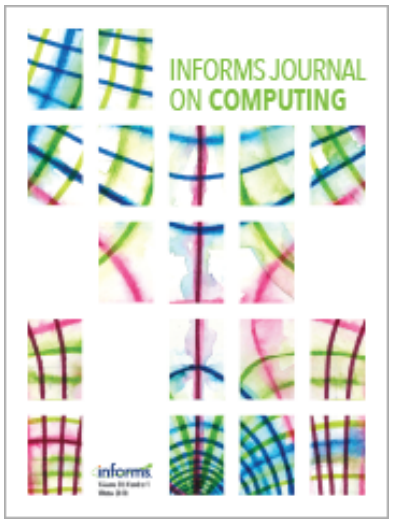

INFORMS Journal on Computing

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

A Branch-and-Bound Algorithm for the Close-Enough Traveling Salesman Problem

Walton Pereira Coutinho, Roberto Quirino do Nascimento, Artur Alves Pessoa, Anand Subramanian

To cite this article:
Walton Pereira Coutinho, Roberto Quirino do Nascimento, Artur Alves Pessoa, Anand Subramanian (2016) A Branch-and-Bound Algorithm for the Close-Enough Traveling Salesman Problem. INFORMS Journal on Computing 28(4):752-765. https://doi.org/10.1287/ijoc.2016.0711

INFORMS Journal on Computing
Vol. 28, No. 4, Fall 2016, pp. 752–765
ISSN 1091-9856 (print) | ISSN 1526-5528 (online)

http://dx.doi.org/10.1287/ijoc.2016.0711

A Branch-and-Bound Algorithm for the Close-Enough Traveling Salesman Problem

Walton Pereira Coutinho
School of Mathematics, University of Southampton, Southampton SO17 1BJ, United Kingdom, w.p.coutinho@soton.ac.uk

Roberto Quirino do Nascimento
Departamento de Computação Científica, Centro de Informática — Universidade Federal da Paraíba, Unidade Lynaldo Cavalcanti de Albuquerque, 58055-000, João Pessoa - PB, Brazil, roberto@ci.ufpb.br

Artur Alves Pessoa
Departamento de Engenharia de Produção — Universidade Federal Fluminense, São Domingos, 24210-240, Niterói - RJ, Brazil, artur@producao.uff.br

Anand Subramanian
Departamento de Engenharia de Produção, Centro de Tecnologia — Universidade Federal da Paraíba, 58051-970, João Pessoa - PB, Brazil, anand@ct.ufpb.br

This paper addresses the close-enough traveling salesman problem. In this problem, rather than visiting the vertex (customer) itself, the salesman must visit a specific region containing such vertex. To solve this problem, we propose a simple yet effective exact algorithm, based on branch-and-bound and second order cone programming. The proposed algorithm was tested in 824 instances suggested in the literature. Optimal solutions are obtained for open problems with up to a thousand vertices. We consider instances both in two- and three-dimensional space.

Keywords: close-enough traveling salesman problem; branch-and-bound; second-order cone programming
History: Accepted by Karen Aardal, Area Editor for Design and Analysis of Algorithms; received January 2015; revised October 2015, December 2015; accepted March 2016. Published online October 5, 2016.

1. Introduction

The traveling salesman problem (TSP) has been widely studied for several decades. In the symmetric TSP one aims to find a shortest *Hamiltonian cycle* in a complete and undirected graph $G = (V, E)$, where V is the set of vertices (customers) and E is the set of edges connecting the vertices. The length of each edge is given by a previously determined metric, for example, the *Euclidean* distance between two vertices p and $q \in V$ that can be defined as $d_{pq} = \|\mathbf{v}^p - \mathbf{v}^q\|_2$, where $\mathbf{v}^p$ and $\mathbf{v}^q$ denote the respective coordinate vectors of p and q and $\|\cdot\|_2$ the Euclidean norm. Due to the importance and applicability of the TSP, many variants and numerical algorithms to compute exact and approximate solutions for these problems have been proposed in the literature. More information about the TSP can be found, e.g., in the book by Applegate et al. (2011).

The close-enough traveling salesman problem (CETSP) is a generalization of the TSP and can be cast, according to Mennell (2009), as a particular case of three other TSP-related problems: the TSP with neighborhoods (TSPN) (Arkin and Hassin 1994), the generalized traveling salesman problem (Silberholz and Golden 2007), and the covering tour problem (Gendreau et al. 1997). In the CETSP, rather than visiting the vertex (*customer*) itself, the salesman must visit a specific region, here denoted as *covering region*, containing such vertex. In this paper we assume that such regions are circles. This is a common assumption in the CETSP literature. Therefore, a vertex $i \in V$ is considered to be covered if the salesman passes through the disc D_i with radius r_i containing the vertex i or at least touches the border of this disc. In a three-dimensional space the regions are considered to be spheres and, in the same sense, the vertices are considered covered if the salesman passes through or at least touches the border of their respective covering spheres.

The CETSP can be formally described as follows: We are given a set of vertices $V = \{0, \ldots, n\}$ in a bidimensional space with coordinates $(\bar{x}_i, \bar{y}_i)$, $i = 0, \ldots, n$. Each vertex i is in the center of a convex region bounded by a circle $\mathcal{D}_i$ with radius r_i. We assume that $(\bar{x}_i, \bar{y}_i) \neq (\bar{x}_j, \bar{y}_j)$, $\forall\, i, j \in V$, $i \neq j$, i.e., there is no repetition of vertices. The problem lies in determining the value of the coordinates of the *hitting points* $(x_i, y_i) \in \mathbb{R}^2$, $i = 0, \ldots, n$, a.k.a. *representative points* (Mennell 2009), and

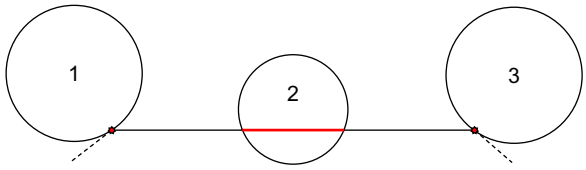

Figure 1 **(Color online) In This CETSP Solution, Vertices 1 and 3 Are Represented by the Coordinates (x_1, y_1) and (x_3, y_3), Respectively, Whereas Vertex 2 Can Be Represented by Any Point of the Line Segment Lying Within Its Boundaries**

a sequence $\mathcal{S} = (k_0, k_1, \ldots, k_n)$, $k_i \in V$, representing the order in which the vertices are covered, such that the tour over the hitting points forms a *Hamiltonian cycle* of minimum length and $(x_i, y_i) \in \mathcal{D}_i$, $\forall\, i \in V$. In the example of Figure 1, the tour "hits" the circle associated with vertices 1 and 3 only in a single point, whereas for the vertex 2, the tour "hits" its associated circle at an infinite number of points. Here we denote the vertex $i = 0$ representing the depot and we assume that $\mathcal{D}_0$ is a circle with $r_0 = 0$. This definition can be easily extended to the three-dimensional space by using three coordinates, that is $(\bar{x}_i, \bar{y}_i, \bar{z}_i)$, instead of two.

Behdani and Smith (2014) proved that any optimal solution to the CETSP can be represented by a set of straight line segments connecting at most n points, called *turn points*, forming a tour. Because the number of such turn points can be smaller than n, some hitting points may be useless in the sense that they do not change the trajectory of the tour. Our method attempts to explore this fact by solving subproblems where some of the covering constraints are relaxed.

Figure 2 illustrates an example of a CETSP solution in the two-dimensional space; Figure 3 depicts a CETSP solution in the three-dimensional space.

The CETSP has several applications in real-world problems. For example, by using Radio Frequency Identification (RFID) tags connected to physical meters one can encode the identification number of the meter and its current reading into digital signals.

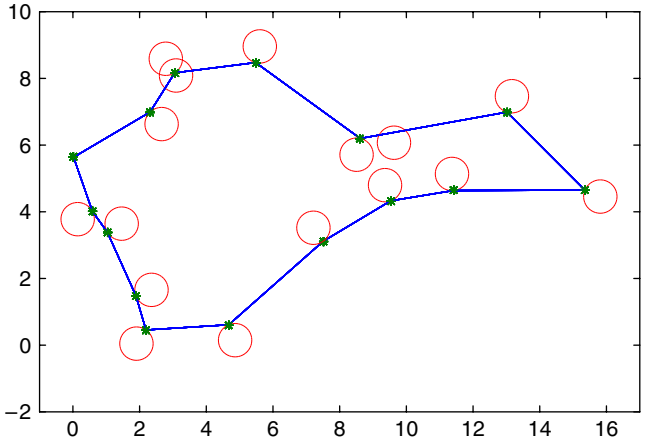

Figure 2 **(Color online) Feasible Solution for the CETSP in $\mathbb{R}^2$**

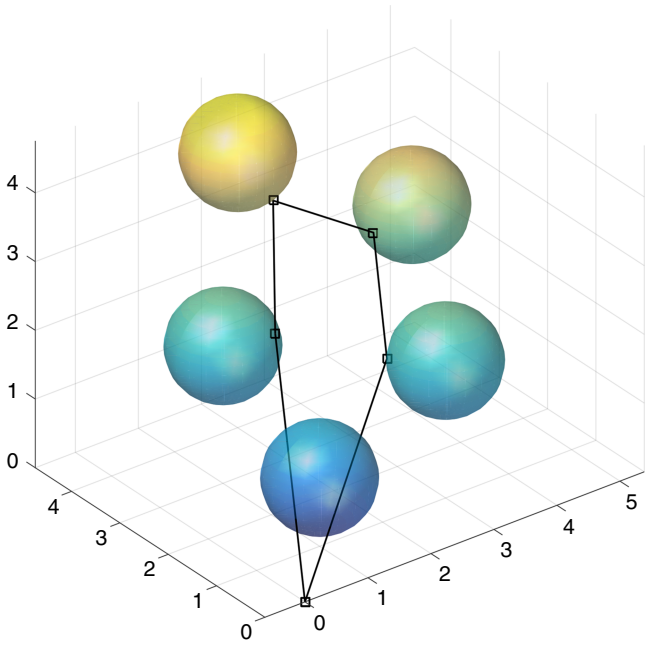

Figure 3 **(Color online) Feasible Solution for the CETSP in $\mathbb{R}^3$**

This way a utility truck, equipped with an Automatic Meter Reading (AMR) system, can remotely collect and transmit data from a certain distance. Hence, in the AMR context, the meter reader is not required to personally visit each customer, but only to get within a certain radius of every customer (Gulczynski et al. 2006, Dong et al. 2007).

Unmanned aerial vehicles (UAVs) have been widely used for military and civil missions, especially when it is unsafe for humans to be on board. UAVs can be used, for example, for aerial reconnaissance, aerial forest fire detection, ship tracking, supply delivering (food, munitions) to targets, geographic region monitoring, and surveillance of pipelines. If the UAV is equipped with a sensor, the vehicle can successfully operate from a certain distance to the target. Another example is when the UAV needs to drop its cargo as close as possible to the target locations, as in special military operations. These are all practical applications that can be modeled as a CETSP.

Recently, Behdani and Smith (2014) pointed out that there is a lack of exact algorithms for solving the CETSP. Moreover, Mennell et al. (2011) state that developing lower bounds for the CETSP is a nontrivial task. To the best of our knowledge, we are the first to present a method that yields exact optimal solutions for the CETSP in a finite number of steps. More specifically, we propose a simple yet effective combinatorial branch-and-bound algorithm, in which the subproblem is based on a second-order cone programming (SOCP) formulation (Lobo et al. 1998, Farid and Goldfarb 2003, Boyd and Vandenberghe 2004), capable of solving instances with up to a thousand nodes.

Our method is also designed to address n-dimensional hyperspheres. However, for practical purposes, we limited our computational experiments to instances in two- and three-dimensional spaces. Still, with further modifications on the branching rules and the feasibility check procedure, this algorithm can also be generalized for the case wherein the covering regions are modeled as ellipsoids.

The remainder of this paper is organized as follows. Section 2 surveys some related work. Section 3 describes the proposed branch-and-bound approach. Computational results are provided in Section 4. Section 5 presents concluding remarks.

2. Related Work

In this section we provide a brief outline of solution approaches proposed for solving the CETSP and some of the related variants.

With a view to obtaining near-optimal solutions, several heuristics have been developed for the CETSP. Gulczynski et al. (2006) and Dong et al. (2007) proposed several heuristic methods for the case wherein all discs have the same radius. Their methods are based on the concept of *super-nodes*. A feasible super-node set S is defined as the set of points in $\mathbb{R}^2$, including the depot, such that each vertex $v_i \in V$, $i = 1, \ldots, n$, is within r units of at least one point of S. The heuristics were based on three distinct phases: (i) generation of the super-node set; (ii) construction of the tour over the super-node set; and (iii) improvement of the tour by moving the super-nodes. Although the heuristics share the same main structure, the procedures adopted in each phase differ from each other.

Yuan et al. (2007) addressed the optimal robot routing problem (ORRP) as a TSPN where the compact sets covering the vertices are disjoint discs with a given radius. The ORRP consists in designing the optimal route of a mobile robot operating in a wireless sensor network in such a way that the robot can collect data from all sensors while minimizing the total distance traveled. Hence, this problem may be cast as a particular CETSP instance where the discs representing the action area are all disjoint. The authors proposed a two-phase algorithm for the TSPN by decomposing the TSPN into a combinatorial problem and a continuous optimization problem. The former aims to find a near-optimal solution for the TSP over the original vertices; the latter is based on an evolutionary algorithm that applies search space reduction techniques to find feasible *hitting points*.

Based on preliminary results in Mennell (2009), Mennell et al. (2011) put forward a *Steiner zone heuristic* based on three phases. The first is the so-called *graph reduction*, where the total number of vertices is reduced to a smaller number of *Steiner zones*. The second is the *tour finding*, where a TSP tour is built over super-nodes selected from each Steiner zone. In the last phase, denoted as *tour improvement*, exact and heuristic approaches are applied to shorten the tour length.

To the best of our knowledge, the only exact approaches devised for the CETSP were suggested by Behdani and Smith (2014). The authors proposed two different partitioning schemes to approximate the continuous covering regions. One of them is the so-called *grid-based* discretization, which approximates arbitrary covering regions by rectangular cells. The other is the so-called *arc-based* partitioning, primarily intended for circular covering regions, which discretizes the borders of the circles into possible hitting points (arc cells). Three Mixed Integer Programming (MIP) formulations based on such approximation schemes were developed to provide lower bounds for the problem.

The first approach solves a TSP over a grid- or arc-based CETSP partitioning set $C = \{C_0, \ldots, C_n\}$, where each C_i is a nonempty and nonoverlapping compact set containing the boundary of a covering region S_i, $i \in V$. In other words, the continuous covering region of each vertex $i \in V$ is replaced by a set C_i of representative points such that the convex hull of C_i contains the whole covering region. Moreover, at least one point of C_i must be visited by a tour to cover the corresponding vertex $i \in V$. In addition to the TSP formulation, cutting-plane generation procedures were developed to avoid subtours and for tightening the lower bounds of the original formulation.

The second MIP-based approach proposed by Behdani and Smith (2014) takes into account the travel distances inside the partitioning cells, thus improving existing lower bounds. In this case, the objective function accounts for the minimum distance between every tuple of three compact sets (C_i, C_j, C_k), where $i, j, k \in V$. For this purpose, auxiliary variables were introduced for each set of three vertices. In addition, the distance between the compact sets was computed as the length of the shortest path that traverses them.

The third and last approach developed by the authors is based on a two-stage MIP formulation. The overall idea is slightly similar to the one presented in this paper in the sense of first identifying the order in which the targets are visited and then optimizing the path built. However, in their case, optimization is performed by determining the shortest path over a grid approximation. More precisely, their method selects a unique representative cell from each compact set that covers each target. Here we optimize the path by solving a SOCP, thus avoiding approximation. This enabled our algorithm to find optimal solutions in a finite number of steps, especially when considering instances with large vertices. Moreover, while the referred authors developed a Benders' decomposition

algorithm, we devised a simple implicit enumeration algorithm to find lower and upper bounds.

Together Behdani and Smith (2014) were capable of finding tight lower and upper bounds for instances with up to 21 vertices. On the one hand, the more the partitioning scheme approximates the circular regions the more the solutions tend to converge towards the actual optimal solution. On the other hand, high partitioning levels make their approaches prohibitively expensive in terms of computing time.

More recently, Hà et al. (2014) studied a CETSP variant called the closeenough arc routing problem (CEARP). In the CEARP there is a predefined directed graph $G = (V, A)$, where $V = \{v_0, v_1, \ldots, v_{n-1}\}$ is the set of vertices and $A = \{(v_i, v_j) \mid v_i, v_j \in V\}$ is the set of arcs connecting these vertices, and another set of vertices in $\mathbb{R}^2$, representing the customers and denoted by $W = \{w_1, w_2, \ldots, w_l\}$, that must be covered. The CEARP consists of finding a minimum-cost cycle over the graph G such that every customer in W is covered, i.e., lies a certain distance from any arc of the cycle. This problem fits exactly in the AMR context; it was originally called *CETSP over a street network* by Shuttleworth et al. (2008), since in their approach the arcs of the graph are associated with streets.

3. Branch-and-Bound Algorithm

This section presents a complete description of the proposed combinatorial branch-and-bound algorithm for the CETSP.

In summary, the method is as follows: Each branch-and-bound node is associated with an optimal partial tour that needs to visit only a given subset of vertices in a particular order. At the root node, the algorithm chooses three vertices to generate an initial sequence of nodes that need to be visited. Because there are only three vertices involved in this sequence and costs are symmetric, their order will not affect the solution. Therefore, the problem of finding an optimal tour that visits these three vertices in the given order is a valid relaxation of the main problem regardless of the choice of the initial sequence. In Section 3.2 we show that this problem can be formulated as a SOCP. If the associated solution is feasible, i.e., all customers are covered, then this solution is optimal and the problem is solved. Otherwise, for this root node, the algorithm branches into three subproblems; in each of them, a vertex that does not belong to the tour is inserted in a different position. The choice of which vertex to insert in the tour can be arbitrary but, as it has a major impact in the practical performance of the algorithm, the strategy used for that is discussed in Section 3.5. A node is discarded if its cost is greater than or equal to the best known upper bound or if its associated solution is feasible. Otherwise, a branching is performed over this node using the same rationale applied in the root node. Note that the number of child nodes (subproblems) for every node in the tree is equal to the number of vertices in the partial sequence of the parent node. Note also that a node cannot be removed from a partial sequence.

Figure 4 shows an example of the execution of the method for an instance involving seven vertices. The set of uncovered vertices is represented by a list alongside its corresponding node, whereas the bold numbers represent the vertices to be inserted. In this

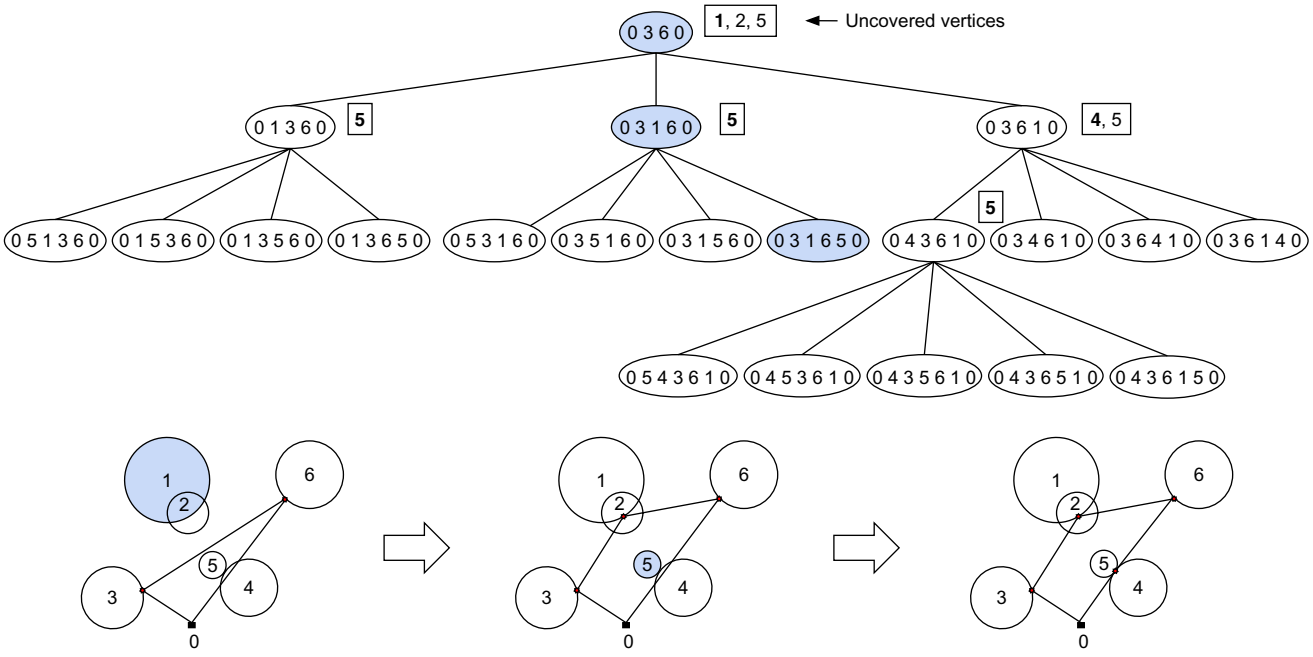

Figure 4 (Color online) Example of the Proposed Branch-and-Bound Algorithm for an Instance with Seven Vertices

case, the relaxed solution found at the root node was $0 \to 3 \to 6 \to 0$. Next, vertex 1 was selected to be inserted in every possible position, thus resulting in three child nodes. When vertex 1 is inserted in the first or second position of the tour it can be verified that vertex 4 is covered; the same does not happen when vertex 1 is inserted in the third position because vertex 4 remains uncovered. Branchings are then performed as long as they are necessary. This particular example depicts a possible branch-and-bound tree associated with this 7-vertex instance. For the sake of simplicity, pruning by bound was not considered in this example. As indicated in the figure, the optimal solution of this instance is $0 \to 3 \to 1 \to 6 \to 5 \to 0$.

Note that the maximum number of nodes in a certain level l of the branch-and-bound tree is given by $(l+2)!/2$. Moreover, in the worst case, the total number of nodes corresponds to $\sum_{l=0}^{n}((l+2)!/2) = \mathcal{O}((n+2)!)$.

3.1. Correctness

This section formalizes the branch-and-bound algorithm previously summarized and gives a proof for its correctness.

Let $\mathcal{N}_k$ be the kth branch-and-bound node generated by the algorithm, defined as a subsequence of vertices, $(i_0, \ldots, i_q)$, which need to be visited. The subproblem corresponding to $\mathcal{N}_k$ consists of finding an optimal tour that visits the vertices $i_0, \ldots, i_q$, in this order. Let $c(\mathcal{N}_k)$ and $\mathcal{V}(\mathcal{N}_k)$ be the cost and the set of vertices covered by this tour, respectively. The proposed algorithm processes node $\mathcal{N}_k$ as follows: If $\mathcal{V}(\mathcal{N}_k) = V$, then the tour is feasible for the original problem and the node is considered solved. If $c(\mathcal{N}_k)$ is not smaller than the cost of the best feasible solution found so far, then a better solution cannot be found by exploring $\mathcal{N}_k$ and it is considered pruned. Otherwise, an uncovered vertex i^* is inserted in the current subsequence and q new nodes are created with corresponding subsequences $(i_0, i^*, i_1, i_2, \ldots, i_q)$, $(i_0, i_1, i^*, i_2, \ldots, i_q)$, $\ldots$, $(i_0, i_1, i_2, \ldots, i_q, i^*)$. In this case, $\mathcal{N}_k$ is considered as processed.

Next, we present a proof of the correctness of the proposed algorithm.

THEOREM 1. *The objective value returned by the proposed algorithm is optimal to the main problem.*

PROOF. First, note that the algorithm only records objectives corresponding to feasible solutions. Hence, the objective returned by the algorithm cannot be smaller than the main problem's optimal solution. Now, it remains to prove that the solution associated to the returned objective is optimal. By contradiction, suppose that the algorithm terminates with a suboptimal solution. Let $\mathcal{S}$ be the sequence of vertices visited in an optimal solution, let c^* be the optimal objective, and let $\mathcal{N}$ be a pruned branch-and-bound node whose vertices are a subsequence of $\mathcal{S}$. Since $\mathcal{N}$ is a subsequence of $\mathcal{S}$, the optimal solution associated with $\mathcal{S}$ is feasible to the subproblem associated with $\mathcal{N}$. Thus, we have that $c(\mathcal{N}) \le c^*$. As a result, this node would only be pruned if another optimal solution had already been found, which contradicts the suboptimality of the algorithm. It remains to prove that any arbitrary sequence (i_0, i_1, i_2) is a subsequence and starts at the same vertex as a sequence that leads to an optimal solution. Let $\mathcal{S} = (k_0^*, \ldots, k_n^*)$ be the sequence of vertices visited in an optimal solution. Because any shifted sequence $(k_{q+1}^*, \ldots, k_n^*, k_0^*, \ldots, k_q^*)$ has the same cost as $\mathcal{S}$, we may assume without loss of generality that $i_0 = k_0^*$. Moreover, since the sequence $(k_0^*, k_n^*, k_{n-1}^*, \ldots, k_1^*)$ has the same cost as $\mathcal{S}$, we may also assume without loss of generality that i_1 precedes i_2 in $\mathcal{S}$. □

3.2. Second-Order Cone Formulation

In this section we provide a mathematical formulation, based on SOCP, for solving the branch-and-bound subproblems. This formulation has been initially proposed by Mennell (2009) for the touring Steiner zones problem when the sequence of visits is given.

Let $\mathcal{S} = (i_0, \ldots, i_q)$, $q < n$, be any partial sequence found during execution of the branch-and-bound algorithm. The subproblems of the branch-and-bound consist in finding the values of the hitting point coordinates (x_{i_k}, y_{i_k}), $k = 0, \ldots, q$, such that the length of the partial tour is minimized. Let us assume that $i_{-1} = i_q$. The formulation is as follows:

$$\min \sum_{k=0}^{q} z_k \tag{1}$$

$$\text{s.t. } w_k = x_{i_{k-1}} - x_{i_k}, \quad k = 0, \ldots, q \tag{2}$$

$$u_k = y_{i_{k-1}} - y_{i_k}, \quad k = 0, \ldots, q \tag{3}$$

$$s_k = \bar{x}_k - x_k, \quad k = 0, \ldots, q \tag{4}$$

$$t_k = \bar{y}_k - y_k, \quad k = 0, \ldots, q \tag{5}$$

$$z_k^2 \ge w_k^2 + u_k^2, \quad k = 0, \ldots, q \tag{6}$$

$$s_k^2 + t_k^2 \le r_k^2, \quad k = 0, \ldots, q \tag{7}$$

$$z_k \ge 0;\ w_k, u_k, s_k, t_k \text{ free} \quad k = 0, \ldots, q \tag{8}$$

$$x_k, y_k \text{ free} \quad k = 0, \ldots, q. \tag{9}$$

In this SOCP formulation the objective function (1) is linear. Variables z_k, $k = 0, \ldots, q$, represent the distance between subsequent hitting points $(x_{i_{k-1}}, y_{i_{k-1}})$ and (x_{i_k}, y_{i_k}). The auxiliary variables w, u, s and t are defined in the linear constraints (2–5) so as to keep the SOCP on its standard form. They represent differences of coordinates used to calculate Euclidean distances. The second-order cone (SOC) constraints (6)

define the length of the edge connecting subsequent hitting points in the sequence $\mathcal{S}$. The quadratic constraints (7) ensure that the hitting points will lie within their respective customers' covering circles. The expressions (8) and (9) are bounding constraints over the variables.

We know that SOCP problems can be solved in polynomial time (Andersen et al. 2003). Furthermore, some well known optimization software are also now capable of addressing this important class of problems.

3.3. Root Relaxation

This section explains how the algorithm determines an initial sequence, i.e., the one generated at the root node.

Three vertices must be chosen to build an initial partial sequence. They are selected using the following steps. Initially, the depot is selected as the first vertex (as already mentioned, in the CETSP literature the radius of the depot is assumed to be zero). Next, the algorithm chooses the most distant vertex from the depot. Finally, the third vertex selected is the one that leads to the largest lower bound value. More specifically, the algorithm solves a SOCP problem for all remaining candidates and chooses the vertex associated with the sequence that yields the best relaxation.

Figure 5 illustrates an example involving seven vertices. Note that the depot (vertex 0) is the first to be selected to be part of the sequence, followed by vertex 6 (the most distant from 0), and vertex 3, whose insertion criterion is the one just mentioned.

3.4. Checking Feasibility

In this section we explain the procedure developed for checking whether a certain branch-and-bound subproblem solution is feasible for the CETSP.

Consider the set of uncovered vertices $\hat{V} = \{i \in V : \hat{d}_i > r_i\}$, where the distance between a vertex $i \in \hat{V}$ and the edge of a subproblem solution that is nearest to i, denoted by $\hat{d}_i$, is greater than the radius r_i. An *uncovered vertex* is one whose convex bounding region, $\mathcal{D}_i$, $i \in V$, is not intercepted by the solution of its corresponding subproblem. Suppose that c and $\overline{p_1p_2}$ are an arbitrary vertex and an arbitrary edge of a partial solution, respectively. We want to determine the coordinates of a point $p \in \overline{p_1p_2}$ that minimizes the distance between $\overline{p_1p_2}$ and c.

The minimum distance $\hat{d}$ between c and $\overline{p_1p_2}$ is computed by solving the optimization problem defined by Equation (10):

$$\hat{d} = \min_{0 \le \theta \le 1} \|(1-\theta)p_1 + \theta p_2 - c\|, \tag{10}$$

whose analytical solution to the unconstrained problem is given by $\theta^* = -(p_2 - p_1)^\top (p_1 - c)/\|p_2 - p_1\|^2$. Therefore, the minimum distance can be computed by Equation (11):

$$\hat{d} = \begin{cases} \|p_1 - c\|, & \text{if } \theta^* \le 0; \\ \|p - c\|, & \text{if } 0 < \theta^* < 1; \\ \|p_2 - c\|, & \text{if } \theta^* \ge 1, \end{cases} \tag{11}$$

where $p = (1-\theta^*)p_1 + \theta^* p_2$.

The feasibility of a subproblem solution can be verified by checking whether the set $\hat{V}$ is empty. If so, it means that there are no uncovered vertices and thus the solution is feasible. Suppose that the number of vertices is given by n and the number of edges is given by e. We can easily determine that this verification can be done in $\mathcal{O}(ne)$ operations. Note that this approach works for instances in $\mathbb{R}^2$ and $\mathbb{R}^3$.

3.5. Branching Rules

In this section we describe the two branching criteria used to select the next vertex to be inserted in a partial solution associated with a node of the branch-and-bound tree. The appropriate criterion is automatically selected based on the radii of the instances.

If all vertices have the same radius, the first method proceeds as follows: The algorithm first computes the value of $\hat{d}_k$ for every $k \in \hat{V}$. Next, the maximum value among all $\hat{d}_k$ is determined, that is, $\max_{k \in \hat{V}}\{\hat{d}_k\}$, and the corresponding vertex k is inserted in the partial solution. Figure 6 depicts an example of how the

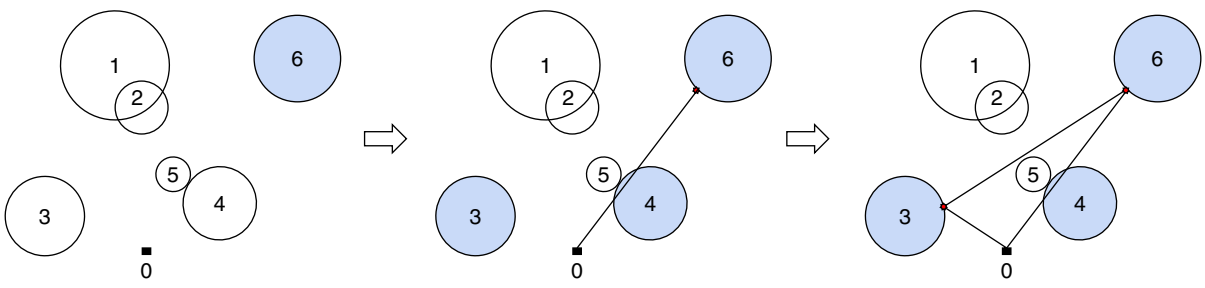

Figure 5 **(Color online) Example of the Procedure to Find a Valid Root Relaxation**

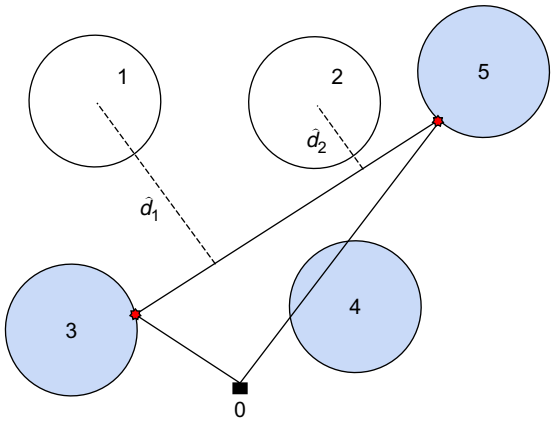

Figure 6 (Color online) Illustration of the Branching Rule Used When All Vertices Have the Same Radius

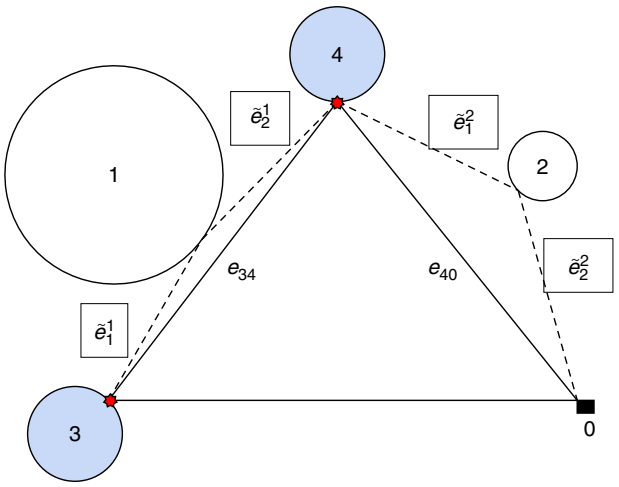

Figure 7 (Color online) Illustration of the Branching Rule Used When the Vertices Have Different Radii

vertex selection is performed. In this case, we are given a partial solution $0 \to 5 \to 3 \to 0$. Note that vertex 4 is covered, but vertices 1 and 2 are not, thus $\hat{V} = \{1, 2\}$. It can be seen that $\hat{d}_1 > \hat{d}_2$. Hence, vertex 1 is the one associated with $\max_{k \in \hat{V}} \{\hat{d}_k\}$.

The first branching rule performs well when all vertices have the same radii because, in this case, the increment on the lower bound of a partial solution is proportional to the minimum distance, measured from the center, between the farthest uncovered vertex and the edges set defined by (x_{i_k}, y_{i_k}), $k = 0, \ldots, q$, where q is the size of a partial sequence. However, when the vertices have different radii the increment on the lower bound is not only proportional to the distance between the farthest uncovered vertex and the aforementioned edges set, but also to the radii of the vertex. In this case, the following branching rule is used.

At first, for all vertices $k \in \hat{V}$, the algorithm computes an estimate, given by γ_k, of how much the lower bound would increase if the vertex k were inserted between its closest neighbors in the sequence. To estimate γ_k, the procedure first calculates the coordinates of the point $\tilde{p}_k$ on the border of the disk D_k that minimizes the distance between this point and its nearest edge $\overline{p_1 p_2}$ on the sequence. Hence, γ_k can be computed as $\gamma_k = \tilde{e}_1^k + \tilde{e}_2^k - e_{ij}$, where $\tilde{e}_1^k$ is the distance between $\tilde{p}_k$ and p_1, $\tilde{e}_2^k$ is the distance between $\tilde{p}_k$ and p_2 and e_{ij} is the length of the edge $\overline{p_1 p_2}$. Figure 7 shows an example of this procedure, where $\hat{V} = \{1, 2\}$. Therefore, $\gamma_1 = \tilde{e}_1^1 + \tilde{e}_2^1 - e_{34}$ and $\gamma_2 = \tilde{e}_1^2 + \tilde{e}_2^2 - e_{40}$. Suppose that $\gamma_2 > \gamma_1$, as the figure suggests. Then, following the second branching rule, vertex 2 would be inserted in the sequence.

4. Computational Experiments

The branch-and-bound algorithm was coded in C++ and the tests were carried out on an Intel Core i7 with 3.40 GHz and 16 GB RAM running under Linux Mint 13. The SOCPs were solved using CPLEX 12.4. Only a single thread was used in our experiments.

We performed a series of experiments to choose the most computationally effective branching strategy to be adopted. Tests were conducted using the following strategies: depth-first search, breadth-first search, and best-first search. The results revealed that the best-first search was the most suitable strategy to be applied in our algorithm.

The proposed algorithm was tested in 824 instances were suggested by Mennell (2009) and Behdani and Smith (2014). The difficulty of each instance is associated with the notion of *overlap ratio*, given by Mennell (2009). This is defined as the ratio between the mean of the radii of all vertices and the largest side ($l_{\max}$) of the rectangle that involves all vertices and their radii, i.e., $\textit{overlap ratio} = ((\sum_{i=0}^{n} r_i)/n)/l_{\max}$.

Mennell (2009) presents the following classifications for his instances: Six problems whose names start with *team*, plus the instance *bonus1000* were denoted *Team Problems*. Those starting with *rotatingDiamonds*, *bubbles*, *concentricCircles*, plus the instance *chaoSingleDep* were gathered in the subset called *Geometric Problems*. Finally, the instances *d493*, *dsj1000*, *kroD100*, *lin318*, *pcb442*, *rat195*, and *rd400* were generated from the TSPLIB; they were called *TSPLIB problems*.

Instances of the groups Team Problems and Geometric Problems were generated with $r_i = r$, $i = 0, \ldots, n$, and different overlap ratios that were not specified. The instances of the group TSPLIB Problems were generated with radii of three different sizes, i.e., with three overlap ratios, i.e., 0.02, 0.10, and 0.30. Only the instances of the groups Team Problems and TSPLIB Problems have 3D versions. Mennell (2009) also provided two groups of instances denoted

Team Random Radius Problems and *TSPLIB Random Radius Problems*, each containing 2D and 3D versions, with radii generated at random in such a way that $r_i \neq r_j$, $\forall i, j \in V$, also denoted by the author as *arbitrary radius* instances.

Behdani and Smith (2014) provided 240 2D test-problems with 7, 9, 11, 13, 15, 17, 19, and 21 vertices. These instances were generated as follows: The position $(\bar{x}_i, \bar{y}_i)$, $i \in V$, of each vertex and the depot i_0 were randomly selected in a limited space of 16 units of length and 10 units of width. The coverage area of each vertex was defined as a circle with radius r. In their experiments, the authors used three radii, i.e., 0.25, 0.50, and 1.00, thus obtaining three distinct groups with overlap ratios 0.015, 0.030, and 0.060, respectively. In this work, we used the same instances presented by Behdani and Smith (2014) to evaluate the performance of the proposed algorithm.

4.1. Results Found for the Small Instances of Behdani and Smith (2014)

Table 1 shows a summary of the results found for the instances of Behdani and Smith (2014). In this table, *Group* denotes the eight groups of instances divided into three different subgroups (overlap ratios = 0.015, 0.030, 0.060), *Group Size* shows the number of instances of each group, *#Opt.* indicates the number of optimal solutions obtained for each group, *Tree Size* corresponds to the average tree size, and *Avg. Time* is the average time, in seconds.

Table 1 Summary Results for Instances of Behdani

Group	*Group size*	*#Opt.*	*Avg. tree size*	*Avg. time*
	$r = 0.25$ and *overlap* = 0.0156			
CETSP-06	30	30	26	0.029
CETSP-08	30	30	51	0.062
CETSP-10	30	30	86	0.107
CETSP-12	30	30	139	0.189
CETSP-14	30	30	219	0.308
CETSP-16	30	30	443	0.673
CETSP-18	30	30	544	0.839
CETSP-20	30	30	1,016	1.656
	$r = 0.50$ and *overlap* = 0.0313			
CETSP-06	30	30	19	0.020
CETSP-08	30	30	39	0.047
CETSP-10	30	30	66	0.085
CETSP-12	30	30	92	0.125
CETSP-14	30	30	144	0.204
CETSP-16	30	30	222	0.334
CETSP-18	30	30	331	0.525
CETSP-20	30	30	437	0.719
	$r = 1.0$ and *overlap* = 0.0625			
CETSP-06	30	30	13	0.017
CETSP-08	30	30	23	0.030
CETSP-10	30	30	38	0.049
CETSP-12	30	30	51	0.066
CETSP-14	30	30	71	0.100
CETSP-16	30	30	95	0.144
CETSP-18	30	30	117	0.176
CETSP-20	30	30	163	0.261

From Table 1, note that all instances were solved to optimality in a matter of seconds. The complete results obtained by our branch-and-bound algorithm for the instances of Behdani and Smith (2014) can be found in the online supplement (available as supplemental material at http://dx.doi.org/10.1287/ijoc.2016.0711).

4.2. Results Found for the 2D Instances of Mennell (2009)

The results for the 2D instances of Mennell (2009) are presented in Table 2. In this table, *Instance* is the name of the instance, $|V|$ indicates the size of the instance, *UB* denotes the upper bound obtained by Mennell (2009), *LB* is the lower bound by a given method, *Opt.* corresponds to the cost of the optimal solutions found by our branch-and-bound algorithm, *Tree Size* represents the size of the branch-and-bound tree, *Gap* (%) indicates the gap between the lower bound and the best known upper bound or the optimal solution, and *Time* (s) is the total computing time in seconds. Note that a time limit of four hours was imposed for each instance. New improved (optimal) solutions are highlighted in boldface. Some associated optimal tours are depicted by Figures 8–10.

By observing the results of Table 2, we can see that our branch-and-bound algorithm found optimal solutions of 22 of 62 instances. We can also verify that all lower bounds available in the literature were dramatically improved. Moreover, the larger the overlap ratio, the better the algorithm works. This was somewhat expected since the number of vertices in the optimal tour tends to be inversely proportional to the overlap ratio value, thus helping the branch-and-bound algorithm to quickly converge towards an optimal solution, since fewer nodes are required to be explored throughout the tree.

4.3. Results Found for the 3D Instances of Mennell (2009)

Table 3 presents the results obtained for the 3D instances of Mennell (2009). The meaning of the columns in this table is the same as in the previous section. Our algorithm found 10 optimal solutions of 42 instances. Moreover, all lower bounds were improved. Some associated optimal tours are shown in Figures 11–13.

4.4. Overlap Ratio's Influence on the Algorithm Performance

As previously mentioned, the instance difficulty is associated with its overlap ratio. In this section we provide some insights on the performance of the algorithm when adopting different overlap ratios for an instance.

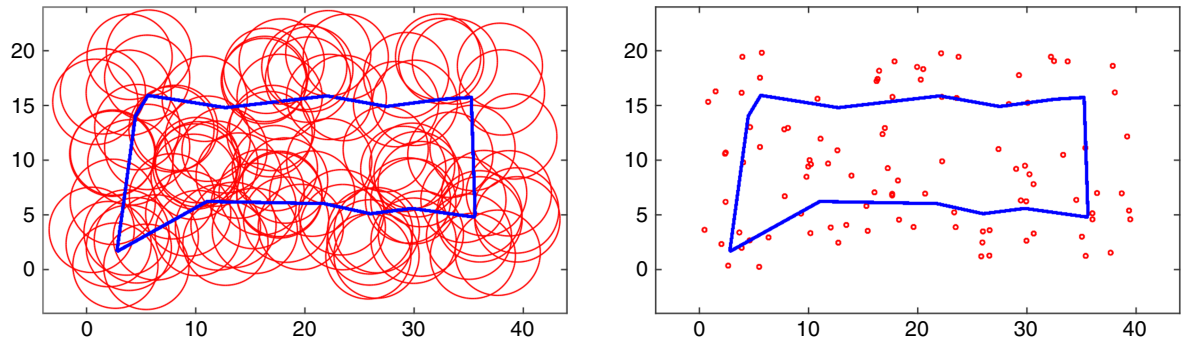

Figure 8 (Color online) Solution for *kroD100*, Equal radius, 2D, Size = 100, Length = 89.668

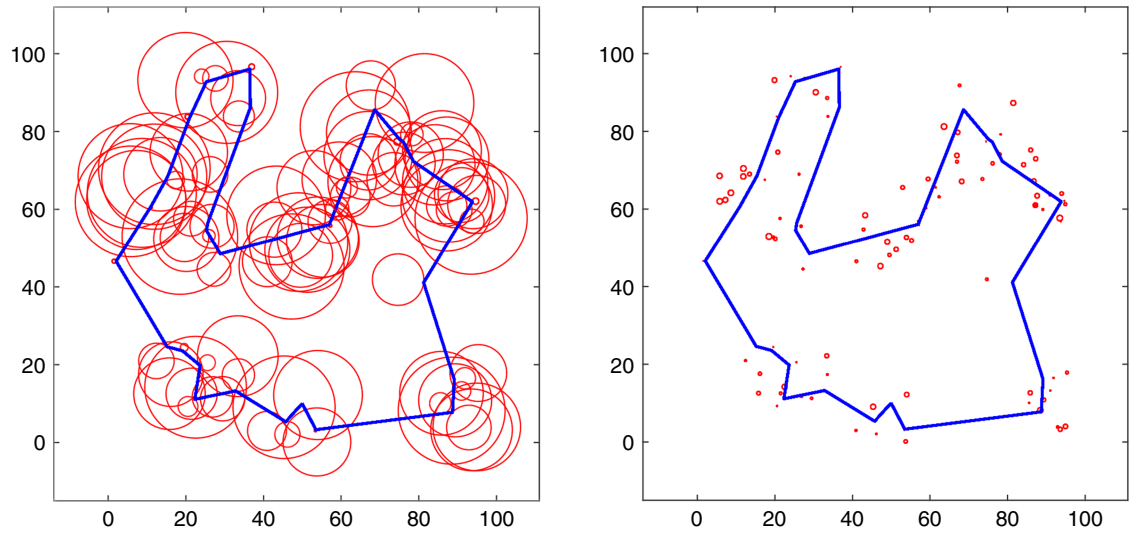

Figure 9 (Color online) Solution for *team1_100rdmRad*, Arbitrary radius, 2D, Size = 101, Length = 388.537

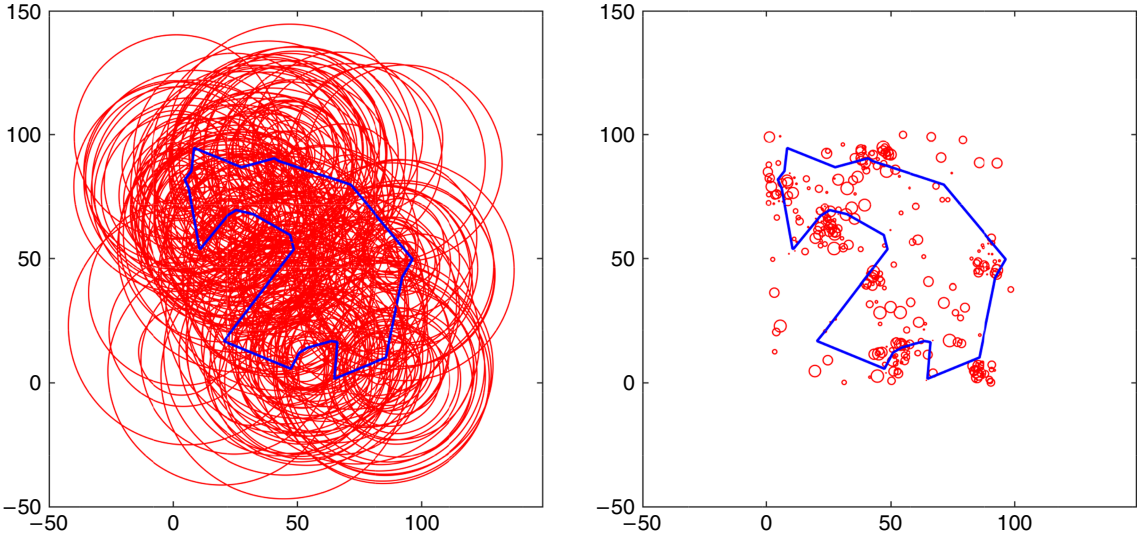

Figure 10 (Color online) Solution for *team3_300rdmRad*, Arbitrary Radius, 2D, Size = 301, Length = 378.087

Table 2 Results for 2D Instances of Mennell

			Mennell (2009)		Branch and bound				
Instance	*Size*	*Overlap ratio*	*UB*	*LB*	*Opt.*	*LB*	*Tree size*	*Gap* (%)	Time (s)
					Low overlap ratio				
d493	493	0.02	202.793	57.460	—	**146.331**	859,216	27.842	14,400.17
dsj1000	1,000	0.02	935.743	100.190	—	**559.114**	585,631	40.249	14,400.24
kroD100	100	0.02	159.046	78.830	—	**142.872**	1,784,161	10.169	14,400.16
lin318	318	0.02	2,863.366	977.430	—	**1,990.902**	933,499	30.470	14,400.08
pcb442	442	0.02	323.034	32.510	—	**185.847**	842,855	42.468	14,400.03
rat195	195	0.02	158.785	28.200	—	**108.104**	1,108,939	31.918	14,400.04
rd400	400	0.02	1,033.414	284.030	—	**567.190**	891,890	45.115	14,400.27
					Moderate overlap ratio				
d493	493	0.10	101.735	31.730	**100.721**	**100.721**	8,343	0.000	53.28
dsj1000	1,000	0.10	376.099	10.170	—	**373.730**	718,104	0.630	14,400.51
kroD100	100	0.10	89.668	0.000	89.668	**89.668**	798	0.000	1.86
lin318	318	0.10	1,408.482	0.000	**1,394.626**	**1,394.626**	1,118,666	0.000	8,541.19
pcb442	442	0.10	147.244	1.870	—	**137.448**	1,383,948	6.653	14,400.09
rat195	195	0.10	68.083	0.000	**67.991**	**67.991**	4,735	0.000	17.32
rd400	400	0.10	466.102	0.000	—	**432.798**	1,436,555	7.145	14,400.03
					High overlap ratio				
d493	493	0.30	69.758	16.750	69.758	**69.758**	3	0.000	0.32
dsj1000	1,000	0.30	199.948	0.000	199.948	**199.948**	17	0.000	0.75
kroD100	100	0.30	58.541	0.000	58.541	**58.541**	3	0.000	0.07
lin318	318	0.30	765.964	0.000	765.964	**765.964**	7	0.000	0.24
pcb442	442	0.30	83.537	0.000	83.537	**83.537**	3	0.000	0.31
rat195	195	0.30	45.702	0.000	45.702	**45.702**	3	0.000	0.13
rd400	400	0.30	224.839	0.000	224.839	**224.839**	24	0.000	0.33
					Varied overlap ratios				
bonus1000	1,001	0.12	402.470	0.000	—	**359.383**	949,465	10.706	14,400.02
bubbles1	37	0.07	349.135	60.620	349.135	**349.135**	66	0.000	0.10
bubbles2	77	0.06	428.279	60.620	428.279	**428.279**	121	0.000	0.22
bubbles3	127	0.06	530.733	60.620	**529.955**	**529.955**	31,995	0.000	193.12
bubbles4	185	0.06	829.888	60.620	—	**690.578**	1,502,931	16.787	14,400.01
bubbles5	251	0.05	1,062.335	60.620	—	**851.822**	1,181,671	19.816	14,400.28
bubbles6	325	0.05	1,383.139	60.620	—	**993.981**	1,025,324	28.136	14,400.15
bubbles7	407	0.05	1,720.214	60.620	—	**1,123.522**	918,674	34.687	14,400.19
bubbles8	497	0.05	2,101.373	60.620	—	**1,252.715**	894,857	40.386	14,400.28
bubbles9	595	0.04	2,426.274	60.620	—	**1,374.407**	765,301	43.353	14,400.33
chaoSingleDep	201	0.03	1,039.610	439.260	—	**1,000.151**	1,760,829	3.796	14,400.09
concentricCircles1	17	0.03	53.158	14.000	53.158	**53.158**	2,943	0.000	5.18
concentricCircles2	37	0.03	153.132	43.590	—	**149.868**	2,699,245	2.131	14,400.03
concentricCircles3	61	0.02	271.076	115.280	—	**247.624**	2,093,145	8.651	14,400.18
concentricCircles4	105	0.02	454.457	161.110	—	**358.887**	1,320,268	21.030	14,400.06
concentricCircles5	149	0.02	645.381	249.980	—	**459.411**	1,112,726	28.815	14,400.16
rotatingDiamonds1	21	0.02	32.389	6.200	32.389	**32.389**	59	0.000	0.09
rotatingDiamonds2	61	0.02	140.477	13.870	140.477	**140.477**	274,967	0.000	730.37
rotatingDiamonds3	181	0.02	380.882	27.870	—	**348.608**	1,612,881	8.474	14,400.07
rotatingDiamonds4	321	0.01	770.660	35.570	—	**593.350**	969,457	23.008	14,400.03
rotatingDiamonds5	681	0.01	1,510.752	69.570	—	**1,106.577**	777,682	26.753	14,400.23
team1_100	101	0.09	307.337	33.520	307.337	**307.337**	2,805	0.000	9.61
team2_200	201	0.20	246.683	0.000	246.683	**246.683**	187	0.000	0.72
team3_300	301	0.07	466.241	8.750	—	**447.534**	1,713,428	4.012	14,400.03
team4_400	401	0.05	680.211	20.590	—	**507.302**	938,101	25.420	14,400.10
team5_499	500	0.02	702.823	231.210	—	**524.589**	833,603	25.360	14,400.05
team6_500	501	0.27	225.216	0.000	225.216	**225.216**	24	0.000	0.43

In summary, the overlap ratio can be simply defined as the radius of each vertex and the largest side of the rectangle that involves it, i.e., *overlap ratio* $= r/l_{\max}$, where r is the size of each covering circle. For the benchmark data set called TSPLIB Problems, suggested by Mennell (2009), all vertices

Table 2 (Continued)

Instance	Size	Overlap ratio	Mennell (2009) UB	Mennell (2009) LB	Branch and bound Opt.	Branch and bound LB	Branch and bound Tree size	Branch and bound Gap (%)	Branch and bound Time (s)
				Arbitrary radius problems					
bonus1000rdmRad	1,001	0.06	987.114	—	—	**506.131**	656,095	48.726	14,400.22
d493rdmRad	493	0.13	140.120	—	—	**125.312**	1,195,835	10.568	14,400.14
dsj1000rdmRad	1,000	0.14	653.128	—	—	**509.740**	798,273	21.954	14,400.30
kroD100rdmRad	100	0.04	141.835	—	—	**136.620**	2,293,903	3.677	14,400.12
lin318rdmRad	318	0.10	2,080.574	—	—	**1,807.681**	1,349,251	13.116	14,400.00
pcb442rdmRad	442	0.09	235.188	—	—	**175.834**	966,808	25.237	14,400.23
rat195rdmRad	195	0.42	68.224	—	68.224	**68.224**	1,628	0.000	5.16
rd400rdmRad	400	0.01	1,252.380	—	—	**571.482**	776,842	54.368	14,400.29
team1_100rdmRad	101	0.08	388.537	—	388.537	**388.537**	86,565	0.000	269.31
team2_200rdmRad	201	0.05	622.738	—	—	**488.182**	1,098,493	21.607	14,400.12
team3_300rdmRad	301	0.24	381.828	—	**378.087**	**378.087**	132,111	0.000	682.39
team4_400rdmRad	401	0.02	1,011.772	—	—	**549.907**	870,580	45.649	14,400.32
team5_499rdmRad	500	0.20	454.327	—	—	**442.637**	1,145,471	2.573	14,400.36
team6_500rdmRad	501	0.10	666.149	—	—	**489.612**	939,270	26.501	14,400.08

Table 3 Results for 3D Instances of Mennell

Instance	Size	Overlap ratio	Mennell (2009) UB	Mennell (2009) LB	Branch and bound Opt.	Branch and bound LB	Branch and bound Tree size	Branch and bound Gap (%)	Branch and bound Time (s)
				Low overlap ratio					
d493	493	0.01	1,353.137	—	—	**469.783**	642,868	65.282	14,400.01
dsj1000	1,000	0.02	3,147.865	—	—	**751.510**	494,579	76.126	14,400.05
kroD100	100	0.02	202.021	—	—	**148.231**	1,183,277	26.626	14,400.12
lin318	318	0.02	3,044.270	—	—	**1,994.372**	934,859	34.488	14,400.14
pcb442	442	0.02	404.490	—	—	**186.381**	725,462	53.922	14,400.18
rat195	195	0.02	291.258	—	—	**126.493**	978,598	56.570	14,400.06
rd400	400	0.02	3,218.198	—	—	**868.188**	753,967	73.023	14,400.12
				Moderate overlap ratio					
d493	493	0.03	665.056	—	—	**421.164**	827,584	36.672	14,400.18
dsj1000	1,000	0.10	1,021.252	—	—	**602.987**	644,546	40.956	14,400.31
kroD100	100	0.10	91.669	—	**91.663**	**91.663**	2,407	0.000	7.51
lin318	318	0.10	1,443.427	—	—	**1,398.254**	1,588,841	3.130	14,400.12
pcb442	442	0.10	154.810	—	—	**137.951**	1,209,228	10.890	14,400.13
rat195	195	0.10	112.405	—	—	**88.721**	1,325,338	21.070	14,400.13
rd400	400	0.10	1,552.723	—	—	**752.423**	851,059	51.542	14,400.24
				High overlap ratio					
d493	493	0.08	335.592	—	**325.207**	**325.207**	5,064	0.000	31.11
dsj1000	1,000	0.30	270.399	—	**267.751**	**267.751**	2,472	0.000	25.07
kroD100	100	0.30	58.926	—	58.926	**58.926**	4	0.000	0.08
lin318	318	0.30	766.831	—	766.831	**766.831**	8	0.000	0.24
pcb442	442	0.30	83.722	—	83.722	**83.722**	4	0.000	0.33
rat195	195	0.30	47.889	—	47.889	**47.889**	4	0.000	0.14
rd400	400	0.30	539.954	—	—	**450.720**	507,442	16.526	14,400.40
				Varied overlap ratios					
bonus1,000	1,001	0.12	941.348	—	—	**472.559**	655,479	49.800	14,400.35
team1_100	101	0.09	820.727	—	—	**690.298**	1,197,716	15.892	14,400.22
team2_200	201	0.20	283.238	—	**273.383**	**273.383**	86,492	0.000	557.94
team3_300	301	0.07	1,484.411	—	—	**762.683**	930,319	48.620	14,400.12
team4_400	401	0.05	753.813	—	—	**509.803**	894,994	32.370	14,400.30
team5_499	500	0.02	1,924.527	—	—	**705.633**	665,171	63.335	14,400.20
team6_500	501	0.27	236.964	—	**230.923**	**230.923**	73	0.000	0.67

Table 3 (Continued)

Instance	Size	Overlap ratio	Mennell (2009) UB	Mennell (2009) LB	Branch and bound Opt.	Branch and bound LB	Branch and bound Tree size	Branch and bound Gap (%)	Branch and bound Time (s)
				Arbitrary radius problems					
bonus1000rdmRad	1,001	0.06	2,689.413	—	—	**578.638**	512,678	78.485	14,400.04
d493rdmRad	493	0.03	761.065	—	—	**438.701**	858,509	42.357	14,400.06
dsj1000rdmRad	1,000	0.14	2,074.844	—	—	**696.289**	656,155	66.441	14,400.05
kroD100rdmRad	100	0.04	171.568	—	—	**137.765**	1,350,550	19.702	14,400.13
lin318rdmRad	318	0.10	2,189.426	—	—	**1,806.783**	1,200,921	17.477	14,400.10
pcb442rdmRad	442	0.09	258.404	—	—	**177.231**	962,861	31.413	14,400.09
rat195rdmRad	195	0.42	84.470	—	**82.105**	**82.105**	152,636	0.000	790.53
rd400rdmRad	400	0.01	3,592.601	—	—	**876.280**	766,218	75.609	14,400.06
team1_100rdmRad	101	0.08	907.593	—	—	**726.685**	1,137,079	19.933	14,400.26
team2_200rdmRad	201	0.05	1,055.948	—	—	**525.310**	1,027,597	50.252	14,400.20
team3_300rdmRad	301	0.24	1,053.380	—	—	**676.184**	1,004,637	35.808	14,400.10
team4_400rdmRad	401	0.02	1,276.896	—	—	**551.046**	793,139	56.845	14,400.08
team5_499rdmRad	500	0.20	840.477	—	—	**599.741**	940,015	28.643	14,400.28
team6_500rdmRad	501	0.10	1,076.352	—	—	**507.122**	870,473	52.885	14,400.12

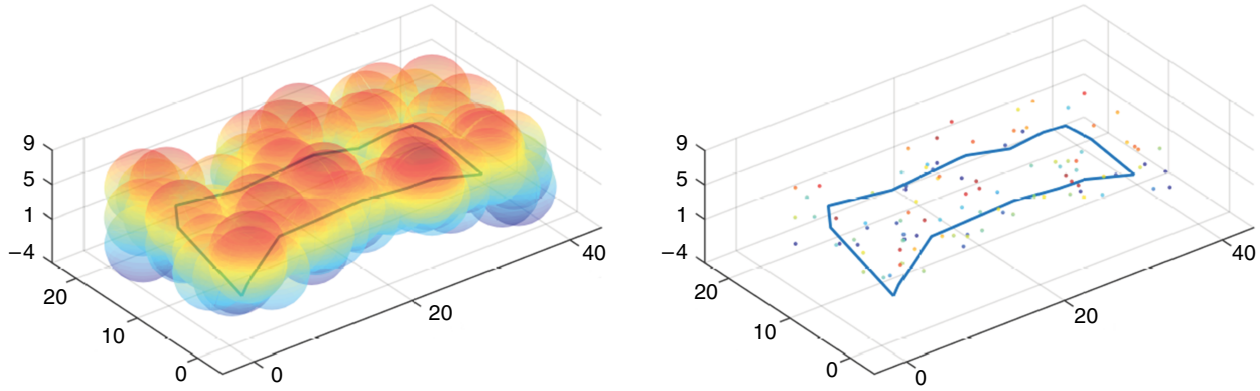

Figure 11 (Color online) Solution for ***kroD100***, Equal Radius, 3D, Size = 100, Length = 91.663

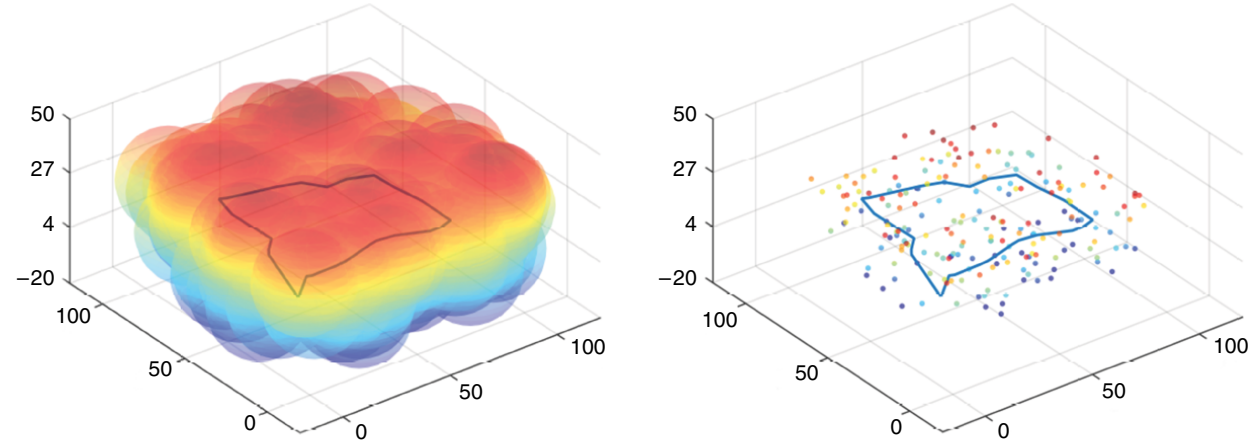

Figure 12 (Color online) Solution for ***team2_200***, Equal Radius, 3D, Size = 201, Length = 273.383

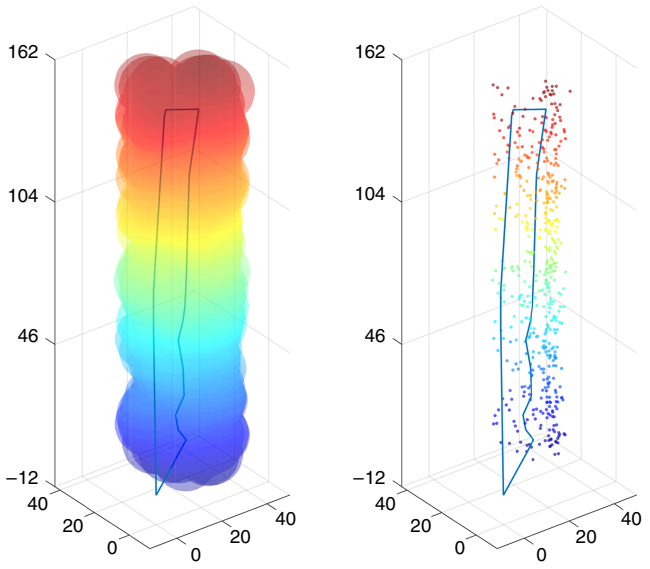

Figure 13 **(Color online) Solution for *d493*, Equal Radius, 3D, Size = 493, Length = 325.207**

have the same radius. Hence, we can vary the overlap ratio of each instance by multiplying r by a constant value f, here denoted as *radius factor*.

In view of this, a simple experiment could be conducted to demonstrate the influence of the overlap ratio on the branch-and-bound algorithm's performance. Specifically, we ran the proposed algorithm for each instance of the aforementioned benchmark data set, but considering different values for f, in particular, values ranging from 0.5 to 1.5.

As expected, the algorithm's performance improves as the overlap among the vertices increases. In addition, the number of branch-and-bound nodes as well as the computing times highly depend on the overlap ratio of each instance. Therefore, for the instances where the radii of the vertices are very small, the enumeration scheme is likely to fail in practice. Note also that these measures of performance might explode if the overlap ratio decreases to a certain value. Results for some instances are shown in Figures 14–17 where

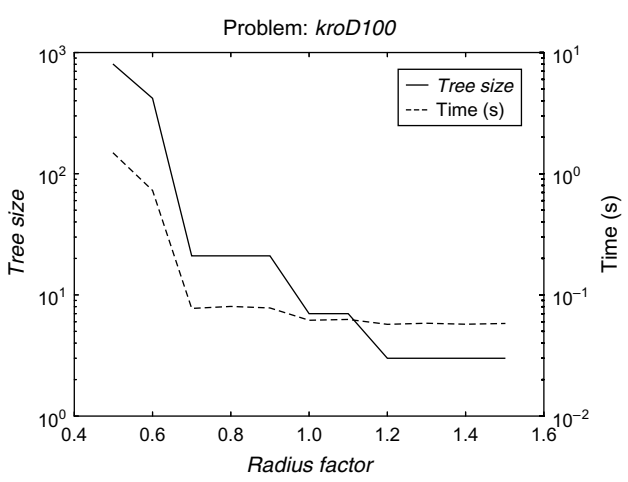

Figure 14 **Tree Size and CPU Time for *kroD100***

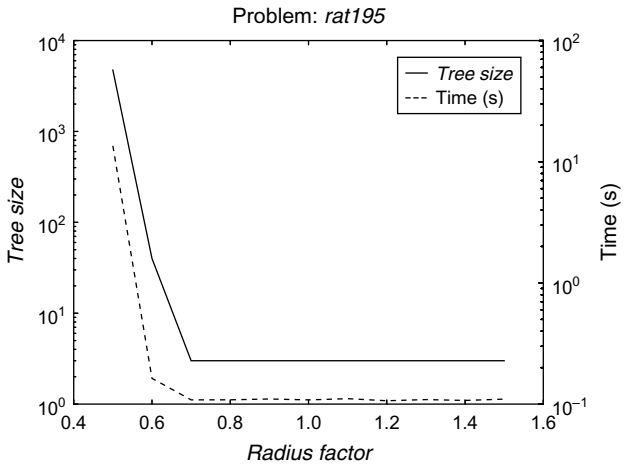

Figure 15 **Tree Size and CPU Time for *rat195***

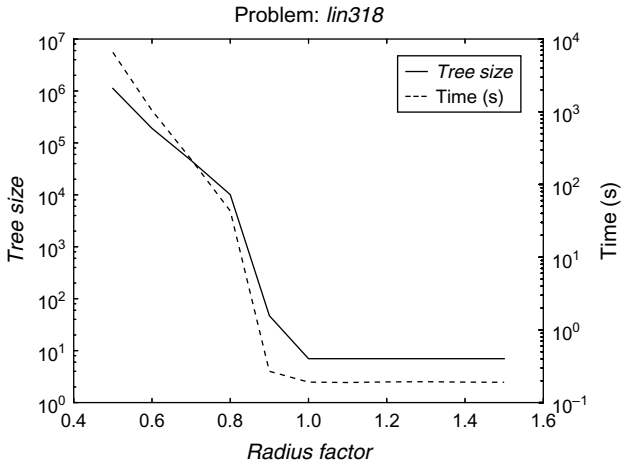

Figure 16 **Tree Size and CPU Time for *lin318***

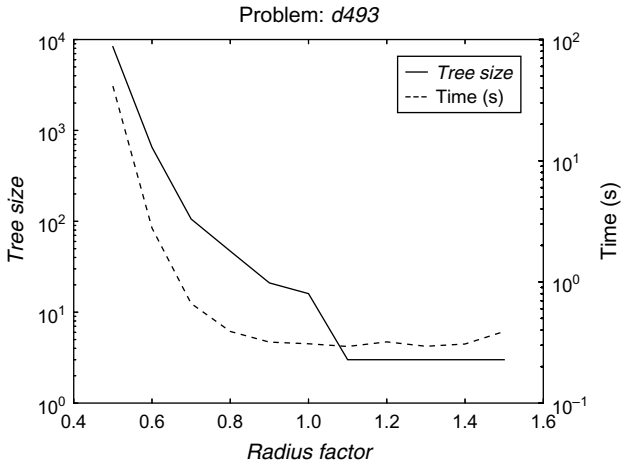

Figure 17 **Tree Size and CPU Time for *d493***

it can be seen, in logarithmic scale, how the tree size (continuous lines) and computing times (dashed lines) decrease for increasing overlap ratios.

5. Conclusions

This work presented a combinatorial branch-and-bound algorithm for the CETSP, where the subproblems solved at each node consists of a SOCP. The proposed algorithm was tested in 824 instances available in the literature, i.e., 720 two-dimensional small sized instances suggested by Behdani and Smith (2014); 62 two-dimensional and 42 three-dimensional instances proposed by Mennell (2009). Our algorithm found optimal solutions for 752 instances, specifically, all 720 instances from Behdani and Smith (2014), 22 two-dimensional and 10 three-dimensional instances from Mennell (2009). The lower bounds of all remaining instances were also improved.

The proposed algorithm performed well particularly on those instances with larger overlap ratios. In practical terms, these types of instances are more closely related to the technological advance. For example, better wireless transmitters and receptors tend to increase the coverage area of the equipment that uses this type of technology, as in the case of AMR and other applications found in the literature, i.e., the radius of each vertex is likely to increase with the development of new technologies.

Promising avenues of research include the development of new approaches to improve the root relaxation, especially for smaller overlap ratios where the problem is closer to the classical TSP.

Supplemental Material

Supplemental material to this paper is available at http://dx.doi.org/10.1287/ijoc.2016.0711.

Acknowledgments

The authors thank Dr. Manuel Iori for valuable comments that helped to improve the quality of the paper and Dr. Behdani for providing the instances generated in Behdani and Smith (2014). The first author received grants from CNPq [Grant no. 202241/2041-9]. The third author received grants from FAPERJ [Grant no. E26-103.206/2011] and CNPq [Grant no. 301278/2013-0]. The last author received grants from CNPq [Grant nos. 471158/2012-7 and 305223/2015-1].

References

Andersen ED, Roos C, Terlaky T (2003) On implementing a primal-dual interior-point method for conic quadratic optimization. *Math. Programming* 95(2):249–277.

Applegate DL, Bixby RE, Chvátal V, Cook WJ (2011) *The Traveling Salesman Problem: A Computational Study*, Princeton Series in Applied Mathematics (Princeton University Press, Princeton, NJ).

Arkin E, Hassin R (1994) Approximation algorithms for the geometric covering salesman problem. *Discrete Appl. Math.* 55(3): 197–218.

Behdani B, Smith JC (2014) An integer-programming-based approach to the close-enough traveling salesman problem. *INFORMS J. Comput.* 26(3):415–432.

Boyd S, Vandenberghe L (2004) *Convex Optimization* (Cambridge University Press, Cambridge, UK).

Dong J, Yang N, Chen M (2007) Heuristic approaches for a TSP variant: the automatic meter reading shortest tour problem. Baker EK, Joseph A, Mehrotra A, Trick MA, eds. *Extending the Horizons: Advances in Computing, Optimization, and Decision Technologies*, Operations Research/Computer Science Interfaces, Vol. 37 (Springer, New York), 145–163.

Farid A, Goldfarb D (2003) Second-order cone programming. *Math. Programming* 95(1):3–51.

Gendreau M, Laporte G, Semet F (1997) The covering tour problem. *Oper. Res.* 45(4):568–576.

Gulczynski DJ, Heath JW, Price CC (2006) The close enough traveling salesman problem: A discussion of several heuristics. Alt FB, Fu MC, Golden B, eds. *Perspectives in Operations Research*, Operations Research/Computer Science Interfaces, Vol. 36 (Springer, New York), 271–283.

Hà MH, Bostel N, Langevin A, Rousseau LM (2014) Solving the close-enough arc routing problem. *Networks* 63(1):107–118.

Lobo M, Vandenberghe L, Boyd S, Lebret H (1998) Applications of second-order cone programming. *Linear Algebra Appl.* 284(1–3): 193–228.

Mennell W (2009) Heuristics for solving three routing problems: Close-enough traveling salesman problem, close-enough vehicle routing problem, sequence-dependent team orienteering problem. Unpublished doctoral dissertation, University of Maryland, College Park.

Mennell W, Golden B, Wasil E (2011) A steiner-zone heuristic for solving the close-enough traveling salesman problem. *12th INFORMS Comput. Soc. Conf.: Oper. Res., Comput., Homeland Defense, Monterey, CA*.

Shuttleworth R, Golden B, Smith S, Wasil E (2008) Advances in meter reading: Heuristic solution of the close enough traveling salesman problem over a street network. Golden B, Raghavan S, Wasil E, eds. *The Vehicle Routing Problem: Latest Advances and New Challenges*, Operations Research/Computer Science Interfaces, Vol. 43 (Springer, New York), 487–501.

Silberholz J, Golden B (2007) The generalized traveling salesman problem: A new genetic algorithm approach. Baker EK, Joseph A, Mehrotra A, Trick MA, eds. *Extending the Horizons: Advances in Computing, Optimization, and Decision Technologies*, Operations Research/Computer Science Interfaces, Vol. 37 (Springer, New York), 165–181.

Yuan B, Orlowska M, Sadiq S (2007) On the optimal robot routing problem in wireless sensor networks. *IEEE Trans. Knowledge Data Engrg.* 19(9):1252–1261.