution from A^*, we have to connect all subcycles K_i. Connecting a subcycle K_i is the inclusion of any connecting arc $(p,q) \in E \backslash A^*$ into the optimal solution A^* that is incident with one of the nodes in a subcycle K_i. The additional cost of including an arc (p,q) is equal to its lower tolerance value $l(p,q) = f_C(A^*_+(p,q)) - f_C(A^*)$.

Take an arbitrary subcycle K_i. There are two possible connections: one that enters K_i, and one that leaves K_i. The sets formed by these arcs are denoted by $IN(K_i)$ and $OUT(K_i)$, respectively.

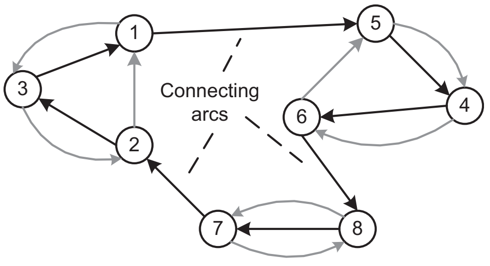

Fig. 1. Hamiltonian cycle (dark color), AP solution (light color) and connecting arcs for an 8 city ATSP.

Formally, for each subcycle $K_i \in K(A^*)$, define:

1. $OUT(K_i) = \{(p,q) : p \in V(K_i), q \in V \backslash V(K_i)\}$, which denotes the set of all connecting arcs directed out of subcycle K_i;
2. $IN(K_i) = \{(p,q) : p \in V \backslash V(K_i), q \in V(K_i)\}$, which denotes the set of all connecting arcs directed into subcycle K_i.

Then, for each $K_i \in K(A^*)$, the minimum cost of connecting subcycle K_i with an outgoing arc $(p,q) \in OUT(K_i)$ is denoted by and defined as

$$l_{out}(K_i) = \min\{l(p,q) : (p,q) \in OUT(K_i)\},$$

and the minimum cost of connecting subcycle K_i with an incoming arc $(p,q) \in IN(K_i)$ as

$$l_{in}(K_i) = \min\{l(p,q) : (p,q) \in IN(K_i)\}.$$

Thus, $l_{out}(K_i)$ and $l_{in}(K_i)$ are the minimum additional costs incurred when K_i is connected with an outgoing or incoming arc, respectively. For each $K_i \in K(A^*)$, we denote and define $l(K_i) = \max\{l_{out}(K_i), l_{in}(K_i)\}$.

It is shown in [23] that one can compute the lower tolerance values of all outgoing arcs from a single vertex $p \in V$ in $O(n^2)$ time, i.e., the lower tolerances of $(p,q) \in E$, with q in another subcycle than the one containing p. Therefore, all lower tolerances of arcs in $OUT(K_i)$ can be computed in $O(n^2 \times |K_i|)$ time, for each $K_i \in K(A^*)$. Unfortunately, the same does not hold for incoming arcs. In order to compute the lower tolerances of all arcs in $IN(K_i)$, we have to calculate the lower tolerance values of all rows of the matrix C, requiring a time complexity of $O(n^3)$. The following theorem shows that it suffices to compute $l_{out}(K_i)$.

Theorem 1. *Let A^* be an optimal AP solution, and let $K(A^*) = \bigcup_{i=1}^{k} K_i$ be such that $k > 1$. Then $l(K_i) = l_{out}(K_i) = l_{in}(K_i)$ for each subcycle $K_i \in K(A^*)$.*

Proof.

1. We prove that $l_{out}(K_i) \le l_{in}(K_i)$. Let $e \in IN(K_i)$ such that $l_{in}(K_i) = l(e)$. If $A_+(e) \cap OUT(K_i) = \emptyset$, there is at least one path going into $V(K_i)$, but no path is coming from $V(K_i)$. The path into $V(K_i)$ cannot form a subcycle; the resulting solution would be infeasible to the AP. Therefore, it must hold that $\exists g \in A^*_+(e) \cap OUT(K_i)$. Since $g \in A^*_+(e)$, it holds that $l_{out}(K_i) \le l(g) = f_C(A^*_+(e)) - f_C(A^*) = l(e) = l_{in}(K_i)$.
2. The proof of $l_{out}(K_i) \ge l_{in}(K_i)$ goes along similar lines. Let $g \in OUT(K_i)$ such that $l_{out}(K_i) = l(g)$. If $A_+(g) \cap IN(K_i) = \emptyset$, there is at least one path going out of $V(K_i)$, but none is coming in. Therefore, $\exists e \in A^*_+(g) \cap OUT(K_i)$. Since $e \in A^*_+(g)$, it holds that $l_{in}(K_i) \le l(e) = f_C(A^*_+(g)) - f_C(A^*) = l(g) = l_{out}(K_i)$. $\square$

Theorem 2 shows that a lower bound for the corresponding ATSP instance is guaranteed if, for any subcycle K_i, $l(K_i)$ is added to the AP value.

Theorem 2. *Let A^* and H^* be optimal solutions to the AP and ATSP instances, respectively, with the same cost matrix C. Assume that A^* consists of at least two subcycles. Then for each $K_i \in K(A^*)$, the following inequalities hold*:

$$f_C(A^*) \le f_C(A^*) + l(K_i) \le f_C(H^*).$$

Proof. The first inequality is obvious, since $l(p,q) \ge 0$ for any $(p,q) \in E$.

To prove the second inequality, we need to prove that, in a given subcycle K_i, we have to add at least one arc from $IN(K_i)$ and at least one from $OUT(K_i)$, so $\forall H \in \mathcal{H}, |H \cap OUT(K_i)| > 0$ and $|H \cap IN(K_i)| > 0$. We prove this by contradiction. Assume that $\exists H \in \mathcal{H}$ such that $|H \cap IN(K_i)| = 0$. This means that there is no path in H from any $p \in V \backslash V(K_i)$ to any $q \in V(K_i)$, thereby contradicting the assumption that H is a Hamiltonian cycle. Along similar lines, it can be shown that $|H \cap OUT(K_i)| > 0$.

To complete the proof, we use that, for any $e \in IN(K_i)$, it holds that $l(e) \ge l(K_i)$. Let $H^* \in \mathcal{H}^*$. For $e \in IN(K_i) \cap H^*$ (from the previous part, we know that e exists), it holds that $f_C(H^*) \ge f_C(A^*) + l(e) \ge f_C(A^*) + l(K_i)$. The proof for $OUT(K_i)$ goes along similar lines. □

The subcycle K_i is taken arbitrarily in Theorem 2, meaning that the value $l(K_i)$ of any subcycle $K_i \in K(A^*)$ can be added to $f_C(A^*)$ in order to obtain a lower bound for the corresponding ATSP solution value. The increase in the lower bound $f_C(A^*) + l(K_i)$ is the largest for the subcycle K_i for which $l(K_i)$ is maximal. However, all lower tolerance values in $\bigcup_{i=1}^{k} OUT(K_i)$ must be computed to guarantee that we obtain an arc with a maximum value of $l(K_i)$ (see Theorem 1). Another promising choice is a subcycle with the smallest cardinality $|K_{i_0}|$, so $K_{i_0} \in \operatorname{argmin}\{|K_i| : i = 1, \ldots, k\}$. Not only is the number of lower tolerance computations needed small, it is also likely that $l(K_{i_0})$ is relatively large compared to the value $l(K_i)$ of any other subcycle K_i. The intuitive explanation is that the expected minimum of a small set of random numbers is greater than the expected minimum of a large set of random numbers.

A bottleneck lower tolerance of a subcycle is the minimum lower tolerance value of all arcs entering or leaving a subcycle. We use the bottleneck tolerances l_a and l_e as defined below to compute the lower tolerance-based lower bounds. We introduce the following two lower tolerance-based lower bounds: the exact bottleneck lower tolerance (EBLT) lower bound denoted by and defined as $LB(EBLT) = f_C(A^*) + l_e$ with $l_e = \max\{l(K_i) : i = 1, \ldots, k\}$; and the approximate bottleneck lower tolerance (ABLT) lower bound denoted by and defined as $LB(ABLT) := f_C(A^*) + l_a$ with $l_a = l(K_{i_0})$.

For the calculation of the lower tolerance-based lower bounds, we have to compute $l(K_i)$ for prespecified subcycles K_i. This requires the computation of all lower tolerance values of the connecting arcs of subcycle K_i, i.e., the lower tolerances of all arcs in the sets $OUT(K_i)$ and $IN(K_i)$. Theorem 1 allows us to compute $l(K_i)$ for all $K_i \in A^*$, by using the lower tolerance values of set $OUT(K_i)$. The time complexity of this operation is $O(n^2 \times |K_i|)$; see [23]. Therefore, the ABLT lower bound can be computed by calculating all lower tolerance values in $OUT(K_{i_0})$, which costs $O(n^2 \times |K_{i_0}|)$ time, which is at most $O(0.5n^3)$. On average, we find that the complexity of the ABLT lower bound is lower than the $O(0.143n^3)$ average complexity of the CLB. For the calculation of the EBLT lower bound, we have to compute all lower tolerance values in $\bigcup_{i=1}^{k} OUT(K_i)$. The overall time complexity of this lower bound is, therefore, $O(n^3)$.

The exact bottleneck upper tolerance (EBUT) and approximate bottleneck upper tolerance (ABUT) lower bounds have been introduced in [11]. We summarize the procedure for computing these two lower bounds. For each subcycle K_i, let $u(K_i)$ denote the minimum upper tolerance value in the subcycle. We use the bottleneck tolerances u_a and u_e as defined below to compute the lower tolerance-based lower bounds. The ABUT lower bound is obtained by adding the minimum upper tolerance value in the shortest subcycle to the value of the AP lower bound; its value is $LB(ABUT) = f_C(A^*) + u_a$, where $u_a = u(K_{i_0})$ is the minimum upper tolerance value of the subcycle with the smallest cardinality, i.e., $K_{i_0} \in \operatorname{argmin}\{|K_i| : K_i \in A^*\}$. The ABUT lower bound can be computed by calculating the upper tolerances of all arcs in a subcycle with the smallest cardinality, which takes $O(n^2 \times |K_{i_0}|)$ time. The EBUT lower bound is defined as $LB(EBUT) = f_C(A^*) + u_e$, where $u_e = \max\{u(K_i) : i = 1, \ldots, k\}$. This means that we compute the minimum upper tolerance values in all subcycles and take the largest of these, which takes $O(n^3)$ time.

Theorem 3 shows that the EBLT and ABLT lower bounds are tighter than their counterparts the EBUT and ABUT lower bounds, respectively.

Theorem 3. *The following assertions hold*:

1. $LB(EBLT) \ge LB(EBUT)$;
2. $LB(ABLT) \ge LB(ABUT)$.

Proof. Take any $K_i \in K(A^*)$. Let $e = (v,w) \in OUT(K_i)$ be such that $f_C(A^*_+(e)) = f_C(A^*) + l(K_i), v \in V(K_i), w \notin V(K_i)$. Then there exists $g = (v,z) \in K_i$ such that $g \notin A^*_+(e)$. So we have that $f_C(A^*_-(g)) \le f_C(A^*_+(e))$, which implies that $u(g) = f_C(A^*_-(g)) - f_C(A^*) \le l(e) = f_C(A^*_+(e)) - f_C(A^*)$ or $u(K_i) \le l(K_i)$.

Since K_i is chosen arbitrarily, it holds that:

1. $LB(ABLT) = f_C(A^*) + l(K_{i0}) \ge f_C(A^*) + u(K_{i0}) = LB(ABUT)$;
2. $LB(EBLT) = f_C(A^*) + \max[l(K_i : K_i \in K(A^*)] \ge f_C(A^*) + \max[u(K_i) : K_i \in K(A^*)] = LB(EBUT)$. □

The lower tolerance-based lower bounds are tighter than their upper tolerance-based counterparts when inequalities hold in Theorem 3. Such a situation can occur if the distances between subcycles are relatively large compared to the distances between nodes in the same subcycle. The cost of breaking a subcycle is then relatively low compared to the cost of connecting two subcycles. Fig. 2 shows a simple example of such an instance. The AP solution consists of the zero cost arcs, and the AP solution value is 0. The upper tolerance values of all arcs in both subcycles are 3, because the removal of an arc in the AP solution leads to a reversion of the orientation in its subcycle. Clearly, the values of both $LB(ABUT)$ and $LB(EBUT)$ are $f_C(A^*) + l(K_1) = 0 + 3 = 3$ (subcycle K_2 could have been selected as well here). In order to determine EBLT and ABLT, it is necessary to compute the lower tolerance values leading out of both subcycles. In this example, there are only two candidate arcs, namely the black arcs between the subcycles. The lower tolerance values of both arcs are 6, and so are the values of $LB(ABLT)$ and $LB(EBLT)$ here. This is clearly higher than the values of their upper tolerance-based counterparts.

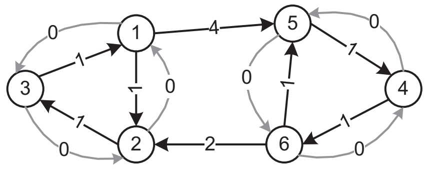

Fig. 2. AP solution (light color) in which lower and upper tolerance-based bounds differ.

4. Classification of the algorithms

In this section, we describe the algorithms to be compared in the computational experiments. We mainly consider BnB algorithms. According to [18], each BnB algorithm can be characterized as a combination of the following four building blocks:

- The *search strategy* determines the order in which the open subproblems are solved, i.e., the subproblems that have been created but have not yet been solved are discarded.
- The *branching rule* determines how the current ATSP should be divided into subproblems.
- The *lower bound* is discussed in Section 1 and 3.
- The *upper bound* is usually the value of the best ATSP tour found so far; in all our algorithms we use Karp–Steele patching at every node of the search tree [17].

The tolerance-based BnB algorithms apply the search strategy depth first search (DFS), in which the most recently generated subproblem is solved first. DFS is commonly applied in practice, because it is relatively easy to implement and it generally obtains a good solution quickly. When a DFS algorithm is terminated prematurely, it is likely that the best solution found until then is of high quality; see [24]. A competing strategy is best first search (BFS), in which the most promising subproblem is solved next. Algorithms with BFS are generally faster than those with DFS, but more difficult to implement.

The *branching rule* determines, firstly, how an unsolved and unfathomed subproblem is divided into subproblems and, secondly, it fixes the order in which these new subproblems are going to be solved in the further BnB process. In case of the ATSP, subproblems are usually defined by the subsets of arcs from the current solution that are forbidden (removed) in the subproblem. The branching rule used in the experiments is based on the "New Branching Scheme" from [4]. It excludes the arcs in a shortest subcycle from the AP solution (i.e., with the smallest number of arcs) one by one.

In particular in DFS, the ordering of the next subproblem matters. The "normal" rule branches on these arcs in a non-increasing order of arc costs. The ABUT branching rule branches on the arcs in a non-decreasing order of arc upper tolerances. It is found in [22] that it makes sense to branch on tolerances in a shortest subcycle. The authors call the algorithm *BnB*(*SCS*), because it determines the bottleneck upper tolerance value from the smallest cycle set (SCS), i.e., from an arbitrary subcycle with the smallest cardinality. In this paper, we choose to denote it by UBnB, to distinguish it from the lower tolerance-based algorithm that uses a smallest cycle set in its lower bounding process. Note that if there are multiple shortest subcycles, we choose a shortest one at random.

In the bounding part, our contribution is the addition of two new lower bounds from the previous section, namely the ABLT and EBLT bounds. These are compared to the following lower bounds: the ABUT and EBUT bounds from [11], the Christofides bound from [6] (CLB) and the AP solution value (APLB). The quality of the lower bounds is measured through the number of nodes that can be fathomed in a search tree, the *search tree reduction*.

Since we only vary the lower bound and the branching rule in our tested algorithms, we characterize our algorithms with *BnB*(*BR*, *LB*), where the branching rules *BR* are ABUT and Normal. We found that it is ineffective to use AP lower tolerances for branching. The candidate lower bounds are ABLT, EBLT, ABUT, EBUT, CLB and APLB. We also include the state-of-the-art methods CDT [5] and Concorde [3] in our experiments. CDT is an AP-based BnB algorithm introduced in [5] which solves in particular random instances rapidly. The Concorde algorithm from [3] uses the LP relaxation. It is designed for the symmetric TSP (STSP) and uses a transformation to formulate the ATSP instance as an STSP instance. Its performance is comparable, though a little slower, to that of the LP-based algorithm FT b&c from [9] and the CZ algorithm from [8].

The codes of FT b&c and CZ were not available to us; therefore, we choose to do the comparison with these algorithms indirectly, based on the CPU times and the solution accuracies reported in [9,8]. The search strategy is BFS and branching rules are in the case of Concorde and FT b&c based on LP relaxations of the ATSP. For this reason, we do not use the *BnB*(*BR*,*LB*) notation, but instead denote these algorithms by their names.

5. Computational experiments with ATSP instances

In this section, we conduct the following computational experiments. Firstly, we determine the search tree reductions obtained with different lower bounds. We investigate whether the use of lower tolerances leads to better lower bounds and then to solution time reductions. After that, we include the algorithms CDT and Concorde in the comparison in Tables 4 and 5.

Our test instances include 10 instances from the ATSPLIB (see [21]) that are solvable within reasonable time limits. Our randomly generated instances ("Random") are of size 500 and 1000 and have intercity distances that are drawn from a uniform distribution supported on {1,…,10 000}.

In the comparison with CDT and Concorde, we have selected computationally difficult instances from Fischetti et al. [9]. Some of these instances are so complex that even state-of-the-art algorithms are only able to find approximate solutions within a prespecified time limit of 10,000 s. The quality of the best solution found until then is used to measure the performance of the algorithms. The experiments are conducted on a Pentium 4 computer with 512 MB RAM memory and 1.66 GHz speed. The algorithms use the AP solver from [16].

First of all, we compare the quality of the different AP-based lower bounds in a BnB algorithm. To this end, we fix the branching rule to Normal and we vary the lower bound. CDT and Concorde are not taken into account here, since they apply a BFS strategy; CDT uses the APLB, whereas the Concorde lower bounds are based on the LP relaxation. Table 1 shows that the search trees obtained with the tolerance-based lower bounds are usually smaller than those obtained with Christofides' lower bound. The ABLT and EBLT lower bounds enable large reductions for the instances `ft53` and `ft70` in particular.

Next, we apply the upper tolerance-based *ABUT* branching rule to the BnB algorithms with the tolerance-based lower bounds *EBLT*, *ABLT* and *EBUT*. When upper or lower tolerance values are

Table 1
Search tree sizes of BnB algorithms with the "normal" branching rule and various lower bounds, for ATSPLIB instances.

Instance	n	Lower bound					
		APLB	CLB	EBUT	EBLT	ABUT	ABLT
ft53	53	20111	2877	7043	1435	7365	1927
ft70	70	25369	2367	5544	401	6582	1644
ftv33	34	6889	3486	2195	1938	3097	3018
ftv35	36	6888	3620	2567	2447	3241	2963
ftv38	39	6195	3195	2247	2089	2927	2585
ftv44	45	619	387	187	181	223	223
ftv47	48	28995	16063	8305	6735	9529	7981
ftv55	56	92445	49171	12279	7515	24841	12123
ftv64	65	43343	17401	7415	2975	9495	6007
ftv70	71	252527	108379	26595	15227	47869	39601

computed for the branching rule, they are available for a lower bound as well without any further computations. In [22], it is reported that branching on upper tolerances generally leads to smaller search trees than normal, cost-based branching strategies. Moreover, the effect of jointly using tolerance-based branching rules and lower bounds is larger than the individual effects on the search trees. The individual reductions amplify each other. This is called the *synergy effect*. The *expected reductions* are the reductions that can be expected when separate reductions do not affect each other; i.e., if the remaining search trees after the individual reductions are 70% and 60% of the original search tree, we expect a total search tree of 42% ($0.7 \times 0.6 \times 100\%$) of the original one, so that the expected reduction is 58%.

Table 2 indicates that the synergy effect also manifests itself when lower tolerance-based bounds and the ABUT branching rule are used in conjunction. For example, for ATSPLIB instances, the combination of the EBLT lower bound and the ABUT branching rule leads to an average search tree reduction of 95.53%, which means that the remaining search tree is only 4.47% of that of *BnB*(*Normal*, *APLB*), although the expected reduction is only 93.24%. For the Christofides lower bound, the ABUT branching rule leads only to small additional search tree reductions for the ATSPLIB instances and for the randomly generated instances, the actual reductions are even smaller than the expected ones. Since separate computations are required for the Christofides lower bound and for the ABUT branching, we will apply the Christofides lower bound in combination with the “normal” branching rule.

Table 3 gives a summary of the solution times of our candidate algorithms for ATSPLIB instances. The algorithm *BnB*(*ABUT*,*ABLT*) obtains the shortest average and median solution times. The ABLT lower bound reduces the search tree greatly for some instances, whereas the additional computation time, compared to ABUT, is short. Although the EBLT lower bound leads to the largest search tree reductions, the tolerance computations take so much time that *BnB*(*ABUT*,*EBLT*) is clearly slower than its competitors.

In the comparison with the CDT and Concorde algorithms, we consider the algorithms *BnB*(*ABUT*, *ABUT*) and *BnB*(*ABUT*, *ABLT*). From now on, we refer to these algorithms as UBnB and LBnB, respectively. The experiments are performed on the same instances as those in [9], introduced in [7]. There are usually five instances of size 100 and one instance of size 316 of each instance type. There are in total 15 clustered `cls` instances, of size 15, 20 and 25, and there are 10 `ran` (random) instances of size 500 and 1000. The results are summarized in Table 4; the average solution time is taken for instances that have been solved within our time limit of 1000 s CPU time, followed by the total number of instances of this type. The number of solved instances is given in between brackets.

Concorde solves the largest number of instances, but has remarkable difficulties with the random instances. CDT and to a lesser degree LBnB and UBnB solve these random instances rapidly. The reverse holds for more difficult instances. For the clustered instances, our LBnB approach achieves the best results. The difficulty with clustered instances is that there are many AP solutions that have multiple subcycles in each cluster of points. In the search process, the lower tolerance-based lower bound includes the additional costs of adding an arc to a vertex in a different cluster. Through the lower tolerance-based lower bound, the LBnB algorithm can fathom many of these solutions.

We consider 22 computationally difficult instances in more detail, of which the bottom seven are from the ATSPLIB. The solution times of the tested algorithms are reported, and, if no optimal solution has been determined after 1000 s, the gap with the best known solution as well. The results in Table 5 show that the tolerance-based algorithms require intermediate CPU times: usually, either CDT or Concorde is faster, but not both simultaneously. Take for example the instance `disk316.10`. the solution times are 13.31 (LBnB), 11.89 (UBnB), 0.94 (CDT), and 37.54 (Concorde) seconds, respectively. For randomly generated instances,

Table 2
Search tree reductions for ATSPLIB and random instances: the synergy effect.

Lower bound	Instance	Actual (%)	Expected (%)
EBLT	ATSPLIB	95.53	93.24
	Random	98.25	94.52
ABLT	ATSPLIB	92.17	87.10
	Random	98.13	94.22
CLB	ATSPLIB	73.76	65.81
	Random	70.43	76.35

Table 3
Solution times in seconds for ATSPLIB instances with different lower bounds and branching rules.

Instance	*n*	*BnB*(*Normal*,.)		*BnB*(*ABUT*,.)		
		APLB	CLB	EBLT	ABUT	ABLT
ft53	53	5.20	1.22	6.33	7.43	1.56
ft70	70	7.89	1.51	1.84	4.65	2.54
ftv33	34	0.45	0.47	0.05	0.01	0.02
ftv35	36	0.46	0.59	1.35	0.74	0.63
ftv38	39	0.51	0.55	1.83	1.05	0.96
ftv44	45	0.06	0.09	0.29	0.08	0.09
ftv47	48	3.11	3.87	5.50	2.11	2.07
ftv55	56	12.01	14.84	21.74	16.35	10.59
ftv64	65	7.51	7.06	16.95	10.12	6.94
ftv70	71	54.07	52.71	26.53	9.43	8.81
Average		9.13	8.29	8.24	5.20	3.42
Median		4.16	1.37	3.67	3.38	1.82

Table 4
Average solution times in seconds for Johnson instances, with number of solved instances between brackets.

Instance type	Nr. of instances	LBnB		UBnB		CDT		Concorde	
		Time	#Solved	Time	#Solved	Time	#Solved	Time	#Solved
ATSPLIB	27	63.90	(25)	77.06	(25)	4.03	(22)	50.38	(27)
coin	6	–	(0)	–	(0)	–	(0)	75.20	(5)
crane	6	664.09	(2)	–	(0)	–	(0)	106.86	(5)
disk	6	11.36	(5)	22.83	(5)	0.32	(5)	14.94	(6)
stilt	6	–	(0)	–	(0)	–	(0)	171.05	(6)
shop	6	15.19	(6)	8.30	(6)	0.36	(6)	26.19	(6)
stilt	6	–	(0)	–	(0)	–	(0)	335.70	(6)
super	6	0.06	(5)	0.04	(5)	0.01	(5)	9.91	(6)
cls	15	20.66	(15)	25.51	(12)	85.88	(11)	5.10	(11)
ran	9	40.81	(9)	28.50	(9)	0.37	(9)	177.97	(8)

Table 5
Comparison of LBnB, UBnB, CDT and Concorde. Time limit of 1000 s.

Name	LBnB			UBnB			CDT			Concorde		
	%Gap	Nodes	Time	%Gap	Nodes	Time	%Gap	Nodes	Time	%Gap	Nodes	Time
coin100.2	5.28	1117990	1000.00	5.65	1611864	1000.00	18.40	233061	1000.00	–	9	137.34
coin316.10	12.73	61163	1000.00	15.57	86836	1000.00	19.14	[a]	1000.00	1.36	[a]	1000.00
crane100.2	1.31	765221	1000.00	2.89	846717	1000.00	8.41	177779	1000.00	–	29	322.29
crane100.4	0.00	533459	1000.00	6.63	608345	1000.00	13.79	175989	1000.00	–	29	200.60
crane316.10	2.51	30820	1000.00	4.60	66333	1000.00	8.14	198395	1000.00	0.01	[a]	1000.00
disk316.10	–	760	13.31	–	917	11.89	–	3397	0.94	–	13	37.54
rtilt100.0	7.71	629364	1000.00	8.98	519722	1000.00	10.62	312583	1000.00	–	43	330.57
rtilt316.10	9.95	36674	1000.00	14.02	23502	1000.00	19.11	[a]	1000.00	–	61	563.71
shop316.10	–	1630	79.35	–	1230	39.69	–	1228	1.32	–	19	75.19
stilt100.4	5.41	555138	1000.00	10.95	394200	1000.00	22.05	206205	1000.00	0.13	[a]	1000.00
stilt316.10	6.88	39301	1000.00	9.01	49062	1000.00	24.21	[a]	1000.00	13.91	[a]	1000.00
super316.10	0.00	94479	1000.00	0.00	165671	1000.00	3.37	192423	1000.00	–	7	32.79
ran500.1	–	32	9.19	–	28	6.08	–	30	0.10	–	73	151.23
ran1000.0	–	61	98.46	–	48	56.62	–	44	0.98	0.19	[a]	1000.00
ran1000.2	–	56	47.15	–	46	28.57	–	158	0.80	–	177	946.63
ft53	–	1490	0.44	–	18354	6.45	15.28	165449	1000.00	–	1	0.27
ftv64	–	2676	1.56	–	14477	11.34	–	4552	0.30	–	5	30.21
ftv70	–	8934	4.89	–	15519	8.61	–	10328	1.23	–	5	8.43
kro124p	2.12	1300525	1000.00	2.12	1610092	1000.00	10.95	175212	1000.00	–	1	5.61
rbg358	–	1	0.03	–	1	0.02	–	1	0.04	–	19	847.10
rbg443	–	1	0.05	–	2	0.05	–	1	0.06	–	3	81.62
ry48p	–	373151	77.35	–	552078	102.09	0.00	201207	1000.00	–	3	11.39

[a] Could not be computed.

Table 6
Best solution for unsolved instances.

Instance	Smallest gap (%)	Obtained with
coin316.10	1.36	Concorde
crane316.10	0.01	Concorde
stilt100.4	0.13	Concorde
stilt316.10	6.88	LBnB

Concorde requires the largest computational times and it is not able to find an optimal solution for `ran1000.0`. Note that CDT solves them extremely fast while LBnB and UBnB are able to find an optimal solution much faster than Concorde. There is one instance for which our BnB algorithms outperform the state-of-the-art, namely the instance `stilt316.10`. UBnB and LBnB return solutions with gaps 6.88% and 9.01% within the time limit of 1000 s, while Concorde and CDT terminate with error margins of 13.91% and 24.21%, respectively (Table 6).

There are four instances for which neither of the methods obtains an optimal solution. Concorde obtains the smallest gap for three instances, but for the instance `stilt316.10`, LBnB terminates with the smallest gap.

The computational experiments do not contain the effective algorithms, CZ and FT b&c, because their codes were not available to us. In Table 1 from [8], it is shown that the solution times of CZ follow a similar pattern to those of Concorde, but are usually somewhat faster. As a consequence, CZ is faster than our LBnB and UBnB algorithms for many of the instance types. However, CZ performs worse than Concorde for super-instances, for which our LBnB and UBnB algorithms perform relatively well, with e.g. a close to optimal solution of `super316.10`.

Our LBnB approach combines the connecting and breaking of subcycles in an AP solution. It is thus able to achieve reasonably short solution times or to terminate with a small gap for many instances. UBnB, which only considers the breaking of subcycles, has great difficulties for many of these instances. In comparison with CDT and Concorde, LBnB performs particularly well for clustered instances.

6. Discussion, summary and future research directions

Our computational experiments show that LBnB outperforms the UBnB algorithm for most instances. The comparison with the state-of-the-art BnB (Carpaneto–Dell'Amico–Toth) and branch-and-cut (Concorde) codes shows that LBnB is faster than branch-and-cut but somewhat slower than CDT for relatively easy instances. The reverse holds for more difficult instances. For some instances, most notably the clustered ones, LBnB achieves the best results. We wish to focus on clustered ATSP instances in a follow-up to this work and we expect that lower tolerance-based algorithms have large potential for such instances.

There is potential to make use of lower tolerances for other problems than the ATSP and its AP relaxation. It takes about the same amount of time to compute upper and lower tolerances for the minimum spanning tree problem [15] and the shortest path problem [20]. There are many NP-hard problems that have these problems as relaxations; see for example [10]. An interesting direction of future research is to use lower tolerances of edges or arcs outside the optimal tree or the shortest paths to determine solutions or lower bounds to these NP-hard problems.

Acknowledgments

The authors would like to thank the editor and the reviewers for their helpful comments. Boris Goldengorin is very grateful for a partial financial support from the project No 11-04-0008: "Calculus of tolerances in combinatorial optimization: theory and algorithms", carried out within the Higher School of Economics' 2011 Academic Fund Program.

References

[1] 〈http://www.tsp.gatech.edu/apps/index.html〉.
[2] Achterberg T, Koch T, Martin A. Branching rules revisited. Operations Research Letters 2004;33(1):42–54.

[3] Applegate D, Bixby RE, Chvátal V, Cook W. On the solution of traveling salesman problems. Documenta Mathematica Extra Volume 1998;ICM III: 645–56.

[4] Carpaneto G, Toth P. Some new branching and bounding criteria for the asymmetric traveling salesman problem. Management Science 1980;21:736–43.

[5] Carpaneto G, Dell'Amico M, Toth P. Exact solution of large-scale, asymmetric traveling salesman problems. ACM Transactions on Mathematical Software 1995;21(4):394–409.

[6] Christofides N. Graph theory an algorithmic approach. Academic Press; 1975. p. 266–75.

[7] Cirasella J, Johnson DS, McGeoch LA, Zhang W. Asymmetric traveling salesman problem: algorithms, instance generators and tests. In: Buchsbaum AL, Snoeyink J, editors. Algorithm engineering and experimentation, third international workshop, Alenex. Lecture notes in computer science, vol. 2153; 2001. p. 32–59.

[8] Climer S, Zhang W. Cut-and-solve: an iterative search strategy for combinatorial optimization problems. Artificial Intelligence 2006;170:714–38.

[9] Fischetti M, Lodi A, Toth P. Exact methods for the asymmetric traveling salesman problem. In: Gutin G, Punnen AP, editors. The traveling salesman problem and its variations. Dordrecht: Kluwer; 2002. p. 169–94. [chapter 2].

[10] Garey MR, Johnson DS. Computers and intractability: a guide to the theory of NP-completeness. San Francisco: W.H. Freeman; 1979.

[11] Goldengorin B, Sierksma G, Turkensteen M. Tolerance based algorithms for the ATSP. In: Lecture notes in computer science, vol. 3353; 2004. p. 222–34.

[12] Goldengorin B, Jäger G, Molitor P. Some basics on tolerances. In: The second international conference on algorithmic aspects in information and management. AAIM'06, Hong Kong, China, 20–22 June 2006. Lecture notes in computer science, vol. 4041; 2006. p. 194–206.

[13] Goldengorin B, Jäger G, Molitor P. Tolerances applied in combinatorial optimization. Journal of Computer Science 2006;2(9):716–34.

[15] Helsgaun K. An efficient implementation of the Lin–Kernighan traveling salesman heuristic. European Journal of Operational Research 2000;12:106–30.

[16] Jonker R, Volgenant A. Improving the Hungarian assignment algorithm. Operations Research Letters 1986;5:171–5.

[17] Karp RM, Steele JM. Probabilistic analysis of heuristics. In: Lawler EL, Lenstra JK, Rinnooy Kan AHG, Shmoys DB, editors. The traveling salesman problem. New York: Wiley; 1990. p. 181–205.

[18] Miller DL, Pekny JF. Exact solution of large asymmetric traveling salesman problems. Science 1991;251:754–61.

[19] Naddef B. Polyhedral theory and branch-and-cut algorithms for the symmetric TSP. In: Gutin G, Punnen AP, editors. The traveling salesman problem and its variations. Kluwer Academic Publishers; 2002. p. 29–116.

[20] Ramaswamy R, Orlin JB, Chakravarti N. Sensitivity analysis for shortest path problems and maximum capacity path problems in undirected graphs. MIT Sloan Working Paper No. 4465-03; 2003.

[21] Reinelt G. TSPLIB—a traveling salesman problem library. ORSA Journal on Computing 1991;3:379–84.

[22] Turkensteen M, Ghosh D, Goldengorin B, Sierksma G. Tolerance-based branch and bound algorithms for the ATSP. European Journal of Operational Research 2008;189:775–88.

[23] Volgenant A. An addendum on sensitivity analysis of the optimal assignment. European Journal of Operational Research 2006;169:338–9.

[24] Zhang W. Truncated branch-and-bound: a case study on the asymmetric TSP. In: Proceedings of the AAAI-93 spring symposium on AI and NP-hard problems, Stanford; 1993. p. 160–6.