# Multi-Goal Path Optimization for Robotic Systems with Redundancy based on the Traveling Salesman Problem with Neighborhoods

Submitted in partial fulfillment of the requirements for

the degree of

Doctor of Philosophy

in

Mechanical Engineering

Iacopo Gentilini

Laurea, Mechanical Engineering, Politecnico di Torino, Italy
Diplom, Mechanical Engineering, Universität Karlsruhe (TH), Germany
M.S., Mechanical Engineering, Carnegie Mellon University

Carnegie Mellon University
Pittsburgh, PA

May, 2012

## Acknowledgments

The author would like to express his deep gratitude and appreciation to his advisor and chair of the doctoral committee, Dr. Kenji Shimada, and to the other member of the committee, Dr. François Margot, Dr. William C. Messner, and Dr. David Bourne.

He also would like to acknowledge DENSO Wave, Inc, Japan, and the Autodesk IDEA Studio program for providing enlightening ideas, the test environments, and partial funding for this research.

Finally, a special thanks is directed to the Bertucci Graduate Fellowship in Engineering and the Mary Jane and Milton C. Shaw Fellowship for their encouraging trust and financial support.

## Abstract

Finding an optimal path for a redundant robotic system to visit a sequence of several goal locations is a complex optimization problem and poses two main technical challenges. Because of the redundancy in the system, the robot can assume an infinite number of goal configurations to reach each goal location. Therefore, not only an optimal sequence of the goals has to be defined, but also, for each goal, an optimal configuration has to be chosen among infinite possibilities. Second, the actual cost for the system to move from one configuration to the next depends on many factors, such as obstacle avoidance or energy consumption, and can be calculated only through the employment of specific path planning techniques.

We first address the optimization problem of finding an optimal sequence of optimal configurations, while assuming the cost function to be analytically defined. This problem is modeled as a Traveling Salesman Problem with Neighborhoods (TSPN), which extends the well-known TSP to more general cases where each vertex (goal configuration) is allowed to move in a given region (neighborhood). In the literature, heuristic solution approaches are available for TSPN instances with only circular or spherical neighborhoods. For more general neighborhood topologies, but limited to the Euclidean norm as edge weighting function, approximation algorithms have also been proposed. We present three novel approaches: (1) a global Mixed Integer Non Linear Programming (MINLP) optimizer and (2) a convex MINLP optimizer are modified to solve to optimality TSPN instances with up to 20 convex neighborhoods, and (3) a hybrid random-key Genetic Algorithm (GA) is developed to address more general problems with a larger number of possibly non-convex neighborhoods and with different types of edge weighting functions. Benchmark tests show that the GA is able to find the same optimal tour calculated by the MINLP solvers while drastically reducing the computational cost, and it always improves the best known solutions for available test problems with up to 1,000 neighborhoods.

Second, we integrate the GA with a probabilistic path planning technique to apply the proposed procedure to two practical applications. We minimize the time currently required by an industrial vision inspection system to complete a multi-goal cycle, where the neighborhoods are defined using piecewise cubic splines in a seven-dimensional configuration space. Afterwards, we optimize the flight path and the energy consumption of a quadrotor Unmanned Aerial Vehicle (UAV) on an urban survey mission. The specifications of the camera installed on the UAV are used here to define the neighborhoods as three-dimensional polyhedra.

*Cum his versare, qui te meliorem facturi sunt.*
*Illos admitte, quos tu potes facere meliores.*
*Mutuo ista fiunt, et homines, dum docent, discunt.*

Seneca, Epistulae Morales, VII, 8

A Lino

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# Chapter 1

# Introduction

In this chapter, we first provide a literature review about recent works relevant to our research, and we illustrate their limitations for practical applications. Then, we state our contribution and summarize the obtained numerical results.

## 1.1 Literature Review

### 1.1.1 The TSPN

There are optimization instances in which the standard Traveling Salesman Problem (TSP) formulation cannot fully capture the exact nature of the problem. Indeed, if the vertices are allowed to move in certain continuous domains (neighborhoods), not only an optimal Hamiltonian cycle has to be found that visits each vertex once, but also the optimal position of each vertex in its neighborhood has to be defined. The combination of an optimal Hamiltonian cycle and optimal vertices positions is called *optimal tour* hereafter. This problem was initially introduced by Arkin and Hassin [9], and it is commonly referred to as the TSP with Neighborhoods (TSPN).

Some technical problems have been recently posed in the literature where the TSPN formulation seems to be a suitable approach to properly capture their nature. Utility companies employ automated meter reading (AMR) based on radio frequency identification (RFID) to read meters from a certain distance. The reader has thus to plan in advance the shortest path that travels within a certain radius from each meter to minimize the reading costs [47, 93]. The same scenario occurs when mobile robots have to acquire data from distributed sensors and therefore they have to approach each sensor from a minimum distance to allow the wireless communication working

properly [105]. Unmanned aerial vehicles (UAV) can be deployed to monitor a set of sites. A flight path has thus to be calculated such that the UAV flies within a certain distance from the center of each site, while minimizing the flying time or the fuel consumption [62]. Industrial manipulators can be used to perform a sequence of multiple tasks during an operating cycle. If the robotic system has 7 or more degrees of freedoms (DOF) there is an infinite number of possible configurations that can be used while performing each task in the sequence. Thus, an optimal sequence of optimal configurations has to be calculated by a multi-goal path planner [44]. In all but the last cited cases the neighborhoods are represented by balls in $\mathbb{R}^2$ or in $\mathbb{R}^3$, and only in the last application the neighborhoods are non-convex regions in the robot configuration space. Depending on the topology of the neighborhoods and on the type of the edge weighting function, specific heuristic approaches are available in the literature.

### 1.1.2 Heuristics

In the case of partially overlapping or disjoint balls in $\mathbb{R}^2$ and Euclidean norm, indicated also as Close Enough TSP (CETSP), Gulczynski et al. [47] propose different heuristics based on tiling, sweeping circles, radial adjacency, and Steiner zones. Dong et al. [28] propose two heuristics to extract representatives vertices for each neighborhood based on tiling or convex hulls, and then simulated annealing is used to search for a near *optimal tour*. An extension of the Steiner zones heuristic is provided by Mennell [74], where also balls in $\mathbb{R}^3$ and Manhattan norm are considered. First a graph reduction is performed by finding the intersections of the partially overlapping balls (Steiner zones), and representative vertices are chosen for each zone. Then a classical TSP is solved using these vertices, and finally the solution is improved by solving a continuous touring problem. Among the several variants of the main procedure, Mennell [74] proposes also to discretize the Steiner zones into several representative points, and to employ a Genetic Algorithm (GA) to solve the resulting Generalized TSP (GTSP) [94]. The proposed procedures are applied on a set of test instances and results are compared to the one obtained applying different approaches proposed by other authors. Instances from the same test set will be used in this work to benchmark our proposed method.

In the case of partially overlapping or disjoint balls in $\mathbb{R}^2$ and where aircraft dynamics is considered in defining the edge weighting function, Klesh [62] discusses necessary conditions for optimality and proposes two heuristics, based on a "rubberband" approach or on a GTSP model. Since edges

are trajectories rather straight lines, not only the position of each vertex has to be considered while searching for a near *optimal tour*, but also the derivative of the trajectory at each vertex location.

In the case of disjoints balls in $\mathbb{R}^2$ and Euclidean or Manhattan norm Yuan et al. [105] propose a two step approach: a permutation of the neighborhoods is found by a traditional TSP algorithm, and then an evolutionary approach is employed to find the best point in each neighborhood.

In the case of multi-goal path planning for redundant robotic systems, where the problem complexity is further increased by the fact that a collision free path between neighborhoods has to be found, Gueta et al. [44] propose to find first a near optimal sequence in three steps: (1) cluster the representative placements in the workspace, (2) solve the resulting TSP in each cluster, and (3) concatenate the resulting paths. Then each neighborhood is sampled, and a configuration is chosen for each neighborhood by combining a greedy nearest neighbor method and the Dijkstra algorithm using a rough-to-smooth procedure. For similar cases, Saha et al. [91] propose first to extract a small number of discrete samples for each neighborhood. The resulting GTSP is then approximated by calculating a minimum group spanning tree as a special case of the Steiner tree problem [85] and by performing a preorder tree walk.

The main limitation of the mentioned heuristic approaches is the fact that only balls in $\mathbb{R}^2$ or in $\mathbb{R}^3$ are employed as neighborhoods. Only in the case of robotic manipulators also non-convex neighborhoods are considered, but a pre-sampling step is performed to transform the TSPN into a GTSP. In this case, to avoid an excessive complexity in the GTSP model, only few samples can be used to replace the continuous neighborhoods with clusters of nodes [91].

### 1.1.3 Approximation algorithms

Besides heuristic approaches, in the computational geometry literature many approximation algorithms have been proposed for the case of the TSPN in $\mathbb{R}^2$ with Euclidean norm. The achieved approximation factors vary for the different cases depending on the fact that the neighborhoods may be connected or non-connected, disjoint or intersecting, with comparable or varying diameter, convex or non-convex, and fat or non-fat. Two definitions of fatness are available in the literature:

- A region $O \subseteq \mathbb{R}^2$ is said to be $\alpha$-fat if for any disk $D$, which does not fully contain $O$ and whose center lies in $O$ as illustrated in Figure 1.1,

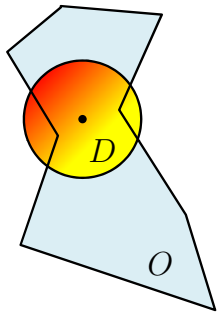



Figure 1.1: A disk $D$ used in the definition of fatness for a region $O \subseteq \mathbb{R}^2$.

the area of the intersection of $O$ and $D$ is al least $1/\alpha$ the area of $D$ [30]. For example $\alpha$ is 1 for plane, 2 for the half plane, 4 for disk, $\infty$ for a line segment.

- A region $O$ is said to be $\alpha$-fat if the ratio of the radius of the smallest circumscribing circle to the radius of largest inscribed circle is bounded by $\alpha$ [77].

Non-fatness and intersection seem to make the problem harder. Moreover, it has been proved that the most general case of TSPN is APX-hard [22, 90], even for the simple case where the neighborhoods are line segments of approximately the same length [30]. For connected, disjoint, varying diameter, and $\alpha$-fat neighborhoods, Elbassioni et al. [30] propose an $O(\alpha)$-approximation algorithm, which can be extended to $O(\alpha/\sqrt{m})$ in $\mathbb{R}^m$. Under the second definition of fatness, Mitchell [77] proposes a polynomial-time approximation scheme (PTAS) for the same problem topology. For connected, intersecting, and comparable diameters neighborhoods a $O(1)$-approximation is proposed by Dumitrescu and Mitchell [29]. Furthermore, if the neighborhoods are convex and $\alpha$-fat an $O(\alpha^3)$ approximation is given in [30]. Finally, in the case of connected and intersecting neighborhoods with varying diameter, an $O(\log(n))$ approximation is proposed for polygons [43] and for more general neighborhoods [30], where $n$ is the number of neighborhoods. The latter approximation becomes $O(1)$ if the neighborhoods have comparable diameters.

The above mentioned approximation algorithms, which are polynomial time in many cases and represent a useful tool for finding a valid upper bound to the solution, can deal with several types of neighborhoods but non-connected. However, their deterministic nature may cause them to provide a near *optimal tour* with an effective approximation factor yet too large

for practical applications. Yuan et al. [105] have shown that for the simple case of disjoint balls in $\mathbb{R}^2$ the solution provided by his evolutionary approach always outperforms the approximation algorithm provided in [30], although it generally requires a larger CPU time. Moreover, approximation algorithms are based on the Euclidean norm and mainly deal with neighborhoods in $\mathbb{R}^2$, except for the extension to $\mathbb{R}^m$ proposed in [30], while in many technical fields different definitions of the objective function in higher dimensional spaces might be required.

## 1.2 Contribution

A robotic system is said to be *redundant* if, for reaching a given goal location, it can assume several, possibly infinite, *configurations*. For example, a robot manipulator typically interacts with objects by using a device mounted at the end of its arm and called *end-effector*. If the manipulator has seven or more joints, for placing its end-effector at a given position and orientation in the workspace, its various joints can assume infinitely many possible positions, or configurations. Another example is an UAV that has to acquire a picture of a target. This picture can be taken from an infinite number of positions, or configurations, as far as some given specifications are fulfilled, such as image resolution or distortion.

In practical applications, a redundant robotic system might be asked to reach not only one but $n$ goal locations within an operation cycle. Each goal location $i$ can be represented by a set $\mathcal{Q}_i$ of configurations in the collision free configuration space, $\mathcal{Q}_{free}$, of the robotic system. The set $\mathcal{Q}_i$ is the *neighborhood* for goal $i$. Given two goals $i \neq j$ and two configurations $\boldsymbol{q}_i \in \mathcal{Q}_i$ and $\boldsymbol{q}_j \in \mathcal{Q}_j$, a cost function for the manipulator to move from $\boldsymbol{q}_i$ to $\boldsymbol{q}_j$ is defined. This function is called hereafter *edge weighting function*, and it is indicated as $\mathrm{d}(\boldsymbol{q}_i, \boldsymbol{q}_j)$. In this work we aim to find configurations $\boldsymbol{q}_i \in \mathcal{Q}_i$ for $i = 1, \ldots, n$ and a tour that connects these $n$ configurations such that its total cycle cost is minimized.

This problem is very complex in its full generality, as neighborhoods can have arbitrary shapes determined by the system specifications or physical constraints. Moreover, computing the optimal path and thus the cost to move between two goal locations is in itself a difficult problem since it involves robot kinematics and obstacle avoidance. In this work we study first a simplified version of this problem using analytically defined cost functions. However, we address most of the limitations of previous approaches illustrated in Section 1.1, allowing wider classes both of neighborhood topologies

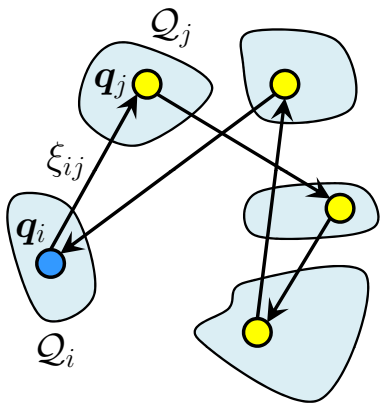



Figure 1.2: An ATSPN instance. The five areas around the vertices shaded in bright blue are the neighborhoods. The directed tour depicted with black arrows is a feasible solution.

and of edge weighting functions. In particular, polyhedra, ellipsoids, and cubic splines in $\mathbb{R}^m$ have been employed as neighborhood, and four types of edge weighting functions have been considered: Manhattan, Euclidean, Quadratic, and Maximum norm. These provide enough flexibility to reasonably estimate the actual system performance for practical purposes. Under these initial assumptions, we propose three novel approaches to search for an *optimal tour*.

The first 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 variables $\xi_{ij}$ for $i, j = 1, \ldots, n$ to represent the possible edges of the tour, as shown in Figure 1.2. The resulting MINLP is non-convex, even when the integrality constraints on the variables $\xi_{ij}$ are relaxed. It follows that only solvers for non-convex MINLP problems can be used for its solution, such as Baron [92], Couenne [12, 20], and Lindoglobal [68].

On the one hand, these solvers struggle to solve relatively small size instances of TSPN. On the other hand, by using a specific feature of the MINLP formulation and customizing the solver by adding specific cut generators and heuristics, we are able to solve instances with up to 16 polyhedral or ellipsoidal neighborhoods far more efficiently. 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. For example, the solver COUENNE (with default settings) requires 733 seconds to solve a TSPN instance with ellipsoids in $\mathbb{R}^2$ (tspn2DP6_2) to optimality, while the proposed approach solves it in a fraction of a second.

In the second approach, the problem is reformulated as a convex MINLP instance under the assumption that the neighborhoods and the edge weighting functions are both convex, and three different formulations are derived. Then, using a convex MINLP optimizer such as BONMIN [17, 20] or MOSEK [4] the problem can be solved to optimality. The best performance is obtained when BONMIN is customized to solve TSPN instances, and a specific cut generator is implemented to efficiently handle specific constraints existing in the MINLP formulation of the problem. Using this exact procedure CPU time is improved up to two orders of magnitude, and instances with up to 20 neighborhoods have been solved to optimality.

A third approach is then proposed to handle larger scale TSPN instances with possibly non-convex neighborhoods. Starting from the MINLP framework used in the previous approaches an hybrid random-key Genetic Algorithm (GA) is specifically developed to search for a near *optimal tour*. The choice of a random-key coding for the GA guarantees feasibility during crossover operations, and avoids to explicitly formulate the subtour elimination constraints of the original MINLP formulation resulting in a more efficient representation of the problem. Moreover, the CPU time of the GA is drastically reduced by replacing commonly used mutation operators with two ad-hoc heuristics: (1) the position of each vertex is fixed, and their sequence is improved by using the Lin-Kernighan heuristic [67]; and (2) the position of each vertex is optimized by solving the Non Linear Programming (NLP) instance resulting from fixing their sequence in the original MINLP formulation.

To evaluate the performance of the proposed GA, TSPN instances were either randomly generated or selected among the CETSP problems proposed by Mennell [74]. In the first case, the GA was able to find the *optimal tour* in all the cases where the solution was also calculated using the MINLP optimizers, while improving the computational performance by orders of magnitude. In the second case, the GA improved the best known near *optimal tour* on average by 1.92%, although the proposed approach is not specifically tailored to solve CETSP instances. Finally, it is worth mentioning that a drawback of the proposed heuristic approach is that no approximation factor for the results can be guaranteed. However, in case an upper bound is required, it is sufficient to first run an approximation algorithm if

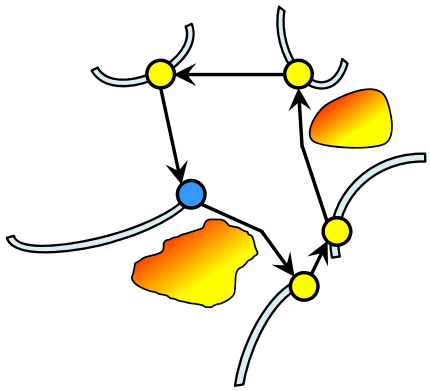


Figure 1.3: Collision-free near *optimal tour* for a TSPN instance. Obstacles are shaded in red/yellow and neighborhoods in bright blue.

available, and then introduce the obtained approximation as a chromosome of the initial population used in the GA.

Finally, to account for collision avoidance and thus to achieve a more realistic evaluation of the edge weighting function for practical applications we embed in the GA a probabilistic path planning technique based on bidirectional Rapidly-exploring Random Trees (RRTs). Figure 1.3 illustrates a TSPN instance where obstacles are considered in the definition of the near *optimal tour*. Moreover, we integrate a dynamic simulator within the GA to optimize the energy consumption of the considered robotic system.

In particular we apply the proposed approach to two test cases. First, we minimize the cycle time of a 7 DOF robotic vision inspection system. The neighborhoods are here approximated using piecewise cubic splines in a seven-dimensional configuration space, and the employed edge weighting function is based on the Quadratic or the Maximum norm. For the specific scenario considered in this work with a 32-goal cycle, experimental tests show an improvement of the current cycle time up to 30%. Second, the flight path and the energy consumption of a quadrotor drone on an urban inspection mission are optimized. The neighborhoods are here defined as three-dimensional polyhedra and the edge weighting function is either the Euclidean or the Quadratic norm. The level of the archived improvement with respect to the results obtained with more traditional optimization techniques generally depends on the number of the neighborhood and their

spatial distribution. Within this work we observe the best improvement on the largest analyzed instance with more than 1,500 goals, where path length and energy consumption are improved up to 38% and 23%, respectively.

The thesis is organized as follows. The used MINLP formulation is presented in chapter 2 together with the first two solution procedures. The hybrid random-key GA is presented in chapter 3. The two considered robotic applications are illustrated in chapters 4 and 5. Finally, chapter 6 contains conclusions and discusses potential future work.

# Chapter 2

# MINLP Solution

In this chapter, we first introduce five different formulations for the considered optimization problem. Then, we propose a heuristic and two exact procedures to solve small scale problems using these formulations and we discuss the attained numerical results.

## 2.1 MINLP Formulation of the TSPN

Dong et al. [28] and Mennell [74] propose a MINLP formulation for the TSPN in case neighborhoods are balls in $\mathbb{R}^2$ and in $\mathbb{R}^3$ and the edge weighting function is the Euclidean or Manhattan norm, but no numerical experiments are reported. In this section a more general formulation is proposed, where convex and non-convex neighborhoods in $\mathbb{R}^m$ are considered, and no constraints are posed on the nature of the edge weighting function. A procedure is then illustrated to reformulate the problem into a convex one if neighborhoods and edge weighting function are convex.

A TSPN instance is given by a set $\mathrm{V} = \{1, 2, \ldots, n\}$ of the indices of the goal locations, a set $\mathcal{Q}_i \subseteq \mathbb{R}^m$ for $i \in \mathrm{V}$ of neighborhoods, and a non-negative distance function $\mathrm{d}(\boldsymbol{u}, \boldsymbol{v})$ for all $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^m$, called edge weighting function hereafter. 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 $n$ variables $\boldsymbol{q}_i \in \mathbb{R}^m$ for all $i \in V$ and $n(n-1)$ binary variables $\xi_{ij}$ 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* [49] (constraints (2.2)-(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 (2.5) below).

We obtain the following MINLP formulation of the ATSPN:

$$\text{minimize :} \quad \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} \xi_{ij} \, \mathrm{d}\left(\boldsymbol{q}_i, \boldsymbol{q}_j\right) \ , \tag{2.1}$$

$$\text{subject to :} \quad \sum_{\substack{i=1 \\ i \neq j}}^{n} \xi_{ij} = 1 \qquad \forall \, j \in \mathrm{V} \ , \tag{2.2}$$

$$\sum_{\substack{j=1 \\ j \neq i}}^{n} \xi_{ij} = 1 \qquad \forall \, i \in \mathrm{V} \ , \tag{2.3}$$

$$\sum_{i \in \mathrm{S}} \sum_{j \in \mathrm{V} \setminus \mathrm{S}} \xi_{ij} \geq 1 \qquad \forall \, \mathrm{S} \subset \mathrm{V} \setminus \{1\}, \, |\mathrm{S}| \geq 2 \ , \tag{2.4}$$

$$\boldsymbol{q}_i \in \mathcal{Q}_i \subseteq \mathbb{R}^m \qquad \forall \, i \in \mathrm{V} \ , \tag{2.5}$$

$$\xi_{ij} \in \{0, 1\} \qquad \forall \, i, j \in \mathrm{V}, \, i \neq j \ , \tag{2.6}$$

$$\boldsymbol{q}_i \in \mathbb{R}^m \qquad \forall \, i \in \mathrm{V} \ . \tag{2.7}$$

The $2n$ assignment constraints (2.2) and (2.3) make sure that each vertex is visited exactly once. The $2^{n-1} - n - 1$ subtour elimination constraints (2.4) 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 (2.5) define the neighborhoods, and the $n(n-1)+n$ constraints (2.6) and (2.7) define the domain of the instance.

The objective function is a non-convex function of the binary and continuous variables. Constraints (2.2)-(2.4) are linear, and the type of the constraints (2.5) 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, convex and quadratic when it is an ellipsoid, or non-convex polynomial equality constraints when it is a cubic spline.

### 2.1.1 Neighborhoods and edge weighting functions

Whereas in the literature mainly balls in $\mathbb{R}^2$ and in $\mathbb{R}^3$ have been used as neighborhoods, in order to address a larger set of technical applications, polyhedra or ellipsoids in $\mathbb{R}^m$ are used in this work as examples of convex and fat domains. Constraints (2.5) becomes thus convex inequality constraints:

$$\mathbf{A}_i \, \boldsymbol{q}_i + \mathbf{b}_i \leq 0 \qquad \forall \, i \in \mathrm{V} \, , \tag{2.8}$$

$$\left(\boldsymbol{q}_i - \mathbf{c}_i\right)^T \, \mathbf{P}_i^{-1} \, \left(\boldsymbol{q}_i - \mathbf{c}_i\right) - 1 \leq 0 \qquad \forall \, i \in \mathrm{V} \, , \tag{2.9}$$

where $\mathbf{A}_i$ is $(l_i \times m)$ matrix, $\mathbf{b}_i$ is a $(l_i \times 1)$ vector with $l_i$ the number of halfspaces of the $i$-th polyhedron, $\mathbf{P}_i$ is a $(m \times m)$ symmetric positive definite matrix, and $\mathbf{c}_i$ is a $(m \times 1)$ vector, center of the $i$-th ellipsoid.

Moreover, for the case of non-convex and non-fat neighborhoods, cubic splines have been used. In this case constraints (2.5) are non-convex equality constraints, where $n$ additional continuous variables $t_i$ and $4n$ spline coefficients $\mathbf{s}_{k,i} \in \mathbb{R}^m$ are employed for the parametrization:

$$\mathbf{s}_{0,i} + \mathbf{s}_{1,i} \, t_i + \mathbf{s}_{2,i} \, t_i^2 + \mathbf{s}_{3,i} \, t_i^3 - \boldsymbol{q}_i = 0 \qquad \forall \, i \in \mathrm{V} \, , \tag{2.10}$$

$$0 \leq \; t_i \leq 1 \qquad \forall \, i \in \mathrm{V} \, , \tag{2.11}$$

$$t_i \in \mathbb{R} \qquad \forall \, i \in \mathrm{V} \, . \tag{2.12}$$

The choice of cubic splines is made based on the fact that these can be easily employed to approximate actual one-dimensional neighborhoods, which might not be analytically defined. Cubic Bézier splines, which are contained in the convex hull of the Bézier polygon defined by the control points $\mathbf{p}_{k,i}$, have been also used as an example:

$$\begin{aligned} \mathbf{p}_{0,i} \, (1 - t_i)^3 + 3 \, \mathbf{p}_{1,i} \, (1 - t_i)^2 \, t_i + \\ 3 \, \mathbf{p}_{2,i} \, (1 - t_i) \, t_i^2 + \mathbf{p}_{3,i} \, t_i^3 - \boldsymbol{q}_i = 0 \qquad \forall \, i \in \mathrm{V} \, . \end{aligned} \tag{2.13}$$

In the present work four types of edge weighting functions $\mathrm{d}\left(\boldsymbol{q}_i, \boldsymbol{q}_j\right)$ have been tested: Manhattan (1), Euclidean (2), Quadratic ($\mathbf{Q}$), and Maximum (inf) norm. The symbol enclosed in parentheses is used hereafter to refer to that specific norm. The four edge weighting functions are:

$$\left\| \boldsymbol{q}_i - \boldsymbol{q}_j \right\|_1 = \sum_{k=1}^{m} \left| q_{i,k} - q_{j,k} \right| \, , \tag{2.14}$$

$$\left\| \boldsymbol{q}_i - \boldsymbol{q}_j \right\|_2 = \sqrt{\left(\boldsymbol{q}_i - \boldsymbol{q}_j\right)^T \, \left(\boldsymbol{q}_i - \boldsymbol{q}_j\right)} \, , \tag{2.15}$$

$$\left\| \boldsymbol{q}_i - \boldsymbol{q}_j \right\|_{\mathbf{Q}} = \sqrt{\left(\boldsymbol{q}_i - \boldsymbol{q}_j\right)^T \, \mathbf{Q} \, \left(\boldsymbol{q}_i - \boldsymbol{q}_j\right)} \, , \tag{2.16}$$

$$\left\| \boldsymbol{q}_i - \boldsymbol{q}_j \right\|_\infty = \max \left\{ \left| q_{i,1} - q_{j,1} \right| , \ldots , \left| q_{i,m} - q_{j,m} \right| \right\} \, . \tag{2.17}$$

where $\mathbf{Q}$ is symmetric positive definite. In industrial robotic applications where the edge weighting function has to account for the traveling time rather than for the distance, Quadratic and Maximum norm are usually employed [44].

### 2.1.2 *First* STSPN formulation

If the graph induced by $\boldsymbol{q}_i$ is undirected, i.e., $\mathrm{d}(\boldsymbol{q}_i, \boldsymbol{q}_j) = \mathrm{d}(\boldsymbol{q}_j, \boldsymbol{q}_i)$, then the TSPN becomes Symmetric (STSPN). Starting from the DFJ formulation of the Symmetric TSP (STSP), the MINLP formulation of the STSPN becomes:

$$\text{minimize :} \quad \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j>i}}^{n} \xi_{ij}\, \mathrm{d}(\boldsymbol{q}_i, \boldsymbol{q}_j) \ , \tag{2.18}$$

$$\text{subject to :} \quad \sum_{j=1}^{i-1} \xi_{ji} + \sum_{j=i+1}^{n} \xi_{ij} = 2 \qquad \forall\, i \in \mathrm{V} \ , \tag{2.19}$$

$$\sum_{i \in \mathrm{S}} \left( \sum_{\substack{j \in \mathrm{V} \setminus \mathrm{S} \\ j<i}} \xi_{ji} + \sum_{\substack{j \in \mathrm{V} \setminus \mathrm{S} \\ j>i}} \xi_{ij} \right) \geq 2 \quad \forall\, \mathrm{S} \subset \mathrm{V} \setminus \{1\},\ |\mathrm{S}| \geq 3 \ , \tag{2.20}$$

$$\xi_{ij} \in \{0, 1\} \qquad \forall\, i, j \in \mathrm{V},\ j > i \ , \tag{2.21}$$

together with constraints (2.5) and (2.7). This formulation is denoted as *first* STSPN formulation hereafter. The binary variables $\xi_{ij}$ are now $n(n-1)/2$ and the assignment problem constraints, (2.19), are $n$.

One difficulty in handling the above formulation is the number of subtour elimination constraints (2.20). Although their number is exponential in the size of the instance $n$, they can be handled efficiently implicitly using a cutting plane approach [81]. A heuristic and an exact solution procedure based on this formulation are proposed in Section 2.2, and they are tested on randomly generated TSPN instances with ellipsoids and polyhedra in $\mathbb{R}^2$ and $\mathbb{R}^3$, and with the Euclidean norm. First the standard Branch-and-Bound algorithm for convex MINLP available in Bonmin [17] is used to find a heuristic solution. This is then used to define the initial point and the upper bound for a customized version of the Branch-and-Bound algorithm available in Couenne [12].

### 2.1.3 *Second* STSPN formulation

A drawback of the MINLP formulation of the TSPN is the non-convexity of the NLP relaxation. This is caused by two factors: the non-convexity of the objective function, (2.1) or (2.18), regardless the nature of the edge weighting function $\mathrm{d}(\boldsymbol{q}_i, \boldsymbol{q}_j)$, and the potential non-convexity of the neighborhoods, (2.5). To overcome this limitation an alternative formulation can be introduced for the STSPN case. If $n(n-1)/2$ additional real variables $d_{ij}$ are introduced, the MINLP formulation becomes:

$$\text{minimize :} \quad \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j>i}}^{n} d_{ij} \ , \tag{2.22}$$

$$\text{subject to :} \quad \begin{cases} d_{ij} = 0 & \text{if} \quad \xi_{ij} = 0 \\ \mathrm{d}(\boldsymbol{q}_i, \boldsymbol{q}_j) - d_{ij} \leq 0 & \text{if} \quad \xi_{ij} = 1 \end{cases} \qquad \forall\, i, j \in \mathrm{V},\, j > i \ , \tag{2.23}$$

$$d_{ij} \in \mathbb{R}_+ \qquad \forall\, i, j \in \mathrm{V},\, j > i \ , \tag{2.24}$$

together with constraints (2.19), (2.20), (2.5), (2.21), and (2.7), where $\mathbb{R}_+ = \{x \in \mathbb{R} \mid x \geq 0\}$. After replacing the disjunction (2.23) with the following big-M relaxation [55]:

$$\mathrm{d}(\boldsymbol{q}_i, \boldsymbol{q}_j) \leq d_{ij} + (1 - \xi_{ij})\, \mathrm{d}_{\mathrm{MAX},ij} \qquad \forall\, i, j \in \mathrm{V},\, j > i \ . \tag{2.25}$$

we denote the resulting formulation as the *second* STSPN.

We observe that the number of additional variables $d_{ij}$ does not depend on the domain dimension $m$, and that the objective function (2.22) is a linear function of these variables. The value of $\mathrm{d}_{\mathrm{MAX},ij}$ corresponds to the maximum possible value of $\mathrm{d}(\boldsymbol{q}_i, \boldsymbol{q}_j)$ for $\boldsymbol{q}_i \in \mathcal{Q}_i$ and $\boldsymbol{q}_j \in \mathcal{Q}_j$. If the edge weighting function is convex, then the $n(n-1)/2$ constraints (2.25) are convex since each one of those is the positive sum of a convex function and an affine function. Finally, if the neighborhoods (2.5) are convex, then the entire problem is convex.

The following lemma holds:

**Lemma 1.** *If $(\boldsymbol{q}_i^\star, \xi_{ij}^\star, d_{ij}^\star)$ is an optimal point for the second STSPN formulation, then $(\boldsymbol{q}_i^\star, \xi_{ij}^\star)$ is optimal for the first STSPN formulation, where $i, j \in \mathrm{V},\, j > i$.*

*Proof.* Let us call $O_{II}^*$ the optimal value of the objective function (2.22) calculated at $d_{ij}^\star$, and let us call $O_I^*$ the value, not necessarily optimal, of the objective function (2.18) calculated at $(\boldsymbol{q}_i^\star, \xi_{ij}^\star)$. Since constraints (2.25) hold, it is true that $O_I^* \leq O_{II}^*$.

Suppose now that there exists a point $(\widetilde{\boldsymbol{q}}_i, \widetilde{\xi}_{ij})$ such that $\widetilde{O}_I < O_I^*$, where $\widetilde{O}_I$ is the value of the objective function (2.18) calculated at $(\widetilde{\boldsymbol{q}}_i, \widetilde{\xi}_{ij})$. Suppose that this point is a feasible point for the *first* formulation. If the following values are assigned to $\widetilde{d}_{ij}$:

$$\begin{cases} \widetilde{d}_{ij} = 0 & \text{if} \quad \widetilde{\xi}_{ij} = 0 \ , \\ \widetilde{d}_{ij} = \mathrm{d}\,(\widetilde{\boldsymbol{q}}_i, \widetilde{\boldsymbol{q}}_j) & \text{if} \quad \widetilde{\xi}_{ij} = 1 \ , \end{cases}$$

then constraints (2.25) are satisfied, and $(\widetilde{\boldsymbol{q}}_i, \widetilde{\xi}_{ij}, \widetilde{d}_{ij})$ is a feasible point also for the *second* formulation. Moreover, it holds:

$$\widetilde{O}_{II} = \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j>i}}^{n} \widetilde{d}_{ij} = \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j>i}}^{n} \widetilde{\xi}_{ij}\ \mathrm{d}\,(\widetilde{\boldsymbol{q}}_i, \widetilde{\boldsymbol{q}}_j) = \widetilde{O}_I < O_I^* \leq O_{II}^* \ .$$

Finally, $\widetilde{O}_{II} < O_{II}^*$ contradicts the definition of optimality, for which it must hold $\widetilde{O}_{II} \geq O_{II}^*$. Therefore, $(\boldsymbol{q}_i^\star, \xi_{ij}^\star)$ is an optimal point for the *first* STSPN formulation.

Note that this Lemma holds also if the original disjunction (2.23) is used in the *second* STSPN formulation instead of the big-M relaxation (2.25). □

An exact solution procedure based on the *second* STSPN formulation is proposed in Section 2.3, and it is tested on randomly generated STSPN instances with ellipsoids in $\mathbb{R}^2$ and $\mathbb{R}^3$, and with the Manhattan, the Euclidean, and the Maximum norm. Using a customized version of Outer-Approximation based Branch-and-Cut algorithm for convex MINLP available in Bonmin [17], STSPN instances with up to 20 convex neighborhoods can be solved. The heuristic results obtained by solving the *first* STSPN formulation are used here as initial point and as upper bound for the algorithm, and all the subtour elimination constraints already found are added to the initial formulation of the problem.

#### 2.1.3.1 *Second* STSPN formulation for different norms

The *second* STSPN formulation can be reformulated in case of Euclidean or Quadratic norm by substituting Equation (2.15) or (2.16) into constraints (2.25), which become Second Order Conic (SOC) constraints.

For the Maximum norm, constraints (2.25) are substituted by $mn(n-1)$

linear constraints:

$$q_{i,k} - q_{j,k} \leq d_{ij} + (1 - \xi_{ij})\, \mathrm{d}_{\mathrm{MAX},ijk} \qquad \forall\, i, j \in \mathrm{V},\, j > i,\, \forall\, k \in \{1, \ldots, m\}\,, \tag{2.26}$$

$$q_{j,k} - q_{i,k} \leq d_{ij} + (1 - \xi_{ij})\, \mathrm{d}_{\mathrm{MAX},ijk} \qquad \forall\, i, j \in \mathrm{V},\, j > i,\, \forall\, k \in \{1, \ldots, m\}\,. \tag{2.27}$$

Finally for the Manhattan norm, the *second* STSPN formulation becomes:

$$\text{minimize :} \quad \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j>i}}^{n} \sum_{k=1}^{m} d_{ijk}\,, \tag{2.28}$$

$$\text{subject to :} \quad q_{i,k} - q_{j,k} \leq d_{ijk} + (1 - \xi_{ij})\, \mathrm{d}_{\mathrm{MAX},ijk} \qquad \forall\, i, j \in \mathrm{V},\, j > i,\, \forall\, k \in \{1, \ldots, m\}\,, \tag{2.29}$$

$$q_{j,k} - q_{i,k} \leq d_{ijk} + (1 - \xi_{ij})\, \mathrm{d}_{\mathrm{MAX},ijk} \qquad \forall\, i, j \in \mathrm{V},\, j > i,\, \forall\, k \in \{1, \ldots, m\}\,, \tag{2.30}$$

$$d_{ijk} \in \mathbb{R}_{+} \qquad \forall\, i, j \in \mathrm{V},\, j > i,\, \forall\, k \in \{1, \ldots, m\}\,, \tag{2.31}$$

together with constraints (2.19), (2.20), (2.5), (2.21), and (2.7). In this case $mn(n-1)/2$ additional variables $d_{ijk}$ are used.

### 2.1.4 *Third* STSPN formulation

In the *second* STSPN formulation a big-M relaxation of the disjunction (2.23) is introduced. However, a stronger relaxation may lead to better performance of the solver especially for MINLP cases where the disjunction is driven by binary (indicator) variables [48].

If the neighborhoods $\mathcal{Q}_i$ are bounded, without loss of generality we can assume that $q_{i,k} \geq 0\ \forall\, i \in \mathrm{V}$ and $\forall k \in \{1, \ldots, m\}$. Moreover, there exists a positive constant $\mathrm{d}_{\mathrm{MAX}}$ such that $d_{ij} \leq \mathrm{d}_{\mathrm{MAX}} \quad \forall\, i, j \in \mathrm{V},\, j > i$. The following lemma holds.

**Lemma 2.** *If $\tilde{\boldsymbol{q}}_{i,j}$ are $n(n-1)$ additional variables, which represent $\boldsymbol{q}_i$ when the edge $i-j$ or $j-i$ is in the tour, constraints (2.23) are equivalent to the*

*following set of constraints:*

$$\begin{cases} d_{ij} = 0 & \\ \tilde{\boldsymbol{q}}_{i,j} = \mathbf{0} & \text{if} \quad \xi_{ij} = 0 \\ \tilde{\boldsymbol{q}}_{j,i} = \mathbf{0} & \\ & \forall\, i, j \in \mathrm{V},\ j > i\ . \\ \mathrm{d}\left(\tilde{\boldsymbol{q}}_{i,j}, \tilde{\boldsymbol{q}}_{j,i}\right) - d_{ij} \leq 0 & \\ \tilde{\boldsymbol{q}}_{i,j} \in \mathcal{Q}_i & \text{if} \quad \xi_{ij} = 1 \\ \tilde{\boldsymbol{q}}_{j,i} \in \mathcal{Q}_j & \end{cases} \tag{2.32}$$

$$\boldsymbol{q}_i = \frac{1}{2} \sum_{\substack{j=1 \\ j \neq i}}^{n} \tilde{\boldsymbol{q}}_{i,j} \qquad \forall\, i \in \mathrm{V}\ , \tag{2.33}$$

$$\tilde{\boldsymbol{q}}_{i,j} \leq \boldsymbol{q}_i \qquad \forall\, i, j \in \mathrm{V},\ j \neq i\ , \tag{2.34}$$

*together with constraints (2.19).*

*Proof.* Constraints (2.19) and (2.21) insure that only two among the $(n-1)$ variables $\xi_{ij}$ incident to a neighborhood are non-zero. Consequently, constraints (2.32) insure that for each neighborhood $\mathcal{Q}_i$ only two among the $(n-1)$ variables $\tilde{\boldsymbol{q}}_{i,j}$ are non-zero, i.e., the two variables corresponding to the two edges of the tour incident to $\mathcal{Q}_i$. Moreover, constraints (2.33) and (2.34) insure that these two variables are both equal to $\boldsymbol{q}_i$, since the linear system $q_a + q_b = 2q,\ q_a \leq q; q_b \leq q$ admits the only solution $q_a = q_b = q$. Therefore, if $\xi_{ij} = 1$, then $\tilde{\boldsymbol{q}}_{i,j} = \boldsymbol{q}_i$, $\tilde{\boldsymbol{q}}_{j,i} = \boldsymbol{q}_j$, and constraints (2.32) become $\mathrm{d}\left(\boldsymbol{q}_i, \boldsymbol{q}_j\right) - d_{ij} \leq 0$. □

We indicate the subset of the domain defined by constraints (2.32) as $D_{ij} = D^0_{ij} \cup D^0_{ij}$, where:

$$D^0_{ij} = \{(\xi_{ij}, d_{ij}, \tilde{\boldsymbol{q}}_{i,j}, \tilde{\boldsymbol{q}}_{j,i}) \mid \xi_{ij} = 0, d_{ij} = 0, \tilde{\boldsymbol{q}}_{i,j} = \mathbf{0}, \tilde{\boldsymbol{q}}_{j,i} = \mathbf{0}\}\ , \tag{2.35}$$

$$D^1_{ij} = \{(\xi_{ij}, d_{ij}, \tilde{\boldsymbol{q}}_{i,j}, \tilde{\boldsymbol{q}}_{j,i}) \mid \xi_{ij} = 1, 0 \leq d_{ij} \leq \mathrm{d}_{\mathrm{MAX}}, \mathrm{d}\left(\tilde{\boldsymbol{q}}_{i,j}, \tilde{\boldsymbol{q}}_{j,i}\right) - d_{ij} \leq 0, \\ \tilde{\boldsymbol{q}}_{i,j} \in \mathcal{Q}_i, \tilde{\boldsymbol{q}}_{j,i} \in \mathcal{Q}_j\}\ . \tag{2.36}$$

Sets $D^0_{ij}$ and $D^1_{ij}$ are clearly bounded, and if the edge weighting function, $\mathrm{d}(\cdot)$, is convex then they are both convex. We are now interested in the convex hull of $D_{ij}$. Similarly to the results presented by Günlük and Linderoth [48], we can introduce the following lemmas.

**Lemma 3.** *If $D^1_{ij}$ is convex, then* $\mathrm{conv}(D_{ij}) = D^0_{ij} \cup D^-_{ij}$*, where:*

$$D^-_{ij} = \{(\xi_{ij}, d_{ij}, \tilde{\boldsymbol{q}}_{i,j}, \tilde{\boldsymbol{q}}_{j,i}) \mid 0 < \xi_{ij} \leq 1\ ,\ 0 \leq d_{ij}/\xi_{ij} \leq \mathrm{d}_{\mathrm{MAX}}, \\ \mathrm{d}\left(\tilde{\boldsymbol{q}}_{i,j}/\xi_{ij}, \tilde{\boldsymbol{q}}_{j,i}/\xi_{ij}\right) - d_{ij}/\xi_{ij} \leq 0, \tilde{\boldsymbol{q}}_{i,j}/\xi_{ij} \in \mathcal{Q}_i, \tilde{\boldsymbol{q}}_{j,i}/\xi_{ij} \in \mathcal{Q}_j\}\ .$$

*Proof.* Since sets $D_{ij}^0$ and $D_{ij}^1$ are bounded and convex, the convex hull of $D_{ij}$ can be calculated as:

$$\begin{aligned}\operatorname{conv}(D_{ij}) = \{&(\xi_{ij}, d_{ij}, \tilde{\boldsymbol{q}}_{i,j}, \tilde{\boldsymbol{q}}_{j,i}) = (1-\alpha)(\xi_{ij}^0, d_{ij}^0, \tilde{\boldsymbol{q}}_{i,j}^0, \tilde{\boldsymbol{q}}_{j,i}^0) + \\ &\alpha(\xi_{ij}^1, d_{ij}^1, \tilde{\boldsymbol{q}}_{i,j}^1, \tilde{\boldsymbol{q}}_{j,i}^1) \mid (\xi_{ij}^0, d_{ij}^0, \tilde{\boldsymbol{q}}_{i,j}^0, \tilde{\boldsymbol{q}}_{j,i}^0) \in D_{ij}^0, \\ &(\xi_{ij}^1, d_{ij}^1, \tilde{\boldsymbol{q}}_{i,j}^1, \tilde{\boldsymbol{q}}_{j,i}^1) \in D_{ij}^1, 0 \leq \alpha \leq 1\} \,. \end{aligned} \tag{2.37}$$

From Equations (2.35) and (2.37) we observe that $\alpha = \xi_{ij}$. On the one hand, if $\xi_{ij} > 0$, we can also observe that $d_{ij}^1 = d_{ij}/\xi_{ij}$, $\tilde{\boldsymbol{q}}_{i,j}^1 = \tilde{\boldsymbol{q}}_{i,j}/\xi_{ij}$, and $\tilde{\boldsymbol{q}}_{j,i}^1 = \tilde{\boldsymbol{q}}_{j,i}/\xi_{ij}$. The additional variables $\alpha$, $\xi_{ij}^0$, $\xi_{ij}^1$, $d_{ij}^0$, $d_{ij}^1$, $\tilde{\boldsymbol{q}}_{i,j}^0$, $\tilde{\boldsymbol{q}}_{j,i}^0$, $\tilde{\boldsymbol{q}}_{i,j}^1$, and $\tilde{\boldsymbol{q}}_{j,i}^1$ can thus be projected out from (2.37), and the set $D_{ij}^-$ is obtained using Equation (2.36). On the other hand, if $\xi_{ij} = 0$, then $(0, d_{ij}, \tilde{\boldsymbol{q}}_{i,j}, \tilde{\boldsymbol{q}}_{j,i}) \in \operatorname{conv}(D_{ij})$ if and only if $(0, d_{ij}, \tilde{\boldsymbol{q}}_{i,j}, \tilde{\boldsymbol{q}}_{j,i}) \in D_{ij}^0$. Therefore, $\operatorname{conv}(D_{ij}) = D_{ij}^0 \cup D_{ij}^-$. □

**Lemma 4.** *It holds:* $\operatorname{closure}(D_{ij}^-) = \operatorname{conv}(D_{ij})$.

*Proof.* Let $\{\alpha_s\} \subset (0,1)$ be an arbitrary sequence converging to 0, and let

$$\begin{aligned} D_p(\alpha) = \{&(d_{ij}, \tilde{\boldsymbol{q}}_{i,j}, \tilde{\boldsymbol{q}}_{j,i}) \mid 0 \leq d_{ij}/\alpha \leq \mathrm{d}_{\mathrm{MAX}}, \\ &\mathrm{d}\left(\tilde{\boldsymbol{q}}_{i,j}/\alpha, \tilde{\boldsymbol{q}}_{j,i}/\alpha\right) - d_{ij}/\alpha \leq 0, \tilde{\boldsymbol{q}}_{i,j}/\alpha \in \mathcal{Q}_i, \tilde{\boldsymbol{q}}_{j,i}/\alpha \in \mathcal{Q}_j\} \,. \end{aligned}$$

Since, by construction, $D_p(\alpha_s) \neq \emptyset$, there exists a corresponding sequence $\{(d_{ij}, \tilde{\boldsymbol{q}}_{i,j}, \tilde{\boldsymbol{q}}_{j,i})_s\}$ such that $(d_{ij}, \tilde{\boldsymbol{q}}_{i,j}, \tilde{\boldsymbol{q}}_{j,i})_s \in D_p(\alpha_s)$. Since $0 \leq d_{ij} \leq \alpha\, \mathrm{d}_{\mathrm{MAX}}$, $(d_{ij})_s$ converges to 0, and since $\mathcal{Q}_i$ and $\mathcal{Q}_j$ are bounded sets, then $(\tilde{\boldsymbol{q}}_{i,j})_s$ and $(\tilde{\boldsymbol{q}}_{j,i})_s$ also converge to $\mathbf{0}$, as illustrated in Appendix A.1. Finally, it holds:

$$\lim_{\alpha \to 0} D_p(\alpha) = \{(d_{ij}, \tilde{\boldsymbol{q}}_{i,j}, \tilde{\boldsymbol{q}}_{j,i}) \mid d_{ij} = 0, \tilde{\boldsymbol{q}}_{i,j} = \mathbf{0}, \tilde{\boldsymbol{q}}_{j,i} = \mathbf{0}\} \,.$$

Therefore, $D_{ij}^0$ is clearly contained in $\operatorname{closure}(D_{ij}^-)$, and, according to Lemma 3, $\operatorname{closure}(D_{ij}^-) = \operatorname{conv}(D_{ij})$. □

Finally, if polyhedra or ellipsoids are used as neighborhoods and a norm is used as edge weighting function the complete relaxed *third* formulation of

the STSPN is given by:

$$
\begin{aligned}
\text{minimize :} \quad & \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j>i}}^{n} d_{ij} \ , & \\
\text{subject to :} \quad & \sum_{j=1}^{i-1} \xi_{ji} + \sum_{j=i+1}^{n} \xi_{ij} = 2 & \forall\, i \in \mathrm{V} \ , \\
& \sum_{i \in \mathrm{S}} \left( \sum_{\substack{j \in \mathrm{V} \setminus \mathrm{S} \\ j<i}} \xi_{ji} + \sum_{\substack{j \in \mathrm{V} \setminus \mathrm{S} \\ j>i}} \xi_{ij} \right) \geq 2 & \forall\, \mathrm{S} \subset \mathrm{V} \setminus \{1\},\, |\mathrm{S}| \geq 3 \ , \\
& \boldsymbol{q}_i = \frac{1}{2} \sum_{\substack{j=1 \\ j \neq i}}^{n} \tilde{\boldsymbol{q}}_{i,j} & \forall\, i \in \mathrm{V} \ , \\
& \tilde{\boldsymbol{q}}_{i,j} \leq \boldsymbol{q}_i & \forall\, i,j \in \mathrm{V},\, j \neq i \ , \\
& \| \tilde{\boldsymbol{q}}_{i,j} - \tilde{\boldsymbol{q}}_{j,i} \| \leq d_{ij} & \forall\, i,j \in \mathrm{V},\, j > i \ , \\
& \tilde{\boldsymbol{q}}_{i,j} \in \tilde{\mathcal{Q}}_i(\xi_{ij}) \subseteq \mathbb{R}^m_+ & \forall\, i,j \in \mathrm{V},\, j > i \ , \\
& \tilde{\boldsymbol{q}}_{j,i} \in \tilde{\mathcal{Q}}_j(\xi_{ij}) \subseteq \mathbb{R}^m_+ & \forall\, i,j \in \mathrm{V},\, j > i \ , \\
& d_{ij} \leq \xi_{ij}\ \mathrm{d_{MAX}} & \forall\, i,j \in \mathrm{V},\, j > i \ , \\
& 0 \leq \xi_{ij} \leq 1 & \forall\, i,j \in \mathrm{V},\, j > i \ , \\
& \boldsymbol{q}_i \in \mathbb{R}^m & \forall\, i \in \mathrm{V} \ , \\
& \tilde{\boldsymbol{q}}_{i,j} \in \mathbb{R}^m & \forall\, i,j \in \mathrm{V},\, i \neq j \ , \\
& d_{ij} \in \mathbb{R}_+ & \forall\, i,j \in \mathrm{V},\, j > i \ ,
\end{aligned}
$$

where $\tilde{\mathcal{Q}}_i(\xi)$ can be defined as:

$$
\begin{aligned}
& \mathbf{A}_i\, \boldsymbol{q}_i + \xi\ \mathbf{b}_i \leq 0 \quad \text{or} \\
& (\boldsymbol{q}_i - \xi\ \mathbf{c}_i)^T\ \mathbf{P}_i^{-1}\ (\boldsymbol{q}_i - \xi\ \mathbf{c}_i) - \xi^2 \leq 0 \ .
\end{aligned}
$$

### 2.1.5 *Fourth* STSPN formulation

In this Section we propose an alternative method to obtain a stronger relaxation for the STSPN deriving directly the convex hull of the disjunction (2.23).

Assuming that the neighborhoods, $\mathcal{Q}_i$, are bounded, and thus $d_{ij} \leq \mathrm{d_{MAX}}\ \forall\, i,j \in \mathrm{V},\, j > i$, we denote the subset of the domain defined by

constraints (2.23) as $\bar{D}_{ij} = \bar{D}^0_{ij} \cup \bar{D}^1_{ij}$, where:

$$\bar{D}^0_{ij} = \{(\xi_{ij}, d_{ij}, \boldsymbol{q}_i, \boldsymbol{q}_j) \mid \xi_{ij} = 0, d_{ij} = 0, \boldsymbol{q}_i \in \mathcal{Q}_i, \boldsymbol{q}_j \in \mathcal{Q}_j\} \,, \tag{2.38}$$

$$\begin{aligned}\bar{D}^1_{ij} = \{(\xi_{ij}, d_{ij}, \boldsymbol{q}_i, \boldsymbol{q}_j) \mid \xi_{ij} = 1, 0 \leq d_{ij} \leq \mathrm{d}_{\text{MAX}}, \mathrm{d}\,(\boldsymbol{q}_i, \boldsymbol{q}_j) \leq d_{ij},\\ \boldsymbol{q}_i \in \mathcal{Q}_i, \boldsymbol{q}_j \in \mathcal{Q}_j\} \,.\end{aligned} \tag{2.39}$$

Sets $\bar{D}^0_{ij}$ and $\bar{D}^1_{ij}$ are clearly bounded, and if the edge weighting function, $\mathrm{d}(\cdot)$, is convex then they are both convex. As in the previous section, we are now interested in the convex hull of $D_{ij}$.

**Lemma 5.** *If $\bar{\boldsymbol{q}}_{i,j}$ are $n(n-1)$ additional variables and $\bar{D}^1_{ij}$ is convex, then* $\mathrm{conv}(\bar{D}_{ij}) = \bar{D}^0_{ij} \cup \bar{D}^-_{ij} \cup \bar{D}^1_{ij}$, *where:*

$$\begin{aligned}\bar{D}^-_{ij} = \{(\xi_{ij}, d_{ij}, \boldsymbol{q}_i, \boldsymbol{q}_j) \mid 0 < \xi_{ij} < 1, 0 \leq d_{ij}/\xi_{ij} \leq \mathrm{d}_{\text{MAX}},\\ \mathrm{d}\,(\boldsymbol{q}_i/\xi_{ij} - \bar{\boldsymbol{q}}_{i,j}/\xi_{ij} \;,\; \boldsymbol{q}_j/\xi_{ij} - \bar{\boldsymbol{q}}_{j,i}/\xi_{ij}) \leq d_{ij}/\xi_{ij},\\ \boldsymbol{q}_i/\xi_{ij} - \bar{\boldsymbol{q}}_{i,j}/\xi_{ij} \in \mathcal{Q}_i, \boldsymbol{q}_j/\xi_{ij} - \bar{\boldsymbol{q}}_{j,i}/\xi_{ij} \in \mathcal{Q}_j,\\ \bar{\boldsymbol{q}}_{i,j}/(1-\xi_{ij}) \in \mathcal{Q}_i, \bar{\boldsymbol{q}}_{j,i}/(1-\xi_{ij}) \in \mathcal{Q}_j\} \,.\end{aligned}$$

*Proof.* Since sets $\bar{D}^0_{ij}$ and $\bar{D}^1_{ij}$ are bounded and convex, the convex hull of $\bar{D}_{ij}$ can be calculated as:

$$\begin{aligned}\mathrm{conv}(\bar{D}_{ij}) = \{(\xi_{ij}, d_{ij}, \boldsymbol{q}_i, \boldsymbol{q}_j) = (1-\alpha)(\xi^0_{ij}, d^0_{ij}, \boldsymbol{q}^0_i, \boldsymbol{q}^0_j) + \alpha(\xi^1_{ij}, d^1_{ij}, \boldsymbol{q}^1_i, \boldsymbol{q}^1_j) \mid\\ (\xi^0_{ij}, d^0_{ij}, \boldsymbol{q}^0_i, \boldsymbol{q}^0_j) \in \bar{D}^0_{ij}, (\xi^1_{ij}, d^1_{ij}, \boldsymbol{q}^1_i, \boldsymbol{q}^1_j) \in \bar{D}^1_{ij}, 0 \leq \alpha \leq 1\} \,.\end{aligned} \tag{2.40}$$

From Equations (2.38) and (2.40) we can observe that $\alpha = \xi_{ij}$ and $d^1_{ij} = d_{ij}/\xi_{ij}$ for $\xi_{ij} > 0$. Therefore, the additional variables $\alpha$, $\xi^0_{ij}$, $\xi^1_{ij}$, $d^0_{ij}$, and $d^1_{ij}$ can be projected out from (2.40). Moreover, if we set $\boldsymbol{q}^0_i = \hat{\boldsymbol{q}}_i \in \mathcal{Q}_i$ and $\boldsymbol{q}^0_j = \hat{\boldsymbol{q}}_j \in \mathcal{Q}_j$, than we obtain $\xi_{ij}\boldsymbol{q}^1_i = \boldsymbol{q}_i + (\xi_{ij} - 1)\hat{\boldsymbol{q}}_i$ and $\xi_{ij}\boldsymbol{q}^1_j = \boldsymbol{q}_j + (\xi_{ij} - 1)\hat{\boldsymbol{q}}_j$, and the following set can be derived using Equation (2.39):

$$\begin{aligned}\hat{D}^-_{ij} = \{(\xi_{ij}, d_{ij}, \boldsymbol{q}_i, \boldsymbol{q}_j) \mid 0 < \xi_{ij} \leq 1, 0 \leq d_{ij}/\xi_{ij} \leq \mathrm{d}_{\text{MAX}},\\ \mathrm{d}\,(\boldsymbol{q}_i/\xi_{ij} + (\xi_{ij}-1)/\xi_{ij}\hat{\boldsymbol{q}}_i \;,\; \boldsymbol{q}_j/\xi_{ij} + (\xi_{ij}-1)/\xi_{ij}\hat{\boldsymbol{q}}_j) \leq d_{ij}/\xi_{ij},\\ \boldsymbol{q}_i/\xi_{ij} + (\xi_{ij}-1)/\xi_{ij}\hat{\boldsymbol{q}}_i \in \mathcal{Q}_i, \boldsymbol{q}_j/\xi_{ij} + (\xi_{ij}-1)/\xi_{ij}\hat{\boldsymbol{q}}_j \in \mathcal{Q}_j,\\ \hat{\boldsymbol{q}}_i \in \mathcal{Q}_i \;, \hat{\boldsymbol{q}}_j \in \mathcal{Q}_j\}\end{aligned} \tag{2.41}$$

Finally, if we introduce the additional variables $\bar{\boldsymbol{q}}_{i,j}$, if we perform the substitutions $(\xi_{ij} - 1)\hat{\boldsymbol{q}}_i = -\bar{\boldsymbol{q}}_{i,j}$ and $(\xi_{ij} - 1)\hat{\boldsymbol{q}}_j = -\bar{\boldsymbol{q}}_{j,i}$ in (2.41), and if we assume $\xi_{ij} < 1$, then the set $\bar{D}^-_{ij}$ can be directly derived from $\hat{D}^-_{ij}$.

Moreover, since $\bar{D}^0_{ij}$ is convex if $\xi_{ij} = 0$ then $(0, d_{ij}, \boldsymbol{q}_i, \boldsymbol{q}_j) \in \mathrm{conv}(\bar{D}_{ij})$ if and only if $(0, d_{ij}, \boldsymbol{q}_i, \boldsymbol{q}_j) \in \bar{D}^0_{ij}$. Similarly, since $\bar{D}^1_{ij}$ is convex if $\xi_{ij} = 1$ then $(1, d_{ij}, \boldsymbol{q}_i, \boldsymbol{q}_j) \in \mathrm{conv}(\bar{D}_{ij})$ if and only if $(1, d_{ij}, \boldsymbol{q}_i, \boldsymbol{q}_j) \in \bar{D}^1_{ij}$. Therefore, $\mathrm{conv}(\bar{D}_{ij}) = \bar{D}^0_{ij} \cup \bar{D}^-_{ij} \cup \bar{D}^1_{ij}$. □

**Lemma 6.** *It holds:* $\mathrm{closure}(\bar{D}^-_{ij}) = \mathrm{conv}(D_{ij})$.

*Proof.* Let $\{\alpha^0_s\} \subset (0,1)$ be an arbitrary sequence converging to 0 and $\{\alpha^1_s\} \subset (0,1)$ be an arbitrary sequence converging to 1. Let

$$\begin{aligned}\bar{D}_p(\alpha) = \{(d_{ij}, \boldsymbol{q}_i, \boldsymbol{q}_j) \mid\ & 0 \le d_{ij}/\alpha \le \mathrm{d_{MAX}},\\ & \mathrm{d}\left((\boldsymbol{q}_i - \bar{\boldsymbol{q}}_{i,j})/\alpha\ ,\ (\boldsymbol{q}_j - \bar{\boldsymbol{q}}_{j,i})/\alpha\right) - d_{ij}/\alpha \le 0,\\ & (\boldsymbol{q}_i - \bar{\boldsymbol{q}}_{i,j})/\alpha \in \mathcal{Q}_i,\ (\boldsymbol{q}_j - \bar{\boldsymbol{q}}_{j,i})/\alpha \in \mathcal{Q}_j,\\ & \bar{\boldsymbol{q}}_{i,j}/(1-\alpha) \in \mathcal{Q}_i,\ \bar{\boldsymbol{q}}_{j,i}/(1-\alpha) \in \mathcal{Q}_j\}\ .\end{aligned}$$

Since, by construction, $D_p(\alpha_s) \neq \emptyset$, there exists a sequence $\{(d_{ij}, \boldsymbol{q}_i, \boldsymbol{q}_j, \bar{\boldsymbol{q}}_{i,j}, \bar{\boldsymbol{q}}_{j,i})^0_s\}$ such that $(d_{ij}, \boldsymbol{q}_i, \boldsymbol{q}_j)^0_s \in \bar{D}_p(\alpha^0_s)$. Since $0 \le d_{ij} \le \alpha\, \mathrm{d_{MAX}}$, $(d_{ij})_s$ converges to 0, and since $\mathcal{Q}_i$ and $\mathcal{Q}_j$ are bounded sets, then $\left((\boldsymbol{q}_i)^0_s - (\bar{\boldsymbol{q}}_{i,j})^0_s\right)$ and $\left((\boldsymbol{q}_j)^0_s - (\bar{\boldsymbol{q}}_{j-1})^0_s\right)$ also converge to $\mathbf{0}$. Finally, it holds:

$$\lim_{\alpha \to 0} \bar{D}_p(\alpha) = \{(d_{ij}, \boldsymbol{q}_i, \boldsymbol{q}_j) \mid d_{ij} = 0,\ \boldsymbol{q}_i \in \mathcal{Q}_i,\ \boldsymbol{q}_j \in \mathcal{Q}_j\}\ .$$

Similarly, there exists a sequence $\{(d_{ij}, \boldsymbol{q}_i, \boldsymbol{q}_j, \bar{\boldsymbol{q}}_{i,j}, \bar{\boldsymbol{q}}_{j,i})^1_s\}$ such that $(d_{ij}, \boldsymbol{q}_i, \boldsymbol{q}_j)^1_s \in \bar{D}_p(\alpha^1_s)$. Since $\mathcal{Q}_i$ and $\mathcal{Q}_j$ are bounded sets, then $(\bar{\boldsymbol{q}}_{i,j})^1_s$ and $(\bar{\boldsymbol{q}}_{j-1})^1_s$ converge to $\mathbf{0}$. Finally, it holds:

$$\begin{aligned}\lim_{\alpha \to 1} \bar{D}_p(\alpha) = \{&(d_{ij}, \boldsymbol{q}_i, \boldsymbol{q}_j) \mid 0 \le d_{ij} \le \mathrm{d_{MAX}},\ \mathrm{d}(\boldsymbol{q}_i,\ \boldsymbol{q}_j) - d_{ij} \le 0,\\ & \boldsymbol{q}_i \in \mathcal{Q}_i,\ \boldsymbol{q}_j \in \mathcal{Q}_j\}\ .\end{aligned}$$

In conclusion, $\bar{D}^0_{ij}$ and $\bar{D}^1_{ij}$ are clearly contained in $\mathrm{closure}(\bar{D}^-_{ij})$, and, according to Lemma 5, $\mathrm{closure}(D^-_{ij}) = \mathrm{conv}(D_{ij})$. □

Finally, if polyhedra or ellipsoids are used as neighborhoods and a norm is used as edge weighting function the complete relaxed *fourth* formulation

of the STSPN is given by:

$$
\begin{aligned}
\text{minimize :} \quad & \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j>i}}^{n} d_{ij} \ , \\
\text{subject to :} \quad & \sum_{j=1}^{i-1} \xi_{ji} + \sum_{j=i+1}^{n} \xi_{ij} = 2 && \forall\, i \in \mathrm{V} \ , \\
& \sum_{i \in \mathrm{S}} \Bigg( \sum_{\substack{j \in \mathrm{V} \setminus \mathrm{S} \\ j<i}} \xi_{ji} + \sum_{\substack{j \in \mathrm{V} \setminus \mathrm{S} \\ j>i}} \xi_{ij} \Bigg) \geq 2 && \forall\, \mathrm{S} \subset \mathrm{V} \setminus \{1\},\ |\mathrm{S}| \geq 3 \ , \\
& \| (\boldsymbol{q}_i - \bar{\boldsymbol{q}}_{i,j}) - (\boldsymbol{q}_j - \bar{\boldsymbol{q}}_{j,i}) \| - d_{ij} \leq 0 && \forall\, i,j \in \mathrm{V},\ j > i \ , \\
& \boldsymbol{q}_i - \bar{\boldsymbol{q}}_{i,j} \in \bar{\mathcal{Q}}_i(\xi_{ij}) && \forall\, i,j \in \mathrm{V},\ j > i \ , \\
& \boldsymbol{q}_j - \bar{\boldsymbol{q}}_{j,i} \in \bar{\mathcal{Q}}_j(\xi_{ij}) && \forall\, i,j \in \mathrm{V},\ j > i \ , \\
& \bar{\boldsymbol{q}}_{i,j} \in \bar{\mathcal{Q}}_i(1 - \xi_{ij}) && \forall\, i,j \in \mathrm{V},\ j > i \ , \\
& \bar{\boldsymbol{q}}_{j,i} \in \bar{\mathcal{Q}}_j(1 - \xi_{ij}) && \forall\, i,j \in \mathrm{V},\ j > i \ , \\
& d_{ij} \leq \xi_{ij}\ \mathrm{d_{MAX}} && \forall\, i,j \in \mathrm{V},\ j > i \ , \\
& 0 \leq \xi_{ij} \leq 1 && \forall\, i,j \in \mathrm{V},\ j > i \ , \\
& \boldsymbol{q}_i \in \mathbb{R}^m && \forall\, i \in \mathrm{V} \ , \\
& \bar{\boldsymbol{q}}_{i,j} \in \mathbb{R}^m && \forall\, i,j \in \mathrm{V},\ i \neq j \ , \\
& d_{ij} \in \mathbb{R}_+ && \forall\, i,j \in \mathrm{V},\ j > i \ .
\end{aligned}
$$

where $\bar{\mathcal{Q}}_i(\xi)$ can be defined as:

$$
\begin{aligned}
& \mathbf{A}_i\, \boldsymbol{q}_i + \xi\ \mathbf{b}_i \leq 0 \quad \text{or} \\
& (\boldsymbol{q}_i - \xi\ \mathbf{c}_i)^T\ \mathbf{P}_i^{-1}\ (\boldsymbol{q}_i - \xi\ \mathbf{c}_i) - \xi^2 \leq 0 \ .
\end{aligned}
$$

### 2.1.6 MTZ formulation

The principal drawback of the proposed STSPN formulations is the exponential number of subtour elimination constraints (2.4) that makes impractical a direct solution of the problem. To address this limitation, we considered other IP formulations of the ATSP presented in the literature such as sequential formulation, flow based formulation, and time staged formulation [79]. In particular the sequential formulation, also known as *MTZ formulation*, introduces $n-1$ new continuous variables $\eta_i$, and it reduces the number of subtour elimination constraint to $n^2 - 2n - 1$ [76]. If constraints (2.4) are

removed and the following constraints are introduced:

$$\eta_i - \eta_j + n\,\xi_{ij} \leq n-1 \qquad \forall\, i,j \in \mathrm{V}\setminus\{1\},\, i \neq j\ , \tag{2.42}$$

$$\eta_i \in \mathbb{R}_+ \qquad \forall\, i \in \mathrm{V}\setminus\{1\}\ , \tag{2.43}$$

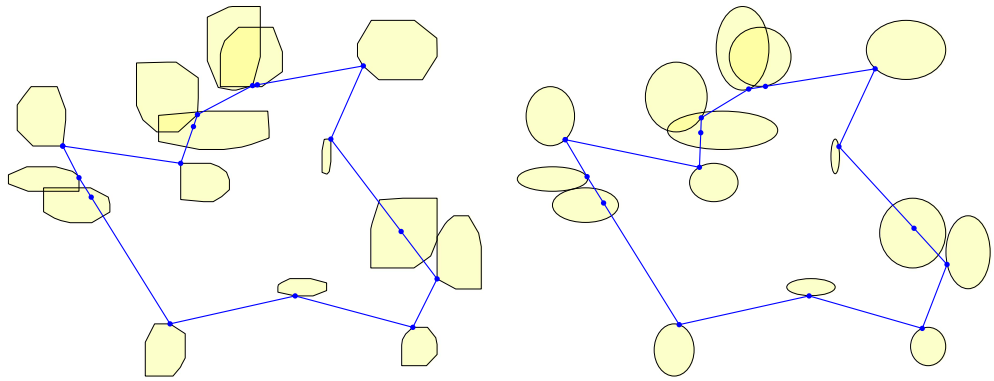
an alternative formulation of the ATSPN is obtained. It is however well known that the MTZ formulation of the ATSP produces a weaker Linear Programming (LP) relaxation than the DFJ formulation [80], though some improvement attempts have been proposed in [25]. Furthermore, the NLP relaxation of the MTZ formulation of the ATSPN is still non convex.

The convex MINLP solver Bonmin [17] is used to perform some preliminary tests. The results are illustrated in Section 2.3. Since Bonmin is an exact solver only for convex problems, the solution returned using the TMZ formulation is feasible, but without any optimality guarantee.

### 2.1.7 Randomly generated STSPN test instances

Differently from the case of the TSP where many public instances are available [86], to benchmark the proposed algorithms only one public repository was found for the case of the TSPN with balls in $\mathbb{R}^2$ and in $\mathbb{R}^3$ [74]. Therefore, the following procedure has been employed to randomly generate more general TSPN instances in $\mathbb{R}^m$ with different types of neighborhoods.

First $n$ random points $\mathbf{c}_i \in [0,100]^m$ are generated. These points are 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, within these boxes, ellipsoids (E), polyhedra (P), or Bézier cubic splines (S) are placed according to Equations (2.8), (2.9), and (2.13). The symbol enclosed in parentheses is used hereafter to refer to that specific topology. Each ellipsoid is generated by aligning its principal axes with the coordinate frame. A corresponding polyhedron is then generated by using the $2m$ facets of the box and 6 additional facets tangent to the ellipsoid at randomly selected locations. Finally, a Bézier spline is generated by randomly selecting its four control points within the box, which guarantees the spline to be fully contained in it. Examples are shown in Figures 2.1 and 3.2.



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

Figure 2.1: Randomly generated STSPN instances of comparable extension with 15 neighborhoods in $\mathbb{R}^2$ and *optimal tours* calculated with Euclidean Norm.

## 2.2 Solution of the *first* STSPN formulation

### 2.2.1 Description of the algorithm

The basis of the algorithm proposed to solve the *first* STSPN formulation with convex neighborhoods is a spatial branch-and-bound as implemented in Couenne [12, 20]. 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 work and can be found in the above paper and references therein. Algorithm 2.1 is a high-level simplified description of the basic algorithm applied on the STSPN, except that only a small number of constraints (2.20) 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

**Algorithm 2.1** A simplified spatial Branch-and-Bound algorithm for solving the MINLP **P**.

Input: Problem **P**
Output: The value $z_{\text{opt}}$ of an optimal solution of **P**

1. Define set $L$ of subproblems; let $L \leftarrow \{\mathbf{P}\}$;
2. Define $z^u$ as an upper bound for **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 $z_{\text{opt}} := z^u$

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 [39]. These alterations yield big improvement of the performance of the solver. The resulting algorithm is refereed to as CouTspn hereafter.

#### 2.2.1.1 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 (2.20) 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 [54]. 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 (we use the LP solver Clp [34])

$$\text{maximize :} \quad \sum_{j=2}^{n} \zeta_{1j} \ , \tag{2.44}$$

$$\text{subject to :} \quad \sum_{j=2}^{n} \zeta_{1j} = \sum_{j=1}^{t-1} \zeta_{jt} - \sum_{j=t+1}^{n} \zeta_{tj} \ , \tag{2.45}$$

$$\sum_{j=1}^{i-1} \zeta_{ji} - \sum_{j=i+1}^{n} \zeta_{ij} = 0 \qquad \forall\, i \in \mathrm{V}\backslash\{1,t\} \ , \tag{2.46}$$

$$-\bar{\xi}_{ij}^{k} \leq \zeta_{ij} \leq \bar{\xi}_{ij}^{k} \qquad \forall\, i,j \in \mathrm{V}, j > i \ , \tag{2.47}$$

$$\zeta_{ij} \in \mathbb{R} \qquad \forall\, i,j \in \mathrm{V}, j > i \ , \tag{2.48}$$

where $\zeta_{ij}$ 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 (2.47) using the solution $\bar{\xi}_{ij}^{k}$ 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 (2.20) exists. If for some terminal $t$ this maximum flow value is strictly less than 2, the set S defining a violated constraint (2.20) is formed by the union of the source 1 and all other vertex $i$ such that the constraint (2.46) for $i$ has a nonzero dual variable in an optimal solution to (2.44)-(2.48). 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 [5, 8], the size of the instances we are interested to solve (i.e. $n \leq 30$) does not require more sophistication.

#### 2.2.1.2 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 (2.21) 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 (2.21) 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 $\xi_{ij}$ to denote either $\xi_{ij}$ if $i < j$ or $\xi_{ji}$ if $i > j$. The rounded vector $\hat{\xi}^k$ has $\hat{\xi}^k_{p_{j-1}p_j}$ and $\hat{\xi}^k_{p_np_1}$ 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 (2.18)-(2.21), (2.5), and (2.7), 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 [20, 98]). To improve the performance of the solver, a permutation $\pi$ 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 $n$):

$$\sum_{i=1}^{n} \mathrm{d}\left(\boldsymbol{q}_{\pi(i)}, \boldsymbol{q}_{\pi(i+1)}\right) \tag{2.49}$$

is minimized subjected to constraints (2.8) and (2.7) if the neighborhoods are polyhedra and constraints (2.9) and (2.7) if they are ellipsoids.

IPOPT requires all functions in the problem formulation to be at least once differentiable, which is not the case when a norm is employed as edge weighting function in Equation (2.49). If the Euclidean norm is considered, we instead use the following edge weighting function:

$$\mathrm{d}\left(\boldsymbol{q}_i, \boldsymbol{q}_j\right) = \begin{cases} \left\|\boldsymbol{q}_j - \boldsymbol{q}_i\right\|_2 & \text{if } \left\|\boldsymbol{q}_j - \boldsymbol{q}_i\right\|_2 \geq \epsilon \ ; \\ \frac{\epsilon}{2} + \frac{1}{2\epsilon}\left\|\boldsymbol{q}_j - \boldsymbol{q}_i\right\|_2^2 & \text{if } \left\|\boldsymbol{q}_j - \boldsymbol{q}_i\right\|_2 < \epsilon \ . \end{cases} \tag{2.50}$$

We use $\epsilon = 0.1$ in the tests. The function (2.50) is continuously differentiable except when $\left\|\boldsymbol{q}_j - \boldsymbol{q}_i\right\|_2 = \epsilon$. 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 as illustrated in Appendix A.2.

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

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} \hat{\xi}^k_{ij} \, \xi_{ij} \leq n - 1 \ . \tag{2.51}$$

Although constraint (2.51) 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 (2.51) 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 [17]. 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 (2.51) 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.

#### 2.2.1.3 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 (2.20) and (2.21) in the model (2.18)-(2.21), (2.5), and (2.7). 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 2.2.2). 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 2.2.1.1 to find the first constraint (2.20) violated by the solution returned by IPOPT, and we add it to $H$. We then call IPOPT again, and this continues until all constraints (2.20) 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 [17, 20]. We then proceed in a similar fashion as with Ipopt, separating constraints (2.20) 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 [35], 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 [21].

While solving the continuous relaxation of a subproblem with Bonmin, the objective function (2.18) 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 (2.18) to prevent this overlapping (we use $\gamma = \delta = 10$ in the tests):

$$\sum_{i=1}^{n} \sum_{\substack{j=1 \\ j>i}}^{n} \xi_{ij} \ \|\boldsymbol{q}_j - \boldsymbol{q}_i\|_2 + \delta \, \mathrm{e}^{-\gamma^2 \|\boldsymbol{q}_j - \boldsymbol{q}_i\|_2^2} \ . \tag{2.52}$$

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 (2.52) is not employed when using CouTspn.)

### 2.2.2 Software settings

Results were obtained using open-source software available from COIN-OR [20]. This section describes precisely which version and additional non-default settings were used. The used software are the following.

The interior point solver Ipopt [20, 98] version stable/3.9 with all default settings plus the option:

- `linear_solver MA57`.

The convex MINLP solver Bonmin [17, 20] 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 [12, 20] 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 [21] version 12.2.0 with all default settings and the algebraic modeling language AMPL [35] version 20110121.

### 2.2.3 Computational results

Tests are performed on 64 random STSPN instances formed by ellipsoids or polyhedra defined in $\mathbb{R}^m$ (with $m = 2$ or 3) and generated as explained in Section 2.1.7. All test instances are available from [38], and an example in $\mathbb{R}^2$ is shown in Figure 2.1. 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`.

First, a heuristic solution is obtained using IPOPT and BONMIN as described in Section 2.2.1. The results are reported in the column with label "BONMIN" in tables 2.2, 2.3, 2.4, and 2.5. 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 2.2, 2.3, 2.4, and 2.5. 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 (2.51) ("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 [12] 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 2.1 reports the results. Although simple branching (either on all variables or restricted

Table 2.1: 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 |

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.

We first observe that, unsurprisingly, the difficulty of solving an instance usually increases with the number $n$ of neighborhoods. We note that the

Table 2.2: STSPN instances with polyhedra in $\mathbb{R}^2$ as neighborhoods.

| instance | $n$ | var. | $h$ | Bonmin | | | CouTspn | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | heuristic value | CPU time [s] | s.e. cuts | optimal value | CPU time [s] overall | CPU time [s] CglTspn | CPU time [s] Ipopt | cuts int. | cuts s.e. | 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 2.3: STSPN instances with polyhedra in $\mathbb{R}^3$ as neighborhoods.

| instance | $n$ | var. | $h$ | Bonmin | | | CouTspn | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | heuristic value | CPU time [s] | s.e. cuts | optimal value | CPU time [s] overall | CglTspn | Ipopt | cuts int. | cuts s.e. | 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 2.4: STSPN instances with ellipsoids in $\mathbb{R}^2$ as neighborhoods.

| instance | $n$ | var. | $h$ | Bonmin | | | CouTspn | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | heuristic value | CPU time [s] | s.e. cuts | optimal value | CPU time [s] overall | CPU time [s] CglTspn | CPU time [s] Ipopt | cuts int. | cuts s.e. | 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 2.5: STSPN instances with ellipsoids in $\mathbb{R}^3$ as neighborhoods.

| instance | $n$ | var. | $h$ | Bonmin | | | CouTspn | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | heuristic value | CPU time [s] | s.e. cuts | optimal value | CPU time [s] overall | CglTspn | Ipopt | cuts int. | cuts s.e. | 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 |

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 2.2.

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

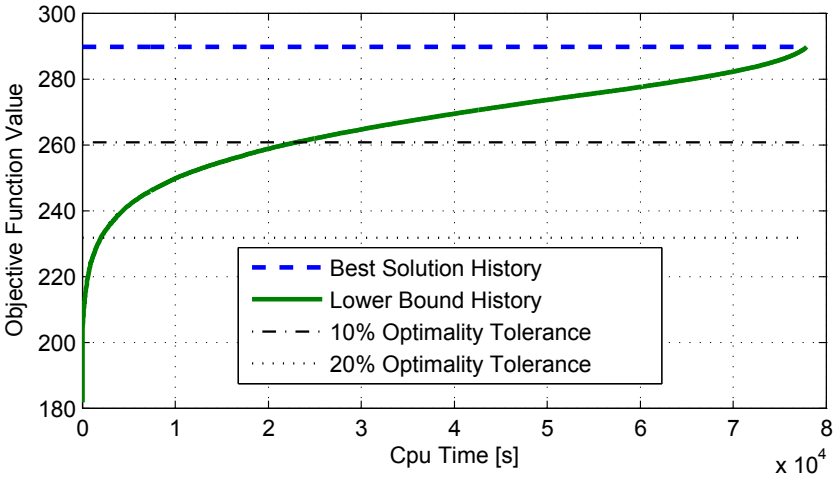



Figure 2.2: Convergence history of COUENNE with CglTspn for the instance tspn2DE15_1.

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 2.2.1.2 is not used can be found in Appendix A.3. Even with termination criteria very favorable for the latter algorithm, we observe that the proposed approach is orders of magnitude faster.

## 2.3 Solution of the *second* STSPN formulation

### 2.3.1 Solution procedure

Using the Outer-Approximation based Branch-and-Cut algorithm for convex MINLP available in Bonmin [17], STSPN instances with convex neighborhoods based on the *second* STSPN formulation can be solved to optimality. The heuristic result obtained by solving the *first* STSPN formulation as illustrated in Section 2.2.1.3, is used here as initial point and as upper bound for the algorithm. Moreover, all the subtour elimination constraints already found are added to the initial formulation of the problem. Finally, an ad-hoc cut generator based on maximum flow computation is embedded in the standard algorithm to check if any of the constraints (2.20) not included in the current problem formulation is violated by the current solution, similarly to the procedure illustrated in Section 2.2.1.1. If such constraint is found, it is added as a global cut.

The edge weighting function becomes non-differentiable when two vertices are identical, as illustrated in Section 2.2.1.3. To overcome this issue, if the Euclidean or the Quadratic norm is used then constraints (2.24) are reformulated as $d_{ij} \geq \mathrm{d_{MIN}}$, and the objective function (2.22) becomes:

$$-\frac{n(n-3)}{2}\,\mathrm{d_{MIN}} + \sum_{i=1}^{n}\sum_{\substack{j=1\\ j>i}}^{n} d_{ij}\ , \tag{2.53}$$

This formulation preserves the convexity of the problem, if convex neighborhoods are employed, while excluding non differentiable points from the feasible set. We use $\mathrm{d_{MIN}} = 0.01$ in the tests. If the Manhattan or the Maximum norm is used, according to Section 2.1.3.1 then constraints (2.26), (2.27), (2.29), and (2.30) are linear and no differentiability issue arises.

Computational test have been performed using also the MTZ formulation and the Branch-and-Bound algorithm for convex MINLP available in Bonmin [17]. Since Bonmin is an exact solver only for convex problems, the solution returned using the TMZ formulation is feasible, but without any optimality guarantee. To avoid vertices from overlapping and consequential convergence problems in the solver, also for this formulation an exponential decaying term is used in the definition of the objective function, as illustrated in Equation (2.52).

### 2.3.2 Software settings

This section describes precisely the additional non-default settings that were used in the solution procedure.

The convex MINLP solver BONMIN [17] stable/1.4 available from *COIN-OR* [20] using CBC releases/2.4.2, CLP releases/1.11.1, and IPOPT releases/3.8.3 as sub-solvers with all the default settings except:

- `linear_solver MA57`
- `max_iter 500`
- `bonmin.integer_tolerance 1e−6`
- `bonmin.algorithm B-BB` (MTZ)
- `bonmin.algorithm B-Hyb` (STSPN II)
- `bonmin.num_retry_unsolved_random_point 1` (STSPN II)
- `bonmin.random_point_perturbation_interval .5` (STSPN II)
- `bonmin.nlp_solve_frequency 1` (STSPN II)
- `bonmin.nlp_solve_max_depth 50` (STSPN II)

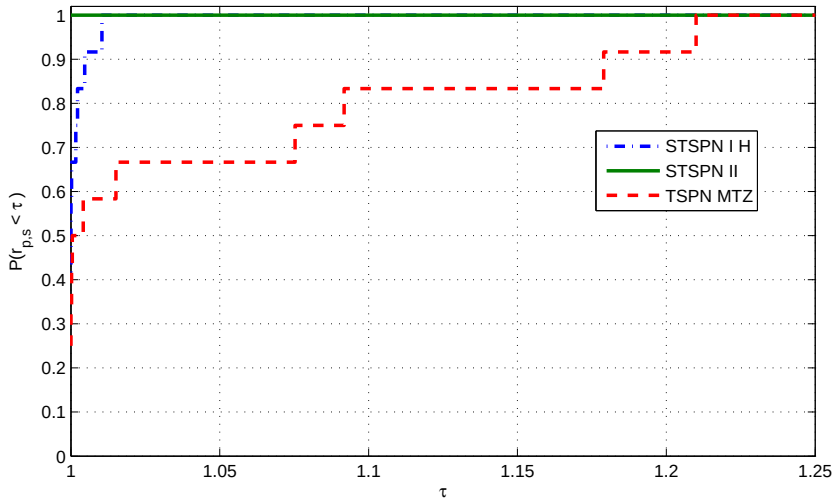
### 2.3.3 Computational results

Tests are performed on 6 random STSPN instances with ellipsoidal neighborhoods, and using three types of edge weighting functions: Manhattan, Euclidean, and Maximum norm. Instances with ellipsoids are in general harder to solve than similar instances with polyhedra as shown in Section 2.2.3, but they can be solved to optimality using the *second* STSPN formulation since the neighborhoods are convex. The computational results are illustrated in Table 2.6. The used edge weighting function is reported in the column with label "d(·)", the total number of variables used in the formulation in the column with label "n. var.", the final tour length in the column with label "obj.", the CPU time in the column with label "CPU", the number of the added subtour elimination constraints in the column with label "s.e.c.", and the number of nodes generated by the solver BONMIN in the column with label "nodes". Results obtained with the heuristic approaches based on the MTZ formulation and on the *first* STSPN formulation are reported in the rows with labels "TSPN MTZ" and "STSPN I H", rispectively. Finally, optimal solutions obtained with the *second* STSPN formulation are reported in the rows with label "STSPN II". The simulations are performed on an `Intel Core i7 920 @2.67 GHz` processor with 6GB of memory running `Windows 7 Ultimate SP1`.

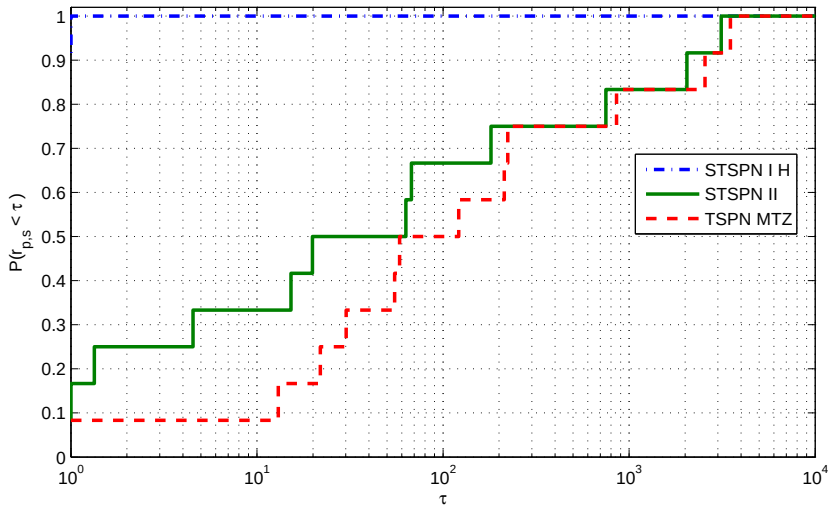
Table 2.6: Comparison of three MINLP solution procedures for randomly generated STSPN instances with ellipsoidal neighborhoods.

| instance | $n$ | $m$ | $h$ | d(·) | formulation | n. var. | obj. | CPU [s] | s.e.c. | nodes |
|---|---|---|---|---|---|---|---|---|---|---|
| tspn2DE10 | 10 | 2 | 0.25 | 2 | TSPN MTZ | 119 | 225.126 | 176 | - | 126 |
| | | | | | STSPN I H | 65 | 225.126 | 1.45 | 1 | 0 |
| | | | | | STSPN II | 110 | 225.126 | 6.57 | 9 | 521 |
| tspn2DE15 | 15 | 2 | 0.25 | 2 | TSPN MTZ | 254 | 311.528 | 3,218 | - | 1,546 |
| | | | | | STSPN I H | 135 | 290.362 | 3.76 | 6 | 0 |
| | | | | | STSPN II | 240 | 289.716 | 681 | 43 | 109,408 |
| tspn2DE20 | 20 | 2 | 0.15 | 2 | TSPN MTZ | 439 | 347.460 | 30,698 | - | 41,271 |
| | | | | | STSPN I H | 230 | 342.276 | 12 | 19 | 0 |
| | | | | | STSPN II | 420 | 342.276 | 9007 | 46 | 766,163 |
| tspn3DE10 | 10 | 3 | 0.25 | 2 | TSPN MTZ | 129 | 328.627 | 234 | - | 645 |
| | | | | | STSPN I H | 75 | 328.627 | 1.05 | 4 | 0 |
| | | | | | STSPN II | 120 | 328.627 | 16 | 10 | 6,930 |
| tspn3DE15 | 15 | 3 | 0.25 | 2 | TSPN MTZ | 269 | 434.150 | 9,175 | - | 21,566 |
| | | | | | STSPN I H | 150 | 434.157 | 2.62 | 3 | 0 |
| | | | | | STSPN II | 255 | 434.150 | 8,186 | 135 | 1,625,274 |
| tspn3DE20 | 20 | 3 | 0.15 | 2 | TSPN MTZ | 459 | 502.913 | 237,643 | - | 500,948 |
| | | | | | STSPN I H | 250 | 502.793 | 2.89 | 8 | 0 |
| | | | | | STSPN II | 440 | 502.793 | 9,126 | 83 | 967,878 |
| tspn2DE10 | 10 | 2 | 0.25 | 1 | TSPN MTZ | 299 | 286.304 | 53 | - | 4 |
| | | | | | STSPN I H | 155 | 286.231 | 4.54 | 1 | 0 |
| | | | | | STSPN II | 200 | 286.164 | 4.07 | 7 | 713 |
| tspn2DE15 | 15 | 2 | 0.25 | 1 | TSPN MTZ | 674 | 379.993 | 172 | - | 4 |
| | | | | | STSPN I H | 345 | 348.634 | 7.85 | 3 | 0 |
| | | | | | STSPN II | 450 | 348.057 | 531 | 16 | 107,692 |
| tspn2DE20 | 20 | 2 | 0.15 | 1 | TSPN MTZ | 1199 | 494.542 | 965 | - | 6 |
| | | | | | STSPN I H | 610 | 423.844 | 32 | 9 | 0 |
| | | | | | STSPN II | 800 | 419.476 | 65,486 | 32 | 6,610,820 |
| tspn2DE10 | 10 | 2 | 0.25 | inf | TSPN MTZ | 209 | 191.218 | 59 | - | 2 |
| | | | | | STSPN I H | 110 | 190.427 | 1.01 | 1 | 0 |
| | | | | | STSPN II | 110 | 190.427 | 1.35 | 4 | 42 |
| tspn2DE15 | 15 | 2 | 0.25 | inf | TSPN MTZ | 464 | 319.125 | 1,558 | - | 43 |
| | | | | | STSPN I H | 240 | 264.964 | 7.30 | 6 | 0 |
| | | | | | STSPN II | 240 | 263.742 | 145 | 15 | 35,833 |
| tspn2DE20 | 20 | 2 | 0.15 | inf | TSPN MTZ | 819 | 423.950 | 1,154 | - | 22 |
| | | | | | STSPN I H | 420 | 314.694 | 21 | 17 | 0 |
| | | | | | STSPN II | 420 | 299.752 | 1,325 | 32 | 167,882 |

Since both the MTZ and the *first* STSPN formulation are non convex, the results obtained using the convex optimizer Bonmin are not always optimal. However the latter seems to perform better than the former. The average optimality gap for the MTZ formulation is 8.25% with a maximum




(a) Performance ratio $r_{p,s}$ based on the objective function value.




(b) Performance ratio $r_{p,s}$ based on the CPU time (a logarithmic scale is used for $\tau$).

Figure 2.3: Performance profiles for the three MINLP solution procedures.

of 41.43% for tspn2DE20 and Maximum norm. For the *first* STSPN formulation the average optimality gap is 0.58% with a maximum of 4.98% for

tspn2DE20 and Maximum norm. Figure 2.3.a illustrates the performance profiles derived using as performance measure the objective function value [27]. Note that the probability for the solution procedure based on the *second* STSPN formulation that its performance ratio $r_{p,s}$ is within factor 1 of the best possible ratio is 1 since the procedure is exact.

Moreover, the CPU time of the MTZ formulation is larger than the one of the *second* STSPN formulation by an average factor of 12 with a maximum factor of 44 for tspn2DE10 and Maximum norm. On the contrary, the CPU time of the *first* STSPN formulation is smaller than the one of the *second* STSPN formulation on average by two orders of magnitude with maximum factors up to three orders of magnitude. These results suggest that the *first* STSPN formulation definitely leads to a more efficient procedure than the MTZ formulation for finding a near *optimal tour* to use as initial point for the *second* STSPN formulation. These results are highlighted in Figure 2.3.b, which illustrates the performance profiles derived using as performance measure the CPU time [27].

Solving to optimality the same instances using the second STSPN formulation with the Euclidean norm seems to be harder than with the Maximum norm. The CPU time with Euclidean norm is larger than the one with Maximum norm by an average factor of 5.45 with a maximum of 6.79 for tspn2DE20. Moreover, the comparison of Euclidean and Manhattan norms does not lead to a certain conclusion. In two cases, tspn2DE10 and tspn2DE15, the CPU time with Euclidean norm is larger than the one with Manhattan norm by an average factor of 1.45, but for tspn2DE20 it is smaller by a factor of 7.14.

Instances in $\mathbb{R}^3$ seem to be harder to solve to optimality than instances in $\mathbb{R}^2$ with the same number of neighborhoods, which is consistent with the results reported in Section 2.2.3. The CPU time for instances in $\mathbb{R}^3$ is larger than the one for instances in $\mathbb{R}^2$ by an average factor of 5.15 with a maximum of 12.02 for tspn2DE15.

Finally, the three instances that were solved to optimality in Section 2.2.3 using the Euclidean norm, i.e., tspn2DE10, tspn2DE15, and tspn3DE10, show that the proposed procedure based on the *second* STSPN formulation performs better in term of computational cost than the one proposed in Section 2.2, while finding the same *optimal tour*. Although we need to take into account that the testing environments are not exactly the same, the CPU times were improved here by a factor of 5, 114, and 50, respectively.

## 2.4 Solution of the *third* and *fourth* STSPN formulations

### 2.4.1 Solution procedure

Using the same procedure illustrated in Section 2.3 for the *second* STSPN formulation, STSPN instances with polyhedral neighborhoods and the Euclidean norm are solved to optimality using the *third* and the *fourth* STSPN formulations. According to Section 2.1.3.1 these formulations lead to Second Order Conic problems for the considered cases, and thus the conic solver MOSEK [4] is also employed. In this case the subtour eliminations constraints are generated using a procedure similar to the one indicated in Section 2.2.1.3 and added one by one to the problem formulation using AMPL.

To avoid convergence issues in the solver BONMIN if two vertices overlaps or if they are both zero vectors, which is always the case for the additional variables in the *third* and the *fourth* STSPN formulations, a constant term is introduced in the Euclidean norm:

$$\|\boldsymbol{q}_i - \boldsymbol{q}_j\|_2 = \sqrt{\epsilon_2 + (\boldsymbol{q}_i - \boldsymbol{q}_j)^T (\boldsymbol{q}_i - \boldsymbol{q}_j)}\,, \tag{2.54}$$

and we use $\epsilon_2 = 0.000001$ in the tests. This is not necessary for the solver MOSEK since conic constraint are fully supported by the solver.

### 2.4.2 Software settings

This section describes precisely the additional non-default settings that were used in the solution procedure.

The convex MINLP solver BONMIN [17] stable/1.6 available from *COIN-OR* [20] using CBC releases/2.7.5, CLP releases/1.14.5, and IPOPT releases/3.10.1 as sub-solvers with all the default settings except:

- `linear_solver MA57`
- `max_iter 500`
- `bonmin.algorithm B-Hyb`
- `bonmin.integer_tolerance 1e−6`
- `bonmin.mir_cuts -99`

The solver MOSEK [4] Version 6.0.0.135 with all the default settings.

Table 2.7: Comparison of the five exact solution procedures for STSPN instances with polyhedral neighborhoods.

| instance | STSPN I | | STSPN II | | | | STSPN III | | | | STSPN IV | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Bonmin | | Mosek | | Bonmin | | Mosek | | Bonmin | | Mosek | |
| | n. | CPU | n. | CPU | n. | CPU | n. | CPU | n. | CPU | n. | CPU | n. | CPU |
| tspn2DP5_1 | 20 | 0.12 | 30 | 0.64 | 60 | 0.31 | 70 | 0.84 | 90 | 0.84 | 70 | 0.81 | 90 | 0.97 |
| tspn2DP5_2 | 20 | 0.14 | 30 | 0.47 | 60 | 0.41 | 70 | 0.75 | 90 | 0.92 | 70 | 0.92 | 90 | 1.00 |
| tspn2DP6_1 | 27 | 0.4 | 42 | 0.59 | 87 | 1.81 | 102 | 1.87 | 132 | 3.24 | 102 | 1.75 | 132 | 4.18 |
| tspn2DP6_2 | 27 | 0.13 | 42 | 0.48 | 87 | 0.70 | 102 | 1.28 | 132 | 1.14 | 102 | 2.15 | 132 | 1.83 |
| tspn2DP7_1 | 35 | 1.72 | 56 | 0.86 | 119 | 12 | 140 | 5.13 | 182 | 4.63 | 140 | 4.37 | 182 | 12 |
| tspn2DP7_2 | 35 | 1.19 | 56 | 0.73 | 119 | 4.96 | 140 | 4.84 | 182 | 4.12 | 140 | 3.82 | 182 | 3.06 |
| tspn2DP8_1 | 44 | 1.79 | 72 | 0.86 | 156 | 11 | 184 | 5.07 | 240 | 4.62 | 184 | 7.77 | 240 | 8.81 |
| tspn2DP8_2 | 44 | 4.04 | 72 | 1.25 | 156 | 32 | 184 | 6.01 | 240 | 13 | 184 | 11 | 240 | 17 |
| tspn2DP9_1 | 54 | 22 | 90 | 1.83 | 198 | 156 | 234 | 12 | 306 | 12 | 234 | 25 | 306 | 42 |
| tspn2DP9_2 | 54 | 2.12 | 90 | 1.08 | 198 | 89 | 234 | 7.71 | 306 | 4.37 | 234 | 12 | 306 | 8.81 |
| tspn2DP10_1 | 65 | 21 | 110 | 2.12 | 245 | 878 | 290 | 18 | 380 | 13 | 290 | 24 | 380 | 26 |
| tspn2DP10_2 | 65 | 3.85 | 110 | 1.47 | 245 | 591 | 290 | 18 | 380 | 5.10 | 290 | 26 | 380 | 8.33 |
| tspn2DP11_1 | 77 | 109 | 132 | 3.98 | 297 | 19,652 | 352 | 98 | 462 | 27 | 352 | 46 | 462 | 49 |
| tspn2DP11_2 | 77 | 11 | 132 | 1.93 | 297 | 3,004 | 352 | 26 | 462 | 6.44 | 352 | 21 | 462 | 17 |
| tspn3DP5 | 25 | 0.15 | 35 | 0.47 | 75 | 0.61 | 95 | 1.00 | 125 | 1.40 | 95 | 0.98 | 125 | 1.39 |
| tspn3DP6 | 33 | 0.6 | 48 | 0.58 | 108 | 2.90 | 138 | 2.39 | 183 | 3.42 | 138 | 3.00 | 183 | 2.03 |
| tspn3DP7 | 42 | 4.25 | 63 | 1.15 | 147 | 23 | 189 | 7.05 | 252 | 23 | 189 | 14 | 252 | 40 |
| tspn3DP8 | 52 | 12 | 80 | 3.71 | 192 | 19 7 | 248 | 55 | 332 | 93 | 248 | 29 | 332 | 68 |
| tspn3DP9 | 63 | 58 | 99 | 3.56 | 243 | 1,598 | 315 | 343 | 423 | 124 | 315 | 112 | 423 | 208 |
| tspn3DP10 | 75 | 230 | 120 | 8.99 | 300 | 8,137 | 390 | 302 | 525 | 252 | 390 | 108 | 525 | 251 |

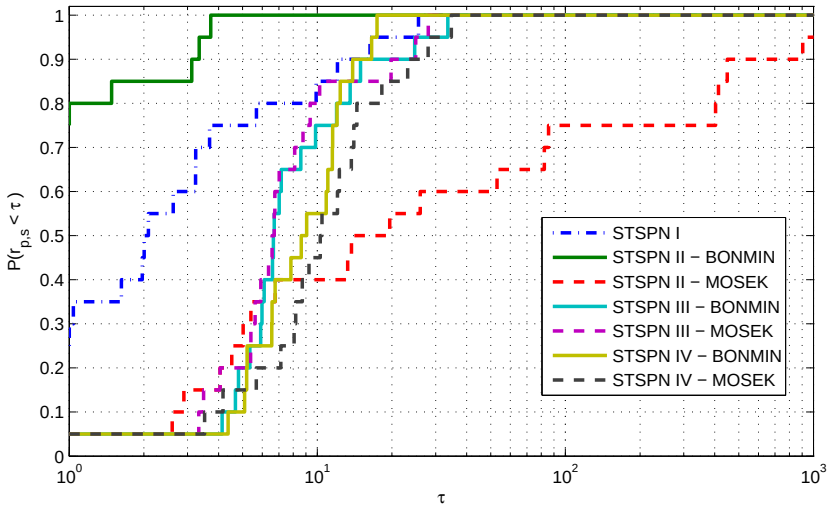



Figure 2.4: Performance profiles based on the CPU time for the five exact MINLP solution procedures (a logarithmic scale is used for $\tau$).

### 2.4.3 Computational results

Tests are performed on 20 STSPN instances with polyhedral neighborhoods from Section 2.2.3. The computational results are illustrated in Table 2.7. The total number of variables used in the formulation is reported in the columns with label "n." and the CPU time in the columns with label "CPU". Results previously obtained with the exact solution procedure for the *first* formulation are reported in the column with label "STSPN I". Results obtained using the *second*, *third*, and *fourth* formulations are reported in columns with labels "STSPN II", "STSPN III", and "STSPN IV", where the used solver is also indicated. The final tour length is not reported since results are always optimal and values can be read from Tables 2.2 and 2.3. The simulations are performed on an `Intel Core i7 920 @2.67 GHz` processor with 12GB of memory running `Windows 7 Ultimate SP1`.

Figure 2.4 illustrates the performance profiles derived using as performance measure the CPU time [27]. Clearly, the *second* formulation solved using Bonmin outperforms all the other procedures. In particular the geometric average of the ratio of the CPU time spent by the *first* formulation solved using Couenne to the *second* formulation solved using Bonmin is

2.39, with a maximum ratio of 27.40 for tspn2DP11_1 and a minimum ratio of 0.19 for tspn2DP5_1. Note that the processors used for the two sets of tests are slightly different, but with an advantage towards the *first* formulation and thus it does not impact our conclusion.

The worst performance is obtained by the *second* formulation is solved using MOSEK, where the CPU time ratio to the *second* formulation solved using BONMIN is 32.26. This result is heavily influenced by the fact that the subtour elimination constraints cannot be dynamically added in the Branch and Bound algorithm available in MOSEK, requiring the problem to be solved several times before a feasible tour is returned.

Finally, the remaining four solution procedures show a very similar performance. The CPU time ratios for the *third* and *fourth* formulations solved using BONMIN to the *second* formulation solved using BONMIN are 7.02 and 7.09, respectively, and the ratios for the *third* and *fourth* formulations solved using MOSEK to the *second* formulation solved using BONMIN are 6.17 and 8.95, respectively. We can observe that the increase in the number of variables due to the additional $\tilde{\boldsymbol{q}}_{i,j}$ or $\bar{\boldsymbol{q}}_{i,j}$ causes the computational cost to increase when BONMIN is used. However, when MOSEK is used this increase might be compensated by the fact that the *third* and *fourth* formulations use a stronger relaxation than the *second*, allowing an overall faster solution procedure.

## 2.5 Conclusion

In this chapter, first a non-convex MINLP formulation of the STSPN 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 alternative heuristic solution procedure based on the MTZ formulation is also tested, but results show that the latter procedure is outperformed by the former both in terms of computational cost and optimality gap.

An approach for solving the problem to optimality using a modified version of the global MINLP solver COUENNE is then presented. An ad hoc cut generator 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 STSPN instances with up to 16 neighborhoods in $\mathbb{R}^2$ and with up to 12 neighborhoods in $\mathbb{R}^3$ can be solved to optimality, and that the proposed approach is orders of magnitude faster than the standard algorithm implemented in Couenne.

A second approach is then proposed by reformulating the STSPN as a convex MINLP instance, under the assumption that neighborhoods and edge weighting function are both convex. Three different formulations are provided, and a modified version of the convex MINLP solver Bonmin is implemented, where subtour elimination constraints are again handled as cutting planes and an ad hoc cut generator is embedded in the standard algorithm. Alternatively, the solver Mosek is also employed to directly handle the conic constraints in the convex formulations.

Numerical tests show that the solution procedure based on the *second* STSPN formulation and the modified solver Bonmin outperforms all the other approaches in case of convex neighborhoods and convex edge weighting function. In particular, it outperforms the procedure based on the *first* STSPN formulation and the modified solver Couenne in terms of computation cost by factors up to two orders of magnitude, while converging to the same optimal value. Using the best solution procedure, instances with up to 20 ellipsoidal neighborhoods in $\mathbb{R}^2$ and $\mathbb{R}^3$ have been solved to optimality using the Euclidean, the Manhattan, and the Maximum norm as edge weighting function.

# Chapter 3

# Hybrid Random-Key Genetic Algorithm

In this chapter, we propose a Genetic Algorithm as a heuristic approach to employ for large scale problems. The formulation is presented, and numerical results are compared with the solutions obtained in chapter 2. Some test problems available in the literature are also analyzed.

## 3.1 Genetic algorithm formulation

The computational cost of the heuristic and exact procedures illustrated in chapter 2 rapidly increases with the number of the neighborhoods. Therefore, an alternative heuristic approach might be useful in practical applications to find a near *optimal tour* for instances with up to $1,000$ neighborhoods. Moreover, the results of the heuristic procedure can be used to define a feasible initial point and an upper bound for the MINLP optimizer and thus to improve its performance, as discussed in Section 2.2.1.3.

Heuristic approaches proposed in the literature model only very specific STSPN instances, such as balls in $\mathbb{R}^2$ or $\mathbb{R}^3$, and they generally generate only one near *optimal tour* while hardly trying to improve it by exploiting the specific nature of the problem. On the contrary, a Genetic Algorithm (GA) can maintain a large pool of near *optimal tour*s, and performs only low cost computations to improve each one of those. Moreover, the general structure of a GA allows to deal with any type of STSPN instances, even with non-connected neighborhoods.

In a general GA formulation the pool of near optimal feasible points currently known is called *population*. Each point is coded in a particu-

lar sequence, called *individual* or *chromosome*, of real, integer, or binary numbers, called *genes*, which univocally represents the point. The encoding strategy depends on the nature of the considered optimization problem, and on the specific *genetic operators* used to generate new *chromosomes*. At each iteration of the algorithm, also called *generation*, newly generated *chromosomes* are grouped into a new pool, which usually contains the same number of *chromosomes* as the initial *population*. The algorithm ends when a termination criterion is fulfilled [11, 51].

The most commonly used *genetic operators* are *selection*, *crossover*, *mutation*, and *immigration*. *Selection* chooses the best *chromosomes* in the current *population* and copies them into the next *population* in order to preserve the best know feasible points throughout the execution of the algorithm. *Crossover* combines in a deterministic or random fashion the *genes* of two randomly chosen *chromosomes* (*parents*) of the current *population* to create two or more new *chromosomes* (*offsprings*), and inserts them in the next *population*, mimicking the biological concept that such combinations may produce superior offsprings. With the same goal, *mutation* varies randomly chosen *genes* in the *chromosomes* of the *population* following a probabilistic pattern. Finally, *immigration* randomly generates new *chromosomes* in the *population* in order to maintain a broad genetic variety.

Several works have been proposed in the literature to find a near optimal point for MINLP problems using a GA. Yokota et al. [104] propose a GA formulation that performs arithmetic *crossover* and accounts for infeasible *chromosomes* by adding a penalty term to the objective function. Deep et al. [24] propose a special truncation procedure to handle integer variables, Laplace *crossover*, Power *mutation*, and a penalty approach for handling constraints. Moreover, many GA with ad-hoc operators have been proposed in the literature for solving TSP instances [64]. Moving from the results proposed by Snyder and Daskin [95] for solving GTSP instances, in the present work a new coding procedure is proposed by adapting to the STSPN the random-key schema proposed by Bean [11]. Finally, a hybrid GA formulation is employed where specific heuristics instead of *mutation* operators enhance the quality of each *chromosome* while maintaining feasibility.

### 3.1.1 Chromosome coding

Section 2.1.2 illustrates that one of the major drawbacks of the *first* or *second* MINLP formulations of the STSPN is the exponential number of subtour elimination constraints. To overcome this issue, a random-key based *chro-*

*mosome* coding scheme is proposed that intrinsically satisfies constraints (2.19) and (2.20). Moreover, instead of keeping the $n(n-1)/2$ binary variables $\xi_{ij}$, only $n$ real numbers in the interval $[0,1]$ are necessary to represent an Hamiltonian cycle. Each *chromosome* is thus a sequence of $n$ *genes*. Each *gene* is composed by a vector part, which represent the position of the vertex in its neighborhood, and fractional part, which represents the position of the vertex in the cycle. The vector part is equal to $\boldsymbol{q}_i$, except for the case of parameterized neighborhoods, such as cubic splines (2.10) or (2.13), where it is equal to the parameter $t_i$. The fractional part is a real number in the interval $[0,1]$, which represents the ordering in ascending order, i.e., the vertex with the smallest fractional part is the first in the cycle and the vertex with the largest one is the last. As an example, in the case $n = 5$ the following chromosome:

$$\boldsymbol{q}_1.22 \qquad \boldsymbol{q}_2.72 \qquad \boldsymbol{q}_3.45 \qquad \boldsymbol{q}_4.87 \qquad \boldsymbol{q}_5.02$$

is decoded into the following tour:

$$\boldsymbol{q}_5 \;\rightarrow\; \boldsymbol{q}_1 \;\rightarrow\; \boldsymbol{q}_3 \;\rightarrow\; \boldsymbol{q}_2 \;\rightarrow\; \boldsymbol{q}_4 \;.$$

Obviously, if a parametric representation of the neighborhoods is used, from the parameter $t_i$ the corresponding location $\boldsymbol{q}_i$ has to be calculated during the decoding procedure.

When a new *chromosome* is generated, the fractional part of each gene is determined by generating a uniformly distributed random number in the interval $[0,1]$. The same is done for the vector part in case of a parametric representation of the neighborhoods (cubic splines). In case of convex neighborhoods, we observe that the edges of the tour always cross the neighborhoods boundaries unless all the neighborhoods are fully contained in a larger one. The optimal position of each vertex is thus likely to be on the boundary of its corresponding neighborhood, as can be easily observed in Figure 2.1. Therefore, for the vector part of each gene, $\boldsymbol{q}_i$, a configuration is randomly chosen in the neighborhood using a beta distribution to obtain a higher density in proximity of the boundary, as illustrated in Appendix A.4. Note that in case of polyhedra, the corresponding ellipsoid as illustrated in Section 2.1.7 is used for generating points in its interior. If this is not available, the maximum volume ellipsoid inscribed in the polyhedron can be used for this purpose, as explained in Appendix A.5.

### 3.1.2 Genetic operators

#### 3.1.2.1 Selection

An elitist strategy is employed to propagate from one generation to the next the best known *chromosomes*, in order to conserve them throughout the execution of the algorithm and to provide the *crossover* operators with a pool of *parents* to choose from. The percentage of the new *population* constituted by these *chromosomes* is indicated by $p_S$. This value is not kept constant in all the *generations* since it has been observed that the convergence rate of the algorithm may benefit from one or more large *immigration* cycles if the best *chromosome* does not improve for a certain number of consecutive *generations*.

#### 3.1.2.2 Crossover

Starting from the pool of *parents* provided by the *selection* operator, new *choromosomes* called *offsprings* are generated using the *crossover* operator, and the percentage of the new *population* constituted by *offsprings* is indicated by $p_X$. The random-key based *chromosome* coding has the advantage that allows performing a direct uniform *crossover* on the *chromosomes* while maintaining the feasibility of the *offsprings* and thus avoiding computationally expensive postprocessing steps to recover feasibility. First, *parents* are selected from the pool using a tournament procedure, where the best among three randomly selected *chromosomes* in the pool becomes one *parent* [41, 51]. Then, a sequence of $n$ random numbers uniformly distributed in the interval $[0, 1]$ is generated and if the $i$-th element of the sequence is less than a given threshold $p_U$, then the *offspring* will inherit the $i$-th gene from the first *parent*, otherwise it will inherit it from the second. As an example, in the case $n = 5$ and $p_U = 0.3$ the following random sequence and two parents:

| | | | | |
|---|---|---|---|---|
| 0.28 | 0.74 | 0.68 | 0.15 | 0.64 |
| $\boldsymbol{q}_1^I.22$ | $\boldsymbol{q}_2^I.72$ | $\boldsymbol{q}_3^I.45$ | $\boldsymbol{q}_4^I.87$ | $\boldsymbol{q}_5^I.02$ |
| $\boldsymbol{q}_1^{II}.89$ | $\boldsymbol{q}_2^{II}.91$ | $\boldsymbol{q}_3^{II}.38$ | $\boldsymbol{q}_4^{II}.12$ | $\boldsymbol{q}_5^{II}.43$ |

generate the following *offspring* (indicated by the tilde):

$$\widetilde{\boldsymbol{q}}_1 = \boldsymbol{q}_1^I.22 \qquad \widetilde{\boldsymbol{q}}_2 = \boldsymbol{q}_2^{II}.91 \qquad \widetilde{\boldsymbol{q}}_3 = \boldsymbol{q}_3^{II}.38 \qquad \widetilde{\boldsymbol{q}}_4 = \boldsymbol{q}_4^I.87 \qquad \widetilde{\boldsymbol{q}}_5 = \boldsymbol{q}_5^{II}.43\ ,$$

which is decoded into the following feasible tour:

$$\widetilde{\boldsymbol{q}}_1 \ \rightarrow\ \widetilde{\boldsymbol{q}}_3 \ \rightarrow\ \widetilde{\boldsymbol{q}}_5 \ \rightarrow\ \widetilde{\boldsymbol{q}}_4 \ \rightarrow\ \widetilde{\boldsymbol{q}}_2\ .$$

The Laplace *crossover* proposed by Deep and Thakur [23] can also be adapted to further evolve the vector part of the *genes*. If $x_i$ for $i = 1, \ldots, n$ are $n$ realizations of a random variable $X \sim \text{Laplace}(\mu, \lambda)$ as illustrated in Appendix A.6, then the vector part of each *gene* of the two *offsprings* generated by the operator is given by:

$$\begin{aligned} \widetilde{\boldsymbol{q}}_i^I &= \boldsymbol{q}_i^I + x_i \left( \boldsymbol{q}_i^{II} - \boldsymbol{q}_i^I \right) , \\ \widetilde{\boldsymbol{q}}_i^{II} &= \boldsymbol{q}_i^{II} + x_i \left( \boldsymbol{q}_i^{II} - \boldsymbol{q}_i^I \right) . \end{aligned} \tag{3.1}$$

It is worth noticing that after applying the Laplace *crossover*, the chromosome may become infeasible, since the vertices may exit from their neighborhoods. However, the touring heuristic, as explained in the next section, will move the vertices back in the feasible set. Moreover, in the numerical tests where no heuristics are employed, the penalty approach on the objective function illustrated by Deep et al. [24] is used to account for infeasibility.

#### 3.1.2.3 Mutation

The *mutation* operator is usually a low computational cost operator that alters the value of randomly selected *genes* in the *population*. However, to fully exploit the nature of the STSPN, in this work an hybrid GA formulation is employed and specific heuristics are rather used, which modify each *gene* in each *chromosome*. These operators are computationally intensive, but also improve the performance of the GA in terms of quality of the found near *optimal tour* and overall computational cost, as illustrated in Section 3.2. In other words, although the computational cost of producing a new *generation* becomes higher, the overall number of *generations* required to converge becomes smaller. In particular, two heuristics are employed: the first modifies the fractional part of the *genes*, keeping the vector part constant (Lin-Keringhan heuristic), and the second does the opposite, i.e., modifies the vector part keeping the fractional constant (Touring heuristic).

In the experimental tests where no heuristics are employed, the Power *mutation* proposed by Deep et al. [24] is used. The $k$-th component of the vector part of the $i$-th *gene* of a *chromosome* in the *population*, $q_{i,k}$, is selected with probability $p_M$, two random numbers, $x_p$ and $x_s$, uniformly distributed in the interval $[0, 1]$ are selected, and the following mutation is

performed:

$$\widetilde{q}_{ik} = \begin{cases} q_{i,k} - (x_p)^{1/\alpha} \left(q_{i,k} - q_{i,k}^L\right) & \text{if} \quad \frac{q_{i,k}-q_{i,k}^L}{q_{i,k}^U-q_{i,k}^L} \leq x_s \ , \\ \\ q_{i,k} + (x_p)^{1/\alpha} \left(q_{i,k}^U - q_{i,k}\right) & \text{if} \quad \frac{q_{i,k}-q_{i,k}^L}{q_{i,k}^U-q_{i,k}^L} > x_s \ , \end{cases} \tag{3.2}$$

where $q_{i,k}^L$ and $q_{i,k}^U$ are the lower and upper bounds on the $k$-th component of the $i$-th neighborhood, and $\alpha > 0$. $(x_p)^{1/\alpha}$ is a realizations of a random variable distributed according to the Power Function distribution $\text{Pow}(\alpha, 1)$, as illustrated in Appendix A.7. Finally, a penalty approach on the objective function is used to account for infeasibility, as illustrated in [24].

**Lin-Keringhan heuristic** If the vector part is kept constant, then the STSPN becomes a well known STSP. Therefore, among the many heuristics available in the literature for finding a near optimal cycle for a STSP, the one that has shown good performance even on large instances is the Lin-Keringhan heuristc [67]. Two different implementations were tested: the one proposed by Helsgaun [52], and the one proposed by Applegate et al. [6]. Since this operator has to be applied on each *chromosome* in each *generation*, although the former showed in general slightly better performance in terms of near *optimal tour* cost, the latter was finally used on the basis of its in average smaller execution time.

**Touring heuristic** If the fractional part is kept constant, then the STSPN becomes a NLP problem, which is convex if the neighborhoods and the edge weighting function are convex. After decoding the neighborhoods sequence from the fractional part into a permutation $\pi(i)$, with $\pi(n+1) = \pi(1)$, the same NLP problem illustrated in Section 2.2.1.2 can be derived using the *first* STSPN formulation, i.e., the objective function (2.49) has to be minimized subject to constraints (2.5) and (2.7).

If the *second* STSPN formulation is used then the Touring heuristic is formulated as follows:

$$\text{minimize :} \quad \sum_{i=1}^{n} d_i \ , \tag{3.3}$$

$$\text{subject to :} \quad \mathrm{d}\left(\boldsymbol{q}_{\pi(i)}, \boldsymbol{q}_{\pi(i+1)}\right) \leq d_i \qquad \forall\, i \in \mathrm{V} \ , \tag{3.4}$$

$$d_i \in \mathbb{R}_+ \qquad \forall\, i \in \mathrm{V} \ , \tag{3.5}$$

together with constraints (2.5) and (2.7). Here $n$ additional variables $d_i$ are used.

If the Euclidean norm or the Quadratic norm is used and if the neighborhoods are polyhedra or ellipsoids, then the above formulation becomes a Second Order Conic Programming (SOCP) problem, whereas if the neighborhoods are cubic splines, it becomes a non-convex NLP problem, which has no guarantee to be solved to optimality using a convex NLP optimizer. In the latter case recent developments achieved in polynomial optimization problems (POP) can be used to find a global optimum, and in particular the sparse semidefinite programming (SDP) relaxation methods proposed by Waki et al. [100] and Lasserre [65] were tested. However, preliminary results showed that the elevated computational cost does not allow to achieve fast convergence for problems beyond 10 neighborhoods. Moreover, on the basis of the structure of the GA, multiple initial points for the convex NLP optimizer are usually tested, which allows it to outperform the SDP based approaches in terms of overall convergence rate while reaching the same optimal point, though without guarantee of global optimality.

If the Manhattan norm or the Maximum norm is used, the following formulation for the Touring heuristic can be derived:

$$\text{minimize :} \quad \sum_{i=1}^{n}\sum_{k=1}^{m} d_{ik} \,, \tag{3.6}$$

$$\text{subject to :} \quad q_{\pi(i),k} - q_{\pi(i+1),k} \leq d_{ik} \qquad \forall\, i \in \mathrm{V},\, \forall\, k \in \{1,\ldots,m\} \,, \tag{3.7}$$

$$q_{\pi(i+1),k} - q_{\pi(i),k} \leq d_{ik} \qquad \forall\, i \in \mathrm{V},\, \forall\, k \in \{1,\ldots,m\} \,, \tag{3.8}$$

$$d_{ik} \in \mathbb{R}_{+} \qquad \forall\, i \in \mathrm{V},\, \forall\, k \in \{1,\ldots,m\} \,, \tag{3.9}$$

together with constraints (2.5) and (2.7). In this formulation $n \times m$ additional variables $d_{ik}$ are used, which become $n$ if the Maximum norm is used (in this case $d_{i1} = \ldots = d_{im} = d_i$). The advantage of this formulation is that the NLP becomes a Linear Programming (LP) problem if the neighborhoods are polyhedra, and a Quadratic Constrained Quadratic Programming (QCQP) problem if the neighborhoods are ellipsoids. In case of cubic splines, it remains a non-convex NLP problem.

In the present work, the optimizer Cplex [21] is employed to solve LP, QCQP, and SOCP problems. For more general NLP instances the open source convex optimizer Ipopt [98] is used where the sparsity of the Lagrangian Hessian matrix allows to achieve fast convergence in the Newton scheme. In this case the objective function and its derivatives are hard-coded into the solver as illustrated in Appendix A.2. In particular, for the case of

Table 3.1: Formulation, Problem Type, and Solver employed for the Touring heuristic.

| Neighborhood | | Edge Weighting Function | | | |
|---|---|---|---|---|---|
| | | 1 | 2 | **Q** | inf |
| Polyhedra | STSPN Form. | II | II | II | II |
| | Problem Type | LP | SOCP | SOCP | LP |
| | Solver | CPLEX | CPLEX | CPLEX | CPLEX |
| Ellipsoids | STSPN Form. | II | II | II | II |
| | Problem Type | QCQP | SOCP | SOCP | QCQP |
| | Solver | CPLEX | CPLEX | CPLEX | CPLEX |
| Cubic Splines | STSPN Form. | II | I + $\epsilon$ | I + $\epsilon$ | II |
| | Problem Type | NLP | NLP | NLP | NLP |
| | Solver | IPOPT | IPOPT | IPOPT | IPOPT |

Euclidean or Quadratic norm the modified edge weighting function (2.50) is used, which ensures all the functions in the problem formulation to be at least once differentiable as required by the solver IPOPT. Here, the objective function (2.49) is employed, and the additional variables $d_i$ are thus not necessary. For the case of Manhattan or Maximum norm differentiability is directly ensured by the employment of the objective function (3.6). Table 3.1 reports the formulation and the solver used for each combination of neighborhood and edge weighting function, where the symbols $\epsilon$ indicates that the modified edge weighting function (2.50) is used.

Finally, it is worth mentioning that even if the original *chromosome* is infeasible due to the *crossover* operator, the employed solver always produces a feasible point and thus assures that each vertex lies within its neighborhood.

#### 3.1.2.4 Immigration

The *immigration* operator is not frequently used in GA implementations. However, since no random *mutations* are performed in the present implementation, and instead heuristics are used to improve the quality of each *chromosome*, in order to maintain sufficient genetic variety at each generation new chromosomes are generated according to the procedure illustrated in Section 3.1.1. The percentage of the new *population* made by these *chromosomes* is indicated by $1 - p_S - p_X$.

Finally, we observed that the convergence rate of the algorithm can be improved by not keeping the *immigration* percentage constant during the

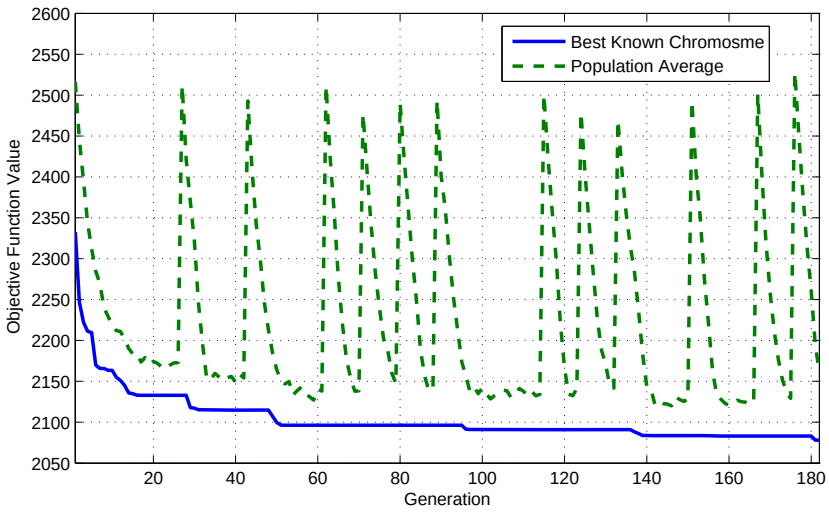



Figure 3.1: Convergence history of the hybrid random-key GA for solving the instance lin318_v in $\mathbb{R}^2$ with Euclidean norm ($g_{I_{\text{MAX}}} = 10$).

entire execution, but by varying it for one ore more generations if the objective value of the best *chromosome* remains constant for a certain sequence of consecutive generations, $g_{I_{\text{MAX}}}$. In this case the percentage of the propagated *chromosomes* is $\bar{p}_S$ and of the new *chromosomes* is $1 - \bar{p}_S$. Figure 3.1 illustrates this behavior. If the population average stabilizes, the GA is not able to further improve the best known chromosome, but if a large immigration cycle is performed, then the algorithm is able to find a new near *optimal tour*.

### 3.1.3 Termination criteria and population management

Two termination criteria are used in the current implementation of the algorithm: maximum number of *generations*, $g_{\text{MAX}}$, and maximum number of *generations* without improving the best known *chromosome*, $g_{C_{\text{MAX}}}$. Moreover, at the end of each *generation* duplicated *chromosomes*, i.e., *chromosomes* that simultaneously correspond to the same cycle and have the same objective function value, are replaced with newly generated *chromosomes*. Finally, in case no Touring heuristic is used, a final feasibility check is performed on each *chromosome*, and if it lies outside the feasible set, a penalty

**Algorithm 3.1** A Hybrid Random-Key Genetic Algorithm for solving STSPN instances.

Input: Neighborhoods $\mathcal{Q}_i, i = 1, \ldots, n$ and edge weighting function $\mathrm{d}(\cdot)$
Output: Near *optimal tour* $\boldsymbol{q}^{\star}_{\pi(i)}, i = 1, \ldots, n$ and its objective value $O^{\star}$

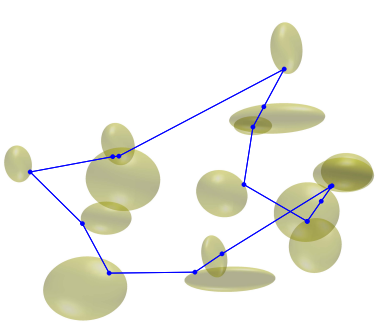
1. generate an initial *population* of *chromosomes*
2. let $O^{\star}$ be the objective of the best *chromosome* in the *population*
3. let $\boldsymbol{q}^{\star}_{\pi(i)}$ be the corresponding decoded tour
4. $g \leftarrow 1$; $g_C \leftarrow 1$; $g_I \leftarrow 1$
5. **repeat**
6. **if** $g_I < g_{I_{\text{MAX}}}$ **then**
7. *selection* of $p_S$ *chromosomes*
8. *crossover* of $p_X$ *chromosomes* with parameters $p_U, \mu, \lambda$
9. *immigration* of $1 - p_S - p_X$ *chromosomes*
10. **else**
11. *selection* of $\bar{p}_S$ *chromosomes*
12. *immigration* of $1 - \bar{p}_S$ *chromosomes*
13. $g_I \leftarrow 1$
14. apply improvement heuristics on $1 - p_S$ or $1 - \bar{p}_S$ *chromosomes*
15. (optional) *mutation* of $p_M$ *genes* components with parameter $\alpha$
16. let $O^{\star}_g$ be the objective of the best *chromosome* in the *generation*
17. **if** $O^{\star}_g \geq O^{\star} - \epsilon_I$ **then** increment $g_C$ and $g_I$
18. **else** let $O^{\star} \leftarrow O^{\star}_g$; update $\boldsymbol{q}^{\star}_{\pi(i)}$; $g_C \leftarrow 1$; $g_I \leftarrow 1$
19. increment $g$
20. remove duplicates and (optional) check feasibility
21. **until** $g > g_{\text{MAX}}$ or $g_C > g_{C_{\text{MAX}}}$
22. output $\boldsymbol{q}^{\star}_{\pi(i)}$ and $O^{\star}$

is added to its objective function value [24]. A high-level description of the algorithm is provided in Algorithm 3.1.

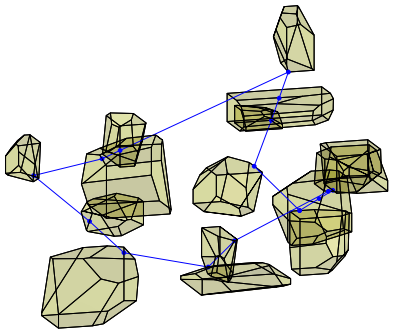
## 3.2 Computational results

### 3.2.1 Random STSPN instances

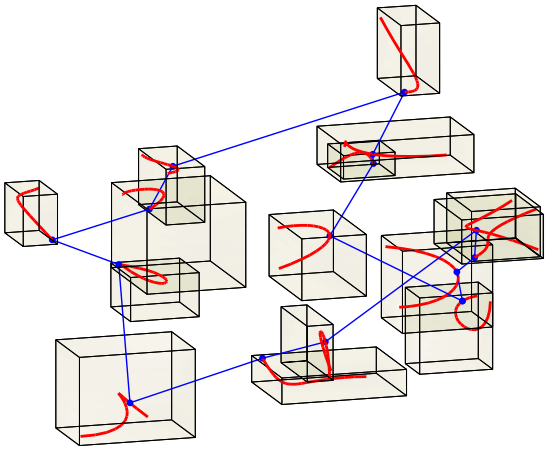
The first set of tests is performed on randomly generated STSPN instances with ellipsoids, polyhedra, and Bézier cubic splines in $\mathbb{R}^2$, $\mathbb{R}^3$, and $\mathbb{R}^7$. An example of such instances is illustrated in Figure 3.2.



(a) Ellipsoids in $\mathbb{R}^3$ (tspn3DE15).



(b) Polyhedra in $\mathbb{R}^3$ (tspn3DP15).



(c) Bézier Splines in $\mathbb{R}^3$ (tspn3DS15).

Figure 3.2: Randomly generated STSPN instances of comparable extension with 15 neighborhoods in $\mathbb{R}^3$ and optimal tours calculated using the Euclidean Norm.

#### 3.2.1.1 Euclidean norm

The computational results obtained using the Euclidean norm as edge weighting function are illustrated in Table 3.2. The neighborhood topology is reported in the column with label "$\mathcal{Q}$", the *population* size in the column with label "pop.", and the termination criteria in the column with label "$g_{\text{MAX}}$", where $g_{C_{\text{MAX}}}$ is enclosed in parentheses. Moreover, the minimum, maximum, and average values of the objective function recorded during a

sequence of five separate runs are reported in the columns with labels "min.", "max.", and "mean", the percentage gap between minimum and maximum attained objective function values in the column with label "gap", and, for the best run, the total number of *generations* and the number of *generations* where the value of the objective function has been improved in the columns with labels "$g$" and "impr.", respectively. The parameters used in the GA are: $p_S = 0.35$, $p_U = 0.30$, $p_X = 0.55$, $\mu = 0.0$, $\lambda = 0.3$, $p_M = 0.005$, $\alpha = 0.20$, and $g_{I_{\text{MAX}}} = g_{C_{\text{MAX}}}$. Due to the last parameter choice, no large *immigration* cycles are executed in these tests, and the algorithm terminates when $g_{C_{\text{MAX}}}$ consecutive *generations* do not improve the best known *chromosome*. The simulations have been performed on an `Intel Core i7 920 @2.67 GHz` processor with 6GB of memory running `Windows 7 Ultimate SP1`.

Although no optimality is guaranteed, the GA always returns the *optimal tour* for each of the six instances with up to 20 ellipsoidal neighborhoods ($n \leq 20$) in $\mathbb{R}^2$ and $\mathbb{R}^3$ solved to optimality using the Euclidean norm and the *second* STSPN formulation. For cases with a larger number of neighborhoods, no *optimal tour* has been calculated using the MINLP optimizer because of the excessive computational cost. However, using the heuristic procedure based on the *first* STSPN formulation a near *optimal tour* for the cases tspn2DE30 and tspn2DE50 has been obtained with objective function values 329.341 and 382.649, and CPU times 119 s and 81,111 s, respectively. For these two cases, the GA outperforms the heuristic procedure based on the MINLP optimizer both in terms of objective function value, by an average factor of 1.92%, and in terms of CPU time, up to two orders of magnitude.

For instances with more than 20 neighborhoods, where no optimality information is available, the GA tends to return more consistent results in case of ellipsoids and in $\mathbb{R}^7$. The average percentage gap between the minimum and maximum value of the objective function is 1.43% overall with a maximum of 3.74% for tspn3DP100. In case of ellipsoids it is 0.95%, in case of polyhedra 2.22%, and in case of Bézier splines 1.13%. Finally, if we consider $n \geq 100$, the average percentage gap between the minimum and maximum value of the objective function for instances in $\mathbb{R}^7$ it is 0.70%, in $\mathbb{R}^3$ 2.14%, and in $\mathbb{R}^2$ 1.47%.

Best known *chromosomes* are typically improved only in a fraction of the overall *generations*, and the employment of strong ad-hoc heuristics instead of more traditional *mutation* operators seems to strengthen this tendency. The *generations* where an improvement is observed are on average 36% of the overall *generations* with a maximum of 78% for tspn2DS200

Table 3.2: Hybrid random-key GA results for randomly generated STSPN instances with Euclidean norm (bold values are proven to be optimal).

| instance | $n$ | $m$ | $h$ | $\mathcal{Q}$ | pop. | $g_{\text{MAX}}$ | min. | max. | mean | gap | $g$ | impr. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| tspn2DE10 | 10 | 2 | 0.25 | E | 10 | 15(5) | **225.126** | 225.126 | 225.126 | 0.00 % | 7 | 2 |
| tspn2DE15 | 15 | 2 | 0.25 | E | 10 | 15(5) | **289.716** | 289.716 | 289.716 | 0.00 % | 4 | 1 |
| tspn2DE20 | 20 | 2 | 0.15 | E | 10 | 15(5) | **342.276** | 342.276 | 342.276 | 0.00 % | 4 | 1 |
| tspn2DE30 | 30 | 2 | 0.25 | E | 15 | 25(8) | 322.395 | 322.395 | 322.395 | 0.00 % | 9 | 2 |
| tspn2DE50 | 50 | 2 | 0.25 | E | 15 | 25(8) | 376.035 | 381.512 | 379.051 | 1.46 % | 17 | 8 |
| tspn2DE100 | 100 | 2 | 0.15 | E | 30 | 50(15) | 513.250 | 513.903 | 513.556 | 0.13 % | 23 | 6 |
| tspn2DE200 | 200 | 2 | 0.15 | E | 30 | 50(15) | 652.823 | 668.318 | 660.110 | 2.37 % | 22 | 8 |
| tspn2DP30 | 30 | 2 | 0.25 | P | 15 | 25(8) | 308.943 | 315.391 | 312.318 | 2.09 % | 25 | 6 |
| tspn2DP50 | 50 | 2 | 0.25 | P | 15 | 25(8) | 363.594 | 372.747 | 368.642 | 2.52 % | 12 | 5 |
| tspn2DP100 | 100 | 2 | 0.15 | P | 30 | 50(15) | 496.571 | 506.539 | 501.452 | 2.01 % | 50 | 10 |
| tspn2DP200 | 200 | 2 | 0.15 | P | 30 | 50(15) | 635.041 | 646.997 | 638.094 | 1.88 % | 50 | 16 |
| tspn2DS30 | 30 | 2 | 0.25 | S | 15 | 25(8) | 346.082 | 348.986 | 347.819 | 0.84 % | 25 | 7 |
| tspn2DS50 | 50 | 2 | 0.25 | S | 15 | 25(8) | 452.115 | 459.470 | 455.614 | 1.63 % | 25 | 13 |
| tspn2DS100 | 100 | 2 | 0.15 | S | 30 | 50(15) | 605.703 | 614.874 | 609.426 | 1.51 % | 50 | 22 |
| tspn2DS200 | 200 | 2 | 0.15 | S | 30 | 50(15) | 829.518 | 836.869 | 833.147 | 0.89 % | 50 | 39 |
| tspn3DE10 | 10 | 3 | 0.25 | E | 15 | 25(8) | **328.627** | 328.627 | 328.627 | 0.00 % | 9 | 2 |
| tspn3DE15 | 15 | 3 | 0.25 | E | 15 | 25(8) | **434.150** | 434.150 | 434.150 | 0.00 % | 7 | 1 |
| tspn3DE20 | 20 | 3 | 0.15 | E | 15 | 25(8) | **502.793** | 502.793 | 502.793 | 0.00 % | 7 | 1 |
| tspn3DE100 | 100 | 3 | 0.15 | E | 30 | 50(15) | 1,209.43 | 1,228.32 | 1,220.44 | 1.56 % | 19 | 6 |
| tspn3DE200 | 200 | 3 | 0.15 | E | 40 | 60(20) | 1,677.94 | 1,706.50 | 1,691.20 | 1.70 % | 32 | 14 |
| tspn3DP100 | 100 | 3 | 0.15 | P | 30 | 50(15) | 1,098.22 | 1,139.24 | 1,119.87 | 3.73 % | 50 | 14 |
| tspn3DP200 | 200 | 3 | 0.15 | P | 40 | 60(20) | 1,490.32 | 1,531.17 | 1,520.35 | 2.74 % | 60 | 21 |
| tspn3DS100 | 100 | 3 | 0.15 | S | 30 | 50(15) | 1,465.40 | 1,490.40 | 1,479.76 | 1.71 % | 29 | 13 |
| tspn3DS200 | 200 | 3 | 0.15 | S | 40 | 60(20) | 2,052.96 | 2,081.89 | 2,066.80 | 1.41 % | 39 | 19 |
| tspn7DE100 | 100 | 7 | 0.15 | E | 40 | 60(20) | 4,328.47 | 4,336.45 | 4,333.10 | 0.18 % | 29 | 5 |
| tspn7DE200 | 200 | 7 | 0.15 | E | 40 | 60(20) | 7,071.81 | 7,084.53 | 7,078.95 | 0.18 % | 30 | 9 |
| tspn7DP100 | 100 | 7 | 0.15 | P | 40 | 60(20) | 3,459.02 | 3,502.08 | 3,486.01 | 1.25 % | 60 | 11 |
| tspn7DP200 | 200 | 7 | 0.15 | P | 40 | 60(20) | 5,514.45 | 5,599.58 | 5,558.01 | 1.54 % | 39 | 18 |
| tspn7DS100 | 100 | 7 | 0.15 | S | 40 | 60(20) | 4,917.85 | 4,944.12 | 4,930.46 | 0.53 % | 32 | 12 |
| tspn7DS200 | 200 | 7 | 0.15 | S | 40 | 60(20) | 8,442.34 | 8,487.78 | 8,459.74 | 0.54 % | 42 | 14 |

Table 3.3: Comparison of the employed heuristics for randomly generated STSPN instances with Euclidean norm (bold values are proven to be optimal

| instance | HRKGA | | GA | | | LK | | | TOUR | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | obj. | CPU [s] | obj. | $g$ | CPU [s] | obj. | $g$ | CPU [s] | obj. | $g$ | CPU [s] |
| tspn2DE10 | **225.126** | 3.82 | 235.477 | 143 | 4.61 | 261.372 | 20 | 4.69 | 274.102 | 2 | 5.62 |
| tspn2DE15 | **289.716** | 6.51 | 382.234 | 235 | 7.85 | 341.178 | 37 | 7.90 | 449.064 | 2 | 9.06 |
| tspn2DE20 | **342.276** | 6.47 | 456.690 | 240 | 7.80 | 395.553 | 24 | 8.01 | 714.965 | 3 | 7.83 |
| tspn2DE30 | 322.395 | 17 | 640.886 | 405 | 20 | 382.462 | 20 | 20 | 797.626 | 8 | 21 |
| tspn2DE50 | 376.035 | 49 | 1,359.15 | 1,159 | 59 | 481.341 | 33 | 59 | 1,484.12 | 15 | 60 |
| tspn2DE100 | 513.250 | 194 | 3,189.91 | 2,064 | 233 | 648.300 | 24 | 234 | 3,275.21 | 17 | 242 |
| tspn2DE200 | 652.823 | 620 | 8,870.79 | 5,286 | 800 | 942.787 | 27 | 804 | 7,898.73 | 43 | 805 |
| tspn2DP30 | 308.943 | 31 | 486.379 | 491 | 38 | 379.917 | 24 | 39 | 533.076 | 24 | 38 |
| tspn2DP50 | 363.594 | 23 | 1,185.17 | 323 | 28 | 475.443 | 9 | 30 | 1,331.01 | 13 | 29 |
| tspn2DP100 | 496.571 | 237 | 2,798.58 | 1,629 | 284 | 646.263 | 23 | 290 | 3,111.32 | 51 | 287 |
| tspn2DP200 | 635.041 | 420 | 6,909.89 | 2,379 | 504 | 975.936 | 11 | 506 | 7,187.92 | 56 | 510 |
| tspn2DS30 | 346.082 | 35 | 654.784 | 512 | 42 | 359.575 | 29 | 42 | 822.531 | 15 | 42 |
| tspn2DS50 | 452.115 | 85 | 1,159.95 | 1,215 | 102 | 465.830 | 53 | 103 | 1,259.32 | 65 | 102 |
| tspn2DS100 | 605.703 | 887 | 3,512.14 | 7,497 | 1064 | 643.355 | 102 | 1,070 | 3,609.57 | 85 | 1,066 |
| tspn2DS200 | 829.518 | 3,631 | 9,227.06 | 10,000 | 1995 | 911.662 | 137 | 4,379 | 7,623.51 | 224 | 4,367 |
| tspn3DE10 | **328.627** | 24 | 335.957 | 651 | 30 | 390.032 | 91 | 30 | 328.692 | 10 | 32 |
| tspn3DE15 | **434.150** | 17 | 498.743 | 448 | 21 | 511.167 | 62 | 21 | 595.434 | 4 | 28 |
| tspn3DE20 | **502.793** | 12 | 656.651 | 306 | 15 | 567.856 | 42 | 15 | 851.883 | 3 | 16 |
| tspn3DE100 | 1,209.43 | 154 | 4,771.23 | 1,590 | 185 | 1,594.43 | 13 | 190 | 4,622.31 | 18 | 195 |
| tspn3DE200 | 1,677.94 | 686 | 1,1345.5 | 3,026 | 824 | 2,392.03 | 16 | 858 | 10,248.2 | 29 | 839 |
| tspn3DP100 | 1,098.22 | 230 | 4,048.34 | 1,471 | 277 | 1,596.70 | 13 | 282 | 4,277.98 | 54 | 279 |
| tspn3DP200 | 1,490.32 | 7694 | 9,046.99 | 2,448 | 923 | 2,429.48 | 10 | 963 | 9,278.15 | 74 | 934 |
| tspn3DS100 | 1,465.40 | 80 | 5,064.10 | 764 | 96 | 1,599.81 | 6 | 105 | 5,495.58 | 20 | 100 |
| tspn3DS200 | 2,052.96 | 443 | 11,952.7 | 1,878 | 531 | 2,367.30 | 9 | 539 | 10,962.2 | 31 | 545 |
| tspn7DE100 | 4,328.47 | 930 | 8,431.68 | 5,971 | 1,117 | 5,239.22 | 74 | 1,128 | 8,328.43 | 29 | 1,137 |
| tspn7DE200 | 7,071.81 | 1531 | 1,8726.5 | 5,690 | 1,837 | 8,893.16 | 34 | 1,871 | 16,716.5 | 27 | 1,855 |
| tspn7DP100 | 3,459.02 | 476 | 7,120.23 | 1,757 | 571 | 5,220.07 | 24 | 571 | 6,861.66 | 61 | 578 |
| tspn7DP200 | 5,514.45 | 783 | 15,106.4 | 2,887 | 941 | 8,987.24 | 14 | 944 | 14,153.0 | 42 | 953 |
| tspn7DS100 | 4,917.85 | 417 | 9,602.17 | 3,130 | 500 | 5,046.07 | 29 | 508 | 9,165.12 | 19 | 507 |
| tspn7DS200 | 8,442.34 | 1364 | 19,390.7 | 5,688 | 1637 | 8,721.37 | 35 | 1,651 | 18,435.3 | 24 | 1,646 |

and a minimum of 14% for tspn3DE20. In particular, in case of convex neighborhoods where the Touring heuristic always returns the optimal value for a fixed cycle, the average percentage is 30%. In case of non-convex neighborhoods, where the Touring heuristic is weaker as optimality is not guaranteed, the average percentage is 46%. Finally, higher dimension seems to influence this tendency, too: for instances in $\mathbb{R}^7$ the average percentage is 30%, whereas in $\mathbb{R}^2$ and in $\mathbb{R}^3$ it is 39%.

Table 3.3 proposes a comparison among the best values of the objective function attained with the proposed GA using both heuristics (HRKGA), using traditional *mutation* operators and no heuristic (GA), using the Lin-Keringhan heuristic only (LK), and using the Touring heuristic only (TOUR). In the last three implementations the algorithm terminates either if $g > 10,000$ or if the CPU time is 20% larger than the CPU time spent by the first implementation. The best obtained value of the objective function is reported in the column with label "obj.", and the measured CPU time in the column with label "CPU". It is worth noticing, that since the GA code has been parallelized the reported CPU time is approximatively seven times larger than the wall clock time observed for the execution of the algorithm on the employed CPU.

The GA implementation strongly benefits from the employment of the ad-hoc heuristics. A more traditional implementation with *mutation* operators leads within comparable CPU time to objective function values that are larger than the one obtained with the proposed approach by an average factor of 4.65 with a maximum of 13.59 for tspn2DE200. This factor tends to increase with the number of neighborhoods and to decrease with the space dimension: the average values range in $\mathbb{R}^2$ from 1.60, for the case $n = 30$, up to 11.86, for the case $n = 200$, and in $\mathbb{R}^7$ from 1.99, for the case $n = 100$, up to 2.56, for the case $n = 200$. Finally, the neighborhood topology does not seem to have a remarkable influence on these factors: the average values are 5.09 for ellipsoids, 4.49 for polyhedra, and 4.36 for Bézier splines.

The GA implementation that uses the Lin-Keringhan heuristic considerably improves the performance of the algorithm and leads to objective function values that are larger than the one obtained with the proposed approach only by an average factor of 1.27 with a maximum of 1.63 for tspn3DP200. Moreover, this implementation seems to perform better with non-fat neighborhoods: the average factor is only 1.07 for Bézier splines, whereas it is 1.30 for ellipsoids and 1.45 for polyhedra. Finally, the number of neighborhoods and the space dimension does not seem to have a strong influence on these factors: the average values range in $\mathbb{R}^2$ from 1.14, for the case $n = 30$, up to 1.36, for the case $n = 200$, and in $\mathbb{R}^7$ from 1.25, for the

case $n = 100$, up to 1.31, for the case $n = 200$.

Finally, the GA implementation that uses the Turing heuristic only has a similar behavior as the first implementation, and leads to objective function values that are larger than the one obtained with the proposed approach by an average factor of 4.59 with a maximum of 12.10 for tspn3DP200. However, when this heuristic is employed after the Lin-Keringhan heuristic it further improves the quality of each *chromosome*, and the GA eventually converges to the *optimal tour* for the cases where $n \leq 20$. This would be very hard using traditional *mutation* operators because of the continuous nature of the optimization problem.

#### 3.2.1.2 Manhattan, Maximum, and Quadratic norm

The following tests are performed on the same instances used in the previous Section, but using the Manhattan, the Maximum, and the Quadratic norm as edge weighting functions. The computational results are illustrated in Tables 3.4 and 3.5. The edge weighting functions is reported in the column with label "d(·)". The parameters used in the GA are the same as the ones listed in Section 3.2.1. For the Quadratic norm in $\mathbb{R}^2$ and $\mathbb{R}^3$, the $Q$ matrices are diagonal matrices with diagonal elements $[1, 1/2]$ and $[1, 1/2, 1/10]$, respectively.

The GA returns the *optimal tour* for each of the instances with up to 20 ellipsoidal neighborhoods solved to optimality using the Manhattan or the Maximum norm and the *second* STSPN formulation. Moreover, the GA outperforms the MINLP optimizer in terms of CPU time up to two orders of magnitude for $n = 20$, although the MINLP optimizer is faster for $n = 10$.

Table 3.4 illustrates that for instances with more than 20 neighborhoods where no optimality information is available the GA tends to return more consistent results in case of Manhattan norm. The average percentage gap between the minimum and maximum value of the objective function recorded during a sequence of five separate runs is 1.37% with a maximum of 3.34% for tspn3DP100. In case of Manhattan norm the average gap is 1.22%, in case of Maximum norm 1.36%, and in case of Quadratic norm 1.52%.

Table 3.5 confirms the importance of the ad-hoc heuristics in the implementation of the GA even with different types of edge weighting functions. The objective function values are larger than the ones obtained by the proposed approach by an average factor of 5.01 for the case of traditional GA implementation, of 1.27 for the case of Lin-Keringhan heuristic only, and of 5.11 for the case of Touring heuristic only. Finally, the type of norm does

Table 3.4: Hybrid random-key GA results for randomly generated STSPN instances with different norms (bold values are proven to be optimal).

| instance | $n$ | $m$ | $h$ | $\mathcal{Q}$ | d$(\cdot)$ | pop. | $g_{\text{MAX}}$ | min. | max. | mean | gap | $g$ | impr. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| tspn2DE10 | 10 | 2 | 0.25 | E | 1 | 10 | 15(5) | **286.164** | 286.164 | 286.164 | 0.00 % | 4 | 1 |
| tspn2DE15 | 15 | 2 | 0.25 | E | 1 | 15 | 25(8) | **348.047** | 353.165 | 352.961 | 1.47 % | 11 | 3 |
| tspn2DE20 | 20 | 2 | 0.15 | E | 1 | 15 | 25(8) | **419.474** | 423.665 | 421.150 | 1.00 % | 7 | 1 |
| tspn2DE100 | 100 | 2 | 0.15 | E | 1 | 40 | 60(20) | 634.213 | 642.087 | 638.719 | 1.24 % | 60 | 14 |
| tspn2DP100 | 100 | 2 | 0.15 | P | 1 | 40 | 60(20) | 635.092 | 638.385 | 636.676 | 0.52 % | 29 | 5 |
| tspn2DS100 | 100 | 2 | 0.15 | S | 1 | 40 | 60(20) | 746.289 | 752.592 | 749.046 | 0.84 % | 41 | 15 |
| tspn2DE10 | 10 | 2 | 0.25 | E | inf | 10 | 15(5) | **190.427** | 190.427 | 190.427 | 0.00 % | 4 | 1 |
| tspn2DE15 | 15 | 2 | 0.25 | E | inf | 15 | 25(8) | **263.742** | 263.742 | 263.742 | 0.00 % | 24 | 7 |
| tspn2DE20 | 20 | 2 | 0.15 | E | inf | 15 | 25(8) | **299.752** | 299.752 | 299.752 | 0.00 % | 12 | 3 |
| tspn2DE100 | 100 | 2 | 0.15 | E | inf | 40 | 60(20) | 449.652 | 450.198 | 449.871 | 0.12 % | 41 | 11 |
| tspn2DP100 | 100 | 2 | 0.15 | P | inf | 40 | 60(20) | 432.691 | 436.740 | 435.008 | 0.94 % | 28 | 6 |
| tspn2DS100 | 100 | 2 | 0.15 | S | inf | 40 | 60(20) | 538.272 | 540.486 | 539.385 | 0.41 % | 52 | 15 |
| tspn2DE100 | 100 | 2 | 0.15 | E | **Q** | 40 | 60(20) | 427.782 | 431.627 | 430.022 | 0.90 % | 25 | 5 |
| tspn2DP100 | 100 | 2 | 0.15 | P | **Q** | 40 | 60(20) | 415.855 | 420.281 | 417.543 | 1.06 % | 60 | 18 |
| tspn2DS100 | 100 | 2 | 0.15 | S | **Q** | 40 | 60(20) | 516.661 | 524.588 | 520.927 | 1.53 % | 36 | 18 |
| tspn3DE100 | 100 | 3 | 0.15 | E | 1 | 40 | 60(20) | 1,671.78 | 1,702.95 | 1,694.86 | 1.86 % | 29 | 9 |
| tspn3DP100 | 100 | 3 | 0.15 | P | 1 | 40 | 60(20) | 1,590.03 | 1,620.01 | 1,608.21 | 1.89 % | 26 | 8 |
| tspn3DS100 | 100 | 3 | 0.15 | S | 1 | 40 | 60(20) | 2,087.85 | 2,108.14 | 2,098.42 | 0.97 % | 50 | 15 |
| tspn3DE100 | 100 | 3 | 0.15 | E | inf | 40 | 60(20) | 931.300 | 951.159 | 938.033 | 2.13 % | 27 | 8 |
| tspn3DP100 | 100 | 3 | 0.15 | P | inf | 40 | 60(20) | 846.255 | 874.481 | 862.279 | 3.34 % | 31 | 11 |
| tspn3DS100 | 100 | 3 | 0.15 | S | inf | 40 | 60(20) | 1,125.69 | 1,139.44 | 1,130.71 | 1.22 % | 31 | 12 |
| tspn3DE100 | 100 | 3 | 0.15 | E | **Q** | 40 | 60(20) | 719.049 | 732.483 | 727.968 | 1.87 % | 29 | 10 |
| tspn3DP100 | 100 | 3 | 0.15 | P | **Q** | 40 | 60(20) | 659.747 | 674.921 | 666.932 | 2.30 % | 25 | 6 |
| tspn3DS100 | 100 | 3 | 0.15 | S | **Q** | 40 | 60(20) | 892.169 | 904.903 | 897.433 | 1.43 % | 36 | 17 |

Table 3.5: Comparison of the employed heuristics for randomly generated STSPN instances with different norms (bold values are proven to be optimal).

| instance | d(·) | HRKGA | | GA | | | LK | | | TOUR | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | obj. | CPU [s] | obj. | $g$ | CPU [s] | obj. | $g$ | CPU [s] | obj. | $g$ | CPU [s] |
| tspn2DE10 | 1 | **286.164** | 5.55 | 309.642 | 209 | 6.68 | 328.380 | 35 | 6.78 | 314.279 | 2 | 7.37 |
| tspn2DE15 | 1 | **348.047** | 23 | 376.723 | 580 | 28 | 430.722 | 87 | 28 | 528.913 | 10 | 29 |
| tspn2DE20 | 1 | **419.474** | 9.45 | 677.591 | 349 | 11 | 489.925 | 40 | 12 | 782.783 | 9 | 12 |
| tspn2DE100 | 1 | 634.213 | 448 | 3,756.70 | 3,410 | 538 | 803.929 | 46 | 546 | 3,839.82 | 51 | 542 |
| tspn2DP100 | 1 | 635.092 | 63 | 4,329.39 | 453 | 76 | 877.804 | 3 | 84 | 4,098.55 | 49 | 76 |
| tspn2DS100 | 1 | 746.289 | 180 | 4,788.16 | 1,470 | 217 | 797.359 | 21 | 219 | 4,642.29 | 32 | 220 |
| tspn2DE10 | inf | **190.427** | 15 | 195.006 | 569 | 18 | 215.405 | 90 | 18 | 262.931 | 6 | 20 |
| tspn2DE15 | inf | **263.742** | 31 | 302.507 | 803 | 37 | 299.206 | 125 | 38 | 359.831 | 8 | 38 |
| tspn2DE20 | inf | **299.752** | 14 | 380.983 | 529 | 17 | 343.694 | 52 | 17 | 501.880 | 13 | 18 |
| tspn2DE100 | inf | 449.652 | 573 | 2,695.83 | 4,322 | 688 | 560.104 | 85 | 694 | 2,961.06 | 51 | 691 |
| tspn2DP100 | inf | 432.691 | 90 | 3,031.29 | 626 | 108 | 597.079 | 6 | 109 | 2,932.34 | 35 | 110 |
| tspn2DS100 | inf | 538.272 | 214 | 3,498.25 | 1,768 | 257 | 562.358 | 18 | 261 | 3,212.25 | 33 | 258 |
| tspn2DE100 | **Q** | 427.782 | 256 | 2,297.92 | 1,800 | 308 | 541.307 | 29 | 316 | 2,967.74 | 20 | 319 |
| tspn2DP100 | **Q** | 415.855 | 465 | 2,457.72 | 3,088 | 559 | 547.027 | 58 | 565 | 2,406.23 | 76 | 564 |
| tspn2DS100 | **Q** | 516.661 | 1247 | 2,896.17 | 9,877 | 1,497 | 555.707 | 135 | 1,506 | 2,535.75 | 175 | 1,507 |
| tspn3DE100 | 1 | 1,671.78 | 462 | 6,578.64 | 3,381 | 555 | 2,249.02 | 41 | 565 | 6,868.81 | 40 | 559 |
| tspn3DP100 | 1 | 1,590.03 | 65 | 5,956.40 | 426 | 78 | 2,408.01 | 3 | 80 | 6,415.42 | 36 | 78 |
| tspn3DS100 | 1 | 2,087.85 | 225 | 6,388.66 | 1,784 | 270 | 2,220.78 | 22 | 276 | 7,483.25 | 30 | 272 |
| tspn3DE100 | inf | 931.300 | 447 | 3,825.08 | 3,287 | 537 | 1,255.31 | 51 | 537 | 3,705.41 | 34 | 549 |
| tspn3DP100 | inf | 846.255 | 67 | 3,151.86 | 433 | 80 | 1,346.33 | 3 | 91 | 3,556.96 | 46 | 81 |
| tspn3DS100 | inf | 1,125.69 | 183 | 4,666.32 | 1,476 | 219 | 1,199.72 | 14 | 224 | 4,395.08 | 22 | 226 |
| tspn3DE100 | **Q** | 719.049 | 389 | 2,992.46 | 2,705 | 467 | 964.035 | 37 | 467 | 2,973.16 | 29 | 475 |
| tspn3DP100 | **Q** | 659.747 | 186 | 2,562.75 | 1,167 | 223 | 974.110 | 16 | 224 | 3,001.75 | 32 | 226 |
| tspn3DS100 | **Q** | 892.169 | 112 | 3,504.64 | 791 | 134 | 977.369 | 9 | 139 | 3,400.60 | 24 | 138 |

not seem to influence the results. In case of traditional implementation of the GA, the average factors are for the Manhattan, the Maximum, and the Quadratic norm 4.98, 5.25, and 4.81, respectively. In case of Lin-Keringhan heuristic, the average factors are 1.27, 1.28, and 1.26, respectively. In case of Touring heuristic, the average factors are 5.08, 5.24, and 5.02, respectively.

### 3.2.2 CETSP Instances

The second set of tests is performed by applying the hybrid random-key GA on the Close Enough TSP (CETSP) instances proposed by Mennell [74]. The neighborhoods are balls in $\mathbb{R}^2$ and $\mathbb{R}^3$, and the Euclidean and Manhattan norms are used as edge weighting functions. Table 3.6 compares the results obtained by the proposed GA (HRKGA) with the best results reported by Mennell [74]. The instances are denoted by "_v" to specify that they have variable radii as explained in [74]. The used edge weighting functions is reported in the column with label "d(·)", the best attained objective function value in the column with label "obj.", and the percentage improvement of the objective function value with respect to results obtained in [74] in the column with label "obj. impr.". Moreover, the CPU time is reported in the column with label "CPU" and the total number of *generations* and the number of *generations* where the value of the objective function has been improved in the columns with labels "$g$" and "impr.", respectively.

A heuristic procedure is initially performed to search for a set of parameters that improve the performance of the hybrid random-key GA. Details on this procedure are reported in Appendix A.9. The final parameters used in the GA are the same as the ones listed in Section 3.2.1 but $p_S = 0.60$, $p_U = 0.40$, $p_X = 0.30$, $g_{I_{\text{MAX}}} = 10$, $\bar{p}_S = 1$ *chromosome*, and $\epsilon_I = 0.1$. The algorithm is terminated here only when a near *optimal tour* better than the one provided in the literature is found.

Although the proposed method is not specifically tailored for CETSP instances, the results show that it outperforms all the algorithms tested in Mennell [74] in terms of attained objective function values, especially for instances in $\mathbb{R}^3$. The best known near *optimal tours* have been improved by an average factor of 1.92% and by a maximum factor of 8.03% for bonus1000_v in $\mathbb{R}^3$ with Euclidean norm. In particular, the average improvement factors are 1.22% in $\mathbb{R}^2$ with Euclidean norm, 3.27% in $\mathbb{R}^3$ with Euclidean norm, and 1.28% in $\mathbb{R}^2$ with Manhattan norm. On the contrary, the comparison of CPU times does not lead to a clear conclusion since the values for some tests have not been reported and the computational environments are different. All experiments reported by Mennell [74] are performed on an

Table 3.6: CETSP instances in $\mathbb{R}^2$ and $\mathbb{R}^3$ with variable radii proposed by Mennell [74].

| instance | $n$ | $m$ | d$(\cdot)$ | HRKGA | | | | | CETSP | | obj. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | obj. | CPU [s] | pop. | $g$ | impr. | obj. | CPU [s] | impr. |
| kroD100_v | 100 | 2 | 2 | 141.824 | 2,003 | 50 | 190 | 28 | 141.834 | 1,008 | 0.01% |
| rat195_v | 195 | 2 | 2 | 68.224 | 81 | 50 | 7 | 5 | 68.224 | 115 | 0.00% |
| lin318_v | 318 | 2 | 2 | 2,078.20 | 9,294 | 50 | 182 | 52 | 2,080.57 | N/A | 0.11% |
| rd400_v | 400 | 2 | 2 | 1,245.43 | 10,138 | 60 | 173 | 27 | 1,252.38 | 12,795 | 0.55% |
| pcb442_v | 442 | 2 | 2 | 226.118 | 1,014 | 60 | 40 | 26 | 235.188 | 1,161 | 3.86% |
| d493_v | 493 | 2 | 2 | 138.291 | 2,166 | 60 | 25 | 22 | 140.120 | 46 | 1.31% |
| dsj1000_v | 1,000 | 2 | 2 | 639.721 | 30,355 | 100 | 76 | 48 | 653.128 | N/A | 2.05% |
| team1_100_v | 101 | 2 | 2 | 388.537 | 108 | 50 | 7 | 6 | 388.537 | 47 | 0.00% |
| team5_499_v | 500 | 2 | 2 | 448.531 | 1,309 | 60 | 15 | 12 | 454.327 | 2,763 | 1.28% |
| bonus1000_v | 1,001 | 2 | 2 | 957.119 | 19,159 | 100 | 54 | 38 | 987.114 | N/A | 3.04% |
| kroD100_v | 100 | 3 | 2 | 169.868 | 127 | 50 | 10 | 9 | 171.568 | 121 | 0.99% |
| rat195_v | 195 | 3 | 2 | 82.358 | 249 | 50 | 12 | 8 | 84.470 | 1945 | 2.50% |
| lin318_v | 318 | 3 | 2 | 2,154.91 | 3,124 | 50 | 50 | 23 | 2,189.43 | 34,640 | 1.58% |
| rd400_v | 400 | 3 | 2 | 3,590.70 | 8,173 | 60 | 100 | 13 | 3,592.60 | N/A | 0.05% |
| pcb442_v | 442 | 3 | 2 | 247.020 | 3,007 | 60 | 100 | 42 | 258.404 | 7,406 | 4.41% |
| d493_v | 493 | 3 | 2 | 727.259 | 2,027 | 60 | 24 | 19 | 761.065 | 25,013 | 4.44% |
| dsj1000_v | 1,000 | 3 | 2 | 1,916.99 | 942 | 60 | 2 | 2 | 2,074.84 | 47 | 7.61% |
| team1_100_v | 101 | 3 | 2 | 905.732 | 89 | 50 | 5 | 4 | 907.593 | 127 | 0.21% |
| team5_499_v | 500 | 3 | 2 | 816.081 | 2,044 | 60 | 19 | 16 | 840.477 | 106,692 | 2.90% |
| bonus1000_v | 1,001 | 3 | 2 | 2,473.50 | 964 | 60 | 2 | 2 | 2,689.41 | 90 | 8.03% |
| kroD100_v | 100 | 2 | 1 | 173.619 | 110 | 50 | 22 | 11 | 174.013 | 22 | 0.23% |
| rat195_v | 195 | 2 | 1 | 82.200 | 108 | 50 | 7 | 6 | 82.200 | 32 | 0.00% |
| lin318_v | 318 | 2 | 1 | 2,484.24 | 844 | 50 | 44 | 26 | 2,505.41 | N/A | 0.85% |
| rd400_v | 400 | 2 | 1 | 1,535.57 | 705 | 60 | 14 | 9 | 1,544.54 | 614 | 0.58% |
| pcb442_v | 442 | 2 | 1 | 277.805 | 885 | 60 | 12 | 10 | 281.254 | 169 | 1.23% |
| d493_v | 493 | 2 | 1 | 172.988 | 2037 | 60 | 46 | 26 | 179.563 | 223 | 3.66% |
| dsj1000_v | 1,000 | 2 | 1 | 744.985 | 23226 | 100 | 119 | 53 | 758.119 | N/A | 1.73% |
| team1_100_v | 101 | 2 | 1 | 490.421 | 401 | 50 | 59 | 16 | 494.114 | 391 | 0.75% |
| team5_499_v | 500 | 2 | 1 | 556.576 | 1715 | 60 | 48 | 26 | 563.110 | 1,899 | 1.16% |
| bonus1000_v | 1,001 | 2 | 1 | 1,193.92 | 14,808 | 100 | 89 | 45 | 1,226.39 | 33,729 | 2.65% |

`Intel Pentium @2.4 GHz` processor with 3GB of memory running `Windows XP Professional`, and thus the frequency of the processor used in this work is higher by a factor of 1.11. Nevertheless, we can observe that the geometric average of the ratio of the CPU time spent by the proposed GA and the one reported in [74] is 1.06 with a maximum ratio of 47 for d493_v in $\mathbb{R}^2$ with Euclidean norm, and a minimum ratio of 0.019 for team5_499_v in $\mathbb{R}^3$ with Euclidean norm.

Finally, the new parameter set and the presence of a large *immigration* operation when $g_I > g_{I_{\text{MAX}}}$ allows the proposed GA to increase the number of *generations* where an improvement of the best *chromosome* is observed. The comparison of the current results with the ones reported in Section 3.2 for TSPN instances with ellipsoidal neighborhoods shows this tendency (the latter results are enclosed in parentheses hereafter). On average, the best *chromosome* has been improved in 62% (27%) of the overall *generations*. In particular, in case of balls in $\mathbb{R}^2$ and Euclidean norm the percentage is 58% (30%), in case of balls in $\mathbb{R}^3$ and Euclidean norm it is 70% (25%), and in case of balls in $\mathbb{R}^2$ and Manhattan norm it is 58% (24%).

## 3.3 Conclusion

In this chapter a hybrid random-key Genetic Algorithm (GA) has been proposed to solve the Traveling Salesman Problem whit Neighborhoods (TSPN). Based on the MINLP formulation presented in chapter 2, this approach uses random-key coding for the chromosomes, and it exploits the efficiency of ad-hoc heuristics to improve the quality of each chromosome rather than more traditional mutation operators.

Numerical results show that the convergence rate of the algorithm can be improved by not keeping the percentage of the newly introduced chromosomes constant during its entire execution. In the GA final implementation this percentage is changed for one generation every time the best known objective function value remains constant for a certain sequence of consecutive generations, i.e., when the population average tends to stabilize.

Finally, the proposed method converges to the optimal tour in all the cases proven to be optimal by using the convex MINLP optimizers. Moreover, applying the GA on TSPN instances available in the literature with circular and spherical neighborhoods, although the proposed method is not tailored for those specific instances, the best known near optimal tours have been improved on average by 1.92%. For instances in $\mathbb{R}^3$ only the average improvement is 3.27%.

# Chapter 4

# 7DOF Industrial Vision Inspection System

In this chapter, we apply the proposed hybrid random-key Genetic Algorithm to an industrial vision inspection system. We illustrate how the neighborhoods are defined and how the edge weighting function is customized to account for cycle time minimization and obstacle avoidance. Finally, simulation and experimental results are discussed.

## 4.1 Problem Formulation

The application analyzed in this chapter is a robotic system used for inspecting assembled industrial components, such as engine blocks or evaporators. The system consists of a 6 Degrees of Freedom (DOF) robotic manipulator and 1 DOF turntable. Its configuration space is thus seven-dimensional. The component that has to be inspected is placed on the turntable, and a camera is mounted at the end-effector of the manipulator. Afterwards, a predefined set of features on the component surface is inspected using the camera and an on-line image processing software. To perform the inspection, the component is rotated and the manipulator is simultaneously actuated to locate the camera at all the relative placements with respect to the component surface where the required images have to be acquired from. Since the sequence of the inspection placements is not fixed, the overall cycle time required to acquire all the images can be optimized by searching for an optimal sequence.

Each relative placement of the camera with respect to the component surface is specified by six parameters. Since the configuration space is seven-

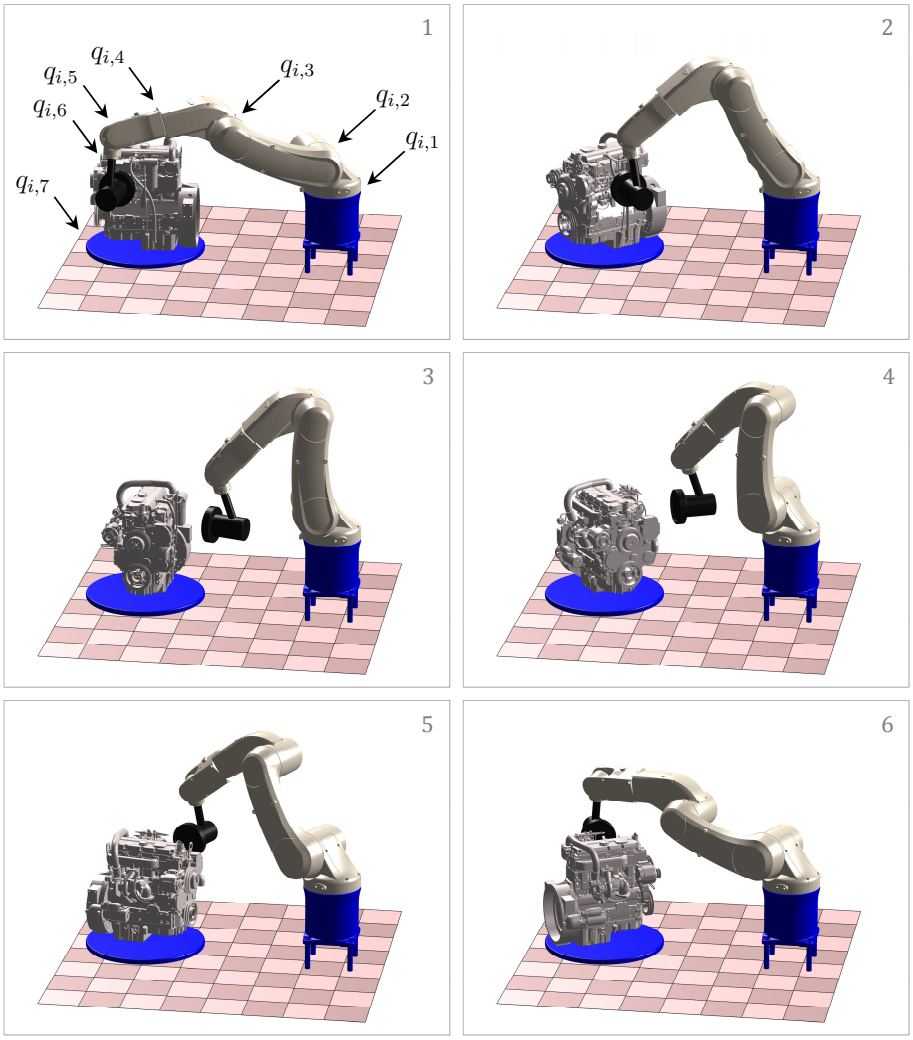



Figure 4.1: Robotic vision inspection system: six different configurations of neighborhood $i = 14$ that correspond to the same relative placement of the camera with respect to the component.

dimensional ($m$=7), this system has one degree of redundancy. Figure 4.1 illustrates this concept by showing six among the infinitely many configurations that correspond to the same relative placement between the camera and the component, and thus to identical images. By exploiting this redun-

dancy of the system, the cycle time can be further improved by searching not only for an optimal sequence, but also for an optimal sequence of optimal configurations.

Heuristic approaches for similar problems are proposed in the literature [44, 45, 46, 91, 102, 103]. However, in these works the continuous nature of the problem is lost by extracting a small number of discrete samples for each neighborhood. Afterwards, Saha et al. [91] propose to approximate the resulting GTSP by calculating a minimum group spanning tree and by performing a preorder tree walk. Collision avoidance is considered using a sampling-based planner. Alternatively, Gueta et al. [44] propose to predetermine a near optimal sequence, and then to choose a configuration in each neighborhood by using a heuristic approach.

We model this optimization problem as a STSPN, where each neighborhood is identified by the set of all the possible configurations that correspond to the same relative placement between the camera and the component, i.e., to the same image. Since the considered system has one degree of redundancy these regions can be represented as possibly non-connected curves in the seven-dimensional configuration space of the robotic system.

Moreover, specific considerations are required about the nature of the edge weighting function. First, since the objective function that needs to be optimized is the overall cycle time, the simple Euclidean norm cannot be used for this purposes, and a more realistic criterion is needed to evaluate the time spent by the manipulator and the turntable to move from one placement to the next. Second, collision avoidance has to be considered.

## 4.2 Objective function evaluation

### 4.2.1 Traveling time

During the motion of the robotic system, the joints move simultaneously. Thus, a good estimate of the traveling time spent to move from configuration $\boldsymbol{q}_i$ to configuration $\boldsymbol{q}_j$ is given by the Weighted Maximum norm [44]:

$$\mathrm{d}\left(\boldsymbol{q}_i, \boldsymbol{q}_j\right) = \max_{k=1,\ldots,7} \left\{ \frac{|q_{i,k} - q_{j,k}|}{\omega_k} \right\} . \tag{4.1}$$

where $\omega_k$ is the angular velocity set by the controller for joint $k$. Moreover, the Quadratic norm defined by the diagonal matrix $Q$ with diagonal elements $[1/\omega_1^2, \ldots, 1/\omega_7^2]$ is also employed. Although the latter norm leads to a minimum path traveled by each joint but not directly to a minimum traveling time, it has been observed that the actual cycle time calculated

by using this norm is smaller than the one calculated using the weighted Maximum norm.

The two edge weighting functions can be evaluated very efficiently and thus are used in the Lin-Keringhan and Touring heuristics, and in the probabilistic path planner. However, the full evaluation of the objective function for each *chromosome* at steps 2. and 16. of the hybrid random-key GA illustrated in Section 3.1.2 requires a more accurate model. When the system moves from configuration $\boldsymbol{q}_i$ to configuration $\boldsymbol{q}_j$, the actual traveling time can be estimated as:

$$\mathrm{d}\left(\boldsymbol{q}_i, \boldsymbol{q}_j\right) = \max_{k=1,\ldots,7} \left\{ \Delta t_0 + \left\{ \begin{array}{ll} \frac{\omega_k}{\alpha_k} + \frac{\left|q_{i,k} - q_{j,k}\right|}{\omega_k} & \text{if } \left|q_{i,k} - q_{j,k}\right| > \frac{\omega_k^2}{\alpha_k} \\ 2\sqrt{\frac{\left|q_{i,k} - q_{j,k}\right|}{\alpha}} & \text{otherwise} \end{array} \right\} \right., \tag{4.2}$$

where $\alpha_k$ is the angular acceleration set by the controller for joint $k$ and $\Delta t_0$ is the controller delay.

Finally, the probabilistic path planner may introduce mid configurations when the robotic system moves from $\boldsymbol{q}_i$ to $\boldsymbol{q}_j$ to avoid collisions. Considering a point to point motion from a generic start configuration $\boldsymbol{q}_s$ to a generic goal configuration $\boldsymbol{q}_g$ with initial and final joint velocities $\dot{\boldsymbol{q}}_s$ to $\dot{\boldsymbol{q}}_g$, respectively, the traveling time can be estimated as:

$$\mathrm{d}\left(\boldsymbol{q}_s, \boldsymbol{q}_g\right) = \max_{k=1,\ldots,7} \left\{ \begin{array}{ll} \frac{2\alpha_k\left|q_{s,k} - q_{g,k}\right| - 2\omega_k^2 + \dot{q}_{s,k}^2 + \dot{q}_{g,k}^2}{2\alpha_k\omega_k} + \frac{2\omega_k - \dot{q}_{s,k} - \dot{q}_{g,k}}{\alpha_k} & \\ \qquad \text{if } \left|q_{s,k} - q_{g,k}\right| > \frac{2\omega_k^2 - \dot{q}_{s,k}^2 - \dot{q}_{g,k}^2}{2\alpha_k} & \\ \frac{\sqrt{4\alpha_k\left|q_{s,k} - q_{g,k}\right| + 2\dot{q}_{s,k}^2 + 2\dot{q}_{g,k}^2} - \dot{q}_{s,k} - \dot{q}_{g,k}}{\alpha_k} & \\ \qquad \text{otherwise} & \end{array} \right\}, \tag{4.3}$$

We require for each joint the velocities $\dot{q}_{s,k}$ and $\dot{q}_{g,k}$ to have the same sign, and thus they are considered to be positive in Equation (4.3). If an inversion in the joint motion direction has to be modeled, a mid configuration with zero joint velocity is introduced.

### 4.2.2 Obstacle avoidance

To obtain a realistic evaluation of the traveling time for the robotic system to move from configuration $\boldsymbol{q}_i$ to configuration $\boldsymbol{q}_j$ a collision-free path connecting them has to be found. This additional requirement drastically increases the complexity of the optimization problem and requires the employment of an ad-hoc path planning technique.

**Algorithm 4.1** BiRRT based Single and Multiple Query Path Planner for the STSPN.

Input: tour $\boldsymbol{q}_{\pi(i)}, i = 1, \ldots, n$, edge weighting function $\mathrm{d}(\cdot)$, indicator matrix $M_{conn}$, sampling threshold $l_{samp}$, distance matrix $D$, and roadmap $G = (V_G, E_G)$
Output: collision-free tour length $O$, updated $D$ and $G$

1. $O \leftarrow 0$
2. **for** $i = 1$ to $n$ **do**
3. **if** $|V_G| > l_{samp}$ **and** $M_{conn}[\pi(i), \pi(i+1)] = 1$ **then**
4. **if** $\boldsymbol{q}_{\pi(i)} \notin V_G$ **then** `connect`$(G, \boldsymbol{q}_{\pi(i)})$
5. **if** $\boldsymbol{q}_{\pi(i+1)} \notin V_G$ **then** `connect`$(G, \boldsymbol{q}_{\pi(i+1)})$
6. $v_s \leftarrow$ `index`$(V_G, \boldsymbol{q}_{\pi(i)})$; $v_g \leftarrow$ `index`$(V_G, \boldsymbol{q}_{\pi(i+1)})$
7. **if** $v_s \neq$ NIL **and** $v_g \neq$ NIL **then**
8. **if** $D(v_s, v_g) = -1$ **then**
9. $D(v_s, v_g) \leftarrow$ `shortest_path`$(G, \boldsymbol{q}_{\pi(i)}, \boldsymbol{q}_{\pi(i+1)})$
10. $O \leftarrow O + D[v_S, v_G]$
11. **else**
12. $O \leftarrow O + \infty$
13. **else**
14. $P = (V_P, E_P)$: $V_P \leftarrow \{\boldsymbol{q}_{\pi(i)}, \boldsymbol{q}_{\pi(i+1)}\}$ and $E_P \leftarrow \emptyset$
15. **if** `local_planner`$(\boldsymbol{q}_{\pi(i)}, \boldsymbol{q}_{\pi(i+1)})$ **then**
16. $E_P \leftarrow \{(\boldsymbol{q}_{\pi(i)}, \boldsymbol{q}_{\pi(i+1)})\}$
17. **else**
18. `biRRT_planner`$(P, G, \boldsymbol{q}_{\pi(i)}, \boldsymbol{q}_{\pi(i+1)})$
19. **if** $|E_P| > 0$ **then**
20. **if** `connect`$(G, P)$ **then** $M_{conn}[\pi(i), \pi(i+1)] \leftarrow 1$
21. $O \leftarrow O +$ `length`$(P)$
22. **else**
23. $O \leftarrow O + \infty$
24. return $O$

Probabilistic sampling-based planners have been intensively used to solve single query or multiple query motion planning problems in high dimensional configuration spaces [60]. Single query motion planners are used when the planning procedure has to be performed only once, and the configuration space is explored using a single or a bi-directional tree. Rapidly-exploring Random Trees (RRTs) are an example of this planning approach [15, 16, 66]. When the planning procedure has to be repeated more than once possibly with different start and goal configurations, multiple query motion planners are used. First, a roadmap is built during a preprocessing phase to explore

**Algorithm 4.2** Function `local_planner`($\boldsymbol{q}_s, \boldsymbol{q}_g$).

Input: start configuration $\boldsymbol{q}_s$, goal configuration $\boldsymbol{q}_g$, and resolution for collision avoidance $q_{res}$
Output: collision status for edge $(\boldsymbol{q}_s, \boldsymbol{q}_g)$

```
1.  if ‖(q_s, q_g)‖_2 < q_res/2 then
2.    return TRUE
3.  else
4.    q_m ← (q_s + q_g)/2
5.    if q_m ∈ Q_free
6.      if local_planner(q_s, q_m) and local_planner(q_m, q_g) then
7.        return TRUE
8.  return FALSE
```

the connectivity of the configuration space. Second a shortest path algorithm is employed to quickly answer to several planning queries navigating through the roadmap. An example of such a planner is the Probabilistic Roadmap Method [3, 56, 61]. Recent attempts have been proposed to integrate these two methods for large scale motion planning. An example is the Sampling Based Roadmaps of Trees (SRT) [82, 83].

In the proposed optimization scheme based on the hybrid random-key GA illustrated in Section 3.1.2 a path planning step has to be executed $n$ times for each *chromosome* in the *population* since in each tour there are $n$ distinct edges. Thus, a multiple query planner seems to be the best choice to efficiently handle this large number of requests. However, the initial and final configurations $\boldsymbol{q}_i$ and $\boldsymbol{q}_j$ for each query are unknown at the beginning and can become any of the infinitely many configurations within each neighborhood. A preprocessing step where a full exploration of the collision-free configuration space $\mathcal{Q}_{free}$ is performed would be thus extremely expensive in terms of computational cost. Similarly to the SRT approach, single query planners are instead used to simultaneously answer the initial queries and incrementally build a roadmap $G = (V_G, E_G)$ over $\mathcal{Q}_{free}$, which afterwards is used to answer multiple queries. A high-level description of the proposed approach is provided in Algorithm 4.1.

For each edge in each tour, $(\boldsymbol{q}_{\pi(i)}, \boldsymbol{q}_{\pi(i+1)})$, first the algorithm checks if the number of configurations in the roadmap, $|V_G|$, is larger than a given threshold, $l_{samp}$, and if the corresponding neighborhoods $\mathcal{Q}_{\pi(i)}$ and $\mathcal{Q}_{\pi(i+1)}$ have been previously connected at least once by using an indicator matrix $M_{conn}$. During the initial calls these two conditions are not verified, and

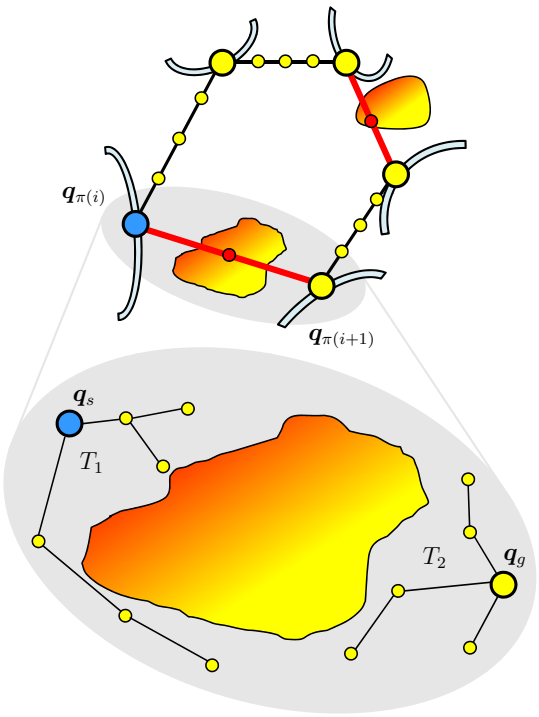



Figure 4.2: Single query planner. Edges depicted in red are not collision-free, and biRRTs are used to generate a collision free path.

thus a single query planner is invoked adding the extracted collision-free path to the roadmap $G$. Once the roadmap is completed according to the given criteria, then a multi query planner is used and the calculated edge lengths are stored in a sparse distance matrix, $D$.

#### 4.2.2.1 Single Query Planner and Roadmap Construction

First, the function `local_planner`, which is described in Algorithm 4.2, is used to verify if the point to point motion along the line segment defined in the configuration space by the two configurations $\boldsymbol{q}_{\pi(i)}$ and $\boldsymbol{q}_{\pi(i+1)}$ is collision-free. This is done by subdividing the line segment using a bisection approach up to a given resolution, $q_{res}$, and checking each intermediate configuration, $\boldsymbol{q}_m$, for collision. Collision evaluation is performed in a hierarchal fashion, first checking for intersection the bounding capsules containing the components of the system [2], and then using bounding-volume trees defined

**Algorithm 4.3** Function `biRRT_planner`$(P, G, \boldsymbol{q}_s, \boldsymbol{q}_g)$.

Input: collision-free path $P = (V_P, E_P)$, roadmap $G = (V_G, E_G)$ in $\mathcal{Q}_{free}$, start configuration $\boldsymbol{q}_s$, goal configuration $\boldsymbol{q}_g$, first tree $T_1 = (V_{T_1}, E_{T_1})$, second tree $T_2 = (V_{T_2}, E_{T_2})$, number of attempts to merge the trees $l_{merge}$, and resolution for collision avoidance $q_{res}$

Output: updated $P$ and $G$

```
1.  V_{T1} ← {q_s}; E_{T1} ← ∅
2.  V_{T2} ← {q_g}; E_{T2} ← ∅
3.  for l = 1 to l_merge do
4.    let q_rand be a randomly chosen configuration in Q
5.    q_{new,1} ← extend(T_1, q_rand)
6.      if q_{new,1} ≠ NIL then
7.        q_{new,2} ← extend(T_2, q_{new,1})
8.          if q_{new,2} ≠ NIL and ‖(q_{new,1}, q_{new,2})‖_2 < q_res then
9.            if local_planner(q_{new,1}, q_{new,2}) then
10.             P ← extract_path(T_1, q_{new,1}, T_2, q_{new,2})
11.             connect(G, T_1); connect(G, T_2)
12.             E_G ← E_G ∪ {(q_{new,1}, q_{new,2})}
13.             return TRUE
14.       swap(T_1, T_2)
15. return FALSE
```

using triangular meshes [32, 42, 78].

Second, if the line segment between the two configurations $\boldsymbol{q}_{\pi(i)}$ and $\boldsymbol{q}_{\pi(i+1)}$ is not collision free, then the function `biRRT_planner` described in Algorithm 4.3, is used to find a collision-free path between these two configurations. Figure 4.2 illustrates the two steps of the single query planner.

**BiRRT Planner** Two RRTs, $T_1$ rooted at the start configuration $\boldsymbol{q}_s = \boldsymbol{q}_{\pi(i)}$ and $T_2$ rooted at a goal configuration $\boldsymbol{q}_g = \boldsymbol{q}_{\pi(i+1)}$, are grown towards each other to build the collision-free path. Initially a random configuration, $\boldsymbol{q}_{rand}$, is generated as follows:

$$\boldsymbol{q}_{rand} = \begin{cases} \boldsymbol{q}_l + ((1+2\eta)\,U - \eta)|\boldsymbol{q}_g - \boldsymbol{q}_s| & \text{if } l < l_{loc} \text{ or } u < p_{loc} \\ \boldsymbol{q}_{\text{MIN}} + U(\boldsymbol{q}_{\text{MAX}} - \boldsymbol{q}_{\text{MIN}}) & \text{otherwise} \end{cases} \tag{4.4}$$

where $q_{l,k} = \min\{q_{s,k}, q_{g,k}\}$, $\boldsymbol{q}_{\text{MIN}}$ and $\boldsymbol{q}_{\text{MAX}}$ are the joint limits vectors, $u$ is a uniformly distributed number in $[0, 1]$, $U$ is a diagonal $(m \times m)$ matrix with diagonal entries uniformly distributed in $[0, 1]$, the parameter $\eta$ defines

**Algorithm 4.4** Function `extend`$(T, \boldsymbol{q})$.

Input: tree $T = (V_T, E_T)$, configuration $\boldsymbol{q}$, step subdivision $h_{step}$ and edge weighting function d$(\cdot)$
Output: new configuration $\boldsymbol{q}_{new}$ towards $\boldsymbol{q}$

1. let $\boldsymbol{q}_{near}$ be the nearest neighbor of $\boldsymbol{q}$ in $V_T$ w.r.t. d$(\cdot)$
2. $\boldsymbol{q}_{new} \leftarrow \boldsymbol{q}_{near} + (\boldsymbol{q} - \boldsymbol{q}_{near})/h_{step}$
3. **if** $\boldsymbol{q}_{new} \in \mathcal{Q}_{free}$ **and** `local_planner`$(\boldsymbol{q}_{near}, \boldsymbol{q}_{new})$ **then**
4. $\quad V_T \leftarrow V_T \cup \{\boldsymbol{q}_{new}\}$
5. $\quad E_T \leftarrow E_T \cup \{(\boldsymbol{q}_{near}, \boldsymbol{q}_{new})\}$
6. $\quad$ **return** $\boldsymbol{q}_{new}$
7. **return** NIL

a reduced sampling domain, $l_{loc}$ is the minimum number of configurations sampled from the reduced domain, and $p_{loc}$ is the probability to sample from the reduced domain. This sampling approach with alternate sampling domains has shown to improve the performance of the planner, confirming similar results presented by Kuffner Jr and LaValle [63] and LaValle and Kuffner Jr [66].

Afterwards, the algorithm attempts to extend the configuration closest to $\boldsymbol{q}_{rand}$ on one tree towards $\boldsymbol{q}_{rand}$ by defining a new configuration $\boldsymbol{q}_{new,1}$. This procedure is performed by the function `extend` illustrated in Algorithm 4.4, where the new configuration is defined by moving from the selected closest configuration on the tree, $\boldsymbol{q}_{near}$, towards the random configuration, $\boldsymbol{q} = \boldsymbol{q}_{rand}$, by a step defined as $(\boldsymbol{q} - \boldsymbol{q}_{near})/h_{step}$, and checking the resulting new edge for collisions. Then the algorithm attempts to extend the configuration closest to $\boldsymbol{q}_{new,1}$ on the second tree towards $\boldsymbol{q}_{new,1}$ by defining a second new configuration $\boldsymbol{q}_{new,2}$ using again the function `extend`.

If the line segment from $\boldsymbol{q}_{new,1}$ to $\boldsymbol{q}_{new,2}$ is not collision free, the two trees are swapped and the same procedure is repeated. Finally, The procedure terminates if the maximum number of attempts to merge the two trees, $l_{merge}$, is reached or if the trees can be merged. In the latter case, a path $P = (V_P, E_P)$, which is defined as a tree rooted at $\boldsymbol{q}_s$ and with only one leaf node at $\boldsymbol{q}_g$, is extracted by back-walking the two trees $T_1$ and $T_2$ from the last added leaf nodes, $\boldsymbol{q}_{new,1}$ and $\boldsymbol{q}_{new,2}$. Moreover, unnecessary internal nodes are removed from the path $P$ and the position of each internal configuration is pulled towards the start and goal configurations using the function `extract_path` illustrated in Algorithm 4.5.

Finally, if the two trees have been merged, they are added to the roadmap

**Algorithm 4.5** Function `extract_path`$(T_1, \boldsymbol{q}_1, T_2, \boldsymbol{q}_2)$.

Input: first tree $T_1 = (V_{T_1}, E_{T_1})$, second tree $T_2 = (V_{T_2}, E_{T_2})$, last added leaf nodes $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$, step subdivision $h_{step}$, and smoothing attempts $l_{smooth}$
Output: collision-free path defined as tree $P = (V_P, E_P)$

1. $V_P \leftarrow \{\boldsymbol{q}_1, \boldsymbol{q}_2\}$; $E_P \leftarrow \{(\boldsymbol{q}_1, \boldsymbol{q}_2)\}$
2. **for** t = 1 to 2 **do**
3. $T \leftarrow T_t$
4. $\boldsymbol{q} \leftarrow \boldsymbol{q}_t$; $\boldsymbol{q}_p \leftarrow$ `parent`$(T_t, \boldsymbol{q}_t)$
5. **while** $\boldsymbol{q}_p \neq$ `root`$(T_t)$ **do**
6. $V_P \leftarrow V_P \cup \{\boldsymbol{q}_p\}$
7. **if** $t = 1$ **then** $E_P \leftarrow \{(\boldsymbol{q}_p, \boldsymbol{q})\}$ **else** $E_P \leftarrow \{(\boldsymbol{q}, \boldsymbol{q}_p)\}$
8. $\boldsymbol{q} \leftarrow \boldsymbol{q}_p$; $\boldsymbol{q}_p \leftarrow$ `parent`$(T_t, \boldsymbol{q})$
9. **if** $|V_P| > 2$
10. $\boldsymbol{q} \leftarrow \boldsymbol{q}_2$
11. **repeat**
12. $\boldsymbol{q}_p \leftarrow$ `parent`$(P, \boldsymbol{q})$
13. **repeat**
14. $\boldsymbol{q}_{pp} \leftarrow$ `parent`$(P, \boldsymbol{q}_p)$
15. **if** `local_planner`$(\boldsymbol{q}_{pp}, \boldsymbol{q})$ **then**
16. $V_P \leftarrow V_P \setminus \{\boldsymbol{q}_p\}$
17. $E_P \leftarrow (E_P \setminus \{(\boldsymbol{q}_{pp}, \boldsymbol{q}_p), (\boldsymbol{q}_p, \boldsymbol{q})\}) \cup \{(\boldsymbol{q}_{pp}, \boldsymbol{q})\}$
18. $\boldsymbol{q}_p \leftarrow \boldsymbol{q}_{pp}$
19. **else break**
20. **until** $\boldsymbol{q}_{pp} =$ `root`$(P)$
21. $\boldsymbol{q} \leftarrow \boldsymbol{q}_p$
22. **until** $\boldsymbol{q}_{pp} =$ `root`$(P)$
23. $\boldsymbol{q} \leftarrow \boldsymbol{q}_2$
24. **repeat**
25. $\boldsymbol{q}_p \leftarrow$ `parent`$(P, \boldsymbol{q})$; $\boldsymbol{q}_{pp} \leftarrow$ `parent`$(P, \boldsymbol{q}_p)$
26. **for** $l = 1$ to $l_{smooth}$ **do**
27. $\boldsymbol{q}_{new} \leftarrow \boldsymbol{q}_p + (\boldsymbol{q}_1 - \boldsymbol{q}_p)/h_{step} + (\boldsymbol{q}_2 - \boldsymbol{q}_p)/h_{step}$
28. **if** $\boldsymbol{q}_{new} \in \mathcal{Q}_{free}$ **and** `local_planner`$(\boldsymbol{q}_{pp}, \boldsymbol{q}_{new})$ **and** `local_planner`$(\boldsymbol{q}_{new}, \boldsymbol{q})$ **then**
29. $\boldsymbol{q}_p \leftarrow \boldsymbol{q}_{new}$
30. **else** $l = l_{smooth}$
31. $\boldsymbol{q} \leftarrow \boldsymbol{q}_p$
32. **until** $\boldsymbol{q}_{pp} =$ `root`$(P)$
33. **return** $P$

$G$ by using the function `connect` illustrated in Algorithm 4.6. The root

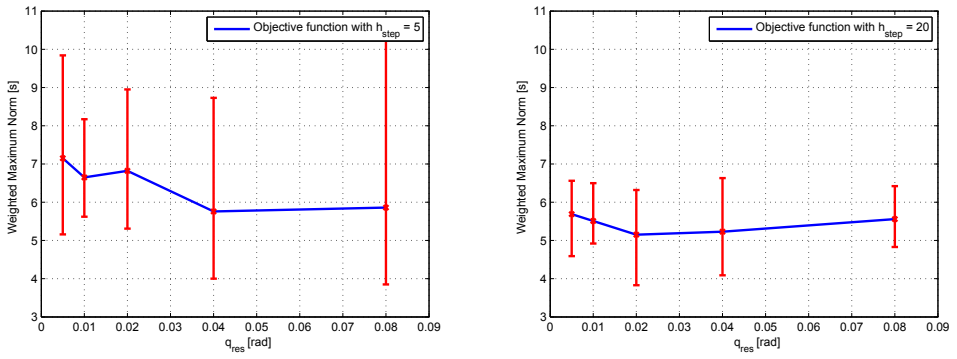
**Algorithm 4.6** Function `connect`$(G, T)$.

Input: roadmap $G = (V_G, E_G)$, tree $T = (V_T, E_T)$,
edge weighting function $\mathrm{d}(\cdot)$,
and connection graph $G_C = (V_C, E_C)$
Output: updated G

1. $V_C \leftarrow \emptyset$
2. $E_C \leftarrow \emptyset$
3. let $V_{conn}$ be a set containing the root node of $T$,
the last added leaf node of $T$, and some internal nodes of $T$
4. **for** $\boldsymbol{q}_{conn}$ in $V_{conn}$ **do**
5. let $\boldsymbol{q}_{near}$ be the nearest neighbor of $\boldsymbol{q}_{conn}$ in $V_G$ w.r.t. $\mathrm{d}(\cdot)$
6. **if** `local_planner`$(\boldsymbol{q}_{conn}, \boldsymbol{q}_{near})$ **then**
7. $V_C \leftarrow V_C \cup \{\boldsymbol{q}_{conn}, \boldsymbol{q}_{near}\}$
8. $E_C \leftarrow E_C \cup \{(\boldsymbol{q}_{conn}, \boldsymbol{q}_{near})\}$
9. **else**
10. $P = (V_P, E_P)$: $V_P \leftarrow \{\boldsymbol{q}_{conn}, \boldsymbol{q}_{near}\}$ and $E_P \leftarrow \emptyset$
11. **if** `biRRT_planner`$(P, G, \boldsymbol{q}_{conn}, \boldsymbol{q}_{near})$ **then**
12. $V_C \leftarrow V_C \cup V_P$
13. $E_C \leftarrow E_C \cup E_P$
14. **if** $|E_C| > 0$ **then**
15. $V_G \leftarrow V_G \cup V_T \cup V_C$
16. $E_G \leftarrow E_G \cup E_T \cup E_C$
17. **return** TRUE
18. **return** FALSE

nodes, the last added leaf nodes, and some internal nodes are connected to the corresponding closest node in the roadmap $G$ by using again either the function `local_planner` or, if it fails, the function `biRRT_planner`.

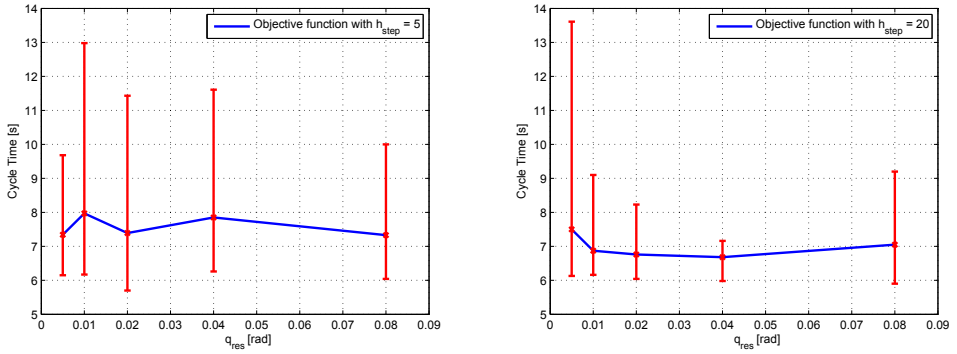
The choice of the resolution for collision avoidance, $q_{res}$, of the step subdivision, $h_{step}$, and of the edge weighting function, $\mathrm{d}(\cdot)$, are critical for the performance of the single query planner [19, 60]. To analyze this relationship a fixed tour of 9 fixed configurations is considered, and a collision-free path is generated for each edge in the tour using the function `biRRT_planner`. This operation is repeated 10 times for each parameter set using either the Weighted Maximum norm or the Quadratic norm. Figures 4.3.a, 4.3.b, 4.4.a, and 4.4.b report the minimum, average, and maximum value of the norm-base objective function evaluated before performing the path smoothing step included in Algorithm 4.5. Figures 4.3.c, 4.3.d, 4.4.c, and 4.4.d report the minimum, average, and maximum value of the corresponding cycle time calculated after the smoothing step according to Equations (4.2) and (4.3).




(a) $h_{step} = 5$




(b) $h_{step} = 20$



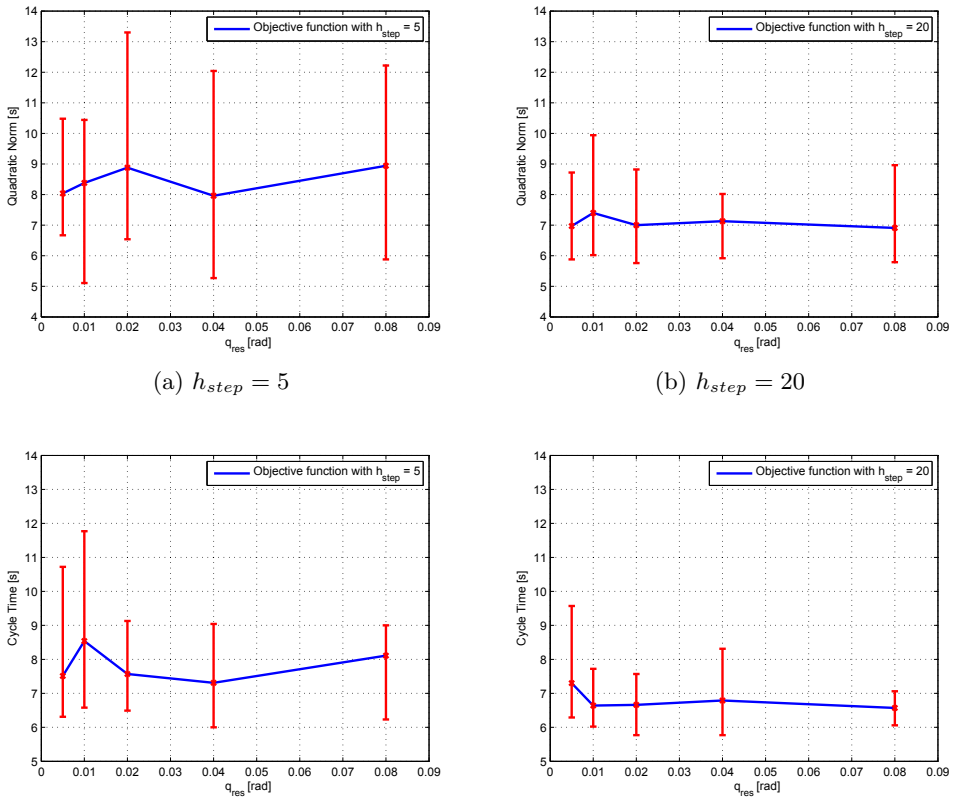

(c) $h_{step} = 5$




(d) $h_{step} = 20$

Figure 4.3: Objective function evaluation for collision-free tour using the Weighted Maximum norm and different parameter sets.

It can be easily observed that the larger is the step subdivision, $h_{step}$, the lower is the average objective function value and the more consistent are the results from the repeated executions with the same parameter set. Moreover, although the resolution $q_{res}$ does not have a clear influence on the results, the computational cost largely increases with the resolution. In the case $h_{step} = 5$ and Quadratic norm, the average CPU time ranges from $10.2s$ to $0.8s$ with $q_{res}$ ranging from 0.005 rad to 0.08 rad. Moreover, results obtained with the tested lower resolution usually have a minimum value in the same range as the one obtained with higher resolution, indicating a similar behavior with respect to collision avoidance. Observing that in the proposed planning scheme the single query planner is mainly used to build the roadmap and




(a) $h_{step} = 5$ (b) $h_{step} = 20$

(c) $h_{step} = 5$ (d) $h_{step} = 20$

Figure 4.4: Objective function evaluation for collision-free tour using the Quadratic norm and different parameter sets.

not as final answer to planning requests, the computational performance is more critical than the objective function value for each planning query. Therefore, parameters are fixed to $h_{step} = 5$ and $q_{res} = 0.08$ hereafter. The detailed results for all the peformed tests are reported in Appendix A.9.

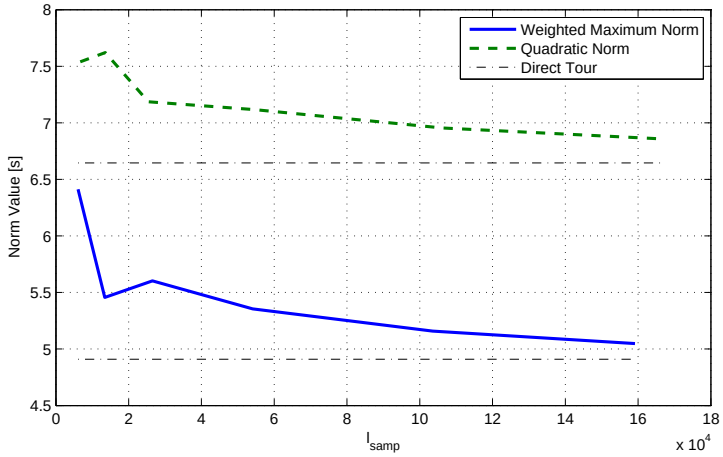
#### 4.2.2.2 Multiple Query Planner

If the neighborhoods corresponding to the two configurations $\boldsymbol{q}_{\pi(i)}$ and $\boldsymbol{q}_{\pi(i+1)}$ have been connected at least once by the single query planner, and if the number of configurations, $|V_G|$, added to the roadmap, $G$, is larger than the threshold $l_{samp}$, then $G$ is used to find a collision-free path between

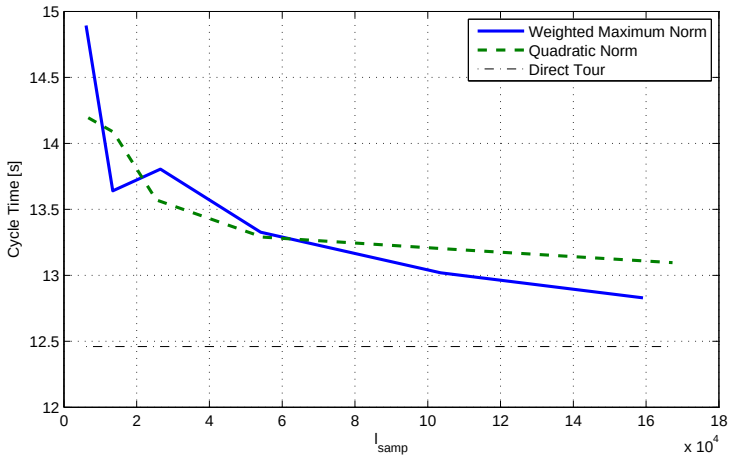
the two configurations. First, if the two configurations are not in $G$ the function `connect` is used to find a collision-free path between them and the corresponding nearest configurations in $G$. Then, using the function `index`, which returns the index of a configuration in $G$ or NIL if it is not in $G$, the corresponding entry in the sparse distance matrix $D$ is checked. If the entry has not been assigned yet, the function `shortest_path` is used to calculate the shortest path in $G$ between the two configurations and to update $D$.

Since the single query planner, upon which the roadmap is build, is a sampling-based thus heuristic procedure, the heuristic search A* is used to retrieve the shortest path [50]. This algorithm is based on the exact search proposed by Dijkstra [26] but it considerably improves the computational performance using a heuristic function, which in the proposed implementation is either the Weighted Maximum norm or the Quadratic norm between the current and the goal configurations. An alternative procedure is to use the all-pairs search for sparse graph proposed by Johnson [58] to first evaluate all the entries of $D$ as soon as the threshold $l_{samp}$ is reached. However, we observed that the number of entries in $D$ that are actually queried is a small fraction of the total number. Moreover, we also observed that adding new configurations and edges to the roadmap even after the threshold $l_{samp}$ is reached improves the performance of the algorithm with respect to the objective function value. In particular, we observed that resetting the connection indicator matrix $M_{conn}$ when $g_I >= g_{I_{\text{MAX}}}$ at step 6. in the main Algorithm 3.1, which causes new configurations and edges to be added to the roadmap, allows the hybrid random-key GA to converge to a shorter near *optimal tour*. Based on these observations, the incremental approach for calculating the entries of $D$ was finally chosen.

The parameter $l_{samp}$, i.e. the minimum number of configurations in the roadmap $G$, strongly influences the performance of the proposed path planner [60]. To illustrate this behavior a roadmap is built repeating Algorithm 4.1 on randomly generated tours of random configurations until the threshold $l_{samp}$ is reached. Afterwards, a fixed tour of fixed configurations is analyzed using the multiple query planner and the value of the objective function is stored. The same procedure is then repeated with a different value of the parameter $l_{samp}$. Figure 4.5.a illustrates how the norm-based objective function value for the fixed tour evolves as function of $l_{samp}$ using the Weighted Maximum norm or the Quadratic norm as edge weighting function. Figure 4.5.b illustrates the results of the same tests where the actual cycle time is calculated as post processing step once the algorithm terminates. The dashed line labeled "Direct Tour" correspond to the limit objective function value for the fixed tour where no collision avoidance is




(a) Norm-base objective function convergence for a fixed tour.




(b) Cycle time convergence for a fixed tour.

Figure 4.5: Multiple query planner convergence as function of the parameter $l_{samp}$. Direct tour is the lower bound for the optimal value of the objective function.

considered. On the one hand, we can clearly observe that the larger is $l_{samp}$, the shorter is the retrieved collision-free tour. On the other hand, the larger is $l_{samp}$, the more the transition between single query and multiple query planner is postponed resulting in an excessive computational cost for the

hybrid random-key GA. Based on these numerical tests the parameter is fixed to $l_{samp} = 100,000$ hereafter, which seems to guarantee a sufficient quality in the retrieved solution.

Finally, we can observe from Figures 4.3, 4.4, and 4.5 that the employed norms show similar convergence trends as the actual cycle time. Therefore, in the actual implementation the cycle time is calculated according to Equations (4.2) and (4.3) only as a final step when the distance matrix, $D$, is updated or the objective function value for the extracted path is returned by the function `length` at steps 10. or 21. of Algorithm 4.1, respectively.

## 4.3 Neighborhood definition

For each image $i$, i.e. for each relative placement between the camera and the component, the position and orientation of the manipulator end-effector can be represented in the turntable coordinate system as a homogenous transformation $(4 \times 4)$ matrix:

$$ {}^{t}R_{e}^{(i)} = \begin{bmatrix} {}^{t}\boldsymbol{\lambda}_{e}^{(i)} & {}^{t}\boldsymbol{\mu}_{e}^{(i)} & {}^{t}\boldsymbol{\nu}_{e}^{(i)} & {}^{t}\boldsymbol{p}_{e}^{(i)} \\ 0 & 0 & 0 & 1 \end{bmatrix} , \tag{4.5} $$

where ${}^{t}\boldsymbol{p}_{e}^{(i)}$ is the end-effector position, and ${}^{t}\boldsymbol{\lambda}_{e}^{(i)}$, ${}^{t}\boldsymbol{\mu}_{e}^{(i)}$, and ${}^{t}\boldsymbol{\nu}_{e}^{(i)}$ are the unit vectors in $\mathbb{R}^3$ that define the end-effector coordinate system with respect to the turntable coordinate system for image $i$.

If ${}^{m}\mathbf{p}_{t}$ is the origin of the turntable coordinate system with respect to the manipulator coordinate system and $q_{i,7}$ is the turntable rotation angle about its $z$-axis, then constraints (2.5) for this robotic system can be derived as:

$$ \begin{bmatrix} \mathrm{rot}(\boldsymbol{e}_z, q_{i,7}) & {}^{m}\mathbf{p}_{t} \\ 0 \quad 0 \quad 0 & 1 \end{bmatrix} \cdot {}^{t}R_{e}^{(i)} = {}^{m}R_{e}^{(i)} = \mathrm{FK}([q_{i,1}, \ldots, q_{i,6}]^T) \quad \forall\, i \in \mathrm{V} , \tag{4.6} $$

where the $(3 \times 3)$ rotation matrix $\mathrm{rot}(\boldsymbol{e}_z, q)$ and the forward kinematic function of the manipulator $\mathrm{FK}(\cdot)$ are defined in Appendix A.10.

Using the manipulator inverse kinematic function $\mathrm{IK}(\cdot)$, constraints (4.6) can be written as:

$$ \mathrm{IK}\left( \begin{bmatrix} \mathrm{rot}(\boldsymbol{e}_z, q_{i,7}) & {}^{m}\mathbf{p}_{t} \\ 0 \quad 0 \quad 0 & 1 \end{bmatrix} \cdot {}^{t}R_{e}^{(i)} \right) = [q_{i,1}, \ldots, q_{i,6}]^T \quad \forall\, i \in \mathrm{V} . \tag{4.7} $$

As illustrated in Appendix A.10, the manipulator inverse kinematic $\mathrm{IK}(\cdot)$ can have up to 8 different solutions, called *figures*. Thus, constraints (4.7)

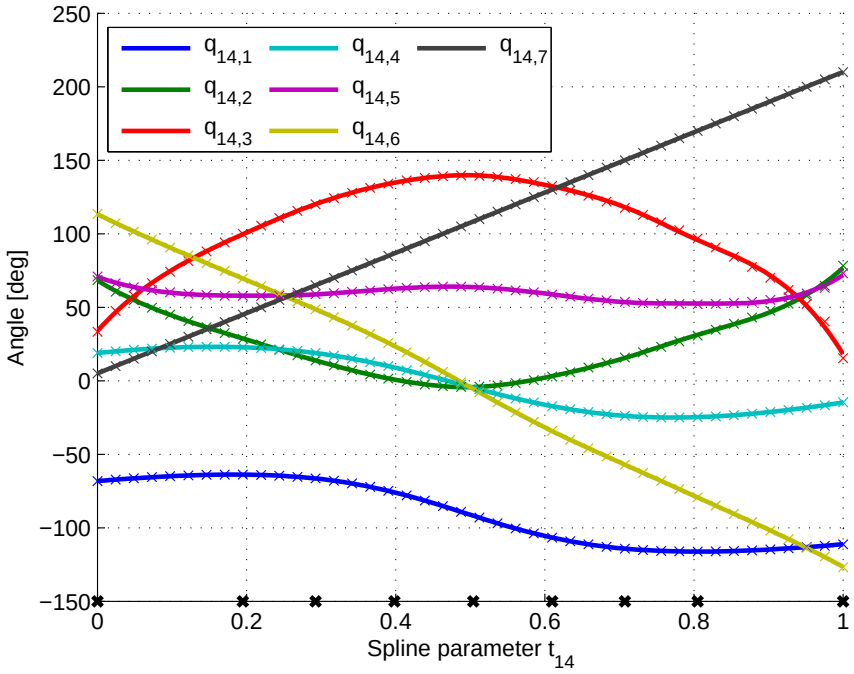



Figure 4.6: Piecewise cubic least-square approximation for neighborhood $i = 14$. The sampled configurations are indicated with x-marks in the same color of the corresponding curve. The hyperspline consists of 8 polynomial pieces with brake points indicated by black x-marks.

represent a set of possibly non-connected curves in the configuration space where $q_{i,7}$ can be viewed as the curve parameter.

In this work, piecewise cubic hypersplines are used to simplify the definition of constraints (4.7), i.e. the neighborhoods, while considering only one *figure* at a time. Thus, for each relative placement ${}^{t}R_{e}^{(i)}$ a set of configurations $\boldsymbol{q}_i$ is obtained by evaluating Equation (4.7) for all the feasible value of $q_{i,7}$ sampled at an interval of 5 deg. The set of consecutive configurations that lie within physical joints limits and are collision-free is then approximated using a piecewise cubic hyperspline, as illustrated in Figure 4.6. The number of pieces, $n_p$, used in the present work is 8. Similarly to Equation (2.10), the spline parameter $t_i \in [0, 1]$ is introduced and constraints (2.5)

Table 4.1: Kinematic parameter for the 7DOF vision inspection system.

| Joint | 1 | 2 | 3 | 4 | 5 | 6 | 7 LS | 7 HS |
|---|---|---|---|---|---|---|---|---|
| $\boldsymbol{q}_{\text{MIN}}$ [rad] | -2.97 | -1.75 | -2.08 | -3.32 | -2/3$\pi$ | -2$\pi$ | -2$\pi$ | -2$\pi$ |
| $\boldsymbol{q}_{\text{MAX}}$ [rad] | 2.97 | 2.36 | 2.95 | 3.32 | -2/3$\pi$ | 2$\pi$ | 2$\pi$ | 2$\pi$ |
| $\omega_k$ [rad/s] | 3.51 | 3.51 | 3.51 | 5.17 | 5.17 | 8.40 | 2.88 | 6.98 |
| $\alpha_k$ [rad/s$^2$] | 21.74 | 21.74 | 21.74 | 33.69 | 33.69 | 52.12 | 14.4 | 34.91 |

are redefined as:

$$\sum_{p=1}^{n_p} \mathbf{1}_p(t_i)\big(\mathbf{s}_{0,i,p} + \mathbf{s}_{1,i,p}\,(t_i - t_{i,p}) + \mathbf{s}_{2,i,p}\,(t_i - t_{i,p})^2 + \mathbf{s}_{3,i,p}\,(t_i - t_{i,p})^3\big) - \boldsymbol{q}_i = 0 \qquad \forall\, i \in \mathrm{V}\ . \tag{4.8}$$

where $t_{i,p}, p = 1, \ldots, n_p + 1$ are the spline break points and $\mathbf{1}_p(t_i) = 1$ if $t_i \in [t_{i,p}, t_{i,p+1})$, 0 otherwise.

## 4.4 Computational results

The proposed procedure is tested using a set of 31 relative placements ${}^tR_e^{(i)}$ and one depot placement provided by Denso Wave, Inc., which owns the vision inspection system examined in this work. The actual data for $\boldsymbol{q}_{\text{MIN}}$, $\boldsymbol{q}_{\text{MAX}}$, $\omega_k$, and $\alpha_k$ are obtained for the DENSO VS 6577E-B manipulator the turntable from the specification sheets [88], [33], and [1]. The values are reported in Table 4.1. For the turntable, i.e. joint 7, two sets of parameters are provided: low and high speed labeled "LS" and "HS", respectively. The controller delay is set to $\Delta t_0 = 0.1s$.

Given the extensive use of the nearest neighbor operation in the proposed procedure, the roadmap $G$ is implemented using an adjacency list based on the kd-tree [13, 14]. Using this data structure the expected cost of such operation is reduced from $O(|V_G|)$ to $O(\log|V_G|)$ [36, 37]. Moreover, the planning algorithm is executed as a separated thread from the hybrid random-key GA using an Inter-Process Communication (IPC) between the two processes based on Remote Procedure Calls (RPC) [75]. This allows the two algorithms to be executed on different cores while maintaining in memory a large roadmap, which can grow above $10^6$ configurations, during different runs of the hybrid random-key GA and thus avoiding to rebuild it at each execution. Finally, the parameter used in Algorithms 4.1 to 4.6 are:

Table 4.2: Simulation and experimental results for the 7DOF vision inspection system.

| Norm | | Original | TSP | TSPN LS | TSPN HS |
|---|---|---|---|---|---|
| Euclidean | norm value [rad] | 30.51 | 24.85 | 19.12 | 20.24 |
| | impr. | - | 18.6% | 37.3% | 33.7% |
| | cycle time [s] | 13.86 | 11.79 | 10.16 | 9.25 |
| | impr. | - | 14.9% | 26.7% | 33.3% |
| | ArmPl. [s] | 18.96 | 15.89 | 13.11 | 12.45 |
| | impr. | - | 16.2% | 30.9% | 34.3% |
| Weighted Maximum | norm value [s] | 5.640 | 4.440 | 3.135 | 2.515 |
| | impr. | - | 21.3% | 44.4% | 55.4% |
| | sim. time [s] | 13.86 | 11.80 | 10.14 | 9.40 |
| | impr. | - | 14.9% | 26.8% | 32.2% |
| | ArmPl. [s] | 18.96 | 15.74 | 14.39 | 12.73 |
| | impr. | - | 17.0% | 24.1% | 32.9% |
| | sys. time [s] | 10.62 | 8.92 | 8.35 | 7.88 |
| | impr. | - | 16.0% | 21.4% | 25.7% |
| Quadratic | norm value [s] | 7.390 | 5.911 | 4.728 | 3.561 |
| | impr. | - | 20.0% | 36.0% | 51.8% |
| | cycle time [s] | 13.86 | 11.74 | 9.98 | 9.13 |
| | impr. | - | 15.3% | 28.0% | 34.1% |
| | ArmPl. [s] | 18.96 | 16.01 | 13.72 | 12.27 |
| | impr. | - | 15.6% | 27.6% | 35.3% |
| | sys. time [s] | 10.62 | 9.10 | 7.87 | 7.42 |
| | impr. | - | 14.3% | 25.9% | 30.1% |

$\eta = 0.3$, $p_{loc} = 0.1$, $l_{loc} = 10$, $h_{step} = 5$, $q_{res} = 0.08$, $l_{merge} = 1,000$, and $l_{samp} = 100,000$.

Table 4.2 illustrates the results of the performed experiments. The column with label "Original" reports the results obtained with the original sequence of 32 configurations provided by Denso Wave and it is used afterwards as reference value. The results reported in the columns with label "TSPN LS" and "TSPN HS" are obtained using the proposed optimization procedure for low and high turntable speed, respectively. Finally, the results reported in the column with label "TSP" are obtained by finding an optimal sequence of the original configurations. First, a distance matrix is calculated by using the planning procedure illustrated in the previous Section on all the possible edges of the full graph defined by the 32 original configurations. Then, the resulting TSP is solved to optimality using the exact TSP solver Concorde [7].

For each one of the four testing framework, three different sets of results are reported:

- norm-based objective function value and simulated cycle time of the near *optimal tour* found by the hybrid random-key GA, which are labeled "norm value" and "cycle time", respectively;
- simulated cycle time for the near *optimal tour* returned by the simulator DENSO WinCAPS III - ArmPlayer Plus [87], which is labeled "ArmPl.";
- experimental cycle time for the near *optimal tour* required by the actual vision inspection system, which is labeled "sys. time".

The tests are performed using the Weighted Maximum norm and the Quadratic norm as underlying edge weighting function for the hybrid random-key GA and for the path planner. The values obtained using the Euclidean norm are also reported, but no test on the actual system are performed in this case.

By exploiting the redundancy of the system while searching for a near *optimal tour*, if the Quadratic norm is used as underlying edge weighting function the actual cycle time can be reduced by 26% for low turntable speed or by 30% for high turntable speed. In this scenario the optimization achieved searching only for an optimal sequence of the original configurations is about 14%. Figure 4.7 shows the original tour, Figure 4.8 the tour of the original configurations optimized by the TSP solver, and Figures 4.9 and 4.10 the near *optimal tours* obtained with the proposed procedure for the two turntable speeds.

The actual improvement measured on the real system is smaller than the one observed for the norm-based objective function, which is 52% in case of Quadratic norm and high turntable speed. Moreover, the actual improvement is also smaller than the ones obtained using the cycle time as defined in Equations (4.2) and (4.3) or using the simulator ArmPlayer Plus, 34% or 35%, respectively. However, we can observe that the trends of simulation and experimental results are very similar, i.e. high turntable speed always leads to smaller cycle time than the one obtained using the low turntable speed or the TSP optimization, independently of the method used to evaluate the cycle time. Moreover, Figure 4.11 illustrates how the calculated cycle time decreases as the turntable speed increases, suggesting that the performance of the system can be adjusted just by changing the dynamic characteristic of the turntable. Results obtained using the Weighted Maximum norm lead to similar observations, confirming that both norms well

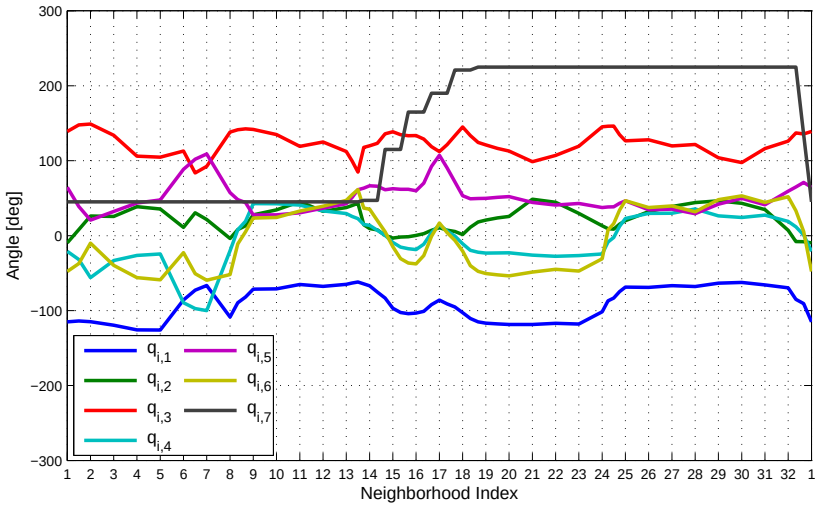



Figure 4.7: Original tour of 32 configurations provided by Denso Wave. The black line corresponds to the turntable joint angle.

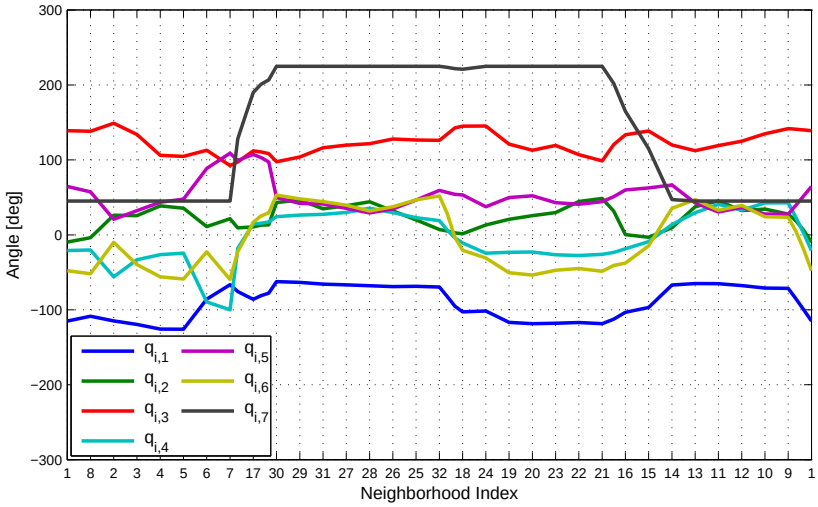



Figure 4.8: Optimal tour of the original configurations obtained using the Quadratic norm.

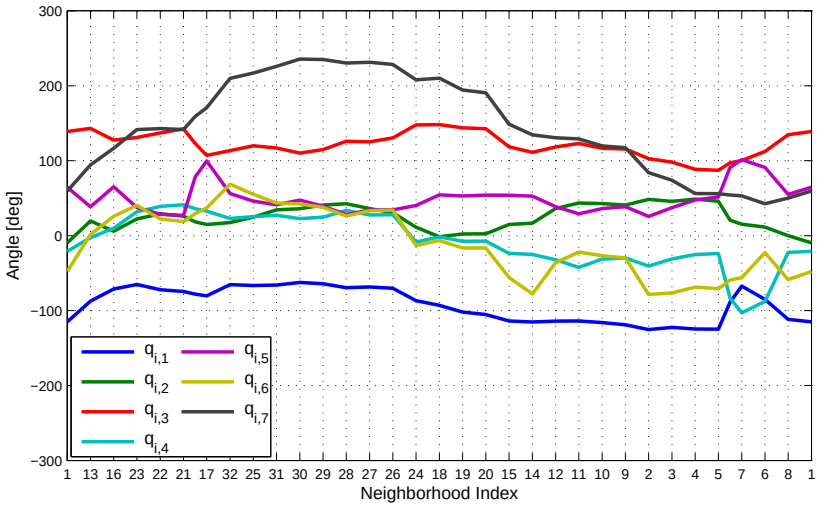



Figure 4.9: Near *optimal tour* of 32 neighborhoods obtained using the Quadratic norm and low turntable speed.

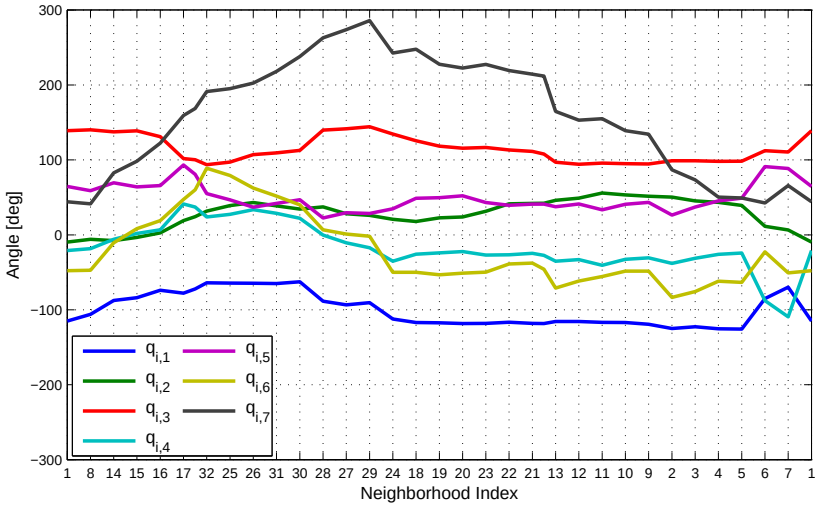



Figure 4.10: Near *optimal tour* of 32 neighborhoods obtained using the Quadratic norm and high turntable speed.

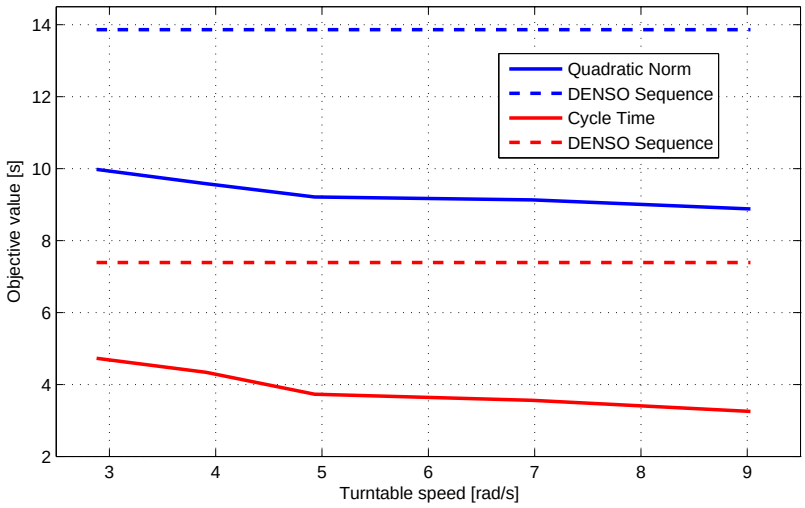



Figure 4.11: Cycle time improvement as function of the turntable speed obtained using the Quadratic norm. Dashed lines represent the corresponding objective function values for the original tour provided by Denso Wave.

represent the behavior of the actual system for the purpose of cycle time minimization.

Finally, we observe that the simulated cycle time is usually larger than the one measured on the actual system. The values calculated with the purely kinematic model defined in Equations (4.2) and (4.3) and with the simulator ArmPlayer Plus are larger than the actual ones on average by 28.1% and 75.8%, respectively, if the Weighted Maximum norm is used, and by 27.6% and 73.3%, respectively, if the Quadratic norm is used. The actual dynamic performance achieved by the manipulator controller with a light camera mounted on its end-effector is probably higher than the one reported in the specification sheet or used by ArmPlayer Plus, which causes the observed reduction in the measured cycle time. At the same time, similar delays in the controller have a larger impact on the shorter cycle time of the actual system, which can originate the observed different improvement levels.

## 4.5 Conclusion

In this chapter a practical applications of the proposed optimization approach is illustrated. The cycle time currently required by a 7 DOF robotic vision inspection system to complete a 32-goal cycle is improved by 30%. The neighborhoods are approximated using piecewise cubic splines in a seven-dimensional configuration space, and the employed edge weighting functions are the Weighted Maximum or the Quadratic norm. Moreover, a kinematic model is used to evaluate the actual cycle time, and an ad-hoc probabilistic path planning technique, which merges single and multiple query sampling-based planners, is embedded in the GA to guarantee each edge in the calculated near *optimal tour* to be collision-free.

# Chapter 5

# Unmanned Aerial Survey System

In this chapter, we apply the proposed hybrid random-key GA to an unmanned aerial survey system. We illustrate how the neighborhoods are defined and how the edge weighting function is customized to account for specific system characteristics, obstacle avoidance, and energy consumption. Finally, simulations are performed and results are discussed.

## 5.1 Problem Formulation

The robotic system considered in this chapter is a quadrotor Unmanned Aerial Vehicle (UAV). This UAV can be used to autonomously carry out rescue missions of large urban areas damaged by natural disasters, or to visually inspect large facilities or infrastructures. An optimal path is needed to maximize the number of critical location covered by the quadrotor during each flight mission, i.e., to accelerate the search operations of rescue teams or to minimize the cost of each inspection procedure.

A similar problem is considered in the literature, where an UAV is constrained to fly within a certain distance from the center of each site that needs to be monitored [62]. The neighborhood are modeled here as circles in $\mathbb{R}^2$, and necessary conditions for optimality are discussed. Finally, two heuristics, based on a "rubberband" approach or on a Generalized TSP (GTSP) model are proposed, but no extensive results are illustrated.

In this work no constraints are formulated for the trajectory that has to be tracked by the UAV given the high manoeuvrability of commercial quadrotors [57, 71, 73]. Theretofore, the problem of finding the optimum

flight path for the quadrotor to visit all the desired locations where the images have to be acquired from can be formulated as a traditional TSP. However, since each image can be acquired by the quadrotor from infinitely many feasible configurations, the problem can be re-formulated as a TSPN. In this case the redundancy of the system is originated by the fact that the relative position between the camera and the object is not fixed, as for the 7DOF industrial vision inspection system illustrated in chapter 4, but it can vary within certain bounds. These can be determined based on the dimensions of the feature to inspect, on the UAV and camera characteristics, and on the image quality specifications, such as resolution or distortion.

Initially, we study a simplified version of this problem using an analytically defined objective function based on the Euclidean or Quadratic norm. Moreover, polyhedra in $\mathbb{R}^3$ are employed to define the neighborhoods, which provide enough flexibility to model the actual constraints for practical purposes. The hybrid random-key GA is then used to find a near *optimal tour* for the resulting TSPN.

Afterwards, bidirectional RRTs are used to account for collision avoidance, and a full dynamic simulation is performed to better estimate within the GA the actual cost in terms of traveled distance or energy consumption. Finally, simulations are performed using two different urban scenarios.

## 5.2 Objective function evaluation

An edge weighting function needs to be defined to evaluate the cost for the quadrotor to fly from one neighborhood to the next. If the objective of the optimization is to minimize the total traveled distance, the Euclidean or the Quadratic norm can be used. In particular, the Quadratic norm can be used when total distance and total traveled yaw angle are minimized simultaneously to balance between the two quantities.

Although a norm allows to evaluate the objective function (2.18) very efficiently, it does not guarantee each edge in the tour to be collision-free. To account for the presence of obstacles along the path of the quadrotor that connects two configurations in the tour, and thus to have a more realistic evaluation of the path length, we use the following procedure based on Oriented Bounding Boxes (OBB). If the straight line connecting the considered two configurations intersects only one OBB, additional collision avoidance configurations are deterministically added to generate a collision free path. If the collision scenario is more complex, the single query planner presented in Section 4.2.2.1 is used to efficiently generate a collision-free path.

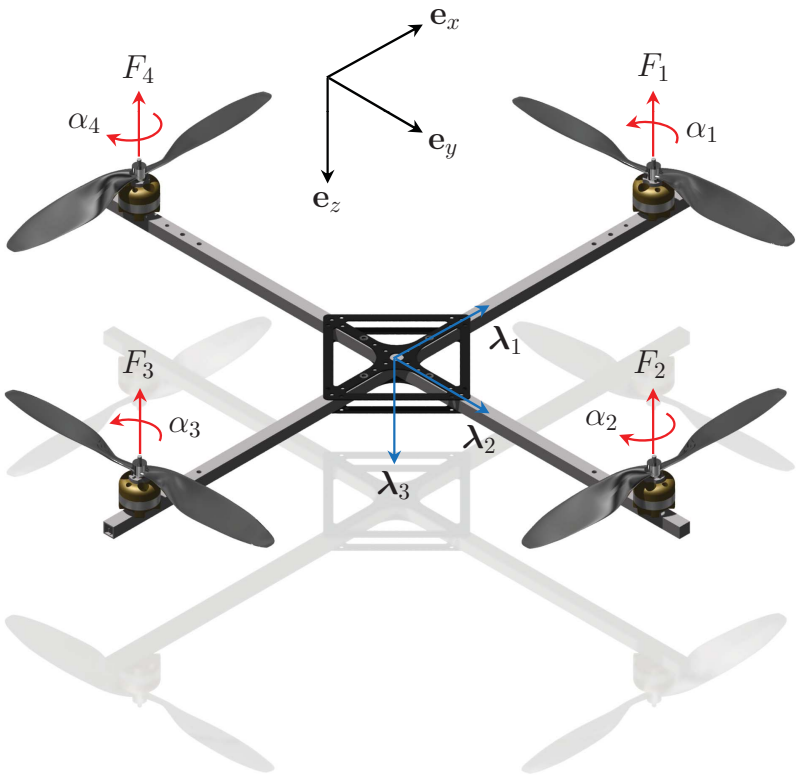



Figure 5.1: Quadrotor schematic.

Finally, if the objective of the optimization is to minimize the total energy used by the quadrotor during the mission, a full dynamic characterization is required. The rest of this Section describes the kinematic and dynamic models of the quadrotor, the aerodynamic forces derivation, and the control scheme used to simulate the flight of the UAV.

### 5.2.1 Kinematic Model

Two coordinate systems are defined as illustrated in Fig. 5.1: the inertial coordinate system, $\{\mathbf{e}_x, \mathbf{e}_y, \mathbf{e}_z\}$, and the body coordinate system, $\{\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2, \boldsymbol{\lambda}_3\}$, centered at the center of mass, $\boldsymbol{r}$, of the quadrotor. The rotation from the body coordinate system to the inertial coordinate system is defined using ZXY Euler angles. The first rotation is about the $z$-axis of the inertial coordinate system by the yaw angle, $\psi$. The second rotation is about an intermediate $x$-axis by the roll angle, $\varphi$. The third rotation is about the

$y$-axis of the body coordinate system by the pitch angle, $\vartheta$. The resulting rotation matrix is:

$$\boldsymbol{R} = \begin{bmatrix} \mathrm{c}\psi\,\mathrm{c}\vartheta - \mathrm{s}\psi\,\mathrm{s}\varphi\,\mathrm{s}\vartheta & -\mathrm{s}\psi\,\mathrm{c}\varphi & \mathrm{c}\psi\,\mathrm{s}\vartheta + \mathrm{s}\psi\,\mathrm{s}\varphi\,\mathrm{c}\vartheta \\ \mathrm{s}\psi\,\mathrm{c}\vartheta + \mathrm{c}\psi\,\mathrm{s}\varphi\,\mathrm{s}\vartheta & \mathrm{c}\psi\,\mathrm{c}\varphi & \mathrm{s}\psi\,\mathrm{s}\vartheta - \mathrm{c}\psi\,\mathrm{s}\varphi\,\mathrm{c}\vartheta \\ -\mathrm{c}\varphi\,\mathrm{s}\vartheta & \mathrm{s}\varphi & \mathrm{c}\varphi\,\mathrm{c}\vartheta \end{bmatrix} \tag{5.1}$$

where $\mathrm{s}(\cdot) = \sin(\cdot)$ and $\mathrm{c}(\cdot) = \cos(\cdot)$.

The velocity of the quadrotor center of mass can be transformed from the body to the inertial coordinate system using the relation $\dot{\boldsymbol{r}} = \boldsymbol{v} = [v_x, v_y, v_z]^T = \boldsymbol{R}\,[v_1, v_2, v_3]^T$. Similarly, the quadrotor angular velocity can be transformed using the relation $\boldsymbol{\omega} = [\omega_x, \omega_y, \omega_z]^T = \boldsymbol{R}\,[\omega_1, \omega_2, \omega_3]^T$. Moreover, the relationship between the angular velocity in the inertial coordinate system and the time derivatives of the Euler angles is given by $\boldsymbol{\omega} = \boldsymbol{M}\,[\dot{\psi}, \dot{\varphi}, \dot{\vartheta}]^T$, where:

$$\boldsymbol{M} = \begin{bmatrix} 0 & \mathrm{c}\psi & -\mathrm{s}\psi\,\mathrm{c}\varphi \\ 0 & \mathrm{s}\psi & \mathrm{c}\psi\,\mathrm{c}\varphi \\ 1 & 0 & \mathrm{s}\varphi \end{bmatrix} . \tag{5.2}$$

Finally, the time derivative of the rotation matrix is given by $\dot{\boldsymbol{R}} = \boldsymbol{\Omega}\,\boldsymbol{R} = \boldsymbol{R}\,\boldsymbol{\Omega}_\lambda$, where

$$\boldsymbol{\Omega} = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}, \quad \boldsymbol{\Omega}_\lambda = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}, \tag{5.3}$$

and $\boldsymbol{\Omega}\,\boldsymbol{u} = \boldsymbol{\omega} \times \boldsymbol{u}$ for any $\boldsymbol{u} \in \mathbb{R}^3$.

### 5.2.2 Dynamic Model

The force equilibrium in the inertial coordinate system is given by:

$$m\dot{\boldsymbol{v}} = mg\,\mathbf{e}_z - (T_1 + T_2 + T_3 + T_4)\,\boldsymbol{\lambda}_3 + \boldsymbol{R}\boldsymbol{F}_d\,, \tag{5.4}$$

and in the body coordinate system:

$$m \begin{bmatrix} \dot{v}_1 \\ \dot{v}_2 \\ \dot{v}_3 \end{bmatrix} = -m\,\boldsymbol{\Omega}_\lambda \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} + mg\,\boldsymbol{R}^T\mathbf{e}_z - \begin{bmatrix} 0 \\ 0 \\ T_1 + T_2 + T_3 + T_4 \end{bmatrix} + \boldsymbol{F}_d\,, \tag{5.5}$$

where $m$ is the quadrotor mass, $\boldsymbol{F}_d$ the drag force in the body coordinate system due to the quadrotor translational velocity and the wind, and $T_k$ the thrust of the $k$-th propeller with $k \in \{1, 2, 3, 4\}$ [31, 84].

The moment equilibrium about the quadrotor center of mass in the inertial coordinate system is given by:

$$\frac{d}{dt}\left(\boldsymbol{I}\boldsymbol{\omega}+\sum_{k=1}^{4}\boldsymbol{I}_{p,k}\boldsymbol{\alpha}_k\right)=\boldsymbol{R}\begin{bmatrix}l(T_4-T_2)\\ l(T_1-T_3)\\ M_1-M_2+M_3-M_4\end{bmatrix}+\boldsymbol{R}\boldsymbol{M}_d\,, \tag{5.6}$$

where $\boldsymbol{I}$ is the quadrotor tensor of inertia with respect to the center of mass, $\boldsymbol{I}_{p,k}$ the cumulative propeller and motor tensor of inertia with respect to the unit center of mass, $\boldsymbol{M}_d$ the drag moment in the body coordinate system due to the quadrotor rotational velocity, $M_k$ the propeller shaft torque, $\boldsymbol{\alpha}_k$ the propeller angular velocity, and $l$ the distance of the propeller center of mass from the quadrotor center of mass.

If we assume that the quadrotor tensor of inertia in the body coordinate system, $\boldsymbol{I}_\lambda=\boldsymbol{R}^T\boldsymbol{I}\boldsymbol{R}$, is constant by averaging the variable contribution of the rotating propellers, that the third principal component, $I_p$, of the tensor $\boldsymbol{I}_{p,k}$ is aligned with $\boldsymbol{\lambda}_3$ and is the same for each propeller, that the propellers angular velocities are given by $\boldsymbol{\alpha}_k=(-1)^k\alpha_k\boldsymbol{\lambda}_3$, and we neglect the drag moment $\boldsymbol{M}_d$, Equation (5.6) becomes in the body coordinate system:

$$\boldsymbol{I}_\lambda\begin{bmatrix}\dot\omega_1\\ \dot\omega_2\\ \dot\omega_3\end{bmatrix}=-\boldsymbol{\Omega}_\lambda\boldsymbol{I}_\lambda\begin{bmatrix}\omega_1\\ \omega_2\\ \omega_3\end{bmatrix}-I_p\begin{bmatrix}\alpha\,\omega_2\\ -\alpha\,\omega_1\\ \dot\alpha\end{bmatrix}+\begin{bmatrix}l(T_4-T_2)\\ l(T_1-T_3)\\ M_1-M_2+M_3-M_4\end{bmatrix}\,, \tag{5.7}$$

where $\alpha=-\alpha_1+\alpha_2-\alpha_3+\alpha_4$.

### 5.2.3 Aerodynamic Forces

The drag force in the body coordinate system can be approximated as:

$$\boldsymbol{F}_d=\frac{\rho}{2}\begin{bmatrix}c_{d,1}A_1(w_1-v_1)|w_1-v_1|\\ c_{d,2}A_2(w_2-v_2)|w_2-v_2|\\ c_{d,3}A_3(w_3-v_3)|w_3-v_3|\end{bmatrix}\,, \tag{5.8}$$

where $w_1$, $w_2$, and $w_3$ are the components of the wind velocity in the body coordinate system and $c_{d,i}$ are the quadrotor drag coefficients.

The propeller thrust can be written as:

$$T_k=c_{T,k}\rho A_p(r_p\alpha_k)^2\,, \tag{5.9}$$

where $c_{T,k}$ is the non-dimensional thrust coefficient, $\rho$ the air density, $A_p$ the propeller disk area, and $r_p$ the propeller radius. Using blade element

theory in case of rigid blade, uniform inflow, and linear twist distribution, the thrust coefficient is given by Johnson [59]:

$$c_{T,k} = \frac{\sigma a}{2}\left(\frac{\theta_{.75}}{3} + \frac{\mu_k^2}{2}\left(\theta_{.75} - \frac{\theta_{\text{tw}}}{4}\right) - \frac{\lambda_k}{2}\right) , \tag{5.10}$$

where $\sigma$ is the propeller solidity ratio, $a$ is the slope of the blade lift curve, $\theta = \theta_{.75} + \theta_{\text{tw}}(r/r_p - 3/4)$ the pitch angle of the blade, $\theta_{.75}$ the pitch of the blade at 75% radius, $\lambda_k$ the inflow ratio, and the advance ratio is defined as:

$$\mu_k = \sqrt{(w_1 - (v_1 + v_{r1,k}))^2 + (w_2 - (v_2 + v_{r2,k}))^2}/(r_p\alpha_k) .$$

The velocity contributions due to the rotation of the quadrotor are $v_{r1,k} \in l\{0, -\omega_3, 0, \omega_3\}$, $v_{r2,k} \in l\{\omega_3, 0, -\omega_3, 0\}$, and $v_{r3,k} \in l\{-\omega_2, \omega_1, \omega_2, -\omega_1\}$.

Using momentum theory, the inflow ratio can be derived as solution of the following quartic equation [59]:

$$\lambda_k = \lambda_{c,k} + \frac{c_{T,k}}{2\sqrt{\mu_k^2 + \lambda_k^2}} , \tag{5.11}$$

where $\lambda_{c,k} = (w_3 - (v_3 + v_{r3,k})/(r_p\alpha_k)$. In vertical flight ($\mu_k = 0$) Equation (5.11) has the following solution assuming the inflow ratio to be positive:

$$\lambda_k = \frac{\lambda_{c,k}}{2} + \sqrt{\left(\frac{\lambda_{c,k}}{2}\right)^2 + \frac{c_{T,k}}{2}} , \tag{5.12}$$

which becomes $\lambda_k = \sqrt{c_{T,k}/2}$ in hover ($\lambda_{c,k} = 0$). In the general case, Equation (5.10) can be substituted in (5.11) and a solution can be found using a Newton-Raphson schema. In vertical flight the closed form solution is given by [59]:

$$\lambda_k = \frac{1}{16}\left(8\lambda_{c,k} - \sigma a + \sqrt{\frac{64}{3}\theta_{.75}\sigma a + (8\lambda_{c,k} - \sigma a)^2}\right) , \tag{5.13}$$

which can be used as the initial value for the numeric procedure.

Finally, we define the hover angular velocity, $\alpha_h$, for the propeller as:

$$\alpha_h = \sqrt{\frac{mg}{4\rho A_p (r_p)^2 c_{T_h}}} , \tag{5.14}$$

where $c_{T_h}$ is obtained from Equations (5.10) and (5.13) and by assuming $\lambda_{c,k} = 0$.

The propeller shaft torques can be written as:

$$M_k = c_{M,k}\rho A_p r_p (r_p \alpha_k)^2 \,, \tag{5.15}$$

where $c_{M,k}$ is the non-dimensional torque coefficient. This is equal to the power coefficient $c_{P,k}$, since for each propeller the power is given by $P_k = M_k \alpha_k$, and it accounts for the induced power required to produce the thrust, the profile power required to turn the propeller, the parasite power required to overcome drag forces, and the climb power. Assuming again rigid blade, uniform inflow, and linear twist distribution, the power coefficient can be obtained as [59]:

$$c_{P,k} = \kappa \lambda_k c_{T,k} + \frac{\sigma c_{d_0}}{8} + \mu_k^2 \left( \frac{\sigma c_{d_0}}{8} - \lambda_k \frac{\sigma a}{4} \left( \theta_{.75} - \frac{\theta_{\text{tw}}}{4} \right) \right) \,, \tag{5.16}$$

where $\kappa$ is an empirical factor to account for additional losses and $c_{d_0}$ is a mean blade drag coefficient.

### 5.2.4 Motor Model

For each propeller-motor unit, the moment equilibrium in the $\boldsymbol{\lambda}_3$ direction is given by:

$$I_p \frac{d\alpha_k}{dt} = c_i (i_k - i_0) - M_k \,, \tag{5.17}$$

where $c_i$ is the motor torque constant, $i_k$ the motor current, and $i_0$ is the zero load current. The governing equation of the motor circuit can be approximated as:

$$L \frac{di_k}{dt} = u_k - R i_k - \frac{\alpha_k}{c_v} \,, \tag{5.18}$$

where $L$ is the motor inductance, $R$ the motor resistance, $u_k$ the motor terminal voltage, and $c_v$ the motor speed constant. We assume $c_v = 1/c_i$ [96]. The terminal motor voltage is defined as a function of the reference angular velocity $\alpha_{R,k}$:

$$u_k = R \left( \frac{M_k(\alpha_{R,k})}{c_i} + i_0 \right) + \frac{\alpha_{R,k}}{c_v} \,. \tag{5.19}$$

A saturation is also introduced to ensure that $i_k \leq i_{\text{MAX}}$ and $u_k \leq u_{\text{MAX}}$.

Finally, the objective function (2.18), which is the total energy used by the quadrotor during the survey mission, can be calculated as:

$$E = \sum_{k=1}^{4} \int i_k u_k dt \tag{5.20}$$

### 5.2.5 Quadrotor Controller

In order to evaluate the total energy, $E$, defined in Equation (5.20), a control law $\alpha_{R,k}(t)$ has to be determined to allow the quadrotor to visit all the inspection configurations $\boldsymbol{q}_i$ in the given sequence. The two layer feedback control loop proposed in [70, 71, 72] is used for this purpose.

First, an inner control loop is implemented using a PD controller to determine the reference propeller angular velocities as function of the reference yaw angle $\psi_R$, the reference roll angle $\varphi_R$, the reference pitch angle $\vartheta_R$, and the altitude error $\Delta z_R$:

$$\begin{bmatrix} \alpha_{R,1} \\ \alpha_{R,2} \\ \alpha_{R,3} \\ \alpha_{R,4} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ -1 & -1 & 0 & 1 \\ 1 & 0 & -1 & 1 \\ -1 & 1 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} k_{p,\psi}(\psi_R - \psi) + k_{d,\psi}(\omega_{3,R} - \omega_3) \\ k_{p,\varphi}(\varphi_R - \varphi) + k_{d,\varphi}(\omega_{1,R} - \omega_1) \\ k_{p,\vartheta}(\vartheta_R - \vartheta) + k_{d,\vartheta}(\omega_{2,R} - \omega_2) \\ \alpha_h - \Delta z_R \end{bmatrix} , \quad (5.21)$$

where $k_p$ and $k_d$ are the gains of the PD controller.

Second, an outer control loop is implemented using a PID controller with a feedforward term to determine the reference roll angle, the reference pitch angle, and altitude error $\Delta z_R$ as function of the position error, $\boldsymbol{e} = \boldsymbol{r}_R - \boldsymbol{r}$, and the reference yaw angle, $\psi_R$. This is achieved by linearizing Equation (5.4) at the hovering state:

$$\varphi_R = -\Delta x_R \sin(\psi_R) + \Delta y_R \cos(\psi_R) , \quad (5.22)$$

$$\vartheta_R = -\Delta x_R \cos(\psi_R) - \Delta y_R \sin(\psi_R) , \quad (5.23)$$

where:

$$\begin{bmatrix} \Delta x_R \\ \Delta y_R \\ \Delta z_R \end{bmatrix} = \boldsymbol{K}_i \int \boldsymbol{e} dt + \boldsymbol{K}_p \boldsymbol{e} + \boldsymbol{K}_d \dot{\boldsymbol{e}} + \boldsymbol{K}_f \ddot{\boldsymbol{r}}_R , \quad (5.24)$$

and $\boldsymbol{K}_i$, $\boldsymbol{K}_p$, and $\boldsymbol{K}_d$ are the gain diagonal matrices of the PID controller, and $\boldsymbol{K}_f$ is a gain matrix for the feed-forward term.

When the quadrotor is hovering all the derivatives of the reference $\boldsymbol{r}_R$ are zero. Afterwards, when the quadrotor moves from one neighborhood to the next the integral gains are set to zero, and the position error is defined by:

$$\boldsymbol{e} = \tilde{\boldsymbol{r}}_R - \boldsymbol{r} - ((\tilde{\boldsymbol{r}}_R - \boldsymbol{r}) \cdot \boldsymbol{t}_R)\boldsymbol{t}_R , \quad (5.25)$$

where $\tilde{\boldsymbol{r}}_R$ is the point on the reference trajectory closest to $\boldsymbol{r}$, and $\boldsymbol{t}_R$ the vector tangent to the reference trajectory at that location. The velocity error is defined as $\dot{\boldsymbol{e}} = \dot{\tilde{\boldsymbol{r}}}_R - \boldsymbol{r}$.

### 5.2.6 Modification to hybrid random-key GA

Within the hybrid random-key GA framework, the computational cost for calculating a collision-free path and its related energy value for each *chromosome* in the *population* is extremely high. Since a large amount of the energy is consumed by the propellers just to generate the thrust, we initially assume that there is a direct correlation between path length and energy consumption.

Afterwards, the following procedure is introduced at each *generation* of the hybrid random-key GA to account for collision avoidance and energy consumption:

1. The best $m$ *chromosomes* not yet verified are selected using the norm-based objective function value (two *chromosomes* are considered to be equal if they simultaneously correspond to the same cycle and have the same objective function value):
   - a tour is decoded from each *chromosome*, and the straight line between each two consecutive configurations is checked for collisions using OBB;
   - if the straight line intersects only one OBB, then additional avoidance points are added around the OBB to avoid collision;
   - if the straight line intersects more than one OBB, then the bi-RRT planner is used to generate a collision free path;
   - using the collision free path, a reference trajectory $[\boldsymbol{r}_T(t), \psi_T(t)]$ is generated, a full simulation of the flight is performed, and if the energy-based objective function value improves the best known value, the calculated collision free tour is stored;
   - The *chromosome* is added to the list of the verified chromosomes.
2. The *chromosomes* in the *population* are sorted again using the energy-based objective function value. If a *chromosome* does not have an energy value, its value is then equated to the maximum value in the population.
3. *Selection*, *immigration*, and *crossover* are performed using the energy-based objective function value, and improvement heuristics are performed using one of the two norms.

## 5.3 Neighborhood definition

Given a feature that needs to be inspected, a neighborhood is defined as the set of all the feasible configurations of the camera that allow to acquire a complete image of such feature with enough resolution and limited distortion. If we assume without loss of generality that the camera center corresponds to the origin of the body coordinate system, each camera configuration is specified by six parameters, $\boldsymbol{q} = [x, y, z, \psi, \varphi, \vartheta]$.

A camera configuration is feasible with respect to a feature, if the convex hull of such feature is contained in the camera field of view (FOV). The FOV is defined in the body coordinate system by the polyhedron:

$$\mathbf{A}_{\text{FOV}} \boldsymbol{p}_b + \mathbf{b}_{\text{FOV}} \leq 0 \ , \tag{5.26}$$

where:

$$\mathbf{A}_{\text{FOV}} = \begin{bmatrix} -1 & 0 & 0 \\ -\text{s}\beta_h & \text{c}\beta_h & 0 \\ -\text{s}\beta_h & -\text{c}\beta_h & 0 \\ -\text{s}\beta_v & 0 & \text{c}\beta_v \\ -\text{s}\beta_v & 0 & -\text{c}\beta_v \\ 1 & 0 & 0 \end{bmatrix} \ ,$$

$$\mathbf{b}_{\text{FOV}} = \begin{bmatrix} -d_{\text{MIN}} & 0 & 0 & 0 & 0 & d_{\text{MAX}} \end{bmatrix}^T \ ,$$

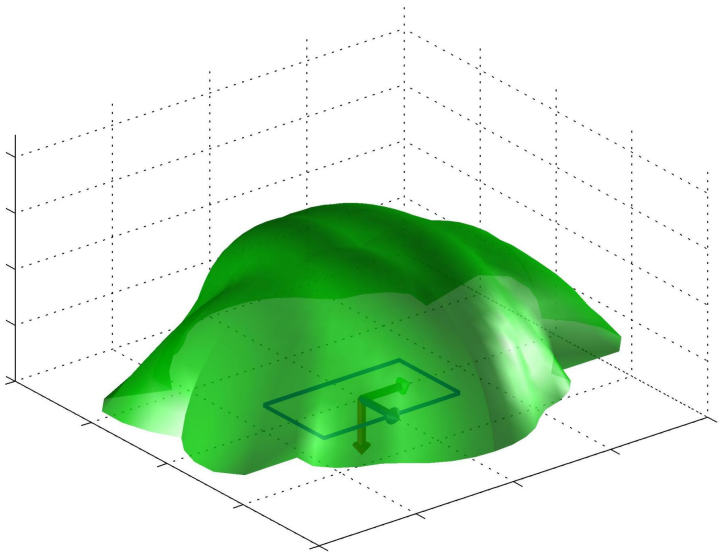
$\boldsymbol{p}_b$ is an arbitrary point in the body coordinate system, $\beta_h$ half the camera horizontal angle of view, $\beta_v$ half the camera vertical angle of view, and $d_{\text{MIN}}$ and $d_{\text{MAX}}$ are the minimum and maximum allowed distances along the camera normal direction between the camera and the feature that guarantee enough resolution in the image.

The convex hull of the $i$-th feature is contained in the camera FOV if its vertices, $\boldsymbol{p}_{i,v}$, satisfy Equation (5.26). Considering the transformation between inertial and body coordinate system illustrated in Section 5.2.1, this condition can be expressed by the following non-linear constraints in the variables $\boldsymbol{q} = [x, y, z, \psi, \varphi, \vartheta]$:

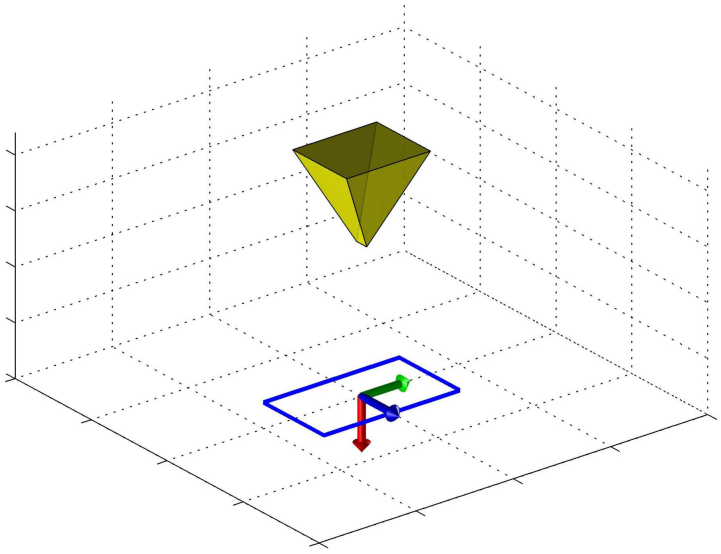
$$\mathbf{A}_{\text{FOV}} \boldsymbol{R}^T \boldsymbol{p}_{i,v} - \mathbf{A}_{\text{FOV}} \boldsymbol{R}^T \boldsymbol{r} + \mathbf{b}_{\text{FOV}} \leq 0 \qquad \forall v \ . \tag{5.27}$$

Moreover, the following constraint has to be satisfied to insure that the camera is placed on the correct side of the feature:

$$\boldsymbol{n}_i \cdot \boldsymbol{\lambda}_1 \leq 0 \ , \tag{5.28}$$



(a) Separation surface.



(b) Polyhedral neighborhood.

Figure 5.2: Neighborhood definition for a rectangular feature with a 35 mm focal length camera.

where $\boldsymbol{\lambda}_1$ is the camera view direction and $\boldsymbol{n}_i$ a representative normal direc-

tion to the feature.

If a rectangle is used to define the convex hull of the feature, the constraints given in Equation (5.27) divide the half-space on one side of the feature into two regions, as illustrated in Figure 5.2a. Below the surface, there is no camera configuration that allows an image of the entire feature to be captured. Above the surface, there is at least one configuration that allows the feature to be entirely captured by the camera.

To avoid excessive distortion, however, the camera should not be placed in the regions corresponding to a large angle between the camera view direction and the normal direction to the feature. In practical applications the camera is usually not fixed on the quadrotor, but a gimbal is used to control its orientation. The orientation of the quadrotor and the camera can thus be controlled separately. Therefore, we assume that the camera roll angle is zero, and that its view direction should remain orthogonal to the feature, i.e., $\boldsymbol{\lambda}_{i,1} = -\boldsymbol{n}_i$. Using this assumption, the constraints given in Equation (5.27) become linear inequalities in the variables $\boldsymbol{r} = [x, y, z]$, and the neighborhoods become polyhedra in $\mathbb{R}^3$, as illustrated in Figure 5.2b.

If $\mathbf{p}_{i,f}$ is the position of the center of the $i$-th feature, and $l_{i,2}$ and $l_{i,3}$ the two principal half dimensions of its rectangular bounding box, the four vertices that define the constraints in Equation (5.27) are $\boldsymbol{p}_{i,v} = \mathbf{p}_{i,f} \pm l_{i,2}\boldsymbol{\lambda}_{i,2} \pm l_{i,3}\boldsymbol{\lambda}_{i,3}$.

Although the total number of linear inequalities per neighborhood is then 24, we can reduce them to 6. Assuming that the orientation of the camera is fixed and determined by the rectangular bounding box of the feature to be inspected, than the matrix $\boldsymbol{R}$ can be written as:

$$\boldsymbol{R}_i = \begin{bmatrix} \boldsymbol{\lambda}_{i,1} & \boldsymbol{\lambda}_{i,2} & \boldsymbol{\lambda}_{i,3} \end{bmatrix} \tag{5.29}$$

If $\boldsymbol{R}_i$ and $\boldsymbol{p}_{i,v} = \mathbf{p}_{i,f} \pm l_{i,2}\boldsymbol{\lambda}_{i,2} \pm l_{i,3}\boldsymbol{\lambda}_{i,3}$ are substituted in Equation (5.27), then the following 24 inequalities are obtained:

$$\begin{aligned}
d_{\text{MIN}} + \boldsymbol{\lambda}_{i,1} \cdot (\boldsymbol{r} - \mathbf{p}_{i,f} \pm l_{i,2}\boldsymbol{\lambda}_{i,2} \pm l_{i,3}\boldsymbol{\lambda}_{i,3}) &\leq 0 \\
(\tan(\beta_h)\boldsymbol{\lambda}_{i,1} - \boldsymbol{\lambda}_{i,2}) \cdot (\boldsymbol{r} - \mathbf{p}_{i,f} \pm l_{i,2}\boldsymbol{\lambda}_{i,2} \pm l_{i,3}\boldsymbol{\lambda}_{i,3}) &\leq 0 \\
(\tan(\beta_h)\boldsymbol{\lambda}_{i,1} + \boldsymbol{\lambda}_{i,2}) \cdot (\boldsymbol{r} - \mathbf{p}_{i,f} \pm l_{i,2}\boldsymbol{\lambda}_{i,2} \pm l_{i,3}\boldsymbol{\lambda}_{i,3}) &\leq 0 \\
(\tan(\beta_v)\boldsymbol{\lambda}_{i,1} - \boldsymbol{\lambda}_{i,3}) \cdot (\boldsymbol{r} - \mathbf{p}_{i,f} \pm l_{i,2}\boldsymbol{\lambda}_{i,2} \pm l_{i,3}\boldsymbol{\lambda}_{i,3}) &\leq 0 \\
(\tan(\beta_v)\boldsymbol{\lambda}_{i,1} + \boldsymbol{\lambda}_{i,3}) \cdot (\boldsymbol{r} - \mathbf{p}_{i,f} \pm l_{i,2}\boldsymbol{\lambda}_{i,2} \pm l_{i,3}\boldsymbol{\lambda}_{i,3}) &\leq 0 \\
-d_{\text{MAX}} - \boldsymbol{\lambda}_{i,1} \cdot (\boldsymbol{r} - \mathbf{p}_{i,f} \pm l_{i,2}\boldsymbol{\lambda}_{i,2} \pm l_{i,3}\boldsymbol{\lambda}_{i,3}) &\leq 0
\end{aligned} \tag{5.30}$$

Since $\boldsymbol{\lambda}_{i,r} \cdot \boldsymbol{\lambda}_{i,s} = 0$ if $r \neq s$ and $l_{i,2/3} > 0$, the following 6 inequalities can

be inferred:

$$\begin{aligned} d_{\text{MIN}} + \boldsymbol{\lambda}_{i,1} \cdot (\boldsymbol{r} - \mathbf{p}_{i,f}) &\leq 0 \\ (\tan(\beta_h)\boldsymbol{\lambda}_{i,1} - \boldsymbol{\lambda}_{i,2}) \cdot (\boldsymbol{r} - \mathbf{p}_{i,f} - l_{i,2}\boldsymbol{\lambda}_{i,2}) &\leq 0 \\ (\tan(\beta_h)\boldsymbol{\lambda}_{i,1} + \boldsymbol{\lambda}_{i,2}) \cdot (\boldsymbol{r} - \mathbf{p}_{i,f} + l_{i,2}\boldsymbol{\lambda}_{i,2}) &\leq 0 \\ (\tan(\beta_v)\boldsymbol{\lambda}_{i,1} - \boldsymbol{\lambda}_{i,3}) \cdot (\boldsymbol{r} - \mathbf{p}_{i,f} - l_{i,3}\boldsymbol{\lambda}_{i,3}) &\leq 0 \\ (\tan(\beta_v)\boldsymbol{\lambda}_{i,1} + \boldsymbol{\lambda}_{i,3}) \cdot (\boldsymbol{r} - \mathbf{p}_{i,f} + l_{i,3}\boldsymbol{\lambda}_{i,3}) &\leq 0 \\ -d_{\text{MAX}} - \boldsymbol{\lambda}_{i,1} \cdot (\boldsymbol{r} - \mathbf{p}_{i,f}) &\leq 0 \end{aligned} \tag{5.31}$$

For the sake of clarity, constraints 5.31, which define each neighborhood, can be summered as:

$$\mathbf{A}_i \boldsymbol{r} + \mathbf{b}_i \leq 0 \ , \tag{5.32}$$

where:

$$\mathbf{A}_i = \begin{bmatrix} \boldsymbol{\lambda}_{i,1} & \mathbf{n}_{i,1} & \mathbf{n}_{i,2} & \mathbf{n}_{i,3} & \mathbf{n}_{i,4} & -\boldsymbol{\lambda}_{i,1} \end{bmatrix}^T \ , \tag{5.33}$$

$$\mathbf{b}_i = - \begin{bmatrix} \boldsymbol{\lambda}_{i,1} \cdot \mathbf{p}_{i,1} \\ \mathbf{n}_{i,1} \cdot \mathbf{p}_{i,2} \\ \mathbf{n}_{i,2} \cdot \mathbf{p}_{i,3} \\ \mathbf{n}_{i,3} \cdot \mathbf{p}_{i,4} \\ \mathbf{n}_{i,4} \cdot \mathbf{p}_{i,5} \\ -\boldsymbol{\lambda}_{i,1} \cdot \mathbf{p}_{i,6} \end{bmatrix} \ , \tag{5.34}$$

the six support points $\mathbf{p}_{i,1\ldots6}$ are defined as:

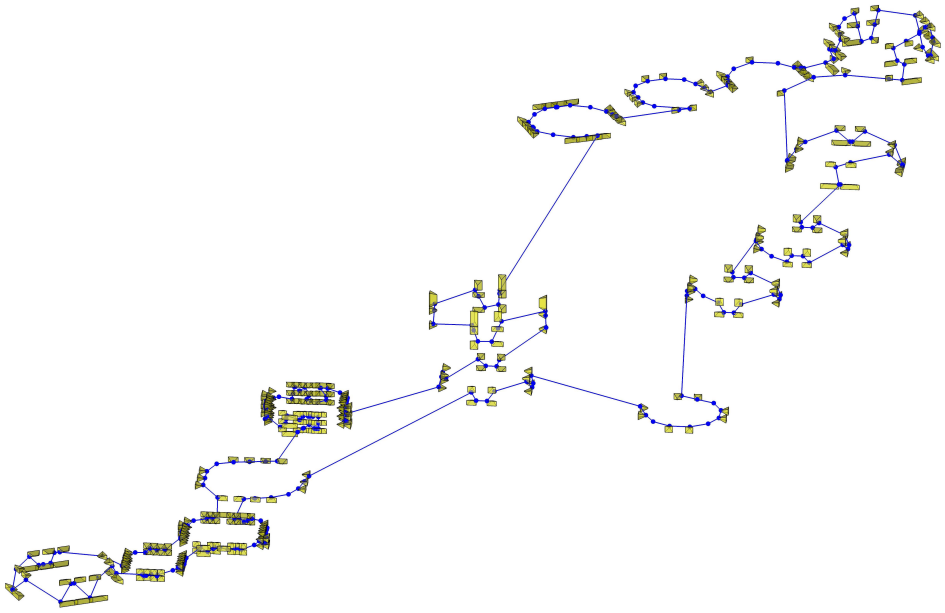
$$\begin{aligned} \mathbf{p}_{i,1} &= \mathbf{p}_{i,f} - d_{\text{MIN}}\boldsymbol{\lambda}_{i,1} \ , \\ \mathbf{p}_{i,2/3} &= \mathbf{p}_{i,f} \pm l_{i,2}\boldsymbol{\lambda}_{i,2} \ , \\ \mathbf{p}_{i,4/5} &= \mathbf{p}_{i,f} \pm l_{i,3}\boldsymbol{\lambda}_{i,3} \ , \\ \mathbf{p}_{i,6} &= \mathbf{p}_{i,f} - d_{\text{MAX}}\boldsymbol{\lambda}_{i,1} \ , \end{aligned}$$

and the four normals $\boldsymbol{n}_{i,1\ldots4}$ are defined as:

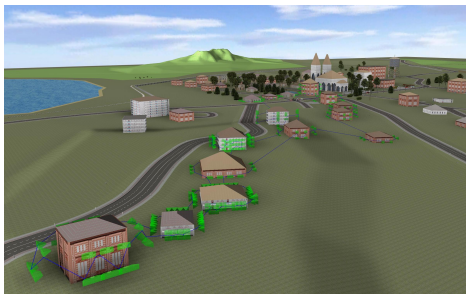
$$\begin{aligned} \boldsymbol{n}_{i,1/2} &= \tan(\beta_h)\boldsymbol{\lambda}_{i,1} \mp \boldsymbol{\lambda}_{i,2} \ , \\ \boldsymbol{n}_{i,3/4} &= \tan(\beta_h)\boldsymbol{\lambda}_{i,1} \mp \boldsymbol{\lambda}_{i,3} \ . \end{aligned}$$

A base point, $\mathbf{p}_{i,0}$, is also defined for each neighborhood:

$$\begin{aligned} d_2 &= \frac{l_{i,2}}{\tan(\beta_h)} \ , \\ d_3 &= \frac{l_{i,3}}{\tan(\beta_v)} \ , \\ \mathbf{p}_{i,0} &= \mathbf{p}_{i,f} - \max\{d_{\text{MIN}}, d_2, d_3\}\, \boldsymbol{\lambda}_{i,1} \ . \end{aligned}$$



(a) Near *optimal tour* for the scenario with low building density.



(b) Urban scenario with 372 neighborhoods.

(c) Path section

Figure 5.3: For the optimization case "Energy With $\psi$", path length and energy consumption are improved by 15.6% and 19.3%, respectively.

The values for the yaw angles $\psi_i$, which are used in the quadrotor control scheme, can be easily determined from $\boldsymbol{\lambda}_{i,2}$ and Equation (5.1).

## 5.4 Computational results

Two test scenarios are generated using the commercial software for urban planning Autodesk Infrastructure Modeler [10]. The quadrotor is required to acquire an image of all the windows and doors of a group of buildings within two different urban environments having different building density, as illustrated in Figures 5.3 and 5.4. The near *optimal tour* is depicted using blue segments, and the locations where the images are acquired from are depicted as blue dots. The neighborhoods are depicted as green polyhedra.

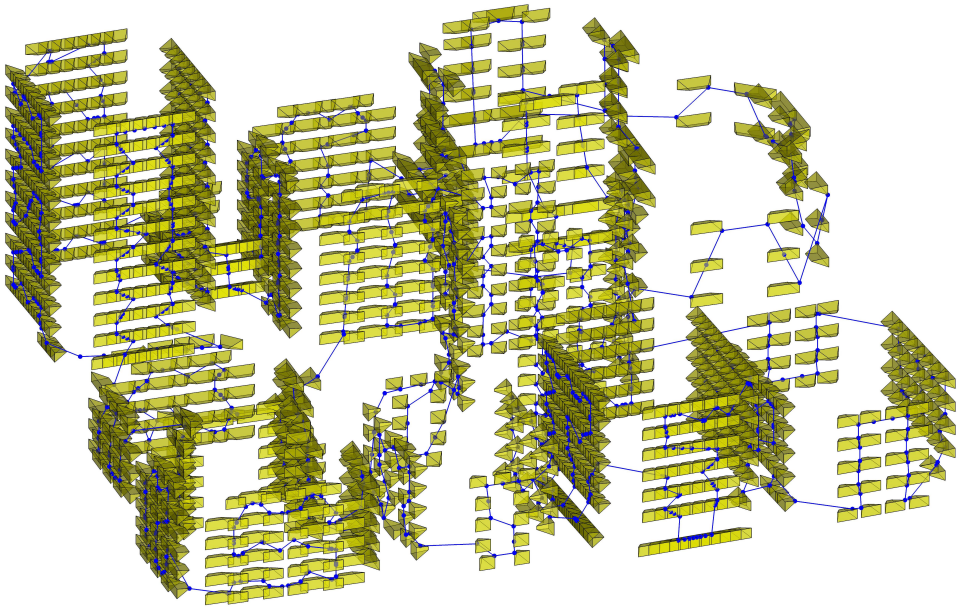
The quadrotor MikroKopter Quadro L4-ME [53] is used as model for the dynamic simulation and the employed parameters are reported in Table 5.1.

Table 5.1: Parameters used in the simulation.

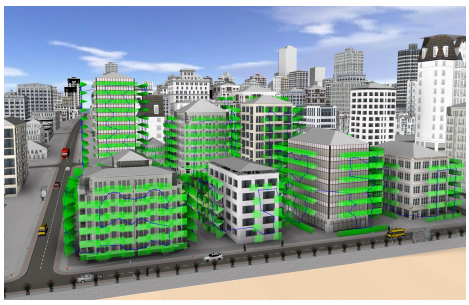
| Parameter | Units | Value | Parameter | Units | Value |
|---|---|---|---|---|---|
| $l$ | m | 0.24 | $c_{d_0}$ | | 0.01 |
| $m$ | kg | 0.8780 | $R$ | $\Omega$ | 0.77 |
| $I_{11}$ | kg m$^2$ | 0.0109 | $L$ | $H$ | 0.0019 |
| $I_{22}$ | kg m$^2$ | 0.0111 | $i_0$ | A | 0.27 |
| $I_{33}$ | kg m$^2$ | 0.0204 | $c_v$ | rad / s V | 79.59 |
| $I_p$ | kg m$^2$ | 1.76e-005 | $c_i$ | N m / A | 0.0126 |
| $r_p$ | m | 0.127 | $i_{\text{MAX}}$ | A | 9 |
| $c_{d,1/2}$ | | 0.01 | $u_{\text{MAX}}$ | V | 14.8 |
| $c_{d,3}$ | | 0.02 | $2\beta_h$ | deg | 54.4 |
| $a$ | | 5.7 | $2\beta_v$ | deg | 37.8 |
| $\sigma$ | | 0.0962 | $d_{\text{MIN}}$ | m | 0 |
| $\theta_{.75}$ | deg | 10.37 | $d_{\text{MAX}}$ | m | 2.2 |
| $\theta_{\text{tw}}$ | deg | -21.27 | $m$ | | 20 |
| $\kappa$ | | 1.1 | $c_\psi$ | | 5.76 |

The reference trajectory $[\boldsymbol{r}_T(t), \psi_T(t)]$ is generated by linearly interpolating the configurations along the collision free path calculated by the GA, and letting the quadrotor hover at each location for 0.6 s, while the image is acquired.

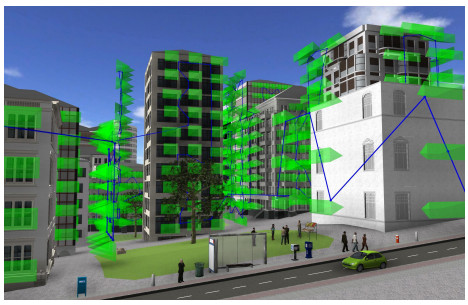
For each scenario, five different cases are considered, as illustrated in Tables 5.2 and 5.3. Moreover, for each case, the results obtained without and with the collision avoidance procedure are reported in the columns labeled with "Direct" and "Collision Free", respectively. The energy values are expressed in kJ and in Ah at a voltage of 14.8 V, which is the reference voltage of the four pack Lipo Battery 2.2 Ah used for the considered quadrotor model.



(a) Near *optimal tour* for the scenario with high building density.



(b) Urban scenario with 1,611 neighborhoods.



(c) Path section

Figure 5.4: For the optimization case "Energy Only", path length and energy consumption are improved by 38.3% and 23.4%, respectively.

In the first case, labeled "TSP Based", the STSP defined by the base points of the neighborhoods, $\mathbf{p}_{i,0}$, is solved using the exact solver CONCORDE [7]. Then a collision free path is calculated and the flight is simulated. The objective function values attained for this tour are used as a reference value to evaluate the performance of the proposed method.

In the second case, labeled "Distance Only", the Euclidean norm is used

Table 5.2: Optimization results with 372 neighborhoods.

| Case | Objective | Direct | | Collision Free | |
|---|---|---|---|---|---|
| | | best | impr. | best | impr. |
| TSP Based | d [m] | 2,195 | | 2,216 | |
| | $\psi$ [deg] | 7,379 | | 7,379 | |
| | $E$ [kJ] | 107.9 | | 114.4 | |
| | $E$ [Ah] | 2.03 | | 2.15 | |
| Distance Only | d [m] | 1,787 | 18.6% | 1,796 | **19.0%** |
| | $\psi$ [deg] | 7,378 | 0.0% | 7,378 | 0.0% |
| | $E$ [kJ] | 89.74 | 16.9% | 94.07 | 17.8% |
| | $E$ [Ah] | 1.68 | | 1.77 | |
| Distance with $\psi$ | d [m] | 1,841 | 16.1% | 1,864 | 15.9% |
| | $\psi$ [deg] | 6,838 | 7.3% | 6,838 | **7.3%** |
| | $E$ [kJ] | 89.66 | 16.9% | 95.39 | 16.6% |
| | $E$ [Ah] | 1.68 | | 1.79 | |
| Energy Only | d [m] | 1,820 | 17.1% | 1,828 | 17.5% |
| | $\psi$ [deg] | 7,199 | 2.4% | 7,199 | 2.4% |
| | $E$ [kJ] | 89.66 | 16.9% | 93.15 | 18.6% |
| | $E$ [Ah] | 1.68 | | 1.75 | |
| Energy with $\psi$ | d [m] | 1,863 | 15.1% | 1,872 | 15.6% |
| | $\psi$ [deg] | 6,838 | 7.3% | 6,838 | 7.3% |
| | $E$ [kJ] | 88.18 | 18.3% | 92.38 | **19.3%** |
| | $E$ [Ah] | 1.66 | | 1.73 | |

as edge weighting function and the yaw angle is not considered in the optimization. In the third case, labeled "Distance With $\psi$", the edge weighting function used in the LK Heuristic evaluation is the Quadratic norm. The $Q$ matrix is a diagonal matrix with elements $[1, 1, 1, c_\psi]$. Furthermore, since the values of the yaw angles $\psi_i$ can not be changed but only their sequence can be changed, as illustrated in Section 5.3, in the evaluation of the Touring Heuristic only the Euclidean norm is used as edge weighting function and the yaw angle is not considered. In these two cases the collision-free path and energy-based objective function values are obtained in a postprocessing step after the GA terminates.

The last two cases labeled "Energy Only" and "Energy With $\psi$" are similar to the previous two cases but the procedure illustrated in Section 5.2.6 is performed at each generation in the GA.

In the scenario with low building density, the total tour length is reduced up to 19%, the total traveled yaw angle up to 7.3%, and the energy consumption up to 19.3%. In the second scenario, the total tour length is

Table 5.3: Optimization results with 1,611 neighborhoods.

| Case | Objective | Direct | | Collision Free | |
|---|---|---|---|---|---|
| | | best | impr. | best | impr. |
| TSP Based | d [m] | 4,417 | | 4,421 | |
| | $\psi$ [deg] | 23,724 | | 23,724 | |
| | $E$ [kJ] | 368.9 | | 370.4 | |
| | $E$ [Ah] | 6.92 | | 6.95 | |
| Distance Only | d [m] | 2,715 | 38.5% | 2,717 | **38.6%** |
| | $\psi$ [deg] | 24,704 | -4.1% | 24,704 | -4.1% |
| | $E$ [kJ] | 288.7 | 21.7% | 289.5 | 21.8% |
| | $E$ [Ah] | 5.42 | | 5.43 | |
| Distance with $\psi$ | d [m] | 2,792 | 36.8% | 2,875 | 35.0% |
| | $\psi$ [deg] | 6,669 | 71.9% | 6,669 | **71.9%** |
| | $E$ [kJ] | 291.4 | 21.0% | 295.9 | 20.1% |
| | $E$ [Ah] | 5.47 | | 5.55 | |
| Energy Only | d [m] | 2,725 | 38.3% | 2,726 | 38.3% |
| | $\psi$ [deg] | 23,174 | 2.3% | 23,174 | 2.3% |
| | $E$ [kJ] | 283.3 | 23.2% | 283.9 | **23.4%** |
| | $E$ [Ah] | 5.32 | | 5.33 | |
| Energy with $\psi$ | d [m] | 2,930 | 33.7% | 2,965 | 33.0% |
| | $\psi$ [deg] | 7,576 | 68.1% | 7,576 | 68.1% |
| | $E$ [kJ] | 288.0 | 21.9% | 290.8 | 21.5% |
| | $E$ [Ah] | 5.40 | | 5.46 | |

reduced up to 38.6%, the total traveled yaw angle up to 71.9%, and the energy consumption up to 23.4%. These values are highlighted using bold fonts in Tables 5.2 and 5.3.

Although a shorter path generally corresponds to a reduced energy consumption, the results show that this trend is not always confirmed. By using the energy evaluation within the GA, the energy minimization can be further improved by 1.5% in the first scenario and by 1.6% in the second scenario, while the path lengths increase by 3.4% and by 0.3%, respectively. Although some oscillations in the results are caused by the heuristic nature of the proposed solver, it is important to notice that this behavior can be observed in all the performed simulations. Moreover, this result underlines the importance of embedding a full energy evaluation in the GA despite an increase in the computational cost.

## 5.5 Conclusion

In this chapter a procedure is proposed to optimize the multi-goal path of a quadrotor UAV on a survey flight mission. The optimization problem is modeled as a TSPN, and the hybrid random-key GA is used to find a near *optimal tour*. The cost function is initially defined using Euclidean or Quadratic norm, and polyhedra are used to represent the neighborhoods. Obstacle avoidance and dynamic simulation are then embedded in the GA to estimate the actual cost in terms of energy consumption for a collision free path. If compared to more traditional solution procedures, the path length and the energy consumption for an urban survey mission with more than 1,500 goals have been improved up to 38% and 23%, respectively.

# Chapter 6

# Conclusion

In this chapter, we conclude the proposed work and discuss future research topics providing a possible timetable. A list of publications is also provided.

## 6.1 Contribution

In this work we address the technical problem of finding an optimal path for a redundant robotic system to visit a sequence of several goal locations. Since the system has some degree of redundancy, not only an optimal sequence of the goals has to be defined, but also, for each goal, an optimal configuration has to be chosen among infinite possibilities. This problem can be modeled as a Traveling Salesman Problem with Neighborhoods (TSPN), which extends the well known TSP to more general cases where each vertex (goal configuration) is allowed to move in a given region (neighborhood).

We first consider the abstract optimization problem of finding an optimal sequence of optimal configurations using analytically defined edge weighing functions, and we present three solution approaches. First, a non-convex Mixed Integer Non Linear Programming (MINLP) formulation for the symmetric TSPN (STSPN) is provided, which has the following property: in case of convex neighborhoods if all the integer variables are fixed to any integer values a convex nonlinear program is obtained. We modify thus the global MINLP optimizer Couenne by implementing an ad-hoc cut generator to exploit this property, and we improve its performance by orders of magnitude. The exact solution is attained for instances with up to 16 convex neighborhoods (ellipsoids and polyhedra) in $\mathbb{R}^2$, and 12 convex neighborhoods in $\mathbb{R}^3$.

Second, an equivalent convex MINLP formulation for the STSPN is de-

rived for the case of convex neighborhoods and convex edge weighting functions. Three different formulations are derived and the convex optimizers Bonmin and Mosek are employed. In particular, after modifying Bonmin to efficiently handle the subtour elimination constraints, instances with up to 20 ellipsoids are solved to optimality. The computational cost is improved up to 2 orders of magnitude with respect to the initial approach.

Third, a hybrid random key genetic algorithm (GA) is proposed to find a near optimal tour for instances with a larger number of possibly non-convex neighborhoods. This approach uses random-key coding for the chromosomes, and it exploits the efficiency of ad-hoc heuristics to improve the quality of each chromosome rather than more traditional mutation operators. Although no optimality is guaranteed, benchmark tests show that the GA is able to find the same optimal tour in all the cases a solution is also retrieved using the MINLP optimizer. Moreover, applying the GA on TSPN instances available in the literature with circular and spherical neighborhoods, although the proposed method is not tailored for those specific problems, the best known near optimal tours have been improved on average by 1.92%.

Finally, the hybrid random-key GA is integrated with a probabilistic path planning technique based on bidirectional Rapidly-exploring Random Trees (RRTs) to better estimate the cost of each edge in the tour while generating collision-free paths. First, the traveling time currently required by a 7 DOF robotic vision inspection system to complete a multi-goal operation cycle is minimized. The neighborhoods are here approximated using piecewise cubic splines in a seven-dimensional configuration space, and the used edge weighting functions are either weighted Maximum norm or Quadratic norm. Experimental results for a given 32-goal cycle provided by Denso Wave show a cycle time improvement up to 30%. Second, the flight path and the energy consumption of a quadrotor drone on an urban survey mission are optimized. In this case the neighborhoods are approximated using three-dimensional polyhedra, and the Euclidean or Quadratic norm is used as edge weighting function. Computational experiments clearly indicate that the performance of the proposed procedure depends on the number of the neighborhoods and their spatial distribution. The best result in this work is obtained for a dense urban scenario with more than 1,500 goal locations. Path length and energy consumption are improved for this specific case up to 38% and 23%, respectively, if compared to the results of more traditional optimization techniques.

## 6.2 Future work

The proposed methods to retrieve an exact solution for STSPN instances suffers from two main limitations, i.e., only convex neighborhoods can be used, and only instances with up to 20 neighborhoods can be efficiently solved. Since for practical applications the number of neighborhoods can increase even beyond 1,000, in the future we will focus on improving the convergence rate of the proposed approach. In particular we will further investigate the convex formulations derived in this work trying to apply a preprocessing step to reduce the number of binary variables, and a projection technique to eliminate the additional variables introduced in the convex hull relaxation. Finally, an exact procedure for the case of non-convex neighborhood should be also implemented.

Second, in the current industrial environment there is an increasing need of real time adaptation of plants productivity levels to actual market requests. On the one hand, when the demand is high, the manufacturing process has to be performed in the shortest amount of time as possible, e.g. by minimizing the cycle time of each individual operation. On the other hand, when the demand is low, there is no need of such a high performing process, and other parameters such as the total energy consumption per manufactured unit can be optimized. Since industrial plants are highly automated through the employment of robotic systems, we want to investigate the possibility not only to optimize their cycle time while exploiting the redundancy in the system, but also to minimize the total energy consumed while allowing a longer cycle time, as we showed for the quadrotor application.

Third, for the case of the unmanned aerial survey system the directionality constraint in the calculation of the neighborhoods should be removed to allow a wider definition of such regions. Moreover, different analytic functions could be tested to find a better correlation between the actual energy consumption and the value of the edge weighting function used to efficiently evaluate the cost of each chromosome in the GA. Field tests using the modeled quadrotor should be also performed to validate the proposed optimization procedure.

Finally, we observe that in practical vision inspection procedures, the number of features, $n_f$, to be inspected is fixed, but the number of images necessary to capture all the features might be optimized. In other words, more than one feature can be acquired using only one image. If the possible camera placements where the images have to be taken from are fixed points, this problem is known in the literature as the View Planning Problem (VPP)

with traveling costs [97, 101]. However, if the system is redundant, i.e., the camera placements are rather regions than points, then this MINLP formulation can be introduced:

$$\text{minimize :} \quad \sum_{i=1}^{n} \mathrm{c}(\boldsymbol{q}_i)\psi_i + \gamma \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j>i}}^{n} \xi_{ij}\, \mathrm{d}\,(\boldsymbol{q}_i, \boldsymbol{q}_j) \tag{6.1}$$

$$\text{subject to :} \quad \sum_{i=1}^{n} \nu_r(\boldsymbol{q}_i)\psi_i \geq 1 \qquad \forall\, r = 1, \ldots, n_f \tag{6.2}$$

$$\sum_{j=1}^{i-1} \xi_{ji} + \sum_{j=i+1}^{n} \xi_{ij} = 2\psi_i \qquad \forall\, i \in \mathrm{V} \tag{6.3}$$

$$\sum_{i \in \mathrm{S}} \Bigg( \sum_{\substack{j \in \mathrm{V}\setminus\mathrm{S} \\ j<i}} \xi_{ji} + \sum_{\substack{j \in \mathrm{V}\setminus\mathrm{S} \\ j>i}} \xi_{ij} \Bigg) \geq 2\psi_p \qquad \begin{array}{l} \forall\, \mathrm{S} \subset \mathrm{V} \setminus \{1\},\ |\mathrm{S}| \geq 3 \\ \forall\, p \in \mathrm{V}\setminus\mathrm{S} \end{array} \tag{6.4}$$

$$\boldsymbol{q}_i \in \mathcal{Q}_i \subseteq \mathbb{R}^m \qquad \forall\, i \in \mathrm{V} \tag{6.5}$$

$$\xi_{ij} \in \{0, 1\} \qquad \forall\, i, j \in \mathrm{V},\ i \neq j \tag{6.6}$$

$$\psi_i \in \{0, 1\} \qquad \forall\, i, \in \mathrm{V} \tag{6.7}$$

$$\boldsymbol{q}_i \in \mathbb{R}^m \qquad \forall\, i \in \mathrm{V} \tag{6.8}$$

where $\mathrm{c}(\boldsymbol{q}_i)$ is a scalar function that retrieves the cost of acquiring an image from configuration $\boldsymbol{q}_i$, the binary variable $\psi_i$ is 1 only if the neighborhood $i$ is visited in the tour, and $\nu_q(\boldsymbol{q}_i)$ is the visibility binary function, which is 1 only if the $r$-th feature is visible from the $i$-th configuration. Constraints (6.2) assure that each feature is visible from at least one configuration in the tour. Constraints (6.3) and (6.4) are the extension of Constraints (2.19) and (2.20) to the case where the number of visited neighborhoods in a tour is not fixed. If we assume that $n = n_r$, $\psi_i = 1\ \forall\, i$, and $\nu_r(\boldsymbol{q}_i) = 1\ \forall\, r$ then the above formulation becomes the *first* STSPN formulation. If we assume that the configurations $\boldsymbol{q}_i$ are fixed then it becomes the VPP fomulation proposed by Wang et al. [101]. The above MINLP problem is clearly highly complex to optimize, but searching for an efficient solution procedure might be a logical extension of the present work.

# Appendix A

## A.1 Convergence of Bounded Set

The following lemma holds [48].

**Lemma 7.** *Let* $Q(\alpha) = \{\boldsymbol{q} \in \mathbb{R}^m \mid \boldsymbol{q}/\alpha \in \mathcal{Q}\} \neq \emptyset$ *for* $0 < \alpha \leq 1$. *If* $\mathcal{Q}$ *is a bounded set, then:*

$$\lim_{\alpha \to 0} Q(\alpha) = \{\boldsymbol{q} \in \mathbb{R}^m \mid \boldsymbol{q} = \mathbf{0}\} \,. \tag{A.1}$$

*Proof.* Let $\{\alpha_s\} \subset (0, 1)$ be an arbitrary sequence converging to 0. Since $Q(\alpha_s) \neq \emptyset$, there exists a corresponding sequence $\{\boldsymbol{q}_s\}$ such that $\boldsymbol{q}_s \in Q(\alpha_s)$.

Since $\mathcal{Q}$ is bounded, there exist two vectors, $\boldsymbol{l}$ and $\boldsymbol{u}$, such that $\alpha l_k \leq q_k \leq \alpha u_k$, $\forall k \in \{1, \ldots, m\}$. Therefore, $\alpha_s l_k \leq q_{k,s} \leq \alpha_s u_k$, and $\{\boldsymbol{q}_s\}$ converges to $\mathbf{0}$. □

## A.2 Coded objective function and its derivatives

This appendix illustrates the formulations used for the implementation of the objective function, its gradient, and its Hessian in the solver IPOPT. The Kronecker's delta $\delta_{i,j}$ is used in the derivation:

$$\delta_{i,j} = \begin{cases} 1 & \text{if} \quad i = j; \\ 0 & \text{otherwise.} \end{cases}$$

### A.2.1 Euclidean and Quadratic Norm

The Euclidean norm (2.15) and the Quadratic norm (2.16) are initially considered as edge weighting functions. Without loss of generality the symmetric positive definite matrix $\mathbf{Q}$ used in Equation (2.16) can be represented by a diagonal matrix having diagonal elements $\mathrm{Q}_k$. If the actual matrix is

not diagonal, a coordinate transformation can always be found such that $\mathbf{Q}$ becomes diagonal in the new coordinate system. Hereafter, the derivation for the Quadratic norm is proposed, and the corresponding derivation for the Euclidean norm can be easily obtained by replacing the elements $\mathrm{Q}_k$ with one.

Using the neighborhoods permutation $\pi(i)$ derived from the fixed binary variables $\bar{\xi}_{ij}$ or from the fractional part of a chromosome, the vertices can be reordered such that $\boldsymbol{q}_{\pi(i)} = \hat{\boldsymbol{q}}_i$. The objective function (2.49) thus becomes:

$$\begin{aligned} O &= \sum_{i=1}^{n} \sqrt{\sum_{k=1}^{m} \mathrm{Q}_k (q_{\pi(i),k} - q_{\pi(i+1),k})^2} \\ &= \sum_{i=1}^{n} \sqrt{\sum_{k=1}^{m} \mathrm{Q}_k (\hat{q}_{i,k} - \hat{q}_{i+1,k})^2} \end{aligned} \tag{A.2}$$

where it holds $\pi(n+1) = \pi(1)$, $\hat{\boldsymbol{q}}_0 = \hat{\boldsymbol{q}}_n$, and $\hat{\boldsymbol{q}}_{n+1} = \hat{\boldsymbol{q}}_1$.

Using the following substitution:

$$\Delta_{i,i+1} = \sqrt{\sum_{k=1}^{m} \mathrm{Q}_k (\hat{q}_{i,k} - \hat{q}_{i+1,k})^2} \tag{A.3}$$

the gradient of the objective function (A.2) can be obtained as:

$$\begin{aligned} \frac{\partial O}{\partial \hat{q}_{r,e}} &= \sum_{i=1}^{n} \frac{\sum_{k=1}^{m} \mathrm{Q}_k (\hat{q}_{i,k} - \hat{q}_{i+1,k})}{\Delta_{i,i+1}} (\delta_{i,r}\delta_{k,e} - \delta_{i+1,r}\delta_{k,e}) \\ &= \sum_{i=1}^{n} \mathrm{Q}_e \frac{\hat{q}_{i,e} - \hat{q}_{i+1,e}}{\Delta_{i,i+1}} \ (\delta_{i,r} - \delta_{i+1,r}) \\ &= \mathrm{Q}_e \frac{\hat{q}_{r,e} - \hat{q}_{r+1,e}}{\Delta_{r,r+1}} - \mathrm{Q}_e \frac{\hat{q}_{r-1,e} - \hat{q}_{r,e}}{\Delta_{r-1,r}} \end{aligned} \tag{A.4}$$

and the Hessian as:

$$\begin{aligned} \frac{\partial^2 O}{\partial \hat{q}_{r,e} \partial \hat{q}_{s,f}} &= \frac{\mathrm{Q}_e}{\Delta_{r,r+1}} \delta_{e,f} \ (\delta_{r,s} - \delta_{r+1,s}) \\ &\quad - \mathrm{Q}_e \mathrm{Q}_f \frac{(\hat{q}_{r,e} - \hat{q}_{r+1,e}) \ (\hat{q}_{r,f} - \hat{q}_{r+1,f})}{\Delta_{r,r+1}^3} \ (\delta_{r,s} - \delta_{r+1,s}) \\ &\quad - \frac{\mathrm{Q}_e}{\Delta_{r-1,r}} \delta_{e,f} \ (\delta_{r-1,s} - \delta_{r,s}) \\ &\quad + \mathrm{Q}_e \mathrm{Q}_f \frac{(\hat{q}_{r-1,e} - \hat{q}_{r,e}) \ (\hat{q}_{r-1,f} - \hat{q}_{r,f})}{\Delta_{r-1,r}^3} \ (\delta_{r-1,s} - \delta_{r,s}) \end{aligned} \tag{A.5}$$

The solver IPOPT requires all the functions in the problem formulation to be at least once differentiable. Since the constraints (2.5) considered in the present work are either linear, quadratic, or cubic, the derivation of their gradient and Hessian is rather trivial, and they are continuously differentiable. However, the objective function (A.2) is $C^0$ since the gradient is not defined if $\Delta_{i,i+1} = 0$, i.e., if two vertices overlap. The following modification is thus proposed to overcome this limitation.

**Modified Edge Weighting Function** The Euclidean or the Quadratic norm in the objective function (A.2) are replaced by the following edge weighting function:

$$\mathrm{d}\left(\boldsymbol{q}_i, \boldsymbol{q}_j\right) = \begin{cases} \left\|\boldsymbol{q}_j - \boldsymbol{q}_i\right\|_{2/\mathbf{Q}} & \text{if } \left\|\boldsymbol{q}_j - \boldsymbol{q}_i\right\|_{2/\mathbf{Q}} \geq \epsilon \\ \frac{\epsilon}{2} + \frac{1}{2\epsilon}\left\|\boldsymbol{q}_j - \boldsymbol{q}_i\right\|^2_{2/\mathbf{Q}} & \text{if } \left\|\boldsymbol{q}_j - \boldsymbol{q}_i\right\|_{2/\mathbf{Q}} < \epsilon \end{cases} \tag{A.6}$$

Using the same permutation $\pi(i)$ as above, the objective function (A.2) is now $C^1$ and it is given by:

$$O = \sum_{i=1}^{n} \begin{cases} \sqrt{\sum_{k=1}^{m} \mathrm{Q}_k (\hat{q}_{i,k} - \hat{q}_{i+1,k})^2} & \text{if } \Delta_{i,i+1} \geq \epsilon \\ \frac{\epsilon}{2} + \frac{1}{2\epsilon} \sum_{k=1}^{m} \mathrm{Q}_k (\hat{q}_{i,k} - \hat{q}_{i+1,k})^2 & \text{if } \Delta_{i,i+1} < \epsilon \end{cases} \tag{A.7}$$

After some simplifications, the gradient of the objective function (A.7) can be obtained as:

$$\begin{aligned} \frac{\partial O}{\partial \hat{q}_{r,e}} = \mathrm{Q}_e \left(\hat{q}_{r,e} - \hat{q}_{r+1,e}\right) & \begin{cases} \frac{1}{\Delta_{r,r+1}} & \text{if } \Delta_{r,r+1} \geq \epsilon \\ \frac{1}{\epsilon} & \text{if } \Delta_{r,r+1} < \epsilon \end{cases} \\ - \mathrm{Q}_e \left(\hat{q}_{r-1,e} - \hat{q}_{r,e}\right) & \begin{cases} \frac{1}{\Delta_{r-1,r}} & \text{if } \Delta_{r-1,r} \geq \epsilon \\ \frac{1}{\epsilon} & \text{if } \Delta_{r-1,r} < \epsilon \end{cases} \end{aligned} \tag{A.8}$$

and the Hessian as:

$$\begin{aligned} \frac{\partial^2 O}{\partial \hat{q}_{r,e} \partial \hat{q}_{s,f}} = \mathrm{Q}_e \left(\delta_{r,s} - \delta_{r+1,s}\right) & \begin{cases} \frac{\delta_{e,f}}{\Delta_{r,r+1}} - \frac{\mathrm{Q}_f (\hat{q}_{r,e} - \hat{q}_{r+1,e}) \left(\hat{q}_{r,f} - \hat{q}_{r+1,f}\right)}{\Delta^3_{r,r+1}} & \text{if } \Delta_{r,r+1} \geq \epsilon \\ \frac{\delta_{e,f}}{\epsilon} & \text{if } \Delta_{r,r+1} < \epsilon \end{cases} \\ - \mathrm{Q}_e \left(\delta_{r-1,s} - \delta_{r,s}\right) & \begin{cases} -\frac{\delta_{e,f}}{\Delta_{r-1,r}} + \frac{\mathrm{Q}_f (\hat{q}_{r-1,e} - \hat{q}_{r,e}) \left(\hat{q}_{r-1,f} - \hat{q}_{r,f}\right)}{\Delta^3_{r-1,r}} & \text{if } \Delta_{r-1,r} \geq \epsilon \\ \frac{\delta_{e,f}}{\epsilon} & \text{if } \Delta_{r-1,r} < \epsilon \end{cases} \end{aligned} \tag{A.9}$$

In case $\|\boldsymbol{q}_i - \boldsymbol{q}_{i+1}\|_{2/\mathbf{Q}} < \epsilon$ for all $i$, the objective function (A.2) becomes $C^2$ and it simplifies to:

$$O = \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j>i}}^{n} \bar{\xi}_{ij}\ \mathrm{d}\left(\boldsymbol{q}_i, \boldsymbol{q}_j\right) = \frac{n\,\epsilon}{2} + \frac{1}{2\,\epsilon} \sum_{i=1}^{n} \sum_{k=1}^{m} \mathrm{Q}_k (\hat{q}_{i,k} - \hat{q}_{i+1,k})^2 \tag{A.10}$$

the gradient to:

$$\begin{aligned} \frac{\partial O}{\partial \hat{q}_{r,e}} &= \frac{1}{\epsilon} \sum_{i=1}^{n} \sum_{k=1}^{m} \mathrm{Q}_k \left(\hat{q}_{i,k} - \hat{q}_{i+1,k}\right)\ \delta_{k,e}\ \left(\delta_{i,r} - \delta_{i+1,r}\right) \\ &= \frac{\mathrm{Q}_e}{\epsilon} \left(2\hat{q}_{r,e} - \hat{q}_{r+1,e} - \hat{q}_{r-1,e}\right) \end{aligned} \tag{A.11}$$

end the Hessian to:

$$\frac{\partial^2 O}{\partial \hat{q}_{r,e} \partial \hat{q}_{s,f}} = \frac{\mathrm{Q}_e}{\epsilon}\, \delta_{e,f}\ \left(2\delta_{r,s} - \delta_{r+1,s} - \delta_{r-1,s}\right) \tag{A.12}$$

### A.2.2 Manhattan and Maximum norm

If the Manhattan norm (2.14) or the Maximum norm (2.17) is considered, then the objective function (3.6), which is linear in the additional variables $d_{ik}$, is used. Since the constraints (2.5) are either linear, quadratic, or cubic, and the additional constraints (3.7) and (3.8) are linear, the derivation of their gradient and Hessian is rather trivial, and they are continuously differentiable. Therefore, in this case no modifications are required to fulfill the solver specifications.

## A.3 Effectiveness of integer cuts

In order to illustrate the effect of the modification described in Section 2.2.1.2, in this appendix we compare two versions of the proposed algorithm. The first one is the algorithm CouTspn described in Section 2.2.1. The second one (named Standard in Table A.1) is obtained from CouTspn by skipping the modification described in Section 2.2.1.2. 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 2.2.1.3.

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

Table A.1: Comparison between CouTspn and Standard.

| instance | CouTspn optimal value | CouTspn CPU time [s] | Standard lower bound | Standard upper bound | Standard percent gap | Standard CPU time [s] | Standard s.e. cuts | Standard 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 |

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 2.2.3. 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 A.1. 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 $\mathbb{R}^3$.

## A.4 Random sampling over an ellipsoid

In this appendix, a method is reported to generate random vectors uniformly distributed over an ellipsoid.

First, we generate random vectors $\boldsymbol{Y}$ uniformly distributed over the hypersphere $\{\boldsymbol{y} \mid \|\boldsymbol{y}\|_2 = 1,\ \boldsymbol{y} \in \mathbb{R}^m\}$ using realizations of the following random variable:

$$\boldsymbol{Y} = \left[\frac{X_1}{\|\boldsymbol{X}\|_2} + \ldots + \frac{X_m}{\|\boldsymbol{X}\|_2}\right]^T , \tag{A.13}$$

where $X_k \sim \mathrm{N}(0,1)$ and $\|\boldsymbol{X}\|_2 = \sqrt{\sum_{k=1}^m X_k^2}$. This property holds because the density function of $\boldsymbol{X}$ is spherically symmetric [89]. Then, random vectors $\boldsymbol{Z}$ uniformly distributed in the ellipsoid $\{\boldsymbol{z} \mid \boldsymbol{z}^T \mathbf{P}^{-1} \boldsymbol{z} \leq 1,\ \boldsymbol{z} \in \mathbb{R}^m\}$ can be obtained as realizations of the random variable $\boldsymbol{Z} = \mathbf{B}\,(\boldsymbol{Y}\, R^{1/m})$, where $R \sim \mathrm{U}(0,1)$, $\mathbf{P}$ is symmetric positive definite, and $\mathbf{B}$ is a unique lower triangular matrix such that $\mathbf{P} = \mathbf{B}\,\mathbf{B}^T$ [89].

Finally, if higher density is required in proximity of the ellipsoid surface rather than in the middle of the ellipsoid, we use the random variable $R = |2B-1|$, where $B \sim \mathrm{Beta}(1/2, 1/2)$. Applying the inverse transform method,

the beta distribution with parameters $\alpha = \beta = 1/2$, i.e. the arcsine law, can be obtained from the uniform distribution $U \sim \mathrm{U}(0,1)$ by using $B = 1/2 - 1/2\cos(\pi\, U)$ [40].

## A.5 Maximum volume inscribed ellipsoid

In this appendix, a method is proposed to extract the maximum volume ellipsoid inscribed in a polyhedron. Given a polyhedron $\{\boldsymbol{x} \mid \mathbf{A}\,\boldsymbol{x} + \mathbf{b} \leq 0,\ \boldsymbol{x} \in \mathbb{R}^m\}$ the maximum volume ellipsoid $\{\boldsymbol{x} \mid (\boldsymbol{x} - \boldsymbol{c})^T\, \boldsymbol{P}^{-1}\, (\boldsymbol{x} - \boldsymbol{c}) \leq 1\}$ inscribed in the polyhedron, where $\boldsymbol{B}$ is symmetric positive definite and $\boldsymbol{P} = \boldsymbol{B}\,\boldsymbol{B}$, can be found by solving the following convex optimization problem:

$$\text{maximize :} \quad \log\det \boldsymbol{B} \tag{A.14}$$

$$\text{subject to :} \quad \|\mathbf{a}_l\, \boldsymbol{B}\|_2 + \mathbf{a}_l\, \boldsymbol{c} + \mathrm{b}_l \leq 0 \qquad l = 1, \ldots, m \tag{A.15}$$

where $\mathbf{a}_l\, \boldsymbol{x} + \mathrm{b}_l \leq 0$ is the $l$-th haflspace of the polyhedron, i.e. $\mathbf{a}_l$ is the $l$-th row of $\mathbf{A}$ [18].

## A.6 Laplace distribution random sampling

In this appendix, a method is proposed to generate random number distributed according to the Laplace distribution. Given a uniformly distributed random variable $T \sim \mathrm{U}(0,1)$, if:

$$X = \begin{cases} \mu + \frac{1}{\lambda}\log(2\,T) & \text{if} \quad T \leq 1/2 \\ \\ \mu - \frac{1}{\lambda}\log(2 - 2\,T) & \text{if} \quad T > 1/2 \end{cases} \tag{A.16}$$

then by using the inverse transform method it can be shown that $X$ is a Laplace distributed random variable with mean $\mu$ and variance $2/\lambda^2$, i.e. $X \sim \mathrm{Laplace}(\mu, \lambda)$ [89].

## A.7 Power function distribution random sampling

In this appendix, a method is proposed to generate random number distributed according to the Power Function distribution. Given a uniformly distributed random variable $T \sim \mathrm{U}(0,1)$, if:

$$X = \frac{1}{\beta}\, T^{1/\alpha} \tag{A.17}$$

where $\alpha > 0, \beta > 0$, then by using the method of distribution funciton it can be shown that $X$ has probability density function $f(x) = \alpha\,\beta^{\alpha}\,x^{\alpha-1}$, i.e., $X \sim \text{Pow}(\alpha, \beta)$ with $x \in [0, 1/\beta]$ [99].

## A.8 GA parameters optimization

In this appendix, a grid search is performed to optimize the parameters to use in the hybrid random-key GA. The CETSP instance rat195 with constant radius in $\mathbb{R}^2$ is solved five times for each of the parameter set listed in Table A.2 using the Euclidean norm. The simulations are executed in a random sequence using each time a different seed for the random numbers generator. Figure A.1 shows a plot of the average and the standard deviation of the best obtained objective function values for each parameter set. Besides the parameters illustrated in Section 3.1.2, in these tests the parameter $p_L$ is also used, which corresponds to the percentage of the new *population* constituted by *chromosomes* from the previous *generation* originally excluded from the *selection* operator. Parameter set 33 is the best found in terms of average objective function value.

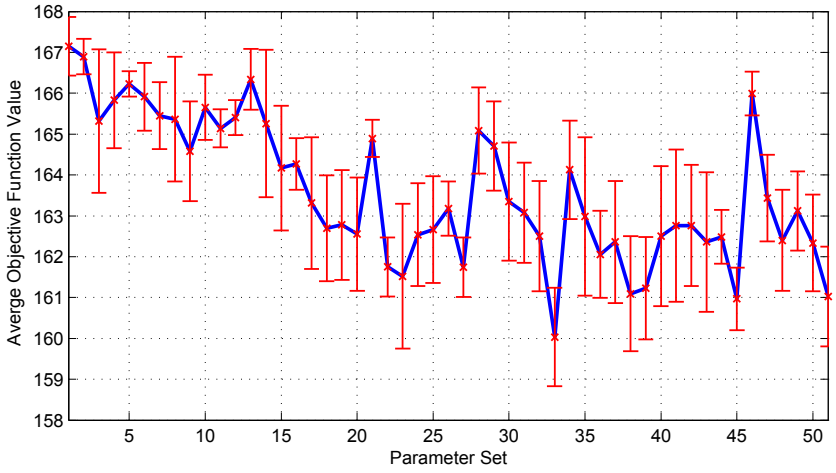
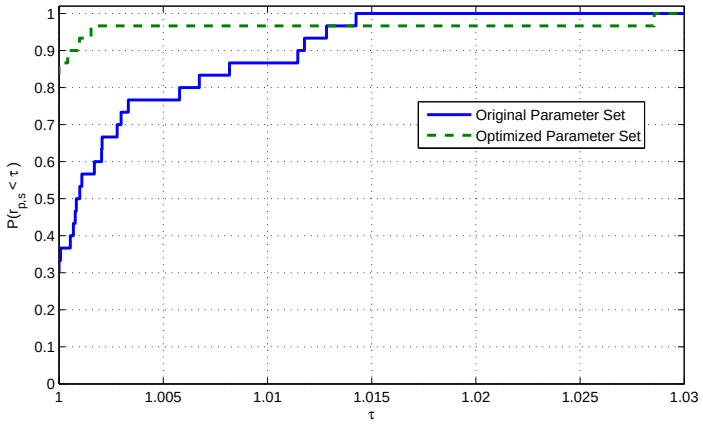


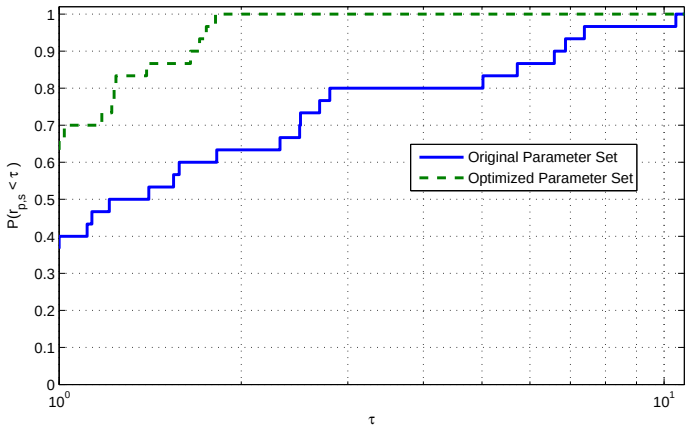

Figure A.1: Parameter Optimization for the CETSP instance rat195 in $\mathbb{R}^2$ with Euclidean norm.

Table A.2: Tested parameters sets.

| set | iter | pop. | $p_L$ | $p_S$ | $p_X$ | $p_U$ | $g_{I_{\text{MAX}}}$ | $\bar{p}_S$ | mean | std. |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 100 | 20 | 0 | 0.2 | 0.7 | 0.2 | 101 | 1 | 167.15 | 0.713 |
| 2 | 100 | 20 | 0 | 0.4 | 0.5 | 0.2 | 101 | 1 | 166.90 | 0.432 |
| 3 | 100 | 20 | 0 | 0.6 | 0.3 | 0.2 | 101 | 1 | 165.32 | 1.756 |
| 4 | 100 | 20 | 0 | 0.2 | 0.7 | 0.3 | 101 | 1 | 165.83 | 1.171 |
| 5 | 100 | 20 | 0 | 0.4 | 0.5 | 0.3 | 101 | 1 | 166.23 | 0.311 |
| 6 | 100 | 20 | 0 | 0.6 | 0.3 | 0.3 | 101 | 1 | 165.91 | 0.829 |
| 7 | 100 | 20 | 0 | 0.2 | 0.7 | 0.4 | 101 | 1 | 165.45 | 0.821 |
| 8 | 100 | 20 | 0 | 0.4 | 0.5 | 0.4 | 101 | 1 | 165.36 | 1.523 |
| 9 | 100 | 20 | 0 | 0.6 | 0.3 | 0.4 | 101 | 1 | 164.58 | 1.217 |
| 10 | 100 | 40 | 0 | 0.2 | 0.7 | 0.2 | 101 | 1 | 165.65 | 0.795 |
| 11 | 100 | 40 | 0 | 0.4 | 0.5 | 0.2 | 101 | 1 | 165.14 | 0.464 |
| 12 | 100 | 40 | 0 | 0.6 | 0.3 | 0.2 | 101 | 1 | 165.41 | 0.426 |
| 13 | 100 | 40 | 0 | 0.2 | 0.7 | 0.3 | 101 | 1 | 166.34 | 0.748 |
| 14 | 100 | 40 | 0 | 0.4 | 0.5 | 0.3 | 101 | 1 | 165.26 | 1.804 |
| 15 | 100 | 40 | 0 | 0.6 | 0.3 | 0.3 | 101 | 1 | 164.17 | 1.523 |
| 16 | 100 | 40 | 0 | 0.2 | 0.7 | 0.4 | 101 | 1 | 164.27 | 0.631 |
| 17 | 100 | 40 | 0 | 0.4 | 0.5 | 0.4 | 101 | 1 | 163.31 | 1.610 |
| 18 | 100 | 40 | 0 | 0.6 | 0.3 | 0.4 | 101 | 1 | 162.70 | 1.294 |
| 19 | 100 | 40 | 0.05 | 0.2 | 0.7 | 0.2 | 101 | 1 | 162.78 | 1.344 |
| 20 | 100 | 40 | 0.05 | 0.4 | 0.5 | 0.2 | 101 | 1 | 162.55 | 1.386 |
| 21 | 100 | 40 | 0.05 | 0.6 | 0.3 | 0.2 | 101 | 1 | 164.89 | 0.457 |
| 22 | 100 | 40 | 0.05 | 0.2 | 0.7 | 0.3 | 101 | 1 | 161.75 | 0.722 |
| 23 | 100 | 40 | 0.05 | 0.4 | 0.5 | 0.3 | 101 | 1 | 161.52 | 1.772 |
| 24 | 100 | 40 | 0.05 | 0.6 | 0.3 | 0.3 | 101 | 1 | 162.54 | 1.259 |
| 25 | 100 | 40 | 0.05 | 0.2 | 0.7 | 0.4 | 101 | 1 | 162.66 | 1.309 |
| 26 | 100 | 40 | 0.05 | 0.4 | 0.5 | 0.4 | 101 | 1 | 163.17 | 0.665 |
| 27 | 100 | 40 | 0.05 | 0.6 | 0.3 | 0.4 | 101 | 1 | 161.75 | 0.728 |
| 28 | 100 | 20 | 0 | 0.2 | 0.7 | 0.3 | 24 | 1 | 165.08 | 1.054 |
| 29 | 100 | 20 | 0 | 0.4 | 0.5 | 0.3 | 24 | 1 | 164.71 | 1.094 |
| 30 | 100 | 20 | 0 | 0.6 | 0.3 | 0.4 | 24 | 1 | 163.35 | 1.448 |
| 31 | 100 | 40 | 0 | 0.2 | 0.7 | 0.3 | 24 | 1 | 163.08 | 1.229 |
| 32 | 100 | 40 | 0 | 0.4 | 0.5 | 0.3 | 24 | 1 | 162.50 | 1.351 |
| **33** | **100** | **40** | **0** | **0.6** | **0.3** | **0.4** | **24** | **1** | **160.03** | **1.205** |
| 34 | 100 | 20 | 0 | 0.2 | 0.7 | 0.3 | 12 | 1 | 164.13 | 1.204 |
| 35 | 100 | 20 | 0 | 0.4 | 0.5 | 0.3 | 12 | 1 | 162.98 | 1.939 |
| 36 | 100 | 20 | 0 | 0.6 | 0.3 | 0.4 | 12 | 1 | 162.06 | 1.068 |
| 37 | 100 | 40 | 0 | 0.2 | 0.7 | 0.3 | 12 | 1 | 162.36 | 1.494 |
| 38 | 100 | 40 | 0 | 0.4 | 0.5 | 0.3 | 12 | 1 | 161.09 | 1.409 |
| 39 | 100 | 40 | 0 | 0.6 | 0.3 | 0.4 | 12 | 1 | 161.23 | 1.253 |
| 40 | 100 | 40 | 0 | 0.2 | 0.7 | 0.3 | 12 | 2 | 162.50 | 1.716 |
| 41 | 100 | 20 | 0 | 0.4 | 0.5 | 0.3 | 12 | 2 | 162.76 | 1.861 |
| 42 | 100 | 20 | 0 | 0.6 | 0.3 | 0.4 | 12 | 2 | 162.76 | 1.485 |
| 43 | 100 | 40 | 0 | 0.2 | 0.7 | 0.3 | 12 | 2 | 162.36 | 1.707 |
| 44 | 100 | 40 | 0 | 0.4 | 0.5 | 0.3 | 12 | 2 | 162.48 | 0.661 |
| 45 | 100 | 40 | 0 | 0.6 | 0.3 | 0.4 | 12 | 2 | 160.97 | 0.765 |
| 46 | 100 | 20 | 0 | 0.2 | 0.7 | 0.3 | 12 | 4 | 165.99 | 0.535 |
| 47 | 100 | 20 | 0 | 0.4 | 0.5 | 0.3 | 12 | 4 | 163.44 | 1.062 |
| 48 | 100 | 20 | 0 | 0.6 | 0.3 | 0.4 | 12 | 4 | 162.40 | 1.240 |
| 49 | 100 | 40 | 0 | 0.2 | 0.7 | 0.3 | 12 | 4 | 163.12 | 0.972 |
| 50 | 100 | 40 | 0 | 0.4 | 0.5 | 0.3 | 12 | 4 | 162.33 | 1.183 |
| 51 | 100 | 40 | 0 | 0.6 | 0.3 | 0.4 | 12 | 4 | 161.03 | 1.225 |




(a) Performance ratio $r_{p,s}$ based on the objective function value.




(b) Performance ratio $r_{p,s}$ based on the CPU time (a logarithmic scale is used for $\tau$).

Figure A.2: Performance profiles for the two parameter sets.

To evaluate the improvement attained with the optimized parameter set, Table A.3 reports for all the instances tested in Table 3.6 the results attained with the original values for the parameters used in Section 3.2.1. Hereafter, the statistics for tests performed the using the original parameter set are enclosed in parentheses.

The best known near *optimal tours* have been improved by an average

factor of 1.80% (1.92%) with a maximum factor of 11.26% for dsj1000 in $\mathbb{R}^3$ with Euclidean norm (8.03% for bonus1000_v in $\mathbb{R}^3$). In particular, the average improvement factors are 1.04% (1.22%) in $\mathbb{R}^2$ with Euclidean norm, 3.26% (3.27%) in $\mathbb{R}^3$ with Euclidean norm, and 1.10% (1.28%) in $\mathbb{R}^2$ with Manhattan norm.

The geometric average of the ratio of the CPU time spent by the proposed GA and the one reported in [74] is 1.33 (1.06) with a maximum ratio of 284 (47) for d493_v in $\mathbb{R}^2$ with Euclidean norm, and a minimum ratio of 0.035 for lin318_v in $\mathbb{R}^3$ with Euclidean norm (0.019 for team5_499_v in $\mathbb{R}^3$ with Euclidean norm).

Figures A.2.a and A.2.b illustrate the performance profiles derived using as performance measure the objective function value and the the CPU time, respectively [27]. It can be easily observed that the optimized parameter set outperforms the original one. In particular, the new parameter set improves the performance of the algorithm, slightly in terms of objective function value but more clearly in terms of overall CPU time.

Finally, the best *chromosome* has been improved in 62% (62%) of the overall *generations*. In particular, in case of balls in $\mathbb{R}^2$ and Euclidean norm the percentage is 46% (58%), in case of balls in $\mathbb{R}^3$ and Euclidean norm it is 76% (70%), and in case of balls in $\mathbb{R}^2$ and Manhattan norm it is 63% (58%).

Table A.3: CETSP instances in $\mathbb{R}^2$ and $\mathbb{R}^3$ with variable radii proposed by Mennell [74].

| instance | $n$ | $m$ | d$(\cdot)$ | HRKGA | | | | | CETSP | | obj. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | obj. | CPU [s] | pop. | $g$ | impr. | obj. | CPU [s] | impr. |
| kroD100_v | 100 | 2 | 2 | 141.829 | 2,425 | 50 | 110 | 24 | 141.834 | 1,008 | 0.00% |
| rat195_v | 195 | 2 | 2 | 68.224 | 598 | 50 | 27 | 13 | 68.224 | 115 | 0.00% |
| lin318_v | 318 | 2 | 2 | 2,079.81 | 100,334 | 50 | 751 | 85 | 2,080.57 | N/A | 0.04% |
| rd400_v | 400 | 2 | 2 | 1,244.21 | 27,329 | 60 | 157 | 34 | 1,252.38 | 12,795 | 0.65% |
| pcb442_v | 442 | 2 | 2 | 229.970 | 579 | 60 | 9 | 9 | 235.188 | 1,161 | 2.22% |
| d493_v | 493 | 2 | 2 | 134.450 | 22,648 | 60 | 123 | 52 | 140.120 | 46 | 4.05% |
| dsj1000_v | 1,000 | 2 | 2 | 647.930 | 152,346 | 100 | 467 | 92 | 653.128 | N/A | 0.80% |
| team1_100_v | 101 | 2 | 2 | 388.537 | 167 | 50 | 10 | 4 | 388.537 | 47 | 0.00% |
| team5_499_v | 500 | 2 | 2 | 449.776 | 8,991 | 100 | 29 | 16 | 454.327 | 2,763 | 1.00% |
| bonus1000_v | 1,001 | 2 | 2 | 970.761 | 10,564 | 200 | 12 | 12 | 987.114 | N/A | 1.66% |
| kroD100_v | 100 | 3 | 2 | 169.611 | 103 | 50 | 4 | 4 | 171.568 | 121 | 1.14% |
| rat195_v | 195 | 3 | 2 | 82.531 | 622 | 50 | 23 | 16 | 84.470 | 1945 | 2.30% |
| lin318_v | 318 | 3 | 2 | 2,157.05 | 1,216 | 50 | 13 | 11 | 2,189.43 | 34,640 | 1.48% |
| rd400_v | 400 | 3 | 2 | 3,589.21 | 4,957 | 60 | 50 | 11 | 3,592.60 | N/A | 0.09% |
| pcb442_v | 442 | 3 | 2 | 249.852 | 3,407 | 60 | 80 | 53 | 258.404 | 7,406 | 3.31% |
| d493_v | 493 | 3 | 2 | 735.820 | 1,453 | 60 | 11 | 11 | 761.065 | 25,013 | 3.32% |
| dsj1000_v | 1,000 | 3 | 2 | 1,841.18 | 1,325 | 60 | 3 | 3 | 2,074.84 | 47 | 11.26% |
| team1_100_v | 101 | 3 | 2 | 906.221 | 223 | 60 | 15 | 10 | 907.593 | 127 | 0.15% |
| team5_499_v | 500 | 3 | 2 | 821.568 | 11,691 | 60 | 97 | 50 | 840.477 | 106,692 | 2.25% |
| bonus1000_v | 1,001 | 3 | 2 | 2,493.73 | 945 | 60 | 2 | 2 | 2,689.41 | 90 | 7.28% |
| kroD100_v | 100 | 2 | 1 | 173.811 | 90 | 50 | 8 | 8 | 174.013 | 22 | 0.12% |
| rat195_v | 195 | 2 | 1 | 82.200 | 87 | 50 | 4 | 4 | 82.200 | 32 | 0.00% |
| lin318_v | 318 | 2 | 1 | 2,491.63 | 1,538 | 150 | 20 | 13 | 2,505.41 | N/A | 0.55% |
| rd400_v | 400 | 2 | 1 | 1,538.17 | 4,641 | 60 | 52 | 25 | 1,544.54 | 614 | 0.41% |
| pcb442_v | 442 | 2 | 1 | 278.041 | 2,051 | 60 | 70 | 35 | 281.254 | 169 | 1.14% |
| d493_v | 493 | 2 | 1 | 173.986 | 1,731 | 60 | 32 | 21 | 179.563 | 223 | 3.11% |
| dsj1000_v | 1,000 | 2 | 1 | 746.513 | 36,679 | 100 | 144 | 65 | 758.119 | N/A | 1.53% |
| team1_100_v | 101 | 2 | 1 | 490.422 | 235 | 50 | 25 | 12 | 494.114 | 391 | 0.75% |
| team5_499_v | 500 | 2 | 1 | 558.432 | 4,804 | 60 | 100 | 45 | 563.110 | 1,899 | 0.83% |
| bonus1000_v | 1,001 | 2 | 1 | 1,194.74 | 16,471 | 100 | 77 | 52 | 1,226.39 | 33,729 | 2.58% |

## A.9 Parameter settings for the single query planner

Tables A.4 and A.5 illustrate the performance of the single query planner with different values for the resolution for collision avoidance, $q_{res}$ and for the step subdivision, $h_{step}$. The results are averaged among 10 different execution of the algorithm using either the Weighted Maximum Norm or the Quadratic Norm and a reduced tour of 9 configurations. The minimum, the maximum, and the average values are reported in columns labeled "max.", "min.", and "avg.", respectively. The average CPU time per execution is reported in columns labeled "CPU". The objective function value of the tour calculated without collision avoidance is indicated as "direct".

Table A.4: Comparison of different parameter settings for the single query planner using the Weighted Maximum Norm.

| | | Weighted Maximum Norm [s] | | | | Cycle Time [s] | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | avg. | min. | max. | CPU [s] | avg. | min. | max. | CPU [s] |
| $h_{step}$ | $q_{res}$ | direct = 2.93 | | | | direct = 5.07 | | | |
| 5 | 0.005 | 7.15 | 5.16 | 9.84 | 10.81 | 7.34 | 6.15 | 9.68 | 20.43 |
| | 0.010 | 6.65 | 5.62 | 8.17 | 5.02 | 7.97 | 6.17 | 12.98 | 12.27 |
| | 0.020 | 6.82 | 5.31 | 8.95 | 2.74 | 7.39 | 5.70 | 11.43 | 5.74 |
| | 0.040 | 5.76 | 4.00 | 8.73 | 1.20 | 7.85 | 6.26 | 11.61 | 2.59 |
| | 0.080 | 5.86 | 3.85 | 10.30 | 0.81 | 7.33 | 6.04 | 10.00 | 1.75 |
| 10 | 0.005 | 6.15 | 4.87 | 7.45 | 13.87 | 6.91 | 6.09 | 7.74 | 29.38 |
| | 0.010 | 5.85 | 4.74 | 6.81 | 6.61 | 6.80 | 5.86 | 7.25 | 14.70 |
| | 0.020 | 5.76 | 3.72 | 7.70 | 2.87 | 6.40 | 5.91 | 7.23 | 8.11 |
| | 0.040 | 6.09 | 5.48 | 6.88 | 1.79 | 7.51 | 5.92 | 10.27 | 3.61 |
| | 0.080 | 6.07 | 5.02 | 8.09 | 1.09 | 7.06 | 5.84 | 8.62 | 2.19 |
| 20 | 0.005 | 5.69 | 4.59 | 6.56 | 22.21 | 7.50 | 6.13 | 13.61 | 55.11 |
| | 0.010 | 5.51 | 4.92 | 6.50 | 12.88 | 6.87 | 6.16 | 9.10 | 27.75 |
| | 0.020 | 5.15 | 3.83 | 6.32 | 6.97 | 6.76 | 6.04 | 8.23 | 13.89 |
| | 0.040 | 5.23 | 4.09 | 6.63 | 3.96 | 6.68 | 5.98 | 7.16 | 8.30 |
| | 0.080 | 5.56 | 4.83 | 6.42 | 2.47 | 7.05 | 5.90 | 9.20 | 4.44 |
| 40 | 0.005 | failed | 4.20 | failed | 45.08 | failed | 6.55 | failed | 92.07 |
| | 0.010 | failed | 4.55 | failed | 24.54 | failed | 6.29 | failed | 47.37 |
| | 0.020 | failed | 4.03 | failed | 12.27 | failed | 6.40 | failed | 24.46 |
| | 0.040 | failed | 4.21 | failed | 8.12 | failed | 6.30 | failed | 13.73 |
| | 0.080 | failed | 4.69 | failed | 5.00 | failed | 6.81 | failed | 7.84 |

Table A.5: Comparison of different parameter settings for the single query planner using the Quadratic Norm.

| | | Quadratic Norm [s] | | | | Cycle Time [s] | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | avg. | min. | max. | CPU [s] | avg. | min. | max. | CPU [s] |
| $h_{step}$ | $q_{res}$ | direct = 3.98 | | | | direct = 5.07 | | | |
| 5 | 0.005 | 8.04 | 6.67 | 10.48 | 10.19 | 7.51 | 6.31 | 10.72 | 23.64 |
| | 0.010 | 8.38 | 5.11 | 10.44 | 4.61 | 8.54 | 6.58 | 11.77 | 10.68 |
| | 0.020 | 8.88 | 6.54 | 13.30 | 2.54 | 7.57 | 6.49 | 9.13 | 5.49 |
| | 0.040 | 7.96 | 5.27 | 12.04 | 1.27 | 7.31 | 6.00 | 9.04 | 2.80 |
| | 0.080 | 8.94 | 5.88 | 12.22 | 0.81 | 8.11 | 6.23 | 9.00 | 1.79 |
| 10 | 0.005 | 6.88 | 5.91 | 7.73 | 11.48 | 7.21 | 6.28 | 8.40 | 33.73 |
| | 0.010 | 8.09 | 6.61 | 12.15 | 6.85 | 7.38 | 5.90 | 8.50 | 15.65 |
| | 0.020 | 7.85 | 5.61 | 9.16 | 3.31 | 6.74 | 5.66 | 8.87 | 7.42 |
| | 0.040 | 7.70 | 5.93 | 9.91 | 2.02 | 6.80 | 5.80 | 7.81 | 3.53 |
| | 0.080 | 7.61 | 6.01 | 9.04 | 0.96 | 6.69 | 6.07 | 7.62 | 2.14 |
| 20 | 0.005 | 6.97 | 5.88 | 8.72 | 25.00 | 7.30 | 6.29 | 9.57 | 53.04 |
| | 0.010 | 7.40 | 6.02 | 9.94 | 12.88 | 6.64 | 6.02 | 7.72 | 26.13 |
| | 0.020 | 7.00 | 5.76 | 8.82 | 7.00 | 6.66 | 5.77 | 7.57 | 13.35 |
| | 0.040 | 7.13 | 5.92 | 8.02 | 3.74 | 6.79 | 5.77 | 8.31 | 6.38 |
| | 0.080 | 6.91 | 5.79 | 8.96 | 2.42 | 6.57 | 6.06 | 7.06 | 3.50 |
| 40 | 0.005 | failed | 5.63 | failed | 42.66 | failed | 6.61 | failed | 87.09 |
| | 0.010 | failed | 5.47 | failed | 24.96 | failed | 5.89 | failed | 43.72 |
| | 0.020 | failed | 6.23 | failed | 14.73 | failed | 6.73 | failed | 24.40 |
| | 0.040 | failed | 5.83 | failed | 8.15 | failed | 6.31 | failed | 12.66 |
| | 0.080 | failed | 5.32 | failed | 5.45 | failed | 6.33 | failed | 8.27 |

## A.10 Manipulator forward and inverse kinematic

In this appendix, we illustrated the forward and inverse kinematic of the used six degrees of freedom Denso VS-6577G-B manipulator [88]. The forward kinematic of the manipulator, ${}^{m}R_{e} = \mathrm{FK}([q_{i,1}, \ldots, q_{i,6}]^{T})$, can be described by the following homogenous transformation:

$$ {}^{m}R_{e} = R_1 R_2 R_3 R_4 R_5 R_6 = \begin{bmatrix} \lambda_x & \mu_x & \nu_x & p_x \\ \lambda_y & \mu_y & \nu_y & p_y \\ \lambda_z & \mu_z & \nu_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.18} $$

where $\boldsymbol{p}$ is the end-effector position, and $\boldsymbol{\lambda}$, $\boldsymbol{\mu}$, and $\boldsymbol{\nu}$ are the unit vectors of the end-effector coordinate system with respect to the manipulator coordi-

nate system. In particular we have:

$$\begin{aligned}
\lambda_x &= \cos(q_1)(\cos(q_4)\cos(q_6) - \cos(q_5)\sin(q_4)\sin(q_6)) \\
&\quad + \sin(q_1)(\cos(q_6)\sin(q_2+q_3)]\sin(q_4) \\
&\quad + (\cos(q_3)\cos(q_4)\cos(q_5)\sin(q_2) + \cos(q_2)\cos(q_4)\cos(q_5)\sin(q_3) \\
&\quad + \cos(q_2+q_3)\sin(q_5))\sin(q_6)) \qquad \text{(A.19)} \\
\lambda_y &= -\sin(q_2+q_3)\sin(q_5)\sin(q_6) + \cos(q_2+q_3)(\cos(q_6)\sin(q_4) \\
&\quad + \cos(q_4)\cos(q_5)\sin(q_6)) \qquad \text{(A.20)} \\
\lambda_y &= \cos(q_6)(-\cos(q_4)\sin(q_1) + \cos(q_1)\sin(q_2+q_3)\sin(q_4)) \\
&\quad + (\cos(q_5)\sin(q_1)\sin(q_4) \\
&\quad + \cos(q_1)(\cos(q_4)\cos(q_5)\sin(q_2+q_3) + \cos(q_2+q_3)\sin(q_5)))\sin(q_6) \\
&\qquad \text{(A.21)} \\
\mu_x &= \cos(q_6)(\cos(q_5)(\cos(q_3)\cos(q_4)\sin(q_1)\sin(q_2) \\
&\quad + \cos(q_2)\cos(q_4)\sin(q_1)\sin(q_3) \\
&\quad - \cos(q_1)\sin(q_4)) + \cos(q_2+q_3)\sin(q_1)\sin(q_5)) \\
&\quad - (\cos(q_1)\cos(q_4) + \sin(q_1)\sin(q_2+q_3)\sin(q_4))\sin(q_6) \qquad \text{(A.22)} \\
\mu_y &= -\cos(q_6)\sin(q_2+q_3)\sin(q_5) \\
&\quad + \cos(q_2+q_3)(\cos(q_4)\cos(q_5)\cos(q_6) - \sin(q_4)\sin(q_6)) \qquad \text{(A.23)} \\
\mu_z &= \sin(q_1)(\cos(q_5)\cos(q_6)\sin(q_4) + \cos(q_4)\sin(q_6)) \\
&\quad + \cos(q_1)(\cos(q_4)\cos(q_5)\cos(q_6)\sin(q_2+q_3) \\
&\quad + \cos(q_2+q_3)\cos(q_6)\sin(q_5) - \sin(q_2+q_3)\sin(q_4)\sin(q_6)) \qquad \text{(A.24)} \\
\nu_x &= \cos(q_2+q_3)\cos(q_5)\sin(q_1) - (\cos(q_3)\cos(q_4)\sin(q_1)\sin(q_2) \\
&\quad + \cos(q_2)\cos(q_4)\sin(q_1)\sin(q_3) \\
&\quad - \cos(q_1)\sin(q_4))\sin(q_5) \qquad \text{(A.25)} \\
\nu_y &= -\cos(q_3)(\cos(q_5)\sin(q_2) + \cos(q_2)\cos(q_4)\sin(q_5)) \\
&\quad + \sin(q_3)(-\cos(q_2)\cos(q_5) \\
&\quad + \cos(q_4)\sin(q_2)\sin(q_5)) \qquad \text{(A.26)} \\
\nu_z &= -\cos(q_1)\cos(q_2+q_3)\cos(q_5) - (\cos(q_1)\cos(q_4)\sin(q_2+q_3) \\
&\quad + \sin(q_1)\sin(q_4))\sin(q_5) \qquad \text{(A.27)} \\
p_x &= d_5\cos(q_1)\sin(q_4)\sin(q_5) \\
&\quad + \sin(q_1)(d_1 + \cos(q_2+q_3)(d_3+d_4+d_5\cos(q_5)) + a_3\sin(q_2+q_3) \\
&\quad - d_5\cos(q_2)\cos(q_4)\sin(q_3)\sin(q_5) \\
&\quad + \sin(q_2)(a_2 - d_5\cos(q_3)\cos(q_4)\sin(q_5))) \qquad \text{(A.28)}
\end{aligned}$$

$$
\begin{aligned}
p_y = {} & a_1 - \cos(q_3)(d_3 + d_4 + d_5\cos(q_5))\sin(q_2) - a_3\sin(q_2)\sin(q_3) \\
& + d_5\cos(q_4)\sin(q_2)\sin(q_3)\sin(q_5) \\
& + \cos(q_2)(a_2 - (d_3 + d_4 + d_5\cos(q_5))\sin(q_3) \\
& + \cos(q_3)(a_3 - d_5\cos(q_4)\sin(q_5))) \qquad \text{(A.29)} \\
p_z = {} & -d_5\sin(q_1)\sin(q_4)\sin(q_5) \\
& + \cos(q_1)(d_1 + \cos(q_2+q_3)(d_3 + d_4 + d_5\cos(q_5)) + a_2\sin(q_2) \\
& + a_3\sin(q_2+q_3) - d_5\cos(q_4)\sin(q_2+q_3)\sin(q_5)) \qquad \text{(A.30)}
\end{aligned}
$$

where the following transformations were used to derive Equations (A.19) to (A.30):

$$
R_1 = \begin{bmatrix} & & & 0 \\ \text{rot}(\boldsymbol{e}_y, q_1) & & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad
R_2 = \begin{bmatrix} & & & 0 \\ \text{rot}(\boldsymbol{e}_x, q_2) & & & a_1 \\ & & & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
R_3 = \begin{bmatrix} & & & 0 \\ \text{rot}(\boldsymbol{e}_x, q_3) & & & a_2 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad
R_4 = \begin{bmatrix} & & & 0 \\ \text{rot}(\boldsymbol{e}_z, q_4) & & & a_3 \\ & & & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
R_5 = \begin{bmatrix} & & & 0 \\ \text{rot}(\boldsymbol{e}_x, q_5) & & & 0 \\ & & & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad
R_6 = \begin{bmatrix} & & & 0 \\ \text{rot}(\boldsymbol{e}_z, q_6) & & & 0 \\ & & & d_5 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
\text{rot}(\boldsymbol{e}_x, q) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(q) & -\sin(q) \\ 0 & \sin(q) & \cos(q) \end{bmatrix}
$$

$$
\text{rot}(\boldsymbol{e}_y, q) = \begin{bmatrix} \cos(q) & 0 & \sin(q) \\ 0 & 1 & 0 \\ -\sin(q) & 0 & \cos(q) \end{bmatrix}
$$

$$
\text{rot}(\boldsymbol{e}_z, q) = \begin{bmatrix} \cos(q) & -\sin(q) & 0 \\ \sin(q) & \cos(q) & 0 \\ 0 & 0 & 1 \end{bmatrix}
$$

The inverse kinematic problem, $[q_{i,1}, \ldots, q_{i,6}]^T = \text{IK}({}^m R_e)$, can be solved analytically in four main steps [69]. First elements (1,3) and (1,4) of the following matrices:

$$
R_1^{-1}\, {}^m R_e = R_2 R_3 R_4 R_5 R_6 \qquad \text{(A.31)}
$$

are equated leading to one equation in the unknown $q_1$:

$$d_5\nu_x\cos(q_1) - d_5\nu_z\sin(q_1) = p_x\cos(q_1) - p_z\sin(q_1) \tag{A.32}$$

The above equation has to two sets of solutions. In the second step, elements (1,4) and (2,4) of the following matrices:

$${}^{m}R_e\ R_6^{-1} = R_1R_2R_3R_4R_5 \tag{A.33}$$

are equated leading to two equations in the unknowns $q_2+q_3$ and $q_2$:

$$\begin{aligned} p_x - d_5\nu_x &= \sin(q_1)(d_1 + (d_3+d_4)\cos(q_2+q_3) \\ &\quad + a_2\sin(q_2) + a_3\sin(q_2+q_3) \qquad \text{(A.34)} \\ p_y - d_5\nu_y &= a_1 + a_2\cos(q_2) + a_3\cos(q_2+q_3) \\ &\quad - d_3\sin(q_2+q_3) - d_4\sin(q_2+q_3) \qquad \text{(A.35)} \end{aligned}$$

The above two equations can be simplified into one equation in $q_2+q_3$:

$$\begin{aligned} &\left(\frac{p_x - d_5n_x}{\sin(q_1)} - d_1\right)^2 + (p_y - d_5n_y - a_1)^2 - a_2^2 + a_3^2 + (d_3+d_4)^2 \\ &- 2\left[\left(\frac{p_x - d_5n_x}{\sin(q_1)} - d_1\right)(d_3+d_4) + (p_y - d_5n_y - a_1)a_3\right]\cos(q_2+q_3) \\ &+ 2\left[(p_y - d_5n_y - a_1)(d_3+d_4) - \left(\frac{p_x - d_5n_x}{\sin(q_1)} - d_1\right)a_3\right]\sin(q_2+q_3) = 0 \end{aligned} \tag{A.36}$$

that has two sets of solutions. Afterwards, the separate solutions $q_2$ and $q_3$ can be easily calculated using the original two equations.

In the third step, elements (1,3), (2,3), and (3,3) of the following matrices:

$$R_3^{-1}R_2^{-1}R_1^{-1}\ {}^{m}R_e = R_4R_5R_6 \tag{A.37}$$

are equated leading to three equations in the unknowns $q_4$ and $q_5$:

$$\begin{aligned} \sin(q_4)\sin(q_5) &= \nu_x\cos(q_1) - \nu_z\sin(q_1) \qquad \text{(A.38)} \\ -\cos(q_4)\sin(q_5) &= \nu_y\cos(q_2+q_3) + (\nu_z\cos(q_1) \\ &\quad + \nu_x\sin(q_1))\sin(q_2+q_3) \qquad \text{(A.39)} \\ \cos(q_5) &= \nu_z\cos(q_1)\cos(q_2+q_3) \\ &\quad + \nu_x\cos(q_2+q_3)\sin(q_1) - \nu_y\sin(q_2+q_3) \qquad \text{(A.40)} \end{aligned}$$

First, the third equation can be easily solved for $q_5$, which leads to two sets of solutions. Then the first two equations can be uniquely solved for $q_4$.

Finally, elements (3,1) and (3,2) of the same matrices used in the previous step are equated leading to two equations in the unknown $q_6$:

$$\begin{aligned}\sin(q_5)\sin(q_6) &= \lambda_z \cos(q_1)\cos(q_2+q_3) \\ &\quad + \lambda_x \cos(q_2+q_3)\sin(q_1) - \lambda_y \sin(q_2+q_3) \qquad \text{(A.41)} \\ \cos(q_6)\sin(q_5) &= \mu_z \cos(q_1)\cos(q_2+q_3) \\ &\quad + \mu_x \cos(q_2+q_3)\sin(q_1) - \mu_y \sin(q_2+q_3) \qquad \text{(A.42)}\end{aligned}$$

The above equations have one unique solution. In conclusion, eight sets of solution for the six joint angles are calculated in the domain $[-\pi, \pi]$ and checked against the corresponding physical joint limit.

# Bibliography


[1] Aerotech. *ALAR Series Rotary Stages*. Aerotech, Inc., 2010.

[2] M.P. Allen, G.T. Evans, D. Frenkel, and BM Mulder. Hard convex body fluids. *Advances in chemical physics*, pages 1–166, 1993.

[3] N.M. Amato, O.B. Bayazit, L.K. Dale, C. Jones, and D. Vallejo. Obprm: An obstacle-based prm for 3d workspaces. *Robotics: The Algorithmic Perspective*, pages 630–637, 1998.

[4] Erling D. Andersen. MOSEK. URL `http://www.mosek.com/`.

[5] 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.

[6] D. Applegate, W. Cook, and A. Rohe. Chained Lin-Kernighan for Large Traveling Salesman Problems. *INFORMS Journal on Computing*, 15(1):82–92, 2003.

[7] D. Applegate, R. Bixby, V. Chvatal, and W. Cook. Concorde tsp solver, 2006. URL `http://www.tsp.gatech.edu/concorde`.

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

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

[10] Autodesk. Autodesk infrastructure modeler, 2011. URL `http://usa.autodesk.com/`.

[11] J.C. Bean. Genetic Algorithms and Random Keys for Sequencing and Optimization. *ORSA Journal on Computing*, 6(2):154–160, 1994.

[12] 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.

[13] J.L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[14] J.L. Bentley. Multidimensional divide-and-conquer. *Communications of the ACM*, 23(4):214–229, 1980.

[15] D. Berenson, S.S. Srinivasa, D. Ferguson, A. Collet, and J.J. Kuffner. Manipulation planning with workspace goal regions. In *IEEE International Conference on Robotics and Automation, 2009 (ICRA'09)*, pages 618–624. IEEE, 2009.

[16] D. Berenson, S.S. Srinivasa, D. Ferguson, and J.J. Kuffner. Manipulation planning on constraint manifolds. In *IEEE International Conference on Robotics and Automation, 2009 (ICRA'09)*, pages 625–632. IEEE, 2009.

[17] 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.

[18] S.P. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge Univ Pr, 2004.

[19] P. Cheng and S.M. LaValle. Reducing metric sensitivity in randomized trajectory design. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2001*, volume 1, pages 43–48. IEEE, 2001.

[20] COIN-OR. Computational Infrastructure for Operations Research project, 2011. URL `http://www.coin-or.org/`.

[21] CPLEX. *IBM ILOG CPLEX Optimization Studio 12.3*, 2011. URL `http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/`.

[22] M. De Berg, J. Gudmundsson, M.J. Katz, C. Levcopoulos, M.H. Overmars, and A.F. van der Stappen. TSP with Neighborhoods of Varying Size. *Journal of Algorithms*, 57(1):22–36, 2005.

[23] K. Deep and M. Thakur. A new Crossover Operator for Real Coded Genetic Algorithms. *Applied Mathematics and Computation*, 188(1): 895–911, 2007.

[24] K. Deep, K.P. Singh, ML Kansal, and C. Mohan. A Real Coded Genetic Algorithm for Solving Integer and Mixed Integer Optimization Problems. *Applied Mathematics and Computation*, 212(2):505–518, 2009.

[25] M. Desrochers and G. Laporte. Improvements and Extensions to the Miller-Tucker-Zemlin Subtour Elimination Constraints. *Operations Research Letters*, 10(1):27–36, 1991.

[26] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[27] Dolan, E.D. and Moré, J.J. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.

[28] J. Dong, N. Yang, and M. Chen. Heuristic Approaches for a TSP Variant: The Automatic Meter Reading Shortest Tour Problem. In *Extending the Horizons: Advances in Computing, Optimization, and Decision Technologies*, volume 37 of *Operations Research/Computer Science Interfaces Series*, pages 145–163. Springer, 2007.

[29] A. Dumitrescu and J.S.B. Mitchell. Approximation Algorithms for TSP with Neighborhoods in the Plane. *Journal of algorithms*, 48(1): 135–159, 2003.

[30] 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.

[31] B. Erginer and E. Altug. Modeling and pd control of a quadrotor vtol vehicle. In *IEEE Intelligent Vehicles Symposium 2007*, pages 894–899. IEEE, 2007.

[32] C. Ericson. *Real-time collision detection*, volume 1. Morgan Kaufmann, 2005.

[33] FIBRO. *FIBROPLAN NC-Rotary tables*. FIBRO GmbH, 2001.

[34] J. Forrest, D. de la Nuez, and R. Lougee-Heimer. Clp user guide. *IBM Research*, 2004.

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

[36] J.H. Friedman, F. Baskett, and L.J. Shustek. An algorithm for finding nearest neighbors. *Computers, IEEE Transactions on*, 100(10):1000–1006, 1975.

[37] J.H. Friedman, J.L. Bentley, and R.A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software (TOMS)*, 3(3):209–226, 1977.

[38] I. Gentilini, F. Margot, and K. Shimada. STSPN Instances, 2011. URL `http://wpweb2.tepper.cmu.edu/fmargot/ampl.html`.

[39] I. Gentilini, F. Margot, and K. Shimada. The Traveling Salesman Problem with Neighborhoods: MINLP Solution. *Optimization Methods and Software*, 2012. Available online, DOI: 10.1080/10556788.2011.648932.

[40] P. Glasserman. *Monte Carlo methods in financial engineering*, volume 53. Springer verlag, 2004.

[41] D.E. Goldberg and K. Deb. A Comparative Analysis of Selection Schemes Used in Genetic Algorithms. In *Foundations of Genetic Algorithms*, volume 1, pages 69–93. Bloomington, IN, 1991.

[42] S. Gottschalk, M.C. Lin, and D. Manocha. Obbtree: a hierarchical structure for rapid interference detection. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 171–180. ACM, 1996.

[43] J. Gudmundsson and C. Levcopoulos. A Fast Approximation Algorithm for TSP with Neighborhoods. *Nordic Journal of Computing*, 6 (4):469–488, 1999.

[44] 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 *IEEE International Conference on Robotics and Automation, 2008 (ICRA'08)*, pages 2252–2258. Institute of Electrical and Electronics Engineers Inc., The, 2008.

[45] L.B. Gueta, R. Chiba, T. Arai, T. Ueyama, and J. Ota. Hybrid design for multiple-goal task realization of robot arm with rotating table. In *IEEE International Conference on Robotics and Automation (ICRA'09)*, pages 1279–1284. IEEE, 2009.

[46] L.B. Gueta, J. Cheng, R. Chiba, T. Arai, T. Ueyama, and J. Ota. Multiple-goal task realization utilizing redundant degrees of freedom of task and tool attachment optimization. In *IEEE International Conference on Robotics and Automation (ICRA'11)*, pages 1714–1719. IEEE, 2011.

[47] D.J. Gulczynski, J.W. Heath, and C.C. Price. The Close Enough Traveling Salesman Problem: A Discussion of Several Heuristics. *Perspectives in Operations Research*, pages 271–283, 2006.

[48] O. Günlük and J. Linderoth. Perspective relaxation of mixed integer nonlinear programs with indicator variables. *Integer Programming and Combinatorial Optimization*, pages 1–16, 2008.

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

[50] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

[51] R.L. Haupt and S.E. Haupt. *Practical genetic algorithms, Second Edition*. Wiley, 2004.

[52] K. Helsgaun. An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000.

[53] HiSystems GmbH. MikroKopter Quadro L4-ME, 2012. URL `http://mikrokopter.de/`.

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

[55] J.N. Hooker. *Integrated methods for optimization*, volume 170. Springer Verlag, 2011.

[56] D. Hsu, J.C. Latombe, and H. Kurniawati. On the probabilistic foundations of probabilistic roadmap planning. *The International Journal of Robotics Research*, 25(7):627, 2006.

[57] H. Huang, G.M. Hoffmann, S.L. Waslander, and C.J. Tomlin. Aerodynamics and control of autonomous quadrotor helicopters in aggressive maneuvering. In *IEEE International Conference on Robotics and Automation, 2009 (ICRA'09)*, pages 3277–3282. IEEE, 2009.

[58] D.B. Johnson. Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM (JACM)*, 24(1):1–13, 1977.

[59] W. Johnson. *Helicopter theory*. Dover Publications (New York), 1994.

[60] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research*, 30(7): 846–894, 2011.

[61] L.E. Kavraki, P. Svestka, J.C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[62] A.T. Klesh. *Optimal Exploration Systems*. PhD thesis, University of Michigan, 2009.

[63] J.J. Kuffner Jr and S.M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation, 2000 (ICRA'00).*, volume 2, pages 995–1001. IEEE, 2000.

[64] P. Larranaga, C.M.H. Kuijpers, R.H. Murga, I. Inza, and S. Dizdarevic. Genetic Algorithms for the Travelling Salesman Problem: a Review of Representations and Operators. *Artificial Intelligence Review*, 13(2):129–170, 1999.

[65] J.B. Lasserre. A Semidefinite Programming Approach to the Generalized Problem of Moments. *Mathematical Programming*, 112(1):65–92, 2008.

[66] S.M. LaValle and J.J. Kuffner Jr. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001.

[67] S. Lin and B.W. Kernighan. An Effective Heuristic Algorithm for the Traveling-Salesman Problem. *Operations Research*, 21(2):498–516, 1973.

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

[69] P. Mckerrow. *Introduction to robotics*. Addison-Wesley Longman Publishing Co., Inc., 1991.

[70] D. Mellinger and V. Kumar. Minimum snap trajectory generation and control for quadrotors. In *IEEE International Conference on Robotics and Automation (ICRA2011)*, pages 2520–2525. IEEE, 2011.

[71] D. Mellinger, M. Shomin, and V. Kumar. Control of quadrotors for robust perching and landing. In *Proceedings of International Powered Lift Conference,(Philadelphia, PA)*, 2010.

[72] D. Mellinger, M. Shomin, N. Michael, and V. Kumar. Cooperative grasping and transport using multiple quadrotors. In *International symposium on distributed autonomous robotic systems*, 2010.

[73] D. Mellinger, N. Michael, and V. Kumar. Trajectory generation and control for precise aggressive maneuvers with quadrotors. *The International Journal of Robotics Research*, 2012.

[74] W.K. Mennell. *Heuristics for Solving Three Routing Problems: Close-Enough Traveling Salesman Problem, Close-Enough Vehicle Routing Problem, Sequence-Dependent Team Orienteering Problem*. PhD thesis, University of Maryland, College Park, 2009.

[75] Microsoft. Remote procedure call, 2012. URL `http://msdn.microsoft.com/en-us/library/ff358900(v=prot.10)`.

[76] C.E. Miller, A.W. Tucker, and R.A. Zemlin. Integer Programming Formulation of Traveling Salesman Problems. *Journal of the ACM (JACM)*, 7(4):329, 1960.

[77] J.S.B. Mitchell. A PTAS for TSP with Neighborhoods among Fat Regions in the Plane. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 11–18. Society for Industrial and Applied Mathematics, 2007.

[78] T. Möller. A fast triangle-triangle intersection test. *Journal of graphics tools*, 2(2):25–30, 1997.

[79] AJ Orman and H.P. Williams. A Survey of Different Integer Programming Formulations of the Travelling Salesman Problem. In *Optimisation, Econometric and Financial Analysis*, volume 9 of *Advances in Computational Management Science*, pages 91–104. Springer, 2007.

[80] M. Padberg and T.Y. Sung. An Analytical Comparison of Different Formulations of the Travelling Salesman Problem. *Mathematical Programming*, 52(1):315–357, 1991.

[81] 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.

[82] E. Plaku and L.E. Kavraki. Distributed sampling-based roadmap of trees for large-scale motion planning. In *IEEE International Conference on Robotics and Automation, 2005 (ICRA'05)*, pages 3868–3873. IEEE, 2005.

[83] E. Plaku, K.E. Bekris, B.Y. Chen, A.M. Ladd, and L.E. Kavraki. Sampling-based roadmap of trees for parallel motion planning. *IEEE Transactions on Robotics*, 21(4):597–608, 2005.

[84] P. Pounds, R. Mahony, and P. Corke. Modelling and control of a quadrotor robot. In *Proceedings Australasian Conference on Robotics and Automation 2006*. Australian Robotics and Automation Association Inc., 2006.

[85] H.J. Promel and A. Steger. *The Steiner Tree Problem*. Vieweg, Braunschweig, 2002.

[86] G. Reinelt and W. Bixby. Tsplib: a library of travelling salesman and related problem instances, 1995. URL `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/`.

[87] Denso Robotics. WinCaps III. URL `http://www.denso-wave.com/`.

[88] Denso Robotics. *VS Series - Specification sheet*. Denso Wave, Inc., Japan, 2003.

[89] R.Y. Rubinstein and D.P. Kroese. *Simulation and the Monte Carlo Method*. Wiley, Hoboken, 2008.

[90] S. Safra and O. Schwartz. On the Complexity of Approximating TSP with Neighborhoods and Related Problems. *Computational Complexity*, 14(4):281–307, 2006.

[91] M. Saha, T. Roughgarden, J.C. Latombe, and G. Sánchez-Ante. Planning Tours of Robotic Arms Among Partitioned Goals. *The International Journal of Robotics Research*, 25(3):207–223, 2006.

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

[93] R. Shuttleworth, B.L. Golden, S. Smith, and E. Wasil. Advances in Meter Reading: Heuristic Solution of the Close Enough Traveling Salesman Problem over a Street Network. *The Vehicle Routing Problem: Latest Advances and New Challenges*, pages 487–501, 2008.

[94] J. Silberholz and B. Golden. The Generalized Traveling Salesman Problem: a New Genetic Algorithm Approach. *Extending the Horizons: Advances in Computing, Optimization, and Decision Technologies*, pages 165–181, 2007.

[95] L.V. Snyder and M.S. Daskin. A random-key genetic algorithm for the generalized traveling salesman problem. *European Journal of Operational Research*, 174(1):38–53, 2006.

[96] A. Tayebi and S. McGilvray. Attitude stabilization of a vtol quadrotor aircraft. *IEEE Transactions on Control Systems Technology*, 14(3): 562–571, 2006.

[97] J.I. Vásquez-Gómez, E. López-Damian, and L.E. Sucar. View planning for 3D object reconstruction. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2009 (IROS'09)*, pages 4015–4020. IEEE, 2009.

[98] 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.

[99] D.D. Wackerly, W. Mendenhall, and R.L. Scheaffer. *Mathematical statistics with applications*. Cengage Learning, 2008.

[100] H. Waki, S. Kim, M. Kojima, and M. Muramatsu. Sums of Squares and Semidefinite Program Relaxations for Polynomial Optimization Problems with Structured Sparsity. *SIAM Journal on Optimization*, 17:218, 2006.

[101] P. Wang, R. Krishnamurti, and K. Gupta. View planning problem with combined view and traveling cost. In *IEEE International Conference on Robotics and Automation, 2007 (ICRA'07)*, pages 711–716. IEEE, 2007.

[102] C. Wurll and D. Henrich. Point-to-point and multi-goal path planning for industrial robots. *Journal of Robotic Systems*, 18(8):445–461, 2001.

[103] Wurll, C. and Henrich, D. and Wörn, H. Multi-goal path planning for industrial robots. In *International Conference on Robotics and Application*, 1999.

[104] T. Yokota, M. Gen, and Y.X. Li. Genetic Algorithm for Non-Linear Mixed Integer Programming Problems and its Applications. *Computers & Industrial Engineering*, 30(4):905–917, 1996.

[105] B. Yuan, M. Orlowska, and S. Sadiq. On the Optimal Robot Routing Problem in Wireless Sensor Networks. *IEEE Transactions on Knowledge and Data Engineering*, 19(9):1252–1261, 2007.