Carnegie Mellon University
Research Showcase @ CMU

Tepper School of Business

7-13-2011

The Traveling Salesman Problem with Neighborhoods: MINLP solution

Iacopo Gentilini
Carnegie Mellon University, gentilini@cmu.edu

François Margot
Carnegie Mellon University, fmargot@andrew.cmu.edu

Kenji Shimada
Carnegie Mellon University, shimada@cmu.edu

Follow this and additional works at: http://repository.cmu.edu/tepper

Part of the Business Commons

This Working Paper is brought to you for free and open access by Research Showcase @ CMU. It has been accepted for inclusion in Tepper School of Business by an authorized administrator of Research Showcase @ CMU. For more information, please contact research-showcase@andrew.cmu.edu.

The Traveling Salesman Problem with Neighborhoods: MINLP solution

Iacopo Gentilini[1,2], François Margot[3], and Kenji Shimada[1]

July 13, 2011

Abstract

The traveling salesman problem with neighborhoods extends the traveling salesman problem to the case where each vertex of the tour is allowed to move in a given region. This NP-hard optimization problem has recently received increasing attention in several technical fields such as robotics, unmanned aerial vehicles, or utility management. In this paper, the problem is formulated as a nonconvex Mixed-Integer NonLinear Program (MINLP) having the property that fixing all the integer variables to any integer values yields a convex nonlinear program. This property is used to modify the global MINLP optimizer Couenne, improving by orders of magnitude its performance and allowing the exact solution of instances large enough to be useful in applications. Computational results are presented where neighborhoods are either polyhedra or ellipsoids in $\mathbb{R}^2$ or $\mathbb{R}^3$ and with the Euclidean norm as distance metric.

Keywords: Nonconvex and nonlinear optimization; Traveling Salesman Problem with Neighborhoods; Spatial branch-and-bound.

2010 Mathematics Subject Classification: 90C26 Nonconvex programming, global optimization; 90C30 Nonlinear programming; 90C57 Polyhedral combinatorics, branch-and-bound, branch-and-cut.

[1]Mechanical Engineering Department, Carnegie Mellon University, Pittsburgh, PA 15213; {gentilini,shimada}@cmu.edu.

[2]Partially supported by a Bertucci Graduate Fellowship in Engineering and by a research grant form DENSO Wave, Inc, Japan; corresponding author.

[3]Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA 15213; fmargot@andrew.cmu.edu. Supported by ONR grant N00014-09-1-0033 and NSF grant NSF-0750826.

1 Introduction

A *manipulator* is a robot designed to handle objects. This interaction is typically carried out by a device mounted at the end of the manipulator arm and called *end-effector*. For computer vision based inspection, a camera or a laser digitizer is installed on the end-effector and pictures of n different features on the object surface have to be acquired. In actual industrial practice this cycle has to be continuously repeated, and a critical task is thus finding an optimal inspection sequence having minimal cycle time to maximize the number of inspected components. Each picture i can be taken from an infinite number of relative positions between object and end-effector, represented by a set $\mathcal{Q}_i$ of end-effector feasible placements. The set $\mathcal{Q}_i$ is the *neighborhood* for taking picture i. Given two pictures $i \neq j$ that have to be taken and two end-effector placements $\boldsymbol{q}_i \in \mathcal{Q}_i$ and $\boldsymbol{q}_j \in \mathcal{Q}_j$, the time for the manipulator to move from $\boldsymbol{q}_i$ to $\boldsymbol{q}_j$ is known. The goal is to find end-effector placements $\boldsymbol{q}_i \in \mathcal{Q}_i$ for $i = 1, \ldots, n$ and a tour visiting these n points such that its total cycle time is minimized. This view planning problem can be modeled as a *Traveling Salesman Problem with Neighborhoods* (TSPN), which was first introduced by Arkin and Hassin [3]. The TSPN has been studied mostly in the approximation algorithm literature [8].

This problem is very complex in its full generality, as neighborhoods can have arbitrary shapes determined by camera specifications and manipulator physical constraints. Moreover, computing the minimum time to move between two end-effector placements is in itself a difficult problem since it involves manipulator kinematic and obstacle avoidance [20]. In this paper we study a simplified version of this problem. First, although each end-effector placement is defined by six parameters (three for its position and three for its orientation), we assume that the relative orientation between camera and object is directly derived from their relative position to limit image distortion. This is a typical restriction in applications and reduces the dimension of each neighborhood to at most three. Neighborhoods are then encoded as either polyhedra or ellipsoids, which are versatile enough to well constraint the set of feasible viewpoints for each picture depending on camera optics limitations and physical view occlusions. Finally, we approximate the travel time between two end-effector placements $\boldsymbol{q}_i$ and $\boldsymbol{q}_j$ by the Euclidean distance denoted by $\mathrm{d}\left(\boldsymbol{q}_i, \boldsymbol{q}_j\right)$ hereafter. This is a reasonable indicator of the dynamic performance for industrial manipulator controllers.

Our approach to solve such instances of TSPN is to formulate it as a non-convex Mixed Integer Non Linear Programming (MINLP) using as variables the coordinates of the vertices $\boldsymbol{q}_i$ for $i = 1, \ldots, n$ as well as binary

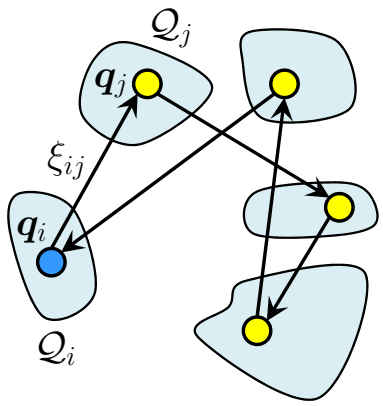


Figure 1: An ATSPN instance. The five shaded areas around the vertices are the neighborhoods of five pictures and the depicted directed tour is a feasible solution.

variables ξ_{ij} for $i, j = 1, \ldots, n$ to represent the possible edges of the tour. The resulting MINLP is nonconvex, even when the integrality constraints on the variables ξ_{ij} are relaxed. It follows that only solvers for nonconvex MINLP problems can be used for its solution. Examples of such solvers are Baron [17], Couenne [4, 6], and Lindoglobal [15]. However, these solvers struggle to solve relatively small size instances of TSPN.

In applications, a typical number of pictures acquired per cycle can range from 5 to 75 [12, 18]. Our benchmark instances have up to $n = 16$ convex neighborhoods of dimension two or three. These instances are thus simplified versions of real ones, but realistic enough to show that the proposed approach clearly outperforms standard ones. For example, the solver Couenne (with default settings) require 733 seconds to solve the instance tspn2DP6_2 to optimality, while the proposed approach solves it in a fraction of a second. Moreover, if a solution provably within a few percent of optimality is satisfactory, solving quickly instances with up to $n = 30$ becomes feasible. See Figure 2 for a typical illustration of the evolution of lower and upper bounds on the optimal solution.

Our aim is thus to show that using a specific feature of the MINLP formulation and customizing the solver by adding specific cut generators and heuristics, it is possible to solve instances of realistic size far more efficiently.

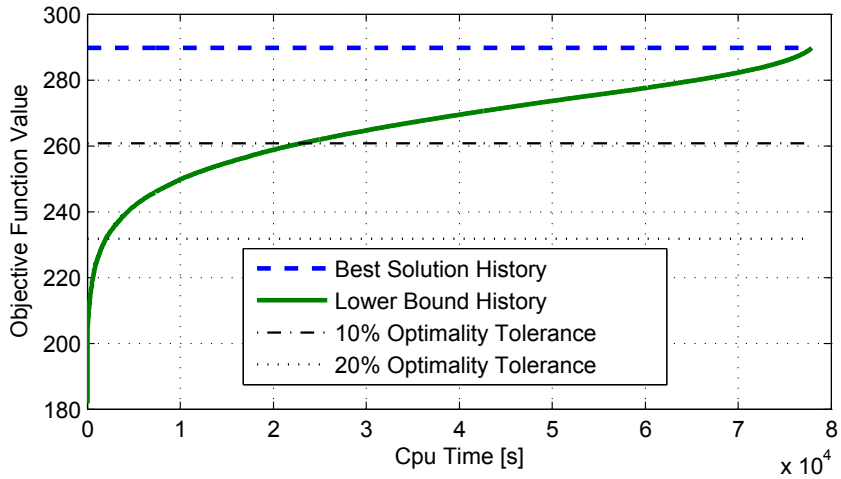


Figure 2: Convergence history of COUENNE with CglTspn for the instance tspn2DE15_1.

The crucial feature that we exploit is that once all the binary variables in the formulation are fixed to 0 or 1 values, the continuous relaxation of the remaining problem is convex. It is thus possible to solve it to optimality using a continuous solver.

The paper is organized as follows: The precise formulation we use is presented in Section 2, and the algorithm is described in Section 3. Specific settings of the various solvers used in the solution process are briefly described in Section 4. In Section 5, we give results on 64 instances of various sizes, as well as comparisons with heuristics solutions obtained by the convex solver BONMIN [5]. Finally, Section 6 contains conclusions and discusses potential future work.

2 MINLP model

An instance is given by a set $\mathrm{V} = \{1, 2, \ldots, n\}$ of the indices of the pictures to be taken, a set $\mathcal{Q}_i \subseteq \mathbb{R}^m$ for $i \in \mathrm{V}$ of neighborhoods, and a nonnegative distance function $\mathrm{d}(\boldsymbol{u}, \boldsymbol{v})$ for all $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^m$. The instance can be *symmetric* (STSPN) or *asymmetric* (ATSPN), depending on the distance function being symmetric, i.e. $\mathrm{d}(\boldsymbol{u}, \boldsymbol{v}) = \mathrm{d}(\boldsymbol{v}, \boldsymbol{u})$ for all $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^m$, or not.

The ATSPN can be formulated using variables $\boldsymbol{q}_i \in \mathbb{R}^m$ for all $i \in V$ and

binary variables ξ_{ij} for all $i, j \in V$ with $i \neq j$ such that

$$\xi_{ij} = \begin{cases} 1 & \text{if neighborhood } j \text{ is visited just after neighborhood } i; \\ 0 & \text{otherwise.} \end{cases}$$

The constraints are either those in an integer programming formulation of the Asymmetric TSP (ATSP) based on the clique packing subtour elimination constraints, also known as *DFJ formulation* [13] (constraints (2)-(4) below), or expressing that variable $\boldsymbol{q}_i \in \mathbb{R}^m$ must be in the neighborhood $\mathcal{Q}_i$ for all $i \in \mathrm{V}$ (constraints (5) below).

We obtain the following MINLP formulation of the ATSPN:

$$\begin{aligned}
\text{minimize :} \quad & \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} \xi_{ij} \, \mathrm{d}\left(\boldsymbol{q}_i, \boldsymbol{q}_j\right) & & (1) \\
\text{subject to :} \quad & \sum_{\substack{i=1 \\ i \neq j}}^{n} \xi_{ij} = 1 & \forall\, j \in \mathrm{V} & (2) \\
& \sum_{\substack{j=1 \\ j \neq i}}^{n} \xi_{ij} = 1 & \forall\, i \in \mathrm{V} & (3) \\
& \sum_{i \in \mathrm{S}} \sum_{j \in \mathrm{V} \setminus \mathrm{S}} \xi_{ij} \geq 1 & \forall\, \mathrm{S} \subset \mathrm{V} \setminus \{1\},\ |\mathrm{S}| \geq 2 & (4) \\
& \boldsymbol{q}_i \in \mathcal{Q}_i \subseteq \mathbb{R}^m & \forall\, i \in \mathrm{V} & (5) \\
& \xi_{ij} \in \{0, 1\} & \forall\, i, j \in \mathrm{V},\ i \neq j & (6) \\
& \boldsymbol{q}_i \in \mathbb{R}^m & \forall\, i \in \mathrm{V}\ . & (7)
\end{aligned}$$

The $2n$ assignment constraints (2) and (3) make sure that each vertex is visited exactly once. The $2^{n-1} - n - 1$ subtour elimination constraints (4) ensure that no subtour is present in a solution by forcing the number of active edges departing from any subgraph induced by a subset of the vertices with cardinality at least 2 to be at least equal to 1. Finally, the n constraints (5) define the neighborhoods, and the $n(n-1) + n$ constraints (6) and (7) define the domain of the instance.

The objective function is a non-convex function of the binary and continuous variables. Constraints (2)-(4) are linear, and the type of the constraints (5) depends on the shape of the neighborhood $\mathcal{Q}_i$. In our test instances, these constraints are either linear when $\mathcal{Q}_i$ is a polyhedron or convex and quadratic when it is an ellipsoid.

If the Euclidean norm is employed as distance function, the problem becomes symmetric, and only half the variables ξ_{ij} used above are necessary. Moreover, if a polyhedron $\{\boldsymbol{x} \in \mathbb{R}^m \mid \mathbf{A}_i \boldsymbol{x} \leq \mathbf{b}_i\}$ is used as neighborhood i, we obtain the following formulation of the STSPN:

$$\text{minimize :} \quad \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j>i}}^{n} \xi_{ij} \ \|\boldsymbol{q}_j - \boldsymbol{q}_i\|_2 \tag{8}$$

$$\text{subject to :} \quad \sum_{j=1}^{i-1} \xi_{ji} + \sum_{j=i+1}^{n} \xi_{ij} = 2 \qquad \forall\, i \in \mathrm{V} \tag{9}$$

$$\sum_{i \in \mathrm{S}} \left(\sum_{\substack{j \in \mathrm{V} \setminus \mathrm{S} \\ j<i}} \xi_{ji} + \sum_{\substack{j \in \mathrm{V} \setminus \mathrm{S} \\ j>i}} \xi_{ij} \right) \geq 2 \quad \forall\, \mathrm{S} \subset \mathrm{V} \setminus \{1\},\ |\mathrm{S}| \geq 3 \tag{10}$$

$$\mathbf{A}_i\, \boldsymbol{q}_i \leq \mathbf{b}_i \qquad \forall\, i \in \mathrm{V} \tag{11}$$

$$\xi_{ij} \in \{0, 1\} \qquad \forall\, i, j \in \mathrm{V},\ j > i \tag{12}$$

$$\boldsymbol{q}_i \in \mathbb{R}^m \qquad \forall\, i \in \mathrm{V}\ . \tag{13}$$

If ellipsoids are used as neighborhoods, Equation (11) becomes:

$$(\boldsymbol{q}_i - \mathbf{c}_i)^T\ \mathbf{P}_i^{-1}\ (\boldsymbol{q}_i - \mathbf{c}_i) - 1 \leq 0 \quad \forall\, i \in \mathrm{V} \tag{14}$$

where $\mathbf{P}_i$ is an $(m \times m)$ symmetric positive definite matrix and $\mathbf{c}_i$ is an $(m \times 1)$ vector, center of the i-th ellipsoid.

One difficulty in handling this formulation is the number of subtour elimination constraints (10). Although their number is exponential in the size of the instance n, they can be handled efficiently implicitly using a cutting plane approach [16].

3 Description of the algorithm

The basis of the algorithm is a spatial branch-and-bound as implemented in Couenne [4, 6]. The main difference with a usual branch-and-bound algorithm for solving mixed-integer linear programs is that branching might occur on a continuous variable. It also uses a linear outer-approximation of the nonlinear problem for bounding purposes. The detailed description of the algorithm is outside the scope of this paper and can be found in the above paper and references therein. Table 1 is a high-level simplified description of the basic algorithm applied on the STSPN, except that only

Input:	Problem $\mathbf{P}$
Output:	The value z_{opt} of an optimal solution of $\mathbf{P}$
1.	Define set L of subproblems; let $L \leftarrow \{\mathbf{P}\}$;
2.	Define z^u as an upper bound for $\mathbf{P}$; let $z^u \leftarrow +\infty$
3.	**while** $L \neq \emptyset$
4.	choose $\mathbf{P}_k \in L$
5.	$L \leftarrow L \setminus \{\mathbf{P}_k\}$
6.	generate a linear relaxation $\mathbf{LP}_k$ of $\mathbf{P}_k$
7.	**repeat**
8.	solve $\mathbf{LP}_k$; let $(\bar{\xi}^k, \bar{\boldsymbol{q}}^k)$ be an optimum and $\bar{z}^k$ its objective value
9.	refine linearization $\mathbf{LP}_k$
10.	**until** $(\bar{\xi}^k, \bar{\boldsymbol{q}}^k)$ is feasible for $\mathbf{P}_k$ or $\bar{z}^k$ does not improve sufficiently
11.	**if** $(\bar{\xi}^k, \bar{\boldsymbol{q}}^k)$ is feasible for $\mathbf{P}_k$, **then** let $z^u \leftarrow \min\{z^u, \bar{z}^k\}$
12.	(optional) find a local optimum $\hat{z}^k$ of $\mathbf{P}_k$; $z^u \leftarrow \min\{z^u, \hat{z}^k\}$
13.	**if** $\bar{z}^k \leq z^u - \epsilon$ **then**
14.	choose a variable $\chi := \xi_{ij}$ or q_{id} where $d \in \{1, \ldots, m\}$
15.	choose a branching point b
16.	create subproblems:
17.	$\mathbf{P}_{k-}$ with $\chi \leq b$,
18.	$\mathbf{P}_{k+}$ with $\chi \geq b$
19.	$L \leftarrow L \cup \{\mathbf{P}_{k-}, \mathbf{P}_{k+}\}$
20.	output $z_{\text{opt}} := z^u$

Table 1: A simplified spatial Branch-and-Bound algorithm for solving the MINLP $\mathbf{P}$.

a small number of constraints (10) are included in the initial formulation $\mathbf{P}$. How we select these constraints is explained below.

Note that when the branching variable selected in Step 14 is an integer variable, the branching point b is taken as an integer value and subproblem $\mathbf{P}_{k+}$ is defined by setting $\chi \geq b + 1$. This ensures that the two subproblems $\mathbf{P}_{k-}$ and $\mathbf{P}_{k+}$ form a partition of subproblem $\mathbf{P}_k$. When the branching variable is a continuous variable, the two generated subproblems overlap on $\chi = b$. This could create a potentially infinite loop, but the choice of the branching point in Step 15 and the refinement Step 9 prevent this to happen.

Next, we describe two major modifications of the basic algorithm as well as a way to generate a very good initial heuristic solution. These alterations yield big improvement of the performance of the solver. The resulting algorithm is refereed to as CouTspn in the remainder of the paper.

3.1 Subtour elimination constraints by cutting planes

The first modification relates to Step 9 of the algorithm. In that step, we check if any of the constraints (10) not included in the current problem is violated by the solution $(\bar{\xi}^k, \bar{\boldsymbol{q}}^k)$. This separation is done using a maximum flow computation [14]. If vertex 1 is defined as *source*, for all *terminals* vertices $t \in \mathrm{V}\backslash\{1\}$, the following max-flow linear problem is solved:

$$\text{maximize}: \quad \sum_{j=2}^{n} \zeta_{1j} \tag{15}$$

$$\text{subject to}: \quad \sum_{j=1}^{i-1} \zeta_{ji} - \sum_{j=i+1}^{n} \zeta_{ij} = 0 \qquad \forall\, i \in \mathrm{V}\backslash\{1,t\} \tag{16}$$

$$\sum_{j=2}^{n} \zeta_{1j} = \sum_{j=1}^{t-1} \zeta_{jt} - \sum_{j=t+1}^{n} \zeta_{tj} \tag{17}$$

$$-\bar{\xi}^k_{ij} \leq \zeta_{ij} \leq \bar{\xi}^k_{ij} \qquad \forall\, i,j \in \mathrm{V}, j > i \tag{18}$$

$$\zeta_{ij} \in \mathbb{R} \qquad \forall\, i,j \in \mathrm{V}, j > i \tag{19}$$

where ζ_{ij} represents the flow between vertices i and j. It is allowed to be negative to account for having a positive flow flowing from j to i on the arc ij with $j > i$. The capacity of each edge is defined in constraints (18) using the solution $\bar{\xi}^k_{ij}$ of the current linearization $\mathbf{LP}_k$.

The maximum flow value between the source 1 and at least one of the terminals is strictly less than 2 if and only if a violated constraint (10) exists. If for some terminal t this maximum flow value is strictly less than 2, the set S defining a violated constraint (10) is formed by the union of the source 1 and all other vertex i such that the constraint (16) for i has a nonzero dual variable in an optimal solution to (15)-(19). If such constraint is generated, it is added as a global cut, i.e., in all problems currently in the list L. The algorithm reaches Step 11 only if no such constraint can be found. While more efficient subtour elimination constraint separation algorithms exist [1, 2], the size of the instances we are interested to solve (i.e. $n \leq 30$) does not require more sophistication.

3.2 Solving a convex relaxation and integer cuts

The second modification concerns Step 12. In the basic algorithm, one try to solve the nonlinear problem (taking all variables as continuous) using the current bounds on the variables. If that solution $(\bar{\xi}^k, \bar{\boldsymbol{q}}^k)$ happens to satisfy

all the integer constraints (12) and the corresponding $\bar{\xi}^k$ variables set to 1 form a tour, the value value $\hat{z}^k$ of that solution is a valid upper bound on the optimal value of the STSPN.

We propose to modify this step as follows: Let $(\bar{\xi}^k, \bar{\boldsymbol{q}}^k)$ be the solution obtained when exiting the loop 7-10. If some of the integer constraints (12) are not satisfied, we round $\bar{\xi}^k$ to a binary vector $\hat{\xi}^k$ representing a tour. This is done in a greedy fashion, by selecting an initial random node p_1 and then for $j = 2, \ldots, n$ selecting p_j as the node with maximum value for the variable $\bar{\xi}^k_{p_{j-1}p_j}$ among all the nodes not yet selected. To simplify notation, we use here ξ_{ij} to denote either ξ_{ij} if $i < j$ or ξ_{ji} if $i > j$. The rounded vector $\hat{\xi}^k$ has $\hat{\xi}^k_{p_{j-1}p_j}$ and $\hat{\xi}^k_{p_n p_1}$ set to 1 and all other variable in $\hat{\xi}$ are set to 0. In that way, $\hat{\xi}^k$ is the characteristic vector of a tour, and it is feasible for the original formulation.

We then check if $\hat{\xi}^k$ already appears in the list of all rounded vectors considered so far. If $\hat{\xi}^k$ does not appear in that list, we solve the initial nonlinear problem (8)-(13) after fixing all binary variables to their rounded value in $\hat{\xi}^k$. As the resulting problem is convex, its optimal solution $\hat{\boldsymbol{q}}^k$ can be computed easily with a nonlinear solver (we use Ipopt [6, 19]). To improve the performance of the solver, a permutation π of V is used to represent the tour defined by $\hat{\xi}^k$, the binary variables are removed from the problem, and the following objective function (indices are taken modulo n):

$$\sum_{i=1}^{n} \mathrm{d}\left(\boldsymbol{q}_{\pi(i)}, \boldsymbol{q}_{\pi(i+1)}\right) \tag{20}$$

is minimized subjected to constraints (11) and (13) if the neighborhoods are polyhedra and constraints (14) and (13) if they are ellipsoids.

Ipopt requires all functions in the problem formulation to be at least once differentiable, which is not the case for the objective function (20). Thus we instead use the following objective function (we use $\epsilon = 0.1$ in the tests):

$$\mathrm{d}\left(\boldsymbol{q}_i, \boldsymbol{q}_j\right) = \begin{cases} \|\boldsymbol{q}_j - \boldsymbol{q}_i\|_2 & \text{if } \|\boldsymbol{q}_j - \boldsymbol{q}_i\|_2 \geq \epsilon \\ \frac{\epsilon}{2} + \frac{1}{2\epsilon}\|\boldsymbol{q}_j - \boldsymbol{q}_i\|_2^2 & \text{if } \|\boldsymbol{q}_j - \boldsymbol{q}_i\|_2 < \epsilon \end{cases} \tag{21}$$

The function (21) is continuously differentiable except when $\|\boldsymbol{q}_j - \boldsymbol{q}_i\|_2 = \epsilon$. In this case, it is only differentiable once. The small error introduced in the latter case is relatively inconsequential for our use of the solution. The above function is hard-coded into the solver.

The rounding operation producing $\hat{\xi}^k$ can potentially produce several times the same binary vector at different iterations. To avoid as much as

possible to generating and solving repeatedly the same continuous problem, we add the linear constraint

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} \hat{\xi}_{ij}^{k} \, \xi_{ij} \leq n - 1 \ . \tag{22}$$

Although constraint (22) is not initially valid, it can be now added as a global cut, i.e. not only to the problem $\mathbf{P}_k$ but to all problems currently in the list L. This is justified, as we have computed the optimal solution when the binary variables take the values in $\hat{\xi}^k$ and the only feasible solutions $(\xi, \boldsymbol{q})$ cut by that constraint have all $\xi = \hat{\xi}^k$. If the rounded vector $\hat{\xi}^k$ appears for the first time, cut (22) is added and we return to Step 7. Otherwise we continue to Step 13. The two above operations are implemented in a cut generator called *CglTspn* based on the COIN-OR CglCutGenerator class.

This modification is related to the local searches of the hybrid algorithm (developed for solving problems that are convex) described in Section 2.3.2 of [5]. There, it is suggested to solve the mixed-integer linear program (MILP) associated with the current subproblem and use that solution to fix integer variables, solve an NLP and get a valid upper bound. The rounding step described above can be seen as a heuristic method to solve the MILP. The integer cuts (22) can be added easily only because all integer variables in our problem are binary. For a problem with general integer variables, that option is not available.

3.3 Initial heuristic solution

Using a good upper bound z^u in Step 2 instead of $z^u = +\infty$ typically improves the solution times of MINLP. We thus devised a heuristic approach that usually generates a very good solution. A by-product of this heuristic is the identification of a set of subtour inequalities that we use in the initial formulation $\mathbf{P}$ used by the algorithm.

The heuristic starts by considering the problem H obtained by dropping all constraints (10) and (12) in the model (8)-(13). Since all variables in H are continuous, H can be solved using the interior point solver Ipopt (precise version numbers and non-default settings for the software used are listed in Section 4). The initial point used as input is constructed by initializing each variable $\boldsymbol{q}_i$ to the center of each neighborhood and the binary variables such that $\xi_{ij} = 1$ only if $j = i + 1$ with $j = 1$ if $i = n$. Note that, as problem H is nonconvex, the solution returned by Ipopt might not be optimal, but this is not a concern for our purposes. Afterward, we use the maximum flow separation algorithm described in Section 3.1 to find the first constraint (10)

violated by the solution returned by Ipopt, and we add it to H. We then call Ipopt again, and this continues until all constraints (10) are satisfied by the solution returned by Ipopt.

At that point, we feed the current formulation H to the NLP Branch-and-Bound algorithm for convex MINLP Bonmin [5, 6]. We then proceed in a similar fashion as with Ipopt, separating constraints (10) violated by the solution returned by Bonmin iteratively. As Bonmin is an exact solver only for convex problems, the solution returned is a feasible solution, but without any optimality guarantee. We nevertheless observe that, in the instances used in our computational tests, the values of the heuristic solutions obtained using this approach are usually very close to the optimal ones. The complete procedure is performed using the algebraic modeling language Ampl [9], and the maximum flow separation in this case is not embedded in a cut generator within Ipopt or Bonmin using Clp, but externally solved using the LP solver Cplex [7].

While solving the continuous relaxation of a subproblem with Bonmin, the objective function (8) might become non-differentiable when neighborhoods overlap. This happens when two vertices q_i and q_j for $i \neq j$ are identical, resulting in convergence problems as Bonmin calls Ipopt as a subroutine. To overcome this issue, we add an exponentially decaying non-convex term to the objective function (8) to prevent this overlapping (we use $\gamma = \delta = 10$ in the tests):

$$\sum_{i=1}^{n} \sum_{\substack{j=1 \\ j>i}}^{n} \xi_{ij} \ \left\| \boldsymbol{q}_j - \boldsymbol{q}_i \right\|_2 + \delta \, \mathrm{e}^{-\gamma^2 \left\| \boldsymbol{q}_j - \boldsymbol{q}_i \right\|_2^2} \ . \tag{23}$$

Finally, the subtour elimination constraints in the initial formulation used by CouTspn contains all the subtour elimination constraints that were introduced in the course of generating the heuristic solution. (Note that the exponentially decaying term (23) is not employed when using CouTspn.)

4 Software settings

Results in this paper were obtained using open-source software available from COIN-OR [6]. This section describes precisely which version and additional non-default settings were used. The used software are the following.

The interior point solver Ipopt [6, 19] version stable/3.9 with all default settings plus the option:

- `linear_solver=MA57`.

The convex MINLP solver Bonmin [5, 6] stable/1.4 using Cbc releases/2.4.2, Clp releases/1.11.1, and Ipopt releases/3.8.3 as sub-solvers with all default settings except:

- `linear_solver=MA57`
- `integer_tolerance=1e−6`
- `allowable_fraction_gap=0`

The MINLP solver Couenne [4, 6] stable/0.3 using Cbc releases/2.4.2, Clp releases/1.11.1, Ipopt releases/3.8.3 as sub-solvers with all default setting except:

- `variable_selection=osi-simple`
- `optimality_bt=no`
- `log_num_obbt_per_level=0`
- `aggressive_fbbt=no`
- `log_num_abt_per_level=0`
- `log_num_local_optimization_per_level=0`
- `local_optimization_heuristic =no`
- `ipopt.linear_solver=MA57`
- `ipopt.max_iter=500`
- `ipopt.mu_strategy=monotone/adaptive`

The meaning of `monotone/adaptive` is that, when solving a given instance, we first use the setting `monotone`. If Ipopt fails somewhere during the solution process, we then try the `adaptive` setting. For all the tested instances at least one of the two settings works.

Finally for the heuristic procedure we also employed the commercial LP solver Cplex [7] version 12.2.0 with all default settings and the algebraic modeling language Ampl [10] version 20110121.

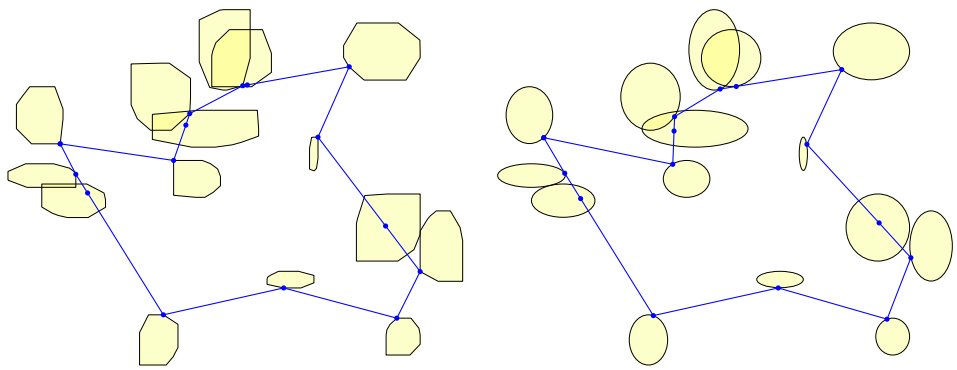
5 Computational results

Tests are performed on 64 random STSPN instances with n neighborhoods formed by ellipsoids or polyhedra defined in in $\mathbb{R}^m$ (with $m = 2$ or 3). All test instances are available from [11].

An instance with ellipsoidal neighborhoods is created as follows. First n random points $\mathbf{c}_i \in [0, 100]^m$ are generated. These points will be the centers of the ellipsoids, and their average distance $\bar{d}$ is computed. Then, for a fixed percentage h of the average distance, a box around $\mathbf{c}_i$ is defined as $\mathbf{c}_i \pm h\,\bar{d}\,\boldsymbol{x}_i/2$, where $\boldsymbol{x}_i$ is a uniformly distributed random vector in $[0, 1]^m$.

Finally, ellipsoids are placed inside these boxes aligning their principal axes with the coordinate frame. That instance is then used to create an instance with polyhedral neighborhoods. Each ellipsoid is approximated by a polyhedron with 10 facets tangent to the ellipsoid at randomly selected points. An example is shown in Figure 3.

The machine used is a `Dell Precision T7500` with an `Intel Xeon @3.33 GHz` processor with 12GB of RAM running `Fedora 14 kernel 2.6.35.13-92`.



(a) Instance tspn2DP15_1. (b) Instance tspn2DE15_1.

Figure 3: STSPN instances of comparable extension with $n = 15$ and optima tours.

First, a heuristic solution is obtained using Ipopt and Bonmin as described in Section 3. The results are reported in the column with label "Bonmin" in tables 3, 4, 5, and 6. The reported Bonmin cpu time includes the time spent for the separation of the subtour elimination constraints. The number of the added constraints is reported in the column with label "s.e. cuts".

Second, using as initial point the heuristic solution calculated in the previous step, each STSPN instance is solved to optimality by using CouTspn. The initial cutoff z^u is set to the value of the heuristic solution provided by Bonmin. The results are reported in the columns labeled "CouTspn" in tables 3, 4, 5, and 6. The overall cpu time, the time spent by CglTspn only, and the time spent within CglTspn to solve NLP instances with Ipopt are reported.

The number of subtour eliminations constraints ("s.e. cuts") generated and the number of integer cuts (22) ("int. cuts"), and the total number of nodes in the tree are also reported.

We compare four branching strategies, using either the option `osi-simple` or `osi-strong` of CouTspn. The former selects the branching variable using a simple ranking function while the latter performs strong branching with pseudo-costs [4] before selecting the variable. In addition, each of these options are tested with and without a modification of the code of CouTspn restricting branching only to binary variables. Table 2 reports the results. Although simple branching (either on all variables or restricted only to binary variables) usually requires a larger number of cuts and nodes, it seems to be the most efficient in terms of overall cpu time. Therefore, all the other tested instances were solved using simple branching on binary variables.

Table 2: Comparison of different branching options in CouTspn.

instance	branching		cpu time [s]			cuts		nodes
	only bin.	osi	overall	CglTspn	Ipopt	int.	s.e.	
tspn2DP12_1	no	simple	128	15	3.79	851	100	2,728
	yes	simple	135	15	3.82	855	102	2,750
	no	strong	604	14	2.06	456	76	3,708
	yes	strong	372	8.07	2.04	458	73	1,564
tspn2DE12_1	no	simple	379	39	12	1,795	150	6,186
	yes	simple	383	38	12	1,777	144	6,170
	no	strong	1,225	37	6.51	986	115	8,990
	yes	strong	760	24	6.83	1,034	120	4,716
tspn3DE10	no	simple	816	58	16	3,298	234	10,602
	yes	simple	804	60	15	3,189	244	10,690
	no	strong	2,665	102	10	2,166	183	30,464
	yes	strong	1,015	49	11	2,242	181	10,168

Table 3: STSPN instances with polyhedra in $\mathbb{R}^2$ as neighborhoods.

instance	n	var.	h	Bonmin heuristic value	Bonmin cpu time [s]	Bonmin s.e. cuts	CouTspn optimal value	CouTspn cpu time [s] overall	CouTspn cpu time [s] CglTspn	CouTspn cpu time [s] Ipopt	CouTspn cuts int.	CouTspn cuts s.e.	CouTspn nodes
tspn2DP5_1	5	20	0.25	184.733	0.15	0	184.733	0.12	0.04	0.02	7	0	16
tspn2DP5_2	5	20	0.15	217.659	0.13	0	217.659	0.14	0.04	0.01	5	0	16
tspn2DP6_1	6	27	0.25	200.469	0.21	1	200.469	0.40	0.11	0.03	12	3	42
tspn2DP6_2	6	27	0.15	247.588	0.18	0	247.588	0.13	0.04	0.01	3	2	12
tspn2DP7_1	7	35	0.25	196.253	0.36	1	196.247	1.72	0.49	0.14	38	15	126
tspn2DP7_2	7	35	0.15	236.444	0.32	2	236.444	1.19	0.30	0.11	26	3	66
tspn2DP8_1	8	44	0.25	188.118	0.31	0	188.108	1.79	0.42	0.09	26	10	96
tspn2DP8_2	8	44	0.15	226.103	0.49	1	226.103	4.04	1.05	0.35	90	14	248
tspn2DP9_1	9	54	0.25	250.939	0.34	1	249.732	22	4.52	1.03	271	48	914
tspn2DP9_2	9	54	0.15	258.450	0.40	1	258.450	2.12	0.42	0.09	23	10	80
tspn2DP10_1	10	65	0.25	220.242	0.48	2	220.242	21	3.51	0.84	180	42	656
tspn2DP10_2	10	65	0.15	268.378	0.34	0	268.378	3.85	0.55	0.13	35	20	150
tspn2DP11_1	11	77	0.25	243.847	0.60	4	243.847	109	14	4.07	855	98	2,918
tspn2DP11_2	11	77	0.15	254.221	0.50	0	254.221	11	1.25	0.31	82	23	264
tspn2DP12_1	12	90	0.25	253.543	0.58	3	253.543	135	15	3.82	855	102	2,750
tspn2DP12_2	12	90	0.15	307.750	0.54	2	306.931	91	12	3.45	699	86	2,116
tspn2DP13_1	13	104	0.25	280.389	1.52	8	273.174	4,895	707	170	25,629	487	73,724
tspn2DP13_2	13	104	0.15	318.432	0.46	0	317.780	93	11	3.14	622	126	1,926
tspn2DP14_1	14	119	0.25	306.338	0.89	4	306.338	6,537	915	204	30,973	776	93,864
tspn2DP14_2	14	119	0.15	266.009	0.92	3	264.164	378	39	11	1,690	123	5,718
tspn2DP15_1	15	135	0.25	285.082	1.72	6	280.202	28,121	5,192	649	94,211	1,197	277,780
tspn2DP15_2	15	135	0.15	299.055	1.42	6	288.467	3,020	352	78	11,937	371	40,002
tspn2DP16_1	16	152	0.15	367.895	3.34	6	365.777	13,654	2,020	331	53,350	998	138,650
tspn2DP16_2	16	152	0.15	292.280	1.43	8	292.280	5,701	633	123	18,753	420	57,292

Table 4: STSPN instances with polyhedra in $\mathbb{R}^3$ as neighborhoods.

instance	n	var.	h	Bonmin heuristic value	Bonmin cpu time [s]	Bonmin s.e. cuts	CouTspn optimal value	CouTspn cpu time [s] overall	CouTspn cpu time [s] CglTspn	CouTspn cpu time [s] Ipopt	CouTspn cuts int.	CouTspn cuts s.e.	CouTspn nodes
tspn3DP5	5	25	0.25	236.214	0.13	0	236.214	0.15	0.04	0.02	5	0	10
tspn3DP6	6	33	0.25	257.551	0.20	1	257.551	0.60	0.13	0.04	14	4	36
tspn3DP7	7	42	0.25	310.691	0.28	2	310.691	4.25	0.71	0.21	56	14	182
tspn3DP8	8	52	0.25	279.257	0.27	0	277.730	12	2.11	0.61	137	27	398
tspn3DP9	9	63	0.25	295.018	0.44	3	290.478	58	6.92	2.23	482	80	1,552
tspn3DP10	10	75	0.25	306.508	0.46	3	301.884	230	25	7.87	1,628	171	4,862
tspn3DP11	11	88	0.25	276.119	0.91	3	276.119	532	50	14	2,507	184	8,128
tspn3DP12	12	102	0.25	300.906	0.96	4	298.779	7,807	791	164	27,705	637	87,840

Table 5: STSPN instances with ellipsoids in $\mathbb{R}^2$ as neighborhoods.

instance	n	var.	h	Bonmin heuristic value	Bonmin cpu time [s]	Bonmin s.e. cuts	CouTspn optimal value	CouTspn cpu time [s] overall	CouTspn cpu time [s] CglTspn	CouTspn cpu time [s] Ipopt	CouTspn cuts int.	CouTspn cuts s.e.	CouTspn nodes
tspn2DE5_1	5	20	0.25	191.255	0.14	0	191.255	0.22	0.07	0.03	8	0	18
tspn2DE5_2	5	20	0.15	219.307	0.13	0	219.307	0.19	0.06	0.03	7	0	16
tspn2DE6_1	6	27	0.25	202.995	0.24	1	202.995	0.67	0.18	0.06	13	4	44
tspn2DE6_2	6	27	0.15	248.860	0.18	0	248.860	0.24	0.08	0.04	5	2	12
tspn2DE7_1	7	35	0.25	201.492	0.30	1	201.492	3.38	1.00	0.28	46	14	160
tspn2DE7_2	7	35	0.15	239.788	0.25	1	239.788	1.72	0.43	0.18	27	3	78
tspn2DE8_1	8	44	0.25	190.243	0.37	0	190.243	2.61	0.45	0.12	23	9	98
tspn2DE8_2	8	44	0.15	229.190	0.40	1	229.150	7.12	1.77	0.60	86	16	286
tspn2DE9_1	9	54	0.25	259.297	0.40	2	259.290	45	7.42	1.62	401	55	1,390
tspn2DE9_2	9	54	0.15	262.815	0.41	1	262.815	3.20	0.55	0.19	26	11	92
tspn2DE10_1	10	65	0.25	225.126	0.41	1	225.126	35	4.45	1.79	224	40	820
tspn2DE10_2	10	65	0.15	273.768	0.35	0	273.192	7.85	0.76	0.24	43	23	182
tspn2DE11_1	11	77	0.25	249.760	0.63	5	247.886	186	19	5.19	954	104	3,542
tspn2DE11_2	11	77	0.15	258.003	0.39	0	258.003	18	2.06	0.63	91	21	340
tspn2DE12_1	12	90	0.25	265.858	0.55	2	265.858	383	38	12	1,777	144	6,170
tspn2DE12_2	12	90	0.15	314.063	0.86	4	312.493	209	22	5.82	941	98	3,148
tspn2DE13_1	13	104	0.25	278.876	1.15	5	278.876	8,813	1,175	295	32,637	533	104,268
tspn2DE13_2	13	104	0.15	324.940	0.49	1	324.271	246	22	5.76	909	151	3,458
tspn2DE14_1	14	119	0.25	310.794	0.95	3	310.794	11,396	1,373	319	38,205	872	127,704
tspn2DE14_2	14	119	0.15	272.157	0.69	3	270.638	840	74	22	2,774	142	9,462
tspn2DE15_1	15	135	0.25	290.362	1.08	6	289.716	77,905	16,276	1,777	184,845	1,652	543,774
tspn2DE15_2	15	135	0.15	293.405	1.20	6	293.357	5,904	610	169	16,602	430	60,230
tspn2DE16_1	16	152	0.15	374.005	2.84	6	369.945	25,663	3,323	543	67,531	1,105	195,806
tspn2DE16_2	16	152	0.15	295.130	1.20	7	295.130	9,847	907	189	21,931	471	73,638

Table 6: STSPN instances with ellipsoids in $\mathbb{R}^3$ as neighborhoods.

instance	n	var.	h	Bonmin heuristic value	Bonmin cpu time [s]	Bonmin s.e. cuts	CouTspn optimal value	CouTspn cpu time [s] overall	CouTspn cpu time [s] CglTspn	CouTspn cpu time [s] Ipopt	CouTspn cuts int.	CouTspn cuts s.e.	CouTspn nodes
tspn3DE5	5	25	0.25	253.495	0.20	0	253.495	0.17	0.05	0.02	4	0	14
tspn3DE6	6	33	0.25	276.996	0.27	1	276.996	1.21	0.20	0.07	19	4	50
tspn3DE7	7	42	0.25	323.689	0.32	2	323.689	7.10	0.91	0.24	66	19	210
tspn3DE8	8	52	0.25	296.918	0.46	0	296.918	28	3.50	1.11	200	33	632
tspn3DE9	9	63	0.25	315.761	0.44	3	312.920	156	12	3.47	790	107	2,722
tspn3DE10	10	75	0.25	328.627	0.73	4	328.627	804	60	15	3,189	244	10,690
tspn3DE11	11	88	0.25	301.307	0.58	1	301.307	1,955	131	32	6,153	323	21,478
tspn3DE12	12	102	0.25	320.575	1.32	5	320.575	24,623	2,442	318	54,913	855	183,388

We first observe that, unsurprisingly, the difficulty of solving an instance usually increases with the number n of neighborhoods. We note that the cpu time to compute the initial heuristic solution is a small fraction (smaller than 10^{-4} on some instances) of the time needed for solving the instance to optimality and that this fraction decreases as n increases. The maximum time used by CouTspn to solve one of the instances is 28,121 seconds for polyhedral neighborhoods (tspn2DP15_1) and 77,905 seconds for ellipsoidal neighborhoods (tspn2DE15_1), while the corresponding values for obtaining the heuristic solutions are respectively 1.72 and 1.08 seconds.

For polyhedral neighborhoods in $\mathbb{R}^2$ (resp. $\mathbb{R}^3$) the average percent gap between heuristic and optimal solution is 0.43% (resp. 0.54%) and the maximum gap is 3.67% for tspn2DP15_2 (resp. 1.56% for tspn3DP9). Moreover, the heuristic solution turns out to be optimal in 14 cases out of 24 in $\mathbb{R}^2$ and in 4 cases out of 8 in $\mathbb{R}^3$.

For ellipsoidal neighborhoods in $\mathbb{R}^2$ (resp. $\mathbb{R}^3$), the average percent gap between heuristic and optimal solution is 0.15% (resp. 0.11%) and the maximum gap is 1.10% for tspn2DE16_1 (resp. 0.91% for tspn3DE9). The heuristic solution turns out to be optimal in 14 cases out of 24 in $\mathbb{R}^2$ and in 7 cases out of 8 in $\mathbb{R}^3$. These numbers attest of the quality of the solutions found by the heuristic algorithm.

Thus, CouTspn usually improves slightly the heuristic solution and gives a guarantee of optimality of its solution. Note that CouTspn usually finds the optimal value within the first few iterations of the cut generator CglTspn, and most of the cpu time is spent afterward to prove its optimality, as illustrated in Figure 2.

Furthermore, the results show that solving instances with ellipsoidal neighborhoods is in general harder than with polyhedral ones. This suggests that in practice, as long as an approximation is necessary, using polyhedral neighborhoods is likely the better option. In $\mathbb{R}^2$, the ratio of the cpu times required to solve an instance with ellipsoidal neighborhoods vs. polyhedral ones is on average 1.91, with a maximum ratio of 2.84 for tspn2DX12_1. A similar pattern occurs in $\mathbb{R}^3$ with an average ratio of 2.52 and a maximum one of 3.67 for tspn3DX11.

We also observe that three-dimensional instances are harder to solve than two-dimensional ones, given the same number n of neighborhoods and the same extension factor h. If one compares the 8 instances with n increasing from 5 to 12 and $h = 0.25$, the ratio of the cpu times for polyhedral neighborhoods in $\mathbb{R}^3$ vs. $\mathbb{R}^2$ is on average 11, with a maximum of 58 for tspnXDP12. For ellipsoidal neighborhoods, the average ratio is 16, with a maximum of 64 for tspnXDE12. In only one instance, tspnXDE5, the cpu

times are approximately the same. Note that in the above comparisons, although the parameters n and h are identical for paired instances, there is no strict correspondence of shape and location of the neighborhoods. Other factors such as overlapping conditions or spatial distribution may thus influence the difference in cpu time, as can be easily noticed by observing the results for the two-dimensional cases with $n = 16$ and $h = 0.15$. The ratio of cpu times for instances with polyhedral neighborhoods tspn2DP16_1 and tspn2DP16_1 is 2.40.

Finally, in most cases, instances with larger neighborhoods ($h = 0.25$) are harder to solve than instances with smaller ones ($h = 0.15$), when the number n of neighborhoods is fixed. For polyhedral neighborhoods, the ratio of cpu times required to solve an instance with $h = 0.25$ vs. $h = 0.15$ is 11, with a maximum of 53 for tspn2DP13. However, there are instances for which the opposite is true, namely tspn2DP5 (ratio 0.85) and tspn2DP8 (ratio 0.44). This confirms that other aspects not considered in this analysis may also influence the overall cpu time. For ellipsoidal neighborhoods, the average ratio is 9, with a maximum of 36 for tspn2DE13. Here also, for instance tspn2DE8 the ratio falls to 0.37, well below one.

A comparison between CouTspn and CouTspn when the modification described in Section 3.2 is not used can be found in Appendix A. Even with termination criteria very favorable for the latter algorithm, we observe that the proposed approach is orders of magnitude faster.

6 Conclusion

In this paper a non-convex MINLP formulation of the Traveling Salesman Problem with Neighborhoods is provided. A very fast heuristic solution procedure using the MINLP solver Bonmin is described. Computational tests show that the generated heuristic solution is usually within 1% of optimality, making it a very efficient and practical tool.

An approach for solving the problem to optimality using a modified version of the global MINLP solver Couenne is also presented. An ad hoc cut generator, called CglTspn, is developed to improve the performance of the standard algorithm while solving STSPN instances.

Two main modifications are proposed. First, subtour elimination constraints are handled as cutting planes and introduced using a maximum flow computation. Second, we observe that, in the proposed formulation, fixing all the binary variables results in a convex problem. We take advantage of that by rounding fractional solutions, solving the corresponding problem

(possibly improving the upper bound), and adding a cut preventing the same rounded solution to be considered. Computational tests show that realistic instances can be solved to optimality and that the proposed approach is orders of magnitude faster than the standard algorithm implemented in Couenne. We note that many applications have the property used in this second modification. The proposed approach can thus be useful in other contexts.

Nevertheless, solving instances with a number of neighborhoods larger than 15 remains challenging. This issue can be partially overcome by asking only for a solution provably within a few percent of optimality instead of an optimal solution. Future work will focus on improving the convergence rate of the proposed approach by enhancing the effectiveness of the cut generator in reducing the domain while adding new cuts.

References

[1] D. Applegate, R.E. Bixby, V. Chvátal, and W. Cook. On The Solution of Traveling Salesman Problems. *Documenta Mathematica*, Extra Volume ICM Berlin 1998(III):645–656, 1998.

[2] D. Applegate, R.E. Bixby, V. Chvátal, and W. Cook. *The Traveling Salesman Problem, A Computational Study.* Princeton University Press, Princeton, 2006.

[3] E.M. Arkin and R. Hassin. Approximation Algorithms for the Geometric Covering Salesman Problem. *Discrete Applied Mathematics*, 55(3): 197–218, 1994.

[4] P. Belotti, J. Lee, L. Liberti, F. Margot, and A. Wächter. Branching and Bound Tightening Techniques for Nonconvex MINLPs. *Optimization Methods and Software*, 24:597–634, 2009.

[5] P. Bonami, L. Biegler, A. Conn, G. Cornuéjols, I. Grossmann, C. Laird, J. Lee, A. Lodi, F. Margot, N. Sawaya, and A. Wächter. An Algorithmic Framework for Convex Mixed Integer Nonlinear Programs. *Discrete Optimization*, 5:186–204, 2008.

[6] COIN-OR. Computational infrastructure for operations research project. URL `http://www.coin-or.org/`.

[7] CPLEX. IBM ILOG CPLEX Optimization Studio 12.3. URL `http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/`.

[8] K.M. Elbassioni, A.V. Fishkin, and R.A. Sitters. Approximation Algorithms for Euclidean Traveling Salesman Problem with Discrete and Continuous Neighborhoods. *International Journal of Computational Geometry and Applications*, 19(2):173–193, 2009.

[9] R. Fourer, D.M. Gay, and B.W. Kernighan. *AMPL: a modeling language for mathematical programming: with AMPL Plus student edition for Microsoft Windows*. Brooks/Cole, 1997.

[10] R. Fourer, D.M. Gay, and B.W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Brooks/Cole Publishing Company / Cengage Learning, 2002.

[11] I. Gentilini, F. Margot, and K. Shimada. STSPN Instances. URL `http://wpweb2.tepper.cmu.edu/fmargot/ampl.html`.

[12] L.B. Gueta, R. Chiba, J. Ota, T. Ueyama, and T. Arai. Coordinated Motion Control of a Robot Arm and a Positioning Table With Arrangement of Multiple Goals. In *Proceedings of the 2008 IEEE international conference on Robotics and Automation*, pages 2252–2258. Institute of Electrical and Electronics Engineers Inc., The, 2008.

[13] G. Gutin and A.P. Punnen. *The Traveling Salesman Problem and its Variations*. Kluwer Academic Publisher, Dordrecht, 2002.

[14] S. Hong. *A Linear Programming Approach for the Travelling Salesman Problem*. PhD thesis, Johns Hopkins University, Baltimore, 1972.

[15] LINDO. Solver Suite. URL `http://www.lindo.com/`.

[16] M.W. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large scale symmetric travelling salesman problems. *SIAM Review*, 33:60–100, 1991.

[17] N.V. Sahinidis. Baron: a General Purpose Global Optimization Software Package. *JOGO*, 8:201–205, 1996.

[18] J.I. Vásquez-Gómez, E. López-Damian, and L.E. Sucar. View planning for 3d object reconstruction. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2009)*, pages 4015–4020. IEEE, 2009.

[19] A. Wächter and L.T. Biegler. On the Implementation of an Interior-Point Filter Line-Search Algorithm for Large-Scale Nonlinear Programming. *Mathematical Programming*, 106(1):25–57, 2006.

[20] P. Wang, R. Krishnamurti, and K. Gupta. View planning problem with combined view and traveling cost. In *2007 IEEE International Conference on Robotics and Automation (ICRA 2007)*, pages 711–716. IEEE, 2007.

A Effectiveness of integer cuts

In order to illustrate the effect of the modification described in Section 3.2, we compare two versions of the proposed algorithm. The first one is the algorithm CouTspn described in Section 3. The second one (named Standard in Table 7) is obtained from CouTspn by skipping the modification described in Section 3.2. Standard is the straightforward adaptation of Couenne in order to handle the subtour elimination constraints by branch-and-cut. Both algorithms are run using the upper bound computed as described in Section 3.3.

Running a few examples using standard, we noted that the cpu time it required was much larger than for CouTspn. We thus modify the condition to stop Standard: If CouTspn requires less than a second, Standard terminates if its optimality gap is less than 0.01% or if its cpu time exceeds 250 seconds. Otherwise, Standard is terminated if its cpu time is 250 times larger than the cpu time used by CouTspn.

Instances used in this comparisons are a subset of the instances used in Section 5. They are selected to cover all types of instances with $n \leq 10$ in $\mathbb{R}^2$ and $n \leq 8$ in $\mathbb{R}^3$.

Results are reported in Table 7. Instances reported in boldface terminate as the optimality gap becomes smaller than 0.01%. On these instances, Standard is on average 370 times slower than CouTspn. Instances tspn2DP6_1 and tspn2DE6_1 terminate as the cpu time becomes larger than 250 seconds. All the other Instances terminate as the cpu time exceeds 250 times the cpu time required by CouTspn. On these instances, the average optimality gap is 2.29% (with a maximum of 6.51% for tspn3DE8).

The difference in cpu time between the two algorithms is thus clearly larger than two order of magnitude. If we consider instances in $\mathbb{R}^2$ with $h = 0.25$ terminated as the cpu time exceeds 250 times the cpu time required

by CouTspn, despite the outlier tspn2DE6_1 showing a gap of 4.13%, the optimality gap increases from 0.62% ($n = 7$) to 3.84% ($n = 10$) for polyhedra, and from 1.16% ($n = 7$) to 5.13% ($n = 10$) for ellipses. Similar trends can be observed in $\mathbb{R}^2$ with $h = 0.15$ and in $\mathbb{R}^3$.

Table 7: Comparison between CouTspn and Standard.

instance	CouTspn		Standard					
	optimal value	cpu time [s]	lower bound	upper bound	percent gap	cpu time [s]	s.e. cuts	nodes
tspn2DP5_1	184.733	0.12	184.714	184.733	0.01%	6.29	0	1,201
tspn2DP5_2	217.659	0.14	217.649	217.659	0.01%	4.32	0	501
tspn2DP6_1	200.469	0.40	199.579	200.469	0.44%	251	1	54,808
tspn2DP6_2	247.588	0.13	247.564	247.588	0.01%	69	2	21,101
tspn2DP7_1	196.247	1.72	195.043	196.247	0.62%	431	9	62,013
tspn2DP7_2	236.444	1.19	236.406	236.444	0.02%	298	3	75,764
tspn2DP8_1	188.108	1.79	180.334	188.108	4.13%	449	8	38,349
tspn2DP8_2	226.103	4.04	224.277	226.103	0.81%	1,012	13	107,197
tspn2DP9_1	249.732	22	245.650	249.732	1.63%	5,504	41	479,662
tspn2DP9_2	258.450	2.12	255.489	258.450	1.15%	531	10	29,151
tspn2DP10_1	220.242	21	211.794	220.242	3.84%	5,258	35	310,880
tspn2DP10_2	268.378	3.85	264.219	268.378	1.55%	964	18	68,130
tspn3DP5	236.214	0.15	236.191	236.214	0.01%	6.14	0	701
tspn3DP6	257.551	0.60	257.526	257.551	0.01%	50	2	11,801
tspn3DP7	310.691	4.25	306.496	310.691	1.35%	1,064	15	81,267
tspn3DP8	277.730	12	265.412	277.730	4.44%	3,004	26	156,345
tspn2DE5_1	191.255	0.22	191.236	191.255	0.01%	134	0	62,801
tspn2DE5_2	219.307	0.19	219.285	219.307	0.01%	9.77	0	2,801
tspn2DE6_1	202.995	0.67	200.548	202.995	1.21%	251	4	42,810
tspn2DE6_2	248.860	0.24	248.836	248.860	0.01%	250	2	70,801
tspn2DE7_1	201.492	3.38	199.158	201.492	1.16%	848	10	128,033
tspn2DE7_2	239.788	1.72	237.668	239.788	0.88%	428	4	47,800
tspn2DE8_1	190.243	2.61	182.538	190.243	4.05%	654	8	48,986
tspn2DE8_2	229.150	7.12	226.580	229.160	1.12%	1,782	14	163,766
tspn2DE9_1	259.290	45	249.809	259.290	3.66%	11,274	54	659,099
tspn2DE9_2	262.815	3.20	257.484	262.815	2.03%	801	9	52,717
tspn2DE10_1	225.126	35	213.581	225.126	5.13%	8,771	36	402,581
tspn2DE10_2	273.192	7.85	265.717	273.192	2.74%	1,968	17	106,578
tspn3DE5	253.495	0.17	253.469	253.495	0.01%	151	0	30,901
tspn3DE6	276.996	1.21	273.329	276.996	1.32%	303	1	29,251
tspn3DE7	323.689	7.10	314.313	323.689	2.90%	1,778	18	105,588
tspn3DE8	296.918	28	277.578	296.918	6.51%	7,015	33	266,846