Efficient Second-Order Cone Programming for the Close Enough Traveling Salesman Problem

Geordan Gutow[1][0000−0002−2137−6280] and Howie Choset[1][0000−0002−2266−8744]

[1]Carnegie Mellon University, Pittsburgh PA 15213, USA

Abstract. The Close Enough Traveling Salesman Problem (CETSP) is a generalization of the Traveling Salesman Problem in which the agent needs only visit a spherical neighborhood surrounding each destination. Prior work has developed an approach that finds globally optimal solutions to instances of the CETSP by solving a sequence of Second-Order Cone Programs (SOCP). We demonstrate it is possible to eliminate 2/3 of the variables and 1/2 of the constraints in these SOCPs, show how to reuse computation and memory allocation across multiple SOCPs in the sequence, and propose a strategy to warm-start the SOCPs using solutions obtained earlier in the sequence. Collectively, these three changes halve the time required to solve 210 random CETSP instances to optimality. We also obtained improved lower bounds on 73 instances from the literature, including solving one instance to optimality for the first time.

Keywords: Traveling Salesman Problems · Convex Programming · Interior Point Methods · Branch and Bound · Motion and Path Planning

1 Introduction

The Traveling Salesman Problem (TSP) is a canonical problem in combinatorial optimization that has been widely studied for both its academic value and its numerous applications [2]. Many variations of the TSP have been proposed to better capture particular problem settings [15], but the essence of the problem is to find the lowest cost sequence of edges (a "tour") in a graph that visits all of the nodes, or in some variations just a subset of the nodes. In the classical TSP, the cost of a tour is the sum of the costs of the edges in the tour and all nodes must be visited. An interesting way to generalize the TSP is to associate a set of waypoints with each node, and allow the edge costs to vary depending on which waypoint is chosen at either end. If there is a discrete set of waypoints for each node, this yields the Generalized Traveling Salesman Problem (GTSP) [19]. If the sets are continuous, this yields the Traveling Salesman Problem with Neighborhoods (TSPN). A particularly well-studied special case of the TSPN restricts the sets (called "neighborhoods") to be spheres and the cost to be the Euclidean distance traveled. This is known as the Close Enough Traveling Salesman Problem (CETSP), as it models a traveling salesman-like scenario where it is sufficient to get "close enough" to each destination. Proposed applications of the CETSP

include aerial reconnaissance [16] and automated meter reading/wireless sensor network monitoring [14, 21].

Prior work on the CETSP can be divided into two categories: heuristic approaches, and exact algorithms. Heuristic approaches like those proposed in [16] rapidly find solutions that are feasible, but do not guarantee optimality and may provide no assessment of solution quality. This second limitation motivated the work in [3], which developed a mixed-integer programming formulation that provides asymptotically tight upper and lower bounds on the optimal cost of a CETSP instance. More recently, [6] and [7] present improved upper and lower bounding algorithms for the CETSP.

The available exact algorithms for the CETSP, which obtain an optimal solution in finite time, are quite limited. To the authors' knowledge there is only the Branch and Bound approach of [9], which was improved upon in [22], and the mixed-integer nonlinear programming approach of [12]. In [9] and [22], the CETSP is solved by searching in a best-first, short to long fashion over possible orders to visit neighborhoods in ("partial tours"). The partial tours are organized in a tree as shown in figure 1. Beginning with a root partial tour, branching occurs by inserting a new neighborhood into each spot in the tour. Bounding is achieved by computing the length of the shortest path that visits the neighborhoods in the order specified by the partial tour; this is guaranteed to be an underestimate of the cost of a complete tour and can be obtained by solving a Second Order Cone Program (SOCP) originally formulated by [16].

Solving these SOCP subproblems for the lower bounds on the complete tour cost makes up the majority of the runtime of the Branch and Bound approach [22]. Solving them quickly is thus very important for the speed of the approach. However, these subproblems have received little attention in the literature; [9] used the formulation of [16] directly, while [22] recognized that the subproblems for child nodes of the same parent share many constraints. In this work, we present three ways to solve the subproblems more efficiently; applying these changes to [22] halves the solution time on 210 randomly generated CETSP instances.

In Section 2 we formally state the Close Enough Traveling Salesman Problem and review the branch and bound algorithm used to solve it in [9, 10, 22]. In Section 3 we show how to *reduce* the size of the SOCP subproblem that must be solved for each node of the branch and bound tree, eliminating 2/3 of the variables and 1/2 of the constraints. Section 4 describes how to *reuse* SOCP solver workspace allocations, constraint matrix construction, and matrix factorization within the branch and bound. Then, in Section 5, we show how to *recycle* the solution to a parent node's subproblem to warm-start the SOCP solver for child node subproblems, at the cost of solving an additional fixed size second order cone program for each child. Section 6 reports an ablation study of the changes proposed in this work, as well as new best known lower bounds on 73 CETSP instances from [22] (including the first optimal solution of one instance). Finally, Section 7 describes future di-

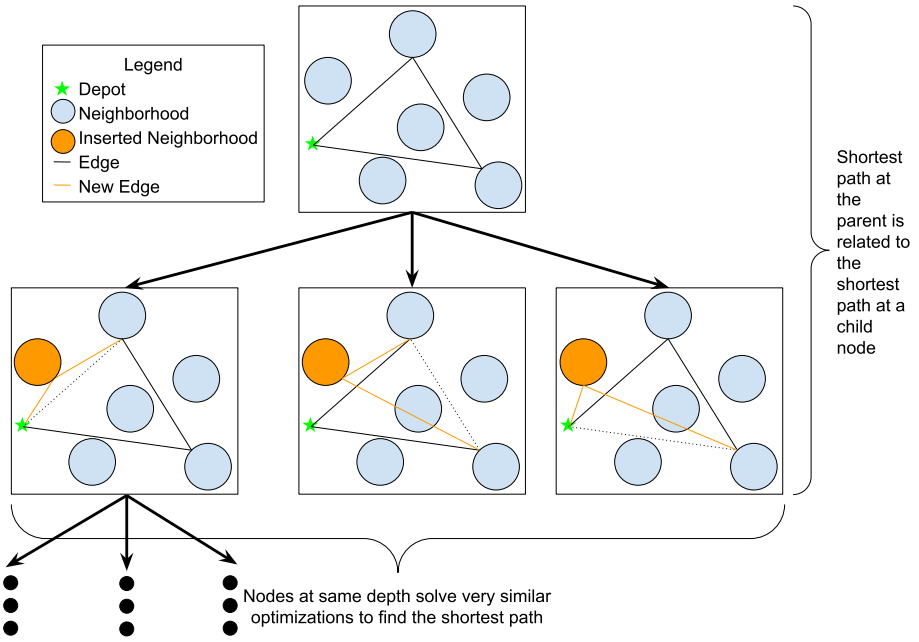

Fig. 1. The branch and bound approach proposed by [9] begins with a partial tour that leaves the depot and visits two other neighborhoods. Then, children are generated by inserting a selected neighborhood at each possible location in the parent partial tour. Observe that the locations of the turning points in the shortest path at the parent are similar to those at the child, and that the child paths are also related to each other. In this work we exploit these similarities to reduce computation time.

rections suggested by the results herein. We release source code and results at `https://github.com/SURI-Shared/BnB_CETSP_CBFS/tree/WAFR24`.

2 Algorithmic and Mathematical Background

The CETSP can be seen as an optimization over *turning points* $\boldsymbol{x}_i \in \mathbb{R}^n$ within each neighborhood and a sequence of neighborhood indices $T = (k_1, \ldots, k_m)$:

$$\begin{aligned} \text{CETSP} \quad \min_{\boldsymbol{x}_{1:m},\, k_{1:m}} \quad & \sum_{i=0}^{m} \|x_{k_{i+1}} - x_{k_i}\| & \\ \text{s.t.} \quad & \|\boldsymbol{c}_i - \boldsymbol{x}_i\| \leq r_i, & i = 1, \ldots, m, \\ & k_i = k_j \iff i = j, & \\ & 1 \leq j \leq m \implies \exists i : k_i = j, & \\ & \boldsymbol{x}_i \in \mathbb{R}^n, k_i \in \mathbb{N} \setminus \{0\} & \end{aligned} \tag{1}$$

where $\boldsymbol{x}_0 = \boldsymbol{x}_{m+1}$ are the coordinates of the depot, which the agent must start and end at. $\boldsymbol{c}_i$ and r_i are the center and radius of the ith neighborhood, respectively. The cost is the Euclidean length of the path between turning points in the order specified by the sequence, and the constraints enforce that each turning point is inside its neighborhood, that each neighborhood is in the sequence no more than once (note that this does not in principle prevent the path from intersecting a neighborhood multiple times), and that each neighborhood is visited at least once. A feasible solution to CETSP is called a tour.

CETSP can be solved to global optimality in finite time [9] by a branch and bound search of a tree whose nodes each correspond to a **partial tour**: a sequence of neighborhoods that begins and ends at the depot but does not necessarily include all neighborhoods[1]. The **children** of a node in the tree are all partial tours containing one new neighborhood inserted into the partial tour. The **lower bound** of a node is the length of the shortest path that intersects the neighborhoods of the partial tour in the specified order. In [9] it was shown that this lower bound is the optimal cost of a second order cone program; in this work we refer to the task of computing the lower bound for a node as the node's **subproblem**.

Second order cone programs are a special case of a class of convex optimization problems called cone programs [5]. Following [13], a generic cone program is specified by its positive semi-definite quadratic cost matrix $\underline{Q}$, linear cost vector $\boldsymbol{q}$, constraint matrix $\underline{A}$, constraint right-hand-side vector $\boldsymbol{b}$, and a composite cone $\mathbb{K}$. A composite cone is the Cartesian product of several primitive convex cones, such as second order cones or the non-negative orthant. The standard form is then:

$$\begin{aligned} \min_{\boldsymbol{x},\,\boldsymbol{s}} \quad & \frac{1}{2}\boldsymbol{x}^T\underline{Q}\boldsymbol{x} + \boldsymbol{q}^T\boldsymbol{x} \\ \text{s.t.} \quad & \underline{A}\boldsymbol{x} + \boldsymbol{s} = \boldsymbol{b}, \\ & \boldsymbol{s} \in \mathbb{K} \end{aligned} \tag{2}$$

and the dual is:

$$\begin{aligned} \max_{\boldsymbol{x},\,\boldsymbol{z}} \quad & -\frac{1}{2}\boldsymbol{x}^T\underline{Q}\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{z} \\ \text{s.t.} \quad & \underline{Q}\boldsymbol{x} + \underline{A}^T\boldsymbol{z} = -\boldsymbol{q}, \\ & \boldsymbol{z} \in \mathbb{K}^* \end{aligned} \tag{3}$$

Here, $\boldsymbol{x} \in \mathbb{R}^{n_p}$ are the primal decision variables, $\boldsymbol{s} \in \mathbb{R}^{n_c}$ are the slack variables, $\boldsymbol{z} \in \mathbb{R}^{n_c}$ are the dual variables, and $\mathbb{K}^*$ is the dual cone of $\mathbb{K}$. Note that the second order cone is self-dual, and so if the composite cone contains only second order cones the primal and dual cones are identical.

[1] These nodes are not the same as the nodes of the TSP graph in the introduction; rather each node here corresponds to a path in the TSP graph

3 Reduced Second-Order Cone Program

In [9] the lower bound of a node was computed via second order cone programming using a formulation initially proposed in [16] for the touring Steiner zones problem with a fixed visit sequence; the same approach was taken in [10] and [22]. The original formulation is given below:

$$\begin{aligned}
\text{MSOCP} \quad \min_{f_{0:m},\, \boldsymbol{x}_{1:m},\, \boldsymbol{w}_{0:m},\, \boldsymbol{v}_{1:m}} \quad & \sum_{i=0}^{m} f_i \\
\text{s.t.} \quad & \boldsymbol{x}_i - \boldsymbol{x}_{i+1} + \boldsymbol{w}_i = 0, && i = 0, \ldots, m, \\
& \|\boldsymbol{w}_i\| - f_i \leq 0, && i = 0, \ldots, m, \\
& \boldsymbol{x}_i + \boldsymbol{v}_i = \boldsymbol{c}_i, && i = 1, \ldots, m, \\
& \|\boldsymbol{v}_i\| \leq r_i, && i = 1, \ldots, m, \\
& f_i \in \mathbb{R}, \boldsymbol{x}_i \in \mathbb{R}^n, \boldsymbol{w}_i \in \mathbb{R}^n, \boldsymbol{v}_i \in \mathbb{R}^n
\end{aligned} \tag{4}$$

where $\boldsymbol{x}_0$ and $\boldsymbol{x}_{m+1}$ are the depot location, $\boldsymbol{c}_i, r_i$ are the center and radius of the ith neighborhood in the partial tour. $\boldsymbol{x}_i$ is the turning point in the ith neighborhood, while f_i is the distance between successive turning points. $\boldsymbol{w}_i$ and $\boldsymbol{v}_i$ are auxiliary variables encoding the displacement between turning points and the displacement from the neighborhood center, respectively. MSOCP has $m + 1 + nm + n(m + 1) + nm = (3m + 1)n + m + 1$ decision variables. If MSOCP's constraints are converted to standard form, the constraint matrix $\underline{A}$ is $\{(4n+2)m+2n+1\} \times \{(3n+1)m+n+1\}$, with $(7n+1)m+2n+1$ non-zero entries (all ± 1).

The auxiliary $\boldsymbol{w}_i$ and $\boldsymbol{v}_i$ variables are redundant. The following reduced formulation (RSOCP), which requires only $(n+1)m+1$ decision variables, is equivalent to MSOCP:

$$\begin{aligned}
\text{RSOCP} \quad \min_{f_{0:m},\, \boldsymbol{x}_{1:m}} \quad & \sum_{i=0}^{m} f_i \\
\text{s.t.} \quad & \|\boldsymbol{x}_{i+1} - \boldsymbol{x}_i\| - f_i \leq 0, && i = 0, \ldots, m, \\
& \|\boldsymbol{c}_i - \boldsymbol{x}_i\| \leq r_i, && i = 1, \ldots, m, \\
& f_i \in \mathbb{R}, \boldsymbol{x}_i \in \mathbb{R}^n
\end{aligned} \tag{5}$$

In standard form, RSOCP's constraint matrix is $\{(2n + 2)m + n + 1\} \times \{(n + 1)m+1\}$ and has only $(3n+1)m+1$ non-zeros (all ± 1). Thus, as m and $n \to \infty$ the number of constraint rows and non-zeros approaches $1/2$ that of MSOCP while the number of variables approaches $1/3$. Additionally, RSOCP involves only second order cones, while MSOCP also needs the zero-cone for its equality constraints. The composite cone for RSOCP is thus self-dual, while that for MSOCP is not.

Commercial presolver routines can recover most of the benefit of RSOCP. The existing branch and bound implementations [9, 22] use CPLEX to solve

MSOCP. The exact transformations used by commercial presolvers are proprietary; table 1 shows the reported rows and columns of MSOCP after presolve for CPLEX 22.1.1 and Gurobi 10.0.3 (another commercial solver capable of addressing MSOCP). Both presolvers reduce the problem size to approximately that of RSOCP, though the rows and columns are switched. The extra column appears to be aggregating the f_i into a single term for the objective. However, warm-starting as in Section 5 requires switching from CPLEX to an open source solver. The selected solver, Clarabel [13], does possess a presolve, but it is currently limited to removing constraint rows with infinite bounds. It was also necessary to disable the presolver to permit both solver reuse (Section 4) and warm-starting (Section 5). Thus, we expect to see a benefit from switching to RSOCP (Section 6 confirms this). Additionally, we found that Clarabel, even solving MSOCP *without* presolve, was faster than CPLEX solving MSOCP *with* presolve (see figure 2), so switching from CPLEX to Clarabel provides a performance improvement even without the changes proposed in this work. We account for this in our results by using Clarabel, solving MSOCP, as our baseline rather CPLEX.

Table 1. The size of the problem actually solved by CPLEX 22.1.1 and Gurobi 10.9.3 when presented with MSOCP, for a $n = 2$ and a range of m, is very similar to RSOCP. Note the transpose of the matrix after presolve by both Gurobi and CPLEX.

		MSOCP		RSOCP		CPLEX		Gurobi	
m	n	Rows	Cols	Rows	Cols	Rows	Cols	Rows	Cols
10	2	105	73	63	31	32	63	32	63
50	2	505	353	303	151	152	303	152	303
100	2	1005	703	603	301	302	603	302	603

4 Reusing Conic Solver Work within Branch and Bound

Let $\underline{0}$ be a square matrix with all zero entries. Let $\mathbb{I}_k$ be a $k \times k$ identity matrix. Let $\mathbf{0}_k$ be a length k vector of zeros. Let $\mathbf{1}_k$ be a length k vector of ones. Writing RSOCP in standard form with decision vector $[\boldsymbol{f}^T, \boldsymbol{x}_1^T, \ldots, \boldsymbol{x}_m^T]$ yields $\underline{Q} = \underline{0}$,

$\boldsymbol{q} = [\mathbf{1}_{m+1}^T, \mathbf{0}_{nm}^T]^T$, and:

$$A = \begin{bmatrix}
0 & 0 & \dots & 0 & 0 & \mathbf{0}_n^T & \mathbf{0}_n^T & \dots & \mathbf{0}_n^T & \mathbf{0}_n^T \\
0 & 0 & \dots & 0 & 0 & \mathbb{I}_n & \underline{0}_n & \dots & \underline{0}_n & \underline{0}_n \\
0 & 0 & \dots & 0 & 0 & \mathbf{0}_n^T & \mathbf{0}_n^T & \dots & \mathbf{0}_n^T & \mathbf{0}_n^T \\
0 & 0 & \dots & 0 & 0 & \underline{0}_n & \mathbb{I}_n & \dots & \underline{0}_n & \underline{0}_n \\
 & & & & & \vdots & & & & \\
0 & 0 & \dots & 0 & 0 & \mathbf{0}_n^T & \mathbf{0}_n^T & \dots & \mathbf{0}_n^T & \mathbf{0}_n^T \\
0 & 0 & \dots & 0 & 0 & \underline{0}_n & \underline{0}_n & \dots & \underline{0}_n & \mathbb{I}_n \\
-1 & 0 & \dots & 0 & 0 & \mathbf{0}_n^T & \mathbf{0}_n^T & \dots & \mathbf{0}_n^T & \mathbf{0}_n^T \\
0 & 0 & \dots & 0 & 0 & -\mathbb{I}_n & \underline{0}_n & \dots & \underline{0}_n & \underline{0}_n \\
0 & -1 & \dots & 0 & 0 & \mathbf{0}_n^T & \mathbf{0}_n^T & \dots & \mathbf{0}_n^T & \mathbf{0}_n^T \\
0 & 0 & \dots & 0 & 0 & \mathbb{I}_n & -\mathbb{I}_n & \dots & \underline{0}_n & \underline{0}_n \\
 & & & & & \vdots & & & & \\
0 & 0 & \dots & -1 & 0 & \mathbf{0}_n^T & \mathbf{0}_n^T & \dots & \mathbf{0}_n^T & \mathbf{0}_n^T \\
0 & 0 & \dots & 0 & 0 & \underline{0}_n & \underline{0}_n & \dots & \mathbb{I}_n & -\mathbb{I}_n \\
0 & 0 & \dots & 0 & -1 & \mathbf{0}_n^T & \mathbf{0}_n^T & \dots & \mathbf{0}_n^T & \mathbf{0}_n^T \\
0 & 0 & \dots & 0 & 0 & \underline{0}_n & \underline{0}_n & \dots & \underline{0}_n & \mathbb{I}_n
\end{bmatrix} \boldsymbol{b} = \begin{bmatrix}
r_1 \\ \boldsymbol{c}_1 \\ r_2 \\ \boldsymbol{c}_2 \\ \vdots \\ r_m \\ \boldsymbol{c}_m \\ 0 \\ \boldsymbol{x}_0 \\ 0 \\ \mathbf{0}_n \\ \vdots \\ 0 \\ \mathbf{0}_n \\ 0 \\ -\boldsymbol{x}_0
\end{bmatrix} \mathbb{K} = \begin{bmatrix}
SOC_{n+1} \\ SOC_{n+1} \\ \vdots \\ SOC_{n+1} \\ SOC_{n+1} \\ SOC_{n+1} \\ \vdots \\ SOC_{n+1} \\ SOC_{n+1}
\end{bmatrix} \quad (6)$$

where SOC_{n+1} is the second order cone in $n+1$ dimensions. Recall that $\boldsymbol{x}_0$ is the depot location. Observe that $\underline{Q}$, $\boldsymbol{q}$, $\underline{A}$, and $\mathbb{K}$ do not depend on either the instance data (neighborhood centers/radii) OR the order of neighborhoods in the partial sequence; only $\boldsymbol{b}$ does. Thus, for a fixed sequence length $\underline{Q}$, $\boldsymbol{q}$, $\underline{A}$, and $\mathbb{K}$ are all identical, while $\boldsymbol{b}$, the decision vector, the slack vector, and the dual vector are all constant size. At every node of equal depth in the branch and bound tree, only $\boldsymbol{b}$ needs to update. $\underline{Q}$, $\boldsymbol{q}$, $\underline{A}$, and $\mathbb{K}$ can be constructed once for each length of partial sequence, and internal solver workspace allocations can be reused. We show this provides a performance benefit in Section 6.

Some cone programming algorithms, like the Douglas-Rachford splitting implemented for SCS [18], could reuse more than just the cost/constraint specification and solver workspace. This algorithm requires factoring a matrix that depends only on $\underline{Q}$ and $\underline{A}$; this factorization is therefore reusable across subproblems with the same length partial sequence. Recent work [4] has even found success in computing these factorizations offline and storing them for scenarios where $\underline{Q}$ and $\underline{A}$ are known ahead of time, which is true for CETSP subproblems. Implementing and benchmarking factorization reuse is left to future work.

5 Recycling Parent Node Solutions for Warm-Starting

At a child node in the branch and bound, the turning points in neighborhoods that were also present in the parent node are often close to their locations at the parent node. This suggests that it should be possible to exploit the solution of the parent node's subproblem to warm-start the solver for the child node. Two difficulties arise. First, the child node subproblem has a new turning point for the inserted neighborhood, two new edge length variables, and associated

constraints, for which the parent does not provide a primal, dual, or slack guess. Second, if an interior point method is used to solve RSOCP, the parent solution is likely a poor initial point as it is on the boundary of the feasible region [17].

Let (P) denote RSOCP specialized to the parent node. Let (C) denote RSOCP specialized to the child node. For the primal variables present in both child and parent, let $\boldsymbol{p}'$ denote their optimal values in (P) and $\boldsymbol{p}^*$ their optimal values in (C). Warm-starting from the parent solution relies on the assumption that $\boldsymbol{p}' \approx \boldsymbol{p}^*$. If we restrict $\boldsymbol{p}^* = \boldsymbol{p}'$, (C) reduces to the following "insertion problem:"

$$\begin{aligned} \text{ISOCP} \quad \min_{f_a, f_b, \boldsymbol{x}_i} \quad & f_a + f_b \\ \text{s.t.} \quad & \|\boldsymbol{x}_i - \boldsymbol{p}_a\| - f_a \leq 0, \\ & \|\boldsymbol{p}_b - \boldsymbol{x}_i\| - f_b \leq 0, \\ & \|\boldsymbol{c}_i - \boldsymbol{x}_i\| \leq r_i, \\ & f_a, f_b \in \mathbb{R}, \boldsymbol{x}_i \in \mathbb{R}^n \end{aligned} \tag{7}$$

where $\boldsymbol{x}_i$ is the (guess at) the turning point in the inserted neighborhood, $\boldsymbol{c}_i$ is the center of the inserted neighborhood, r_i is the radius of the inserted neighborhood, $\boldsymbol{p}_a$ is the optimal turning point in (P) for the neighborhood preceding i, and $\boldsymbol{p}_b$ is the optimal turning point in (P) for the neighborhood succeeding i. The optimal primal, dual, and slack variables of ISOCP provide initial guesses for the new primal, dual, and slack variables in (C). ISOCP has only $2+n$ decision variables and so solving it is extremely cheap, especially in comparison to RSOCP for large m. Additionally, for ISOCP, the only thing that changes when specializing to a new parent-child pair is which radius, neighborhood center, and parent node turning points appear in the right-hand side constraint vector. Thus, we can reuse work between all ISOCP solves in the same way we reuse work between RSOCP solves of equal length partial tours.

The discussion in sections 3 and 4, as well as the initial guess construction above, are agnostic to the algorithm actually used to solve RSOCP and ISOCP. However, actually using an initial guess at a solution to RSOCP depends on the choice of algorithm. The most popular class of algorithm for SOCP is the interior point method, variants of which are implemented in (among others) Gurobi, CPLEX, and Clarabel [13]. A major limitation when providing an initial guess to interior point methods is that, even if the guess is close to the optimal solution, if it is also close to the boundary of the feasible region the algorithm may require numerous very small steps to finish converging [17, 11, 20, 8]. To mitigate this, we adopt the modified first interior point iteration implemented in [13]. If the affine step size α_{aff} is less than one at the first iteration (which it often is because of the badly centered initial guess), scale the affine step by α_{aff} when computing the combined step. For further details see [13].

6 Numerical Results and Discussion

We adapt the branch and bound implementation of [22], specifically the `Cout+` variant; the source code was obtained from `https://github.com/UranusR/BnB_`

`CETSP_CBFS`. Our enhanced code is available at `https://github.com/SURI-Shared/BnB_CETSP_CBFS/tree/WAFR24`. `Cout+` uses CPLEX 22.1.1 to solve instances of MSOCP, but CPLEX is closed-source. To implement the warm-start, we need to replace CPLEX with the recent open source Clarabel solver [13], to which we could make the necessary small modifications. Adopting the naming conventions of [22], replacing CPLEX with Clarabel v0.7.1 but making no other changes yields `Cout+Clarabel`. Clarabel turns out to be much faster for this application than CPLEX, as shown in figure 2. We generated 30 CETSP instances with 5, 10, 15, 20, 25, 30, and 35 neighborhoods (plus the depot) using the standard Behdani method [3]. For `Cout+Clarabel`, the cumulative solve time over all 210 instances was 55% of that required by `Cout+`. This improvement is particularly notable as `Cout+Clarabel` has no presolve and so is actually solving MSOCP. Accordingly, we benchmark the impact of our proposed changes against `Cout+Clarabel` instead of `Cout+`. Timing results are for an Intel i7-11800H CPU with 16 GB of RAM.

We show the cumulative impact of our proposed changes in figure 3. 5 neighborhood instances are omitted as the small CPU times ($<$ 3 milliseconds) on these problems leads to extreme variability in relative performance. The most important change is replacing MSOCP with RSOCP, labeled "Reduce" in the figure; this saves around 40% of `Cout+Clarabel`'s CPU time at all problem sizes. Reusing the solver workspace as detailed in Section 4 is also consistently beneficial. By contrast, warm-starting by solving ISOCP increases the geometric mean runtime for 10 and 15 neighborhoods; solve time does decrease at 20 or more neighborhoods. For small numbers of neighborhoods solving ISOCP is expensive compared to the cost of a few iterations on RSOCP, so it is not surprising that warm-starting in this way is not beneficial for small problems.

We ran `Cout+Clarabel+Reduce+Reuse+Recycle` on 152 problem instances from [22], using an Intel i9-10980XE CPU with a time limit of 14400 seconds and a memory limit of 16GB. We increased the memory limit from the 8GB permitted in [22] as we observed that all but three instances that were unsolved by `Cout+` in [22] were due to exceeding the memory limit rather than the time limit (concentricCircles2, concentricCircles3, and kroD100 with overlap ratio 0.02). `Cout+Clarabel+Reduce+Reuse+Recycle` also requires slightly more memory than `Cout+` as it must retain the primal dual and slack solutions for nodes with unexpanded children, as well as the reusable solver workspaces. With our improved algorithm, we solve one, previously time limited, instance to optimality for the first time (concentricCircles2, 37 neighborhoods); 10,232 seconds and 1.4GB of RAM were required for this instance. rd400 and d493 with overlap ratio 0.02 exhaust the time limit; all others are optimal or memory limited. We report new best lower bounds in Table 2; comprehensive results are available at `https://github.com/SURI-Shared/BnB_CETSP_CBFS/tree/WAFR24/BnB_CETSP/Results/16GB`. OR stands for overlap ratio; 3D indicates this is the 3D version of the instance. For the Behdani instances (CETSP-size-number), variants with neighborhood radius 0.25, 0.5, and 1.0 are all tested.

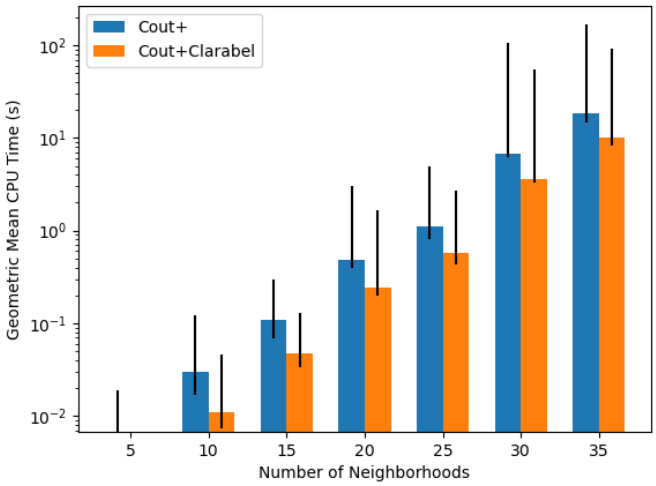

Fig. 2. CPU time for branch and bound to find an optimal solution with CPLEX (`Cout+`) and with Clarabel (`Cout+Clarabel`) using MSOCP. Bar height is the geometric mean over 30 random Behdani instances of the specified size. The vertical line shows the minimum and maximum time for any instance of that size. Note that the vertical axis is logarithmic.

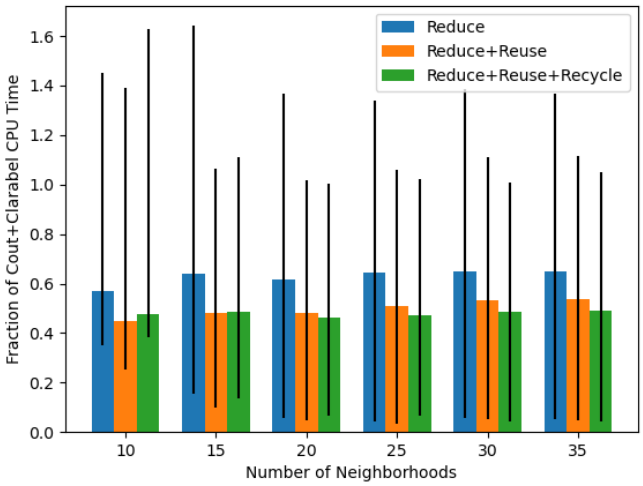

Fig. 3. The bars report the geometric mean of the CPU time for the algorithm variant on 30 random Behdani instances of the specified size, divided by the geometric mean of the CPU time for `Cout+Clarabel` on the same instances. The vertical line shows the minimum and maximum ratio of algorithm variant CPU time to `Cout+Clarabel` CPU time for any instance of that size.

Table 2: Instances from [22] for which `Cout+Clarabel+Reduce+Reuse+Recycle` obtained a new best known lower bound, with a time limit of 14400 seconds and a memory limit of 16GB.

Instance	Old Best	Old LB	New LB	CPU Time (s)	Memory (kB)	Optimal?
bubbles4	Cout	696.36	696.49	3188	16777304	No
bubbles5	Cout	852.80	857.76	2241	16777352	No
bubbles6	Cout	992.84	999.50	2038	16777396	No
bubbles7	Cout	1118.81	1129.46	2168	16777232	No
bubbles8	Cout	1245.88	1257.38	2210	16777380	No
bubbles9	Cout	1367.00	1379.60	2181	16777328	No
concentricCircles2	Cout+	152.29	153.13	10232	1375724	**Yes**
concentricCircles3	Cout+	251.52	252.89	11034	16777288	No
concentricCircles5	Cout+	466.62	468.58	3371	16777396	No
rotatingDiamonds3	Cout	353.85	356.29	4009	16777404	No
rotatingDiamonds4	Cout+	594.10	600.01	2570	16777284	No
rotatingDiamonds5	Cout	1096.71	1111.27	2395	16777244	No
d493, 3D, OR 0.02	Cout+	146.01	491.15	14400	15050448	No
dsj1000, 3D, OR 0.02	Cout	554.18	803.56	5704	16777228	No
kroD100, 3D, OR 0.02	Cout	146.50	164.45	5139	16777436	No
lin318, 3D, OR 0.02	Cout	1998.34	2155.53	4281	16777472	No
pcb442, 3D, OR 0.02	Cout	186.24	201.06	5344	16777448	No
rat195, 3D, OR 0.02	Cout+	109.04	139.45	4549	16777448	No
rd400, 3D, OR 0.02	Cout	568.90	905.69	14400	10140604	No
pcb442, 3D, OR 0.1	Cout	137.37	189.43	4147	16777408	No
rd400, 3D, OR 0.1	Cout	433.73	893.18	4436	16777220	No
CETSP-150-1, R0.25	Cout+	71.34	71.62	3338	16777380	No
CETSP-150-10, R0.25	Cout+	67.47	67.75	3455	16777456	No
CETSP-150-2, R0.25	Cout+	70.32	70.66	3151	16777380	No
CETSP-150-3, R0.25	Cout+	66.77	67.10	3352	16777236	No
CETSP-150-4, R0.25	Cout+	70.50	70.84	3327	16777440	No
CETSP-150-5, R0.25	Cout+	70.13	70.45	3213	16777400	No
CETSP-150-6, R0.25	Cout+	69.64	69.93	3402	16777448	No
CETSP-150-7, R0.25	Cout+	68.68	68.96	3396	16777336	No
CETSP-150-8, R0.25	Cout+	66.77	67.05	3301	16777360	No
CETSP-150-9, R0.25	Cout+	70.11	70.42	3340	16777392	No
CETSP-100-1, R0.5	Cout+	64.25	64.30	2656	16777476	No
CETSP-100-10, R0.5	Cout+	64.83	64.91	2504	16777340	No
CETSP-100-2, R0.5	Cout+	64.41	64.51	2418	16777264	No
CETSP-100-3, R0.5	Cout+	63.22	63.36	2376	16777448	No
CETSP-100-4, R0.5	Cout+	59.87	59.95	2474	16777240	No
CETSP-100-6, R0.5	Cout+	64.41	64.48	2494	16777404	No
CETSP-100-7, R0.5	Cout+	63.91	64.00	2520	16777468	No
CETSP-100-8, R0.5	Cout+	62.70	62.81	2404	16777260	No
CETSP-100-9, R0.5	Cout+	62.73	62.80	2548	16777336	No

CETSP-150-1, R0.5	Cout+	65.50	65.73	2456	16777260	No
CETSP-150-10, R0.5	Cout+	62.18	62.40	2545	16777300	No
CETSP-150-2, R0.5	Cout+	65.31	65.50	2305	16777404	No
CETSP-150-3, R0.5	Cout+	61.70	61.96	2428	16777308	No
CETSP-150-4, R0.5	Cout+	65.63	65.83	2353	16777340	No
CETSP-150-5, R0.5	Cout+	65.27	65.46	2407	16777356	No
CETSP-150-6, R0.5	Cout+	64.94	65.22	2518	16777272	No
CETSP-150-7, R0.5	Cout+	63.62	63.81	2399	16777356	No
CETSP-150-8, R0.5	Cout+	62.13	62.34	2409	16777280	No
CETSP-150-9, R0.5	Cout+	65.14	65.35	2350	16777264	No
CETSP-150-1, R1	Cout+	55.82	55.96	1849	16777272	No
CETSP-150-10, R1	Cout+	51.49	51.56	1953	16777248	No
CETSP-150-3, R1	Cout+	52.15	52.37	1824	16777292	No
CETSP-150-4, R1	Cout+	55.59	55.74	1870	16777224	No
CETSP-150-5, R1	Cout+	55.54	55.73	1848	16777248	No
CETSP-150-6, R1	Cout+	55.60	55.80	1800	16777292	No
bonus1000, 3D	Cout	464.71	471.64	1903	16777612	No
bonus1000rdmRad, 3D	Cout+	612.78	627.18	2997	16777280	No
d493rdmRad, 3D	Cout+	444.04	448.39	2187	16777228	No
dsj1000rdmRad, 3D	Cout+	704.03	716.67	1279	16777268	No
lin318rdmRad, 3D	Cout+	1856.27	1876.68	1721	16777276	No
pcb442rdmRad, 3D	Cout+	180.15	181.88	1219	16777472	No
rd400rdmRad, 3D	Cout+	896.19	911.25	4623	16777384	No
team1_100, 3D	Cout	708.29	708.30	3236	16777252	No
team1_100rdmRad, 3D	Cout+	764.37	764.42	2937	16777324	No
team2_200rdmRad, 3D	Cout+	553.84	556.23	3112	16777444	No
team3_300, 3D	Cout	771.77	780.29	3833	16777252	No
team3_300rdmRad, 3D	Cout+	695.58	706.70	1532	16777428	No
team4_400, 3D	Cout	510.07	513.62	2715	16777324	No
team4_400rdmRad, 3D	Cout+	575.32	582.09	4022	16777312	No
team5_499, 3D	Cout	714.33	725.16	4863	16777352	No
team5_499rdmRad, 3D	Cout+	605.46	615.47	1038	16777228	No
team6_500rdmRad, 3D	Cout+	529.74	536.41	1762	16777404	No

7 Conclusion

This work has presented three alterations to the solution of Second-Order Cone subproblems in the state of the art branch and bound approach to the Close Enough Traveling Salesman Problem. By reducing the size of the optimization problem, reusing solver workspace, and warm-starting we obtain approximately a factor of two improvement in solution time across a range of instances, including solving one instance originally from [16] to optimality for the first time. Reducing the problem formulation appears to have been the most impactful, which is consistent with the experience in the mixed integer programming setting that "presolve [is]...highly important" [1]. Commercial presolvers appear to

be effective in this setting; Clarabel and other open-source solvers would benefit from integrating more sophisticated presolve strategies.

The choice of an interior point method to solve RSOCP limited how much solver work could be reused and resulted in only small speed-ups from warm-starting. A first order cone programming algorithm could reuse matrix factorizations between nodes at the same depth in the branch and bound tree, and may be easier to warm-start with a recycled parent solution. Recent work has even had success precomputing matrix factorizations for first order methods in the quadratic programming setting [4]; future work should investigate replacing Clarabel with, e.g., SCS [18] to evaluate the tradeoffs.

In the vast majority of cases which we could not solve to optimality, we reach the memory limit prior to the time limit. This suggests three possible future directions: first, using memory efficient strategies to explore the branch and bound tree such as the CBFS approach in [22]; second, developing strategies for efficient swapping to and from disk to permit much higher memory limits; third, identifying alternative subproblems (viewed broadly as combinations of restrictions and relaxations to the full CETSP) that can prune more of the tree.

Acknowledgments. Geordan Gutow acknowledges the support of an Intelligence Community Postdoctoral Fellowship. This work was supported by AFRL and the AFOSR.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Achterberg, T., Wunderling, R.: Mixed Integer Programming: Analyzing 12 Years of Progress, pp. 449–481. Springer Berlin Heidelberg, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38189-8˙18, https://doi.org/10.1007/978-3-642-38189-8_18
2. Applegate, D.L.: The traveling salesman problem: a computational study, vol. 17. Princeton university press (2006)
3. Behdani, B., Smith, J.C.: An integer-programming-based approach to the close-enough traveling salesman problem. INFORMS Journal on Computing **26**(3), 415–432 (2014)
4. Bishop, A.L., Zhang, J.Z., Gurumurthy, S., Tracy, K., Manchester, Z.: Relu-qp: A gpu-accelerated quadratic programming solver for model-predictive control (2023)
5. Boyd, S.P., Vandenberghe, L.: Convex optimization. Cambridge university press (2004)
6. Carrabs, F., Cerrone, C., Cerulli, R., D'Ambrosio, C.: Improved upper and lower bounds for the close enough traveling salesman problem. In: Green, Pervasive, and Cloud Computing: 12th International Conference, GPC 2017, Cetara, Italy, May 11-14, 2017, Proceedings 12. pp. 165–177. Springer (2017)
7. Carrabs, F., Cerrone, C., Cerulli, R., Golden, B.: An adaptive heuristic approach to compute upper and lower bounds for the close-enough traveling salesman problem. INFORMS Journal on Computing **32**(4), 1030–1048 (2020)
8. Cay, S.B., Pólik, I., Terlaky, T.: Warm-start of interior point methods for second order cone optimization via rounding over optimal jordan frames. Tech. Rep. 7T-006, Department of Industrial and Systems Engineering, Lehigh University (2017)
9. Coutinho, W.P., Nascimento, R.Q.d., Pessoa, A.A., Subramanian, A.: A branch-and-bound algorithm for the close-enough traveling salesman problem. INFORMS Journal on Computing **28**(4), 752–765 (2016). https://doi.org/10.1287/ijoc.2016.0711, https://doi.org/10.1287/ijoc.2016.0711
10. Deckerová, J.: Generalized routing problems with continuous neighborhoods. Master's thesis, Ceské vysoké ucení technické v Praze (2021)
11. Engau, A., Anjos, M.F., Vannelli, A.: On interior-point warmstarts for linear and combinatorial optimization. SIAM Journal on Optimization **20**(4), 1828–1861 (2010). https://doi.org/10.1137/080742786, https://doi.org/10.1137/080742786
12. Gentilini, I., Margot, F., Shimada, K.: The travelling salesman problem with neighbourhoods: Minlp solution. Optimization Methods and Software **28**(2), 364–378 (2013)
13. Goulart, P.J., Chen, Y.: Clarabel: An interior-point solver for conic programs with quadratic objectives (2024)
14. Gulczynski, D.J., Heath, J.W., Price, C.C.: The close enough traveling salesman problem: A discussion of several heuristics. Perspectives in Operations Research: Papers in Honor of Saul Gass' 80 th Birthday pp. 271–283 (2006)
15. Gutin, G., Punnen, A.P.: The traveling salesman problem and its variations, vol. 12. Springer Science & Business Media (2006)
16. Mennell, W.K.: Heuristics for solving three routing problems: Close-enough traveling salesman problem, close-enough vehicle routing problem, and sequence-dependent team orienteering problem. Ph.D. thesis, University of Maryland (2009), https://www.proquest.com/dissertations-theses/

heuristics-solving-three-routing-problems-close/docview/275627168/se-2
17. Nocedal, J., Wright, S.J.: Numerical Optimization. Springer, 2 edn. (2006)
18. O'Donoghue, B.: Operator splitting for a homogeneous embedding of the linear complementarity problem. SIAM Journal on Optimization **31**(3), 1999–2023 (2021). https://doi.org/10.1137/20M1366307, https://doi.org/10.1137/20M1366307
19. Pop, P.C., Cosma, O., Sabo, C., Sitar, C.P.: A comprehensive survey on the generalized traveling salesman problem. European Journal of Operational Research **314**(3), 819–835 (2024). https://doi.org/https://doi.org/10.1016/j.ejor.2023.07.022, https://www.sciencedirect.com/science/article/pii/S0377221723005581
20. Skajaa, A., Andersen, E.D., Ye, Y.: Warmstarting the homogeneous and self-dual interior point method for linear and conic quadratic problems. Mathematical Programming Computation **5**, 1–25 (2013)
21. Yuan, B., Orlowska, M., Sadiq, S.: On the optimal robot routing problem in wireless sensor networks. IEEE Transactions on Knowledge and Data Engineering **19**(9), 1252–1261 (2007). https://doi.org/10.1109/TKDE.2007.1062
22. Zhang, W., Sauppe, J.J., Jacobson, S.H.: Results for the close-enough traveling salesman problem with a branch-and-bound algorithm. Computational Optimization and Applications **85**(2), 369–407 (2023)