Relation between computational complexity and integrality gap in branch-and-bound-based methods: the case of the Travelling Salesman Problem

Doctoral Dissertation submitted to the

Department of Mathematics (University of Pavia) & Faculty of Informatics (Università della Svizzera Italiana)

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

presented by

E. Vercesi

under the supervision of

S. Gualandi and L.M. Gambardella

October 2023

Abstract

In this work, we study how to generate "hard" instances for the Traveling Salesman Problem (TSP), both in the asymmetric and in the symmetric case.

To evaluate hardness from a theoretical point of view, we rely on the well-known measure of the "integrality gap" with respect to linear relaxation. The integrality gap, in an integer linear programming with an objective function that should be minimized, is the ratio between the optimal solution and the approximation obtained through linear relaxation. If the integrality gap is equal to 1, then solving the linear relaxation is enough to compute the optimal solution.

In parallel, regarding "hardness" from a practical perspective, we search for the worst-case for Concorde, the state-of-the-art TSP solver. Our goal is to find instances that foil it and to understand why certain types of instances are harder than others.

The major contributions of this thesis are the investigation of the relation between empirical hardness and integrality gap, the proposal of methodologies to generate hard instances (both from a theoretical and empirical point of view), and the study in depth of the properties of hard instances with respect to easy ones from a structural level.

The logical flow of this thesis is as follows. After revising the existing literature, we propose a new family of Metric Symmetric TSP (STSP) instances having a small integrality gap, but hard to solve for Concorde. This is the first example of hard-to-solve instances with a bounded integrality gap. By observing the proposed instances along with the ones yet present in the literature, we notice some common patterns. We hence propose a family of inequalities based on an optimality argument that leads to a speed-up in a branch-and-cut implementation using Gurobi callback.

The limit of the instances we proposed is that they are structured, hence, we propose a random instance generator of hard instances where the only constraint to the instances is to be metric. Our generator proposes instances that are challenging and also have a high integrality gap. As the input of our procedure is a vertex of the Symmetric Subtour Elimination Problem (SSEP) Polytope, we also propose a strategy for randomly sampling vertices.

We then classify the difficulty of STSP instances based on statistical features of the problem's cost matrix, such as standard deviation, skewness, and the Gini index of the cost coefficients. The fourth feature is based on a new statistic, called the tight triangle inequality index, which gives the percentage over all possible triangle inequalities in

the cost matrix that are satisfied with equality. After computing the four features for all the instances in the training set, we designed a decision tree that predicts the empirical hardness of instances in the test set with high values of the Matthew Correlation Coefficient prediction. Our decision tree achieves high performance in predicting hard-to-solve instances.

We then focus on the behavior of our "hard" instances with respect to the Dantzig-Fulkerson-Johnson formulation. Even though this model could theoretically have many subtour elimination constraints, we find that, for many benchmarking instances, only a few are needed in practice to find the best solution. We introduce a new solution routine that does not use linear relaxation but relies on the subsequent solution of 2-matching problems. Even with this approach, the hard instances are still challenging. As a by-product, this strategy provides support for a longstanding mathematical conjecture about the complexity of cycle covering without triangles.

We move to the Asymmetric Traveling Salesman Problem (ATSP). We begin with a polyhedral study and, as a consequence, we design a heuristic procedure to generate vertices of the Asymmetric Subtour Elimination Problem (ASEP) Polytope. These are used as input to a procedure that investigates the integrality gap for the ATSP. Although we explore vertices that have been never taken into account earlier, we found the same lower bounds yet present in the literature, In support of an existing conjecture about the structure that vertices must have to achieve the maximum integrality gap. A closer look at the procedure allows us to introduce small and challenging instances for the ATSP.

We conclude this thesis with open problems and some possible future directions.

Contents

Chapter 1

Introduction

In this chapter, we fix the notation, revise the background material, and propose an outline of this thesis.

1.1 The Traveling Salesman Problem

The Traveling Salesman Problem is probably the most popular problem in combinatorial optimization. This popularity arises from its ease of being enunciated, which stands in strong contrast to its computational complexity. The problem can be described as follows: a traveling salesman must visit n localities and return to the city he started from and must do this at the minimum cost. In this thesis, we will call such kind of closed paths both *tours* and *Hamiltonian cycles* (HC) and the tours that have the minimum cost will be called both *optimal tours* and *optimal HCs*. Each instance of this problem can be represented using a weighted graph $G = (V,A)$, which may be either directed or undirected. The vertices V are the cities and weighted arcs A represent the links between them, with the weight c_{ij} representing the cost of moving from one city to another. Note that, if $m := |A|$, then $\mathbf{c} \in \mathbb{R}^m$.

TSP can be modeled using Integer Linear Programming (ILP). Dantzig, Fulkerson, and Johnson (DFJ) provided one of the first formulations of the problem in their work [19]. In the following paragraph, we will focus on the formulation related to the Asymmetric case (ATSP). Then, we focus on the case $c_{ij} = c_{ji}$, namely the symmetric one.

Let S be a subset of the vertices of the graph V. Let $a \in A$ an arc. We will refer to $a = (i, j)$ when we want to specify the start and the end point of arc a.

Let us denote

$$\delta(S) = \{a = (i, j) \mid i \in S \text{ and } j \notin S\}.$$

With a slight abuse of notation, we will denote

$$\delta(i) = \delta(\{i\}).$$

Let x_{ij} be a binary variable referring to arc (i, j). The collection of variables x_{ij}, $(i, j) \in A$ are thus used to describe all the edges of the graph, and more specifically whether or not each arc is part of an optimal solution.

In particular

$$x_{ij} = \begin{cases} 1 & \text{if the tour visits city } j \text{ right after city } i \\ 0 & \text{otherwise.} \end{cases}$$

Note that we have precisely $(n-1)!$ feasible tours, as the "starting point" can be thought fixed to node 1. Hence, the DFJ formulation consists in solving the following ILP:

$$\min \sum_{(i,j)\in A} x_{ij} c_{ij} \tag{1.1}$$

$$\text{subject to} \quad \sum_{i=1}^{n} x_{ij} = 1, \qquad j = 1, \dots, n \tag{1.2}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad i = 1, \dots, n \tag{1.3}$$

$$\sum_{(i,j)\in\delta(S)} x_{ij} \geq 1, \qquad \forall S \subset V \text{ such that } 2 \leq |S| \leq n-2 \tag{1.4}$$

$$x_{ij} \in \{0, 1\} \qquad \forall a \in A. \tag{1.5}$$

Constraints (1.2)–(1.3) are called *degree constraints* and ensure that each node is crossed exactly once. Constraints (1.4) are called *subtour elimination constraints* and ensure that any feasible solution does not decompose in different subtours. Note that we can only focus on sets of cardinality $2 \leq |S| \leq n-2$: considering the empty set makes the problem infeasible and considering sets of cardinality 1 is equivalent to the in-degree constraints, while considering sets of cardinality $n-1$ is equivalent to the outdegree constraints.

1.1.1 The symmetric TSP

We will denote with Symmetric TSP (STPS) the case when $c_{ij} = c_{ji}$, $\forall (i, j) \in A$. In this case, the directed graph becomes undirected, and thus the set of arcs A becomes undirected, and we will denote it with E. In this case, the number of feasible solutions is $\frac{(n-1)!}{2}$, as the direction is irrelevant. The ILP formulation can be simplified in the symmetric case. In this case, let

$$\delta(S) = \{e = \{i, j\} \mid i \in S \text{ and } j \notin S\}.$$

With a slight abuse of notation, we will denote

$$\delta(i) = \delta(\{i\}).$$

Thus, the ILP formulation can be simplified to

$$
\begin{aligned}
&\min \sum_{\{i,j\}\in E} x_{ij}c_{ij} && && (1.6)\\
\text{subject to} \quad & \sum_{\{i,j\}\in \delta(i)} x_{ij} = 2, && j = 1,\dots,n && (1.7)\\
& \sum_{\{i,j\}\in \delta(S)} x_{ij} \geq 2, && \forall S \subset V \text{ such that } 3 \leq |S| \leq n-3 && (1.8)\\
& x_{ij} \in \{0,1\}. && \forall \{i,j\} \in E. && (1.9)
\end{aligned}
$$

In this case, we have the so-called *degree constraints* (1.7), that enforce to cross each node. The subtour elimination constraints (1.8) should now be omitted for the empty set and the full set V and the sets of cardinality 1 and $n-1$ - This makes the problem infeasible. The subtour elimination constraints for sets of cardinality 2 and $n-2$ are implied by the degree constraints. From now on, we will abbreviate the generic TSP as ATSP (namely Asymmetric TSP), while the symmetric one will be abbreviated with STSP.

1.1.2 Theoretical results on the complexity of the TSP

In this thesis, we will use some concepts from complexity theory, such as NP-Hardness, NP-completeness (In Chapter 3), and Fixed-Parameter Tractability (In Chapter 5). As a comprehensive discussion of these topics is beyond the scope of this thesis, interested readers are referred to the book of Garey and Johnson [32].

Here, we recall instead the main theoretical results about the complexity class where TSP belongs. These theorems highlight that the TSP is a hard problem *in theory*, while the empirical hardness of this problem is a central topic of this thesis.

Theorem 1 ([46]). *The TSP as a decision problem is NP-complete.*

Theorem 2 ([60]). *No polynomial time algorithm that approximates the general TSP within a constant factor exists unless P=NP.*

1.1.3 The metric case

One interesting case is achieved when the costs on the edges satisfy the *triangle inequality*.

$$c_{ij} \leq c_{ik} + c_{jk} \qquad \forall i \neq j \neq k \in V. \tag{1.10}$$

Each instance of the TSP that satisfies this family of inequalities will be called *Metric* TSP. Note that in Chapter 6, the definition of *Metric* is not given in the standard way, namely involving also the axiom of symmetry ($c_{ij} = c_{ij}\ \forall i,j \in V$) and identity, ($c_{ij} = 0 \iff i = j\ i,j \in V$). We made this choice to keep the notation as smooth as possible and distinguish between the symmetric and non-symmetric cases. The non-negativity of the costs will always be an assumption in this thesis, in every chapter. The assumption of metric on the costs - which, as already specified, only relates to the triangular inequality - is valid in Chapters 2, 3, and 6.

In terms of complexity, the following result holds:

Theorem 3 ([53])**.** *The metric TSP in NP-Hard.*

On the contrary, we do have a polynomial time algorithm that computes a tour in polynomial time within a constant factor. Among them, the most famous is the Christofides algorithm [17], which guarantees a solution within a factor of $\frac{3}{2}$. Recently such bound has been slightly improved, e.g., it has been shown equal to $3/2 - 10^{-36}$ [45].

There are two special cases of the metric TSP that are worth mentioning: the *Euclidean* TSP for which is possible to provide, for each node, the 2-dimensional (or 3-dimensional) coordinates in $\mathbb{R}^2$ (or $\mathbb{R}^3$) and then compute the cost between edges by using the L^2 norm; the *Rectilinear* TSP, that has the same spirit of the Euclidean one, but the distance is computed according to the L^1 norm.

1.2 The integrality gap

Let us denote by ATSP($\boldsymbol{c}$) the optimal value of (1.1)–(1.5) and with STSP($\boldsymbol{c}$) the optimal value of (1.6)–(1.9). Note that if we relax the integrality constraints in both problems, namely (1.5) and (1.9), by enforcing instead

$$0 \le x_{ij} \le 1 \quad \forall (i,j) \in A, \tag{1.11}$$

$$0 \le x_{ij} \le 1 \quad \forall \{i,j\} \in E. \tag{1.12}$$

Optimizing on this relaxed set of constraints will lead to the so-called Subtour Elimination Problem (SEP). Since we have moved from the discrete to the continuous, the set of feasible solutions is no longer a lattice but it is a polytope. In this thesis, we will extensively use these definitions.

Definition 1 (Asymmetric Subtour Elimination Problem (ASEP) on *n* vertices polytope)**.**

$$P^n_{\text{ASEP}} = \{x \in [0,1] \mid (1.2)\text{–}(1.4)\}.$$

Let $\mathscr{X}^n_{\text{ASEP}}$ be the set of all the vertices.

Definition 2 (Symmetric Subtour Elimination Problem (SSEP) on *n* vertices polytope)**.**

$$P^n_{\text{SSEP}} = \{x \in [0,1] \mid (1.7)\text{–}(1.8)\}.$$

Let $\mathscr{X}^n_{\text{SSEP}}$ be the set of all the vertices.

Optimizing over each of these polytopes provides a lower bound w.r.t the value of the ILP. An important value to consider when dealing with the relaxation of an ILP is the so-called *Integrality Gap* (IG). Naively speaking, this is the ratio between the optimal value of an instance of an ILP, and its linear relaxation. It is trivial to note that IG ≥ 1, and the closer this value is to one, the higher the quality of the relaxation in comparison to the optimal solution.

Let

$$\text{ASEP}(\boldsymbol{c}) := \min_{\boldsymbol{x} \in P^n_{\text{ASEP}}} \boldsymbol{c}\boldsymbol{x}.$$
$$\text{SSEP}(\boldsymbol{c}) := \min_{\boldsymbol{x} \in P^n_{\text{SSEP}}} \boldsymbol{c}\boldsymbol{x}$$

The following definitions of integrality gap will be extensively used in this thesis.

Definition 3 (Integrality gap for the ATSP on a given instance).

$$\alpha_{\text{ATSP}}(\boldsymbol{c}) := \frac{\text{ATSP}(\boldsymbol{c})}{\text{ASEP}(\boldsymbol{c})}.$$

Definition 4 (Integrality gap for the STSP on a given instance).

$$\alpha_{\text{STSP}}(\boldsymbol{c}) := \frac{\text{STSP}(\boldsymbol{c})}{\text{SSEP}(\boldsymbol{c})}.$$

Now let's focus on the metric case.

Definition 5 (Integrality gap for the ATSP on a n vertices).

$$\alpha^n_{\text{ATSP}} := \max_{\boldsymbol{c} \geq 0 \text{ is metric on } n \text{ vertices}} \alpha_{ATSP}(\boldsymbol{c}).$$

Definition 6 (Integrality gap for the STSP on a n vertices).

$$\alpha^n_{\text{STSP}} := \max_{\boldsymbol{c} \geq 0 \text{ is metric on } n \text{ vertices}} \alpha_{STSP}(\boldsymbol{c}).$$

For $n \leq 3$, we have $\alpha_{\text{ATSP}}(n) = 1$. Similarly, $n \leq 5$, we have that $\alpha^n_{\text{STSP}} = 1$. In both cases, this means that the linear programming relaxation is integer, that is, no subtour elimination constraints are actually needed for solving both STSP and ATSP.

Definition 7 (Integrality gap for the ATSP).

$$\alpha_{\text{ATSP}} := \sup_{n \geq 4} \alpha^n_{ATSP}.$$

Definition 8 (Integrality gap for the STSP).

$$\alpha_{\text{STSP}} := \sup_{n \geq 6} \alpha^n_{STSP}.$$

1.2.1 Results and Conjecture on α_{STSP}

Real-world instances of the STSP have been observed to possess low values of $\alpha_{\text{STSP}}(\boldsymbol{c})$, as we have observed by computing it on the TSPLIB instances (see Table 3.2). Note also that Johnson and McGeoch [43] are not aware of any testbed instance with an integrality gap exceeding 1.03. Wolsey [69] demonstrated that the IG for STSP is at most $\frac{3}{2}$, while Williamson conjectured in his master's thesis that for metric TSP, the integrality gap of subtour elimination relaxation is equal to $\frac{4}{3}$ [68]. Although this conjecture has only been proven for a specific class of instances ([8]), we know that $\frac{4}{3}$ is an upper bound for the integrality gap in cubic graphs ([10]). Boyd and Benoit have computed the exact integrality gap for all $n \leq 12$ in [6, 9], having verified the conjecture for all Metric TSP instances with at most 12 nodes through exhaustive enumeration. Furthermore, they have proposed a family of Metric STSP instances where the integrality gap converges to $\frac{4}{3}$. To the best of our knowledge, no instance with a higher integrality gap has been found for a given number of nodes. To conclude, we know that

$$\frac{4}{3} \leq \alpha_{\text{STSP}} \leq \frac{3}{2}.$$

1.2.2 Results and Conjecture on α_{ATSP}

A conjecture formulated by [12] stated that $\alpha_{\text{ATSP}} \leq \frac{4}{3}$ also for the ATSP. However, both [24] and [13] have independently disproved this conjecture. They both show that $\alpha_{ATSP} \geq 2$, by presenting two different families of metric ATSP with $\alpha_{ATSP}(\boldsymbol{c}) \rightarrow 2$ when $n \rightarrow \infty$. More specifically, [24] also provides specific lower bounds for $n \leq 15$. This is the first time an IG greater than $\frac{4}{3}$ is shown for $n = 9$. In terms of the upper bound, the best known bound depends on n and is equal to $2 + \frac{8\log(n)}{\log\log n}$ [4].

Thus, the state-of-art is the following:

$$\alpha_{\text{ATSP}} \geq 2 \text{ and } \forall n \in \mathbb{N}, \alpha^n_{\text{ATSP}} \leq 2 + \frac{8\log(n)}{\log\log n}.$$

1.3 Known families of instances with high integrality gap and related pratical computational complexity

In this section, we list all the known families of instances of STSP and ATSP built iteratively by varying one or more parameters. In this dissertation, the computational complexity is assessed using Concorde [1]. Concorde is a powerful optimization software designed specifically for solving the STSP. Concorde has been at the forefront of TSP research and is widely recognized as the most efficient solver for this problem. The Concorde software utilizes a branch-and-cut algorithm, which combines linear programming techniques with intelligent branching and cutting strategies. This approach allows

Concorde to explore the vast solution space of the TSP efficiently, significantly reducing the time required to find near-optimal or optimal solutions. Concorde is capable of solving TSP instances with up to thousands of cities: Its ability to handle large-scale instances efficiently has contributed to breakthroughs in optimizing supply chain management, logistics planning, and network design.

Regarding the STSP, these instances have been formulated in different articles published at different times. To provide a fair comparison of the hardness of such families of instances, we re-generate all the instances they propose and perform 10 runs with Concorde with different seeds and collect runtime and number Branch and Bound (B&B) nodes. Then, for each family, we fit a linear regression on the number of nodes and the runtime, to observe the trend at the increase of the number of nodes. Concorde has been compiled with CPLEX, a widely used mathematical optimization software package known for its efficiency in solving mixed-integer linear programming problems. This is also the experimental setting used throughout this thesis. All the computations described in this section (and also in the following chapters) have been executed on a single node of an HPC cluster running CentOS, having an Intel CPU working at 2.1 GHz. Figure 1.1 anticipates the comparison between such families, both in terms of integrality gap and runtime in seconds (we report the $\log_{10}$ value).

1.3.1 STSP

TSPLIB [56]. is a library of TSP instances from various sources and with various numbers of nodes, that is commonly used as a benchmark. We select all the instances with less than 100 nodes and run Concorde on them 10 times each, by changing the seed. We hence perform a linear regression on the $\log_{10}$ of the times, obtaining

$$t = 10^{0.011n-2.008} = 1.026^n \cdot 0.009 \tag{1.13}$$

However, the largest integrality gap for the instances in the TSPLIB is equal to 1.095 [55], which is largely less than the conjectured $\frac{4}{3} = 1.\overline{333}$. Note that Concorde can solve to optimality all the instances of the TSPLIB with $n \leq 1380$ in less than a minute.

The foundational work of [6]. In this work, the authors precisely compute α^n_{STSP} for $n \leq 10$. To achieve this, they exhaustively enumerate the vertices of P^n_{SSEP} and solve an optimization problem having as optimal value the integrality gap. As a by-product they introduce a family of STSP instances with a value of IG that increases with the number of nodes, and that asymptotically tends to $\frac{4}{3}$. Figure 1.2 displays the general element of the introduced instance family.

The top path has a edges, the middle path has b edges and the bottom path has c edges. The value of a, b, and c are computed as a function of the number of nodes as follows:

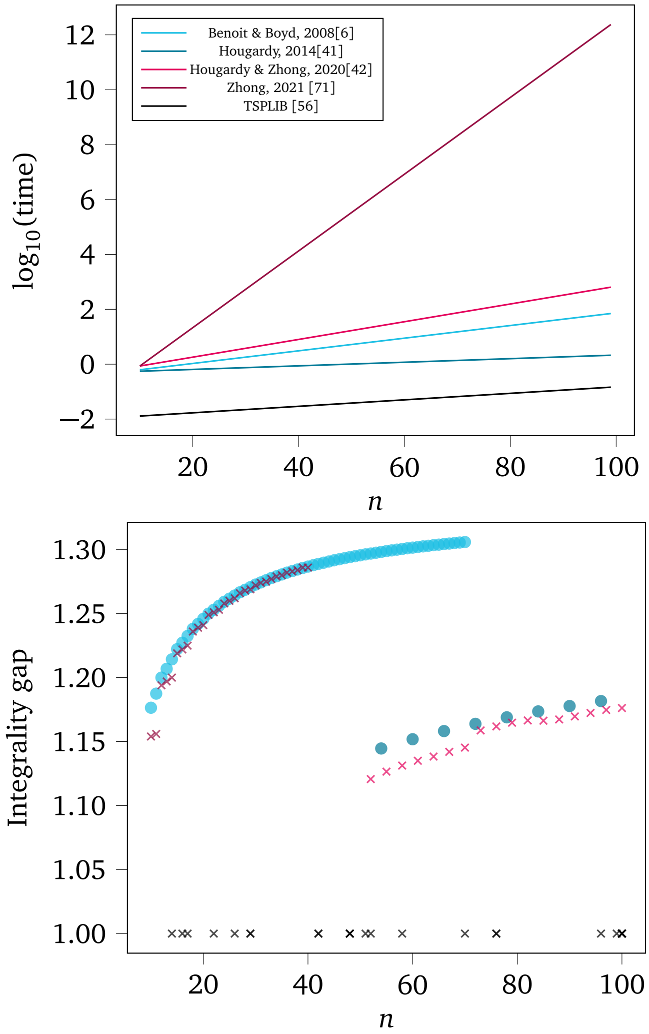


Figure 1.1. Top: Linear regression on the computational times (in seconds) of instance families with high integrality gap published in the literature. Bottom: Integrality gap of these instances.

$$\begin{cases} a = b = c = (n-3)/3 & \text{if } n \equiv 0 \mod 3 \\ a = \lfloor n/3 \rfloor, b = c = \lfloor n/3 \rfloor - 1 = a - 1 & \text{if } n \equiv 1 \mod 3 \\ a = b = \lfloor n/3 \rfloor, c = \lfloor n/3 \rfloor - 1 = a - 1 & \text{if } n \equiv 2 \mod 3 \end{cases} \tag{1.14}$$

Furthermore, they propose a stronger conjecture for the value of the IG, which de-

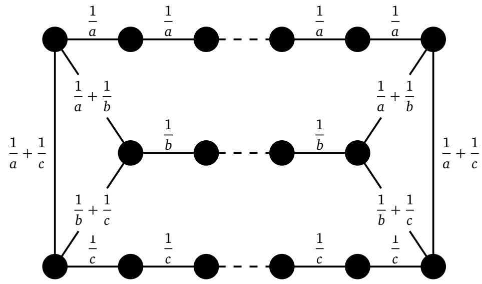


Figure 1.2. Weighted support graphs of the family introduced by [6], depending on the parameters $a, b, c,$ that are chosen according to Equation 1.14. all the missing edges have weight equal to the shortest path between the associated vertices.

pends on the number of nodes.

$$\beta_n = \begin{cases} \frac{4n+6}{3n+9} & \text{if } n \equiv 0 \bmod 3, \\ \frac{4\lfloor n/3\rfloor^2+2\lfloor n/3\rfloor-2}{3\lfloor n/3\rfloor^2+3\lfloor n/3\rfloor-2} & \text{if } n \equiv 1 \bmod 3 \\ \frac{4\lfloor n/3\rfloor^2+2\lfloor n/3\rfloor-4}{3\lfloor n/3\rfloor^2+3\lfloor n/3\rfloor-4} & \text{if } n \equiv 2 \bmod 3. \end{cases}$$

To the best of our knowledge, nobody has been able yet to disprove such conjecture.

Noteworthy, these instances are easy to solve with Concorde. We run Concorde 10 times on each instance by changing the seed and recording the runtime. Then, we perform a linear regression in the same way as the one described in the previous paragraph, obtaining

$$t = 10^{0.023n-0.434} = 1.054^n \cdot 0.368. \tag{1.15}$$

We observe a trend similar to the one of Equation (1.13), and hence, both families are easy.

Euclidean instances [41]. The paper presents a family of Euclidean TSP instances, where the nodes lie on three parallel lines on the Euclidean plane, and the distance is computed according to the L^2 norm. Each instance of the family contains equidistant points on three parallel lines. More precisely, it contains the $3k$ points with coordinates (i, jd), $i = 1, \ldots k$, $j = 1, 2, 3$, and d is the distance between the three parallel lines. The authors prove that, under a suitable choice of d as a function of n, the integrality ratio for these instances converges to $\frac{4}{3}$, disproving a long-standing conjecture suggesting that such value is $\frac{8}{7}$ (mentioned in [69]). The authors describe the construction of these instances and show that an optimal tour for these instances can be found by inserting segments of points between adjacent points on a lower line of the instance. For this type of instance, there exists a dedicated algorithm. We are in the context of convex-hull-and-line TSP, and there exists a polynomial algorithm to solve them [21]. However, even when utilizing a B&B based solver, as Concorde is, we achieve very small computational

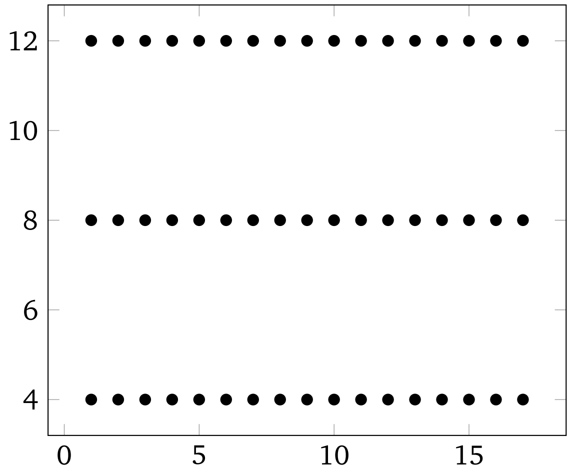


Figure 1.3. An example of the instances introduced in [41], with $k = 17$ and $d = \sqrt{k-1}$.

times. In fact, the linear regression associated with the tests we conducted indicates that the

$$t = 10^{0.006n-0.321} = 1.014^n \cdot 0.478. \tag{1.16}$$

Even in this case, the trend is close to the one exhibited by the TSPLIB. An example of this instance can be found in Figure 1.3

Tetrahedron instances [42]. The generic Tetrahedron instance is an Euclidean TSP that depends on two parameters, $n, m \in \mathbb{N}$, where n is related to the number of the internal lines, while m with the external ones. Note that, in this case, n is a parameter and not the number of nodes. The generic instance is labeled by $T_{n,m}$. The shape of a generic instance recalls the projection of a tetrahedron in the 2-dimensional space. The number of nodes is equal to $3(n+m)-2$. These instances are very challenging. The linear regression of computational time we got by performing the same steps described in the previous paragraphs is:

$$t = 10^{0.033n-0.389} = 1.079^n \cdot 0.409 \tag{1.17}$$

Here both the the base of the exponential as well as the multiplicative factor are higher, and indeed such instances are hard to solve.

The authors also proved that for a specific choice of m, we have

$$\frac{4n + 4n/\sqrt{3} - 17}{3n + 3n/\sqrt{3} - 33} \leq IG \leq \frac{4n + 4n/\sqrt{3} - 69}{3n + 3n/\sqrt{3}}.$$

Hence, it asymptotically tends to $\frac{4}{3}$.

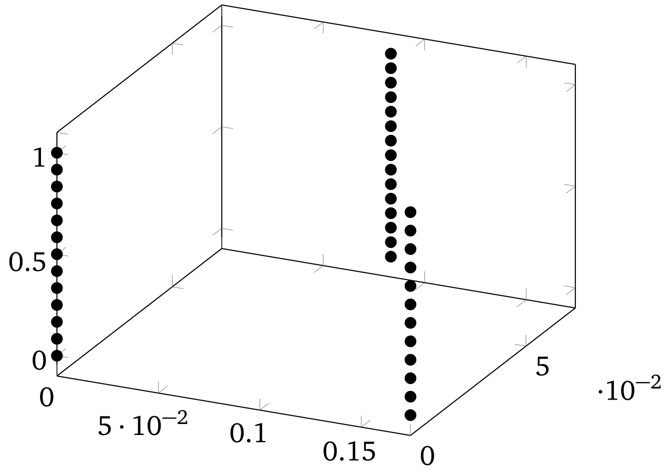


Figure 1.4. The rectilinear three-dimensional instance with 11,10,13.

Rectilinear 3-dimensional instances [71]. The generic Rectilinear 3-dimensional instance is a 3-paths Euclidean TSP where the distance between points is measured according to the L^1 norm and the points lie in $\mathbb{R}^3$. An example for $n = 40$ is reported in Figure 1.4. The definition requires three parameters, i, j, k, that are associated with the number of nodes on the three paths, which are three parallel lines in the 3-dimensional space. More specifically, each of the three indexes is equal to the number of nodes on each path minus two. For each instance, there are several ways of arranging the nodes on the three paths. However, according to the author, the number of nodes to make these instances hard to solve for Concorde depends on its class of congruence $\mod 3$:

- $j = i - 1,\ k = i + 1$ if $n \equiv 0 \mod 3$
- $j = i - 1\ k = i + 2$ if $n \equiv 1 \mod 3$
- $j = i - 1\ k = i + 3$ if $n \equiv 2 \mod 3$

Figure 1.4 represents an instance with 40 nodes.

To the best of our knowledge, these instances are the hardest available in the literature. The linear regression, in this case, is

$$t = 10^{0.139n-1.464} = 1.47^n \cdot 0.006. \tag{1.18}$$

1.3.2 Asymmetric Traveling Salesman Problem

Differently from the case of STSP, a native state-of-art solver is not available for ATSP. Previous work by [26] has shown promising results using a branch-and-cut algorithm that exploits facet-defining inequalities for ATSP. However, recent studies [27, 57] suggest that Concorde remains the most efficient method for solving ATSP. It is important to note that Concorde can only handle symmetric costs, but it is possible to transform any ATSP into an STSP by doubling the number of nodes using the method proposed in [44]. This transformation consists of the following step.

Let $\overline{\boldsymbol{C}} = (\overline{c}_{ij})$ be a matrix derived from the cost vector as follows:

$$\overline{c}_{ij} = \begin{cases} c_{ij} & i \neq j \\ -M & i = j, \end{cases} \tag{1.19}$$

where M denotes a large positive number. Consider the matrix $\boldsymbol{U}$, where all entries are set to infinity. We construct the following $\mathbb{R}^{2n\times 2n}$ matrix: We can create the following $\mathbb{R}^{2n\times 2n}$ matrix:

$$\tilde{\boldsymbol{C}} = \begin{bmatrix} \bar{\boldsymbol{C}} & \boldsymbol{U} \\ \boldsymbol{U} & \bar{\boldsymbol{C}} \end{bmatrix}.$$

Note that $\tilde{\boldsymbol{C}}$ may contain negative costs, which we do not want, as Concorde performs well only with positive costs. Therefore, we shift all costs forward by M, namely making the minimum cost equal to 0. The point at which the minimum is attained does not change. The relationship between the original values of ATSP and STSP is hence

$$\text{ATSP}(\boldsymbol{c}) = \text{STSP}(\tilde{\boldsymbol{c}}) - nM. \tag{1.20}$$

Remark 4. The relationship between the optimal values of the symmetric case and its corresponding asymmetric case holds only for integer solutions. The only relation we get between the associated integrality gap is

$$\frac{\alpha_{\text{STSP}}(\tilde{\boldsymbol{c}})}{\text{ASEP}(\boldsymbol{c})} - \frac{\alpha_{\text{ATSP}}(\boldsymbol{c})}{\text{SSEP}(\tilde{\boldsymbol{c}})} = \frac{nM}{\text{SSEP}(\tilde{\boldsymbol{c}})\text{ASEP}(\boldsymbol{c})}. \tag{1.21}$$

Investigating computationally how the integrality gap changes in the original space compared to the space of symmetric transformation is the subject of Section 6.6.4. Furthermore, we notice that when we conduct the experiments with Concorde, i.e., performing linear regression on 10 runs with 10 different seeds, the reference node count is in the original space and not in the space of duplicated nodes.

Differently from the symmetric case, to the best of our knowledge, no known parametric families of ATSP have been studied in terms of the relation between computational complexity and IG. However, some parametric families and bench-marking instances have been introduced in the literature, and the next paragraphs are devoted to describing them.

ATSPLIB [55] In the TSPLIB are also available instances for the Asymmetric TSP. We will call this benchmark of instances ATSPLIB. It consists of 19 instances, among which the biggest has 443 nodes. All of them have been solved and the optimal solution is known. The linear regression for the associated computational time is hence

$$t = 10^{0.001n-0.273} = 1.002^n \cdot 0.533. \tag{1.22}$$

Thus, such instances can be considered easy.

The instances of Charikar, Goemans, and Karloff [13]. Let $r \in \mathbb{N}$ and let G_1 be a digraph consisting of a bidirected path with $r + 2$ nodes, named s and t, that is, a sequence of loops between two nodes of length r. For each $k \geq 2$, authors build the graph recursively, by adding r copies of G_{k-1} and connect them using a cycle traversing the extreme nodes. Let these extreme nodes be labelled with $u_j, v_j,\ j \in \{1, \dots, r\}$. Starting from this sequence, the family they proposed is hence named L_k and is obtained by removing nodes s, t and reconnecting the graph with the arcs (u_r, u_1), (v_r, v_1). Authors proved that the integrality gap for every k, r is at least

$$\frac{(r-1)(2k-1)}{(r+1)k}$$

which can be made arbitrarily closed to 2 for sufficiently large values of r, k. Another contribution of this work is to outline the following. Let $\boldsymbol{c}$ an ATSP instance and let $\boldsymbol{x}$ be the optimal solution of ASEP($\boldsymbol{c}$). Assume that is *half-integer* that is $x_{ij} \in \{0, \frac{1}{2}\}$. If we multiply all the entries by 2 and define $\hat{\boldsymbol{x}} = 2\boldsymbol{x}$, we observe that $\hat{\boldsymbol{x}}$ contains a tour that is feasible for ATSP($\boldsymbol{c}$), even tough possibly non-optimal. Let $\boldsymbol{z}$ this tour. Hence, it holds

$$\alpha_{\text{ATSP}}(\boldsymbol{c}) = \frac{\text{ATSP}(\boldsymbol{c})}{\text{ASEP}(\boldsymbol{c})} \leq \frac{\boldsymbol{c}^T \boldsymbol{z}}{\text{ASEP}(\boldsymbol{c})} \leq \frac{\boldsymbol{c}^T \hat{\boldsymbol{x}}}{\text{ASEP}(\boldsymbol{c})} = \frac{2\boldsymbol{c}^T \boldsymbol{x}}{\text{ASEP}(\boldsymbol{c})} = \frac{2\text{ASEP}(\boldsymbol{c})}{\text{ASEP}(\boldsymbol{c})} = 2$$

Hence, in general, the integrality gap is at most two for any instance having a half-integer ASEP solution, and there exists a family of instances having a half-integer ASEP solution whose integrality gap tends to 2. Thus, for any instance having a half-integer ASEP solution the integrality gap is exactly two.

The instances of Elliott-Magwood [24]. In this work, the author presents two different families of ATSP with a high integrality gap. The first family has an integrality gap asymptotically tending to $\frac{3}{2}$. All the vertices building such families are half integer and they are constructed iteratively from different cycles, connected between chains of 2 cycles. The second family is remarkable for two reasons. First, the integrality gap tends to 2 faster with respect to one introduced by [13]. Second, they show for the first time a vertex with an integrality gap equal to $\frac{3}{2}$, obtained with $n = 18$.

Remark 5. Differently from the other families of STSP and the ATSPLIB, the two families of instances yet proposed have a number of nodes that grow exponentially with the increasing of the parameters from which they depend. For instance, by looking at [13], we see that the second instance in the family has 24 nodes, while the third one has 96 nodes. As our interest is in finding hard small instances, we retain the family proposed by [13] more interesting from a theoretical perspective with respect to a practical one.

1.4 Outline

Chapter 2 introduces a parametric family of Metric STSP instances that are hard to solve for Concorde. In detail, our newly introduced family is: (1) much harder than the instances available in TSPLIB and with respect to those proposed by [6]; (2) harder than the instances introduced by [42]; (3) similar to the rectilinear TSP introduced by [71]. We show that there is no clear relation between having a large IG and being hard from a computational perspective for exact methods. Lastly, by observing such instances, we show a set of inequalities based on the cost vector of the STSP that are valid for an optimal solution.

Chapter 3 introduces a complexification procedure, namely starting from an instance $\boldsymbol{c}$ of the STSP, we generate another STSP instance $\boldsymbol{c}'$. Differently from the family of instances already present in the literature, we do not enforce any structure *a priori*, but the triangle inequality. The method is based on the solution of an integer linear programming problem that takes as input a fractional solution of the Subtour Elimination Problem on an STSP instance and computes an STSP instance having an integrality gap larger than or equal to the integrality gap of the first instance. The decision variables of this program are the entries of the STSP cost matrix, and the constraints are defined by the intersection of the metric cone with an exponential number of inequalities, one for each possible STSP tour. Then, by sampling cost vectors over the metric polytope and by solving the corresponding SEP, we can generate random fractional vertices of the SEP polytope. Among the generated instances, we can select the generated TSP instance (i.e., cost vector), yielding the most extended runtime for Concorde, the state-of-the-art TSP solver. Our computational results show that our method is very effective in producing challenging instances. As a by-product, we release the Hard-TSPLIB, a library of 41 small metric TSP instances that have a large integrality gap and are challenging in terms of runtime for Concorde. This Chapter has been published [65].

Chapter 4 proposes to classify the empirical hardness of metric Travelling Salesman Problem instances by using four features that depend only on the distribution of values in the cost matrix. The first three features are the standard deviation, the skewness, and the Gini index of the cost coefficients. The fourth feature is based on a new statistic, called the *tight triangle inequality index*, which gives the percentage over all possible triangle inequalities in the cost matrix that are satisfied with equality. As a dataset, we used over 45 000 instances, both collected from the literature and randomly generated with the procedure introduced in Chapter 3, which we labeled as *easy* or *hard* by using the number of branches and bound nodes of Concorde. After computing the four features for all the instances in the training set, we designed a decision tree that predicts the empirical hardness of instances in the test set with high values of the Matthew Correlation Coefficient prediction score. This Chapter has been published [31].

In Chapter 5, we study the complexity of solving the STSP with the formulation proposed by DFJ. It is known that the DFJ formulation with zero subtour constraints (i.e., the minimum-weight 2-matching (MW2M)) is polynomial-time solvable. We show that if k is a constant, the MW2M problem with k subtour elimination constraints can be solved by reducing the problem to a sequence of MW2M problems. This also implies that the STSP with the DFJ formulation can be solved with this algorithm. As a consequence, the minimum-weight triangle-free 2-factor problem is Fixed-Parameter Tractable (FPT) with respect to the number t of triangles; the running time is 2^t poly(n). Whether this problem is polynomial or NP-hard is a longstanding open problem, first posed by [66], and remains open to this day [61, 47], making the FPT status interesting. Although the theoretical result is certainly interesting in itself, we do have some interest regarding the solving procedure of the proposed algorithm. Since this algorithm presents a paradigm that is very different from the standard branch and cut algorithms, we, therefore, question whether the complexity of an instance is an intrinsic property independent of the algorithm. Results on TSPLIB and Hard-TSPLIB demonstrate that this is indeed the case. An extended version of this chapter will be submitted within one year to a journal.

In Chapter 6, we focus on the metric case of ATSP, where the costs satisfy the triangle inequality. We study the integrality gap, providing lower bounds for instances with up to 18 nodes. The method is based on the solution of a linear programming problem that takes as input fractional vertices of P^n_{ASEP} and returns a cost vector maximizing the integrality gap on that vertex. Differently from what we did in Chapter 6, to find the vertices of P^n_{ASEP}, we combine a pivoting-based algorithm that exploits the symmetries of P^n_{ASEP}, and an inductive strategy to generate vertices of P^{n+1}_{ASEP} from P^n_{ASEP}. Then we analyze our instances from a computational point of view, observing that hard-to-solve instances are also present in this generative procedure. Even in this case, high integrality gaps are not associated with computational hardness. An extended version of this chapter will be submitted within one year to a journal.

Chapter 2

Motivation

In this chapter, we motivate our work by providing an example inspired by the literature. We introduce a family of hard-to-solve instances by choosing a different cost matrix on the same graph structure of the instances already introduced in [6]. Then we analyze the relation between costs on edges and practical computational complexity. This was the very first motivation of our work: hence, generate distances with different structures to deeply understand the cause of hardness in Metric TSP. Let $G = (V, A)$ be a weighted connected digraph on n nodes, where the cost function on the edges c_{uv} satisfy the following:

$$c_{uv} \leq c_{uw} + c_{vw} \qquad \forall (u,v),(u,w),(v,w) \in A.$$

Definition 9 (Metric completion). We call the *metric completion* of G the complete digraph $G' = (V, A')$ where each arc $(u, v) \in A' \setminus A$ has a weight equal to the shortest path between (u, v).

Note that the above definition also holds for the non-oriented graph, which is actually at the core of this chapter. As yet observed in [6] for a specific family of graphs, the following holds.

Lemma 1. *The metric completion $G' = (V, A')$ of a connected digraph $G = (V, A)$ leads to a graph whose costs satisfy the metric condition.*

As in [6] a formal proof is missing, we provide it.

Proof. We want to show that

$$c_{uv} \leq c_{uw} + c_{wv}$$

This holds if all among $(u, v), (u, w), (w, v) \in A$.

1. **Case 1:** $(u, v) \notin A$, $(u, w), (w, v) \in A$.. Thus, there exists a shortest path $u \to w_1 \to w_2 \to \ldots \to w_k \to v$ between u and v and.

$$c_{uv} := c_{uw_1} + c_{w_1 w_2} + \ldots + c_{w_k v}$$

Note also that $u \to w \to v$ is a feasible path between u and w. Hence is longer than the shortest path. Thus

$$c_{uw} + c_{wv} \geq c_{uw_1} + c_{w_1 w_2} + \ldots + c_{w_k v} = c_{uv}.$$

2. **Case 2:** (u, v) and one of the edges of the right-hand side is not in A, let's say $(u, w) \notin A$, $(w, v) \in A$.. Thus, we have two shortest paths $u \to w_1 \to w_2 \to \ldots \to w_k \to v$ and $u \to u_1 \to u_2 \to \ldots \to u_q \to w$. Note that, $u \to u_1 \to u_2 \to \ldots \to u_q \to w \to v$ is a feasible path for (u, v), hence is larger than the shortest.

$$\begin{aligned} c_{uv} &= c_{uw_1} + \ldots + c_{w_k v} \\ &\leq c_{uu_1} + \ldots + c_{u_{q-1} u_q} + c_{u_{qw}} + c_{wv} \\ &= c_{uw} + c_{wv} \end{aligned}$$

3. **Case 3:** None among $(u, v), (u, w), (w, v)$ are in A. The argument is similar: there exist three shortest path $u \to w_1 \to w_2 \to \ldots \to w_k \to v$ and $u \to u_1 \to u_2 \to \ldots \to u_q \to w$ and $w \to v_1 \to v_2 \to \ldots \to v_p \to v$, and it is possible to build a longer path between u and v, namely $u \to u_1 \to u_2 \to \ldots \to u_q \to w \to w \to v_1 \to v_2 \to \ldots \to v_p \to v$.

□

2.1 Computational Analysis of a Family of 3-Paths Metric STSP Instances

We define a family of 3-paths metric TSP, that we called P_n, that took inspiration from the one provided in [6]. Each element of the family consists of a graph structure and an arrangement of costs on the edges. Similarly to the ones in [6], our family has members that are the metric completion the graph shown in Figure 2.1.

To further clarify, we constrain each graph to have two side triangles, each of which requires 3 nodes. Thus, we have to arrange $n - 3 \cdot 2 = n - 6$ different nodes on the three paths. We denote with x the number of internal nodes on the first path, y the number of internal nodes on the second path, and z the number of internal nodes on the third path. Thus, the first path has $x + 2$ nodes; similarly, the second path has y internal nodes and $y + 2$ nodes, and the third path has z internal nodes and $z + 2$ total nodes.

Remark 6. The comparison between Figures 2.1 and 1.2 illustrates that the structure of the two graphs is the same. The focus of [6] is on the integrality gap. Hence they choose the parameters a, b, c in order to maximize such measure. On the other hand, our focus is on the empirical hardness, and hence we will choose such parameters in a different way. The remainder of this chapter is devoted to explaining how we choose such parameters and their effectiveness in leading to long runtime.

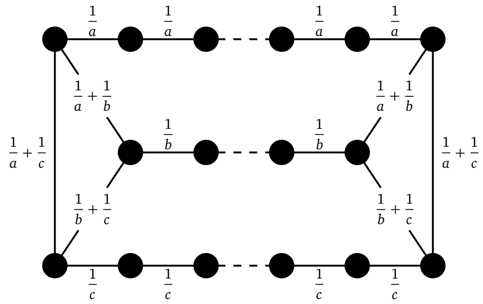


Figure 2.1. One example of an element $p_{x,y,z}$; all the missing edges have weight equal to the shortest path between the associated vertices. In this picture, $a = x+1,\ b = y+1, c = z+1$.

We label each instance of the P_n family with $p_{x,y,z}$, where $x+y+z+6=n$. Note that there are also the extra constraints that $x,y,z \in \mathbb{N}$. Thus, it is possible to represent each family P_n, with $N(n)$ points in $\mathbb{N}^3$ by identifying P_n with the following subset:

$$P_n = \{(x,y,z) \mid x+y+z+6=n \text{ and } x,y,z \in \mathbb{N}\}$$

See Figure 2.2 for a visual representation of the elements of P_n for different values of n as points in $\mathbb{N}^3$. The meaning of the colors of each point will be explained in Section 2.2.1.

We know *a priori* the exact number of elements in this family: for any pair of positive integers f and g, the number of distinct g-tuples of positive integers whose sum is f is given by

$$\binom{f+g-1}{g-1}$$

we can use this formula with $f = n-6$ and $g = 3$, thus

$$|P_n| = \binom{n-6+3-1}{3-1} = \binom{n-4}{2} = \frac{(n-4)(n-5)}{2}$$

Figure 2.1 shows a visual representation of the arrangement of costs on the edges. We assign to each edge in the graph $p_{x,y,z}$ a cost that is inversely proportional to the number of edges on that path. In the first path, we have $x+2$ nodes, consequently $x+1$ edges; we set the cost of each edge on this path equal to $\frac{1}{x+1}$. Similarly, each edge on the second path has a cost of $\frac{1}{y+1}$ and each edge on the third path has a cost equal to $\frac{1}{z+1}$. In conclusion, we connect any two extreme nodes of the graph that lie on different paths with an edge that has a weight equal to the sum of weights building the connected paths. For instance, the (two) diagonal edges that connect the first path with the second one, have weight equal to $\frac{1}{x+1}+\frac{1}{y+1}$. Similarly, the (two) straight edges that connect the first path with the third one, have weight equal to $\frac{1}{x+1}+\frac{1}{z+1}$. The metric completion makes such a graph complete.

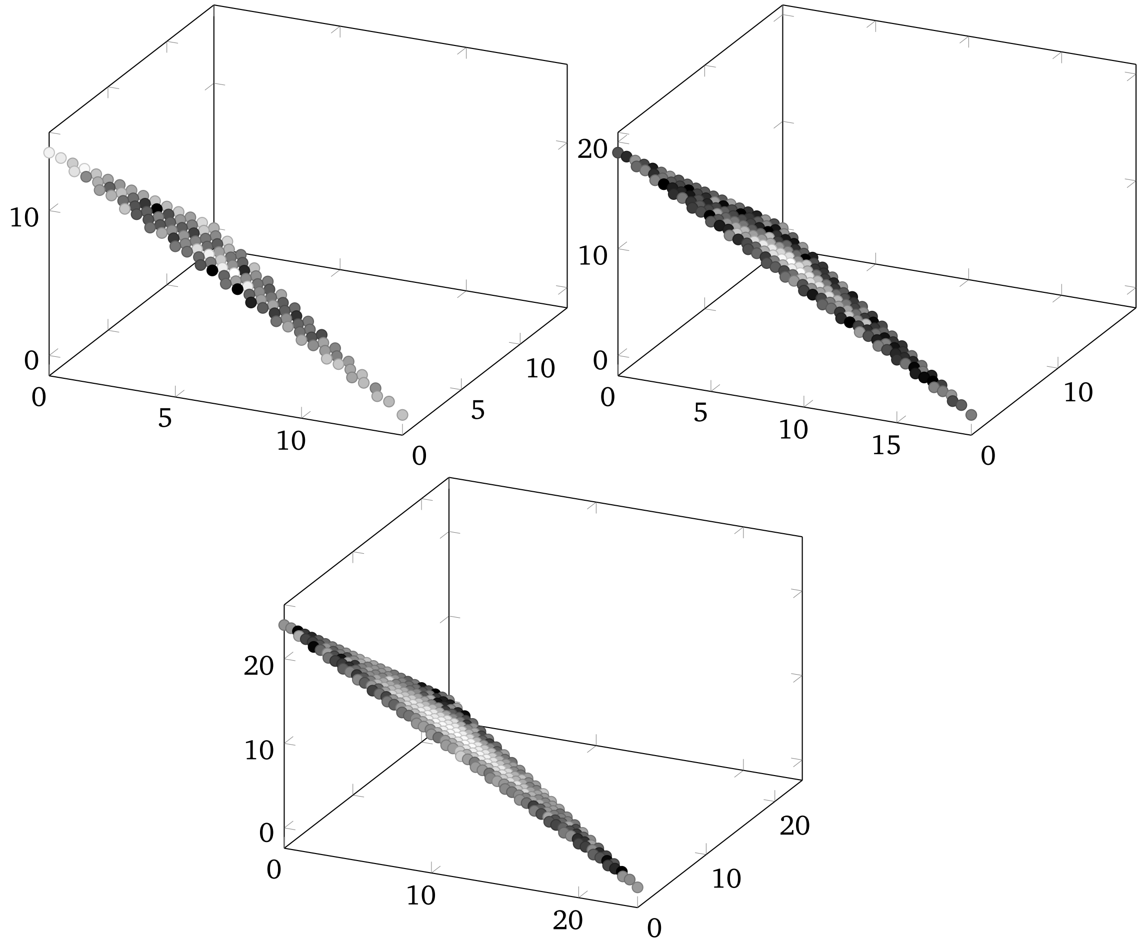


Figure 2.2. The arrangement of triplets (x, y, z) that constitute the P_n families in $\mathbb{N}^3$. The color of each dot is associated with the difficulty of the instance. The black dots are associated with the maximum computational time, while the white ones represent the minimum. Top left: $n = 20$. Top, right: $n = 25$. Bottom: $n = 30$.

Example 7. Figure 2.3 shows the $\frac{(7-4)(7-5)}{2} = 3$ instances that build the P_7 family.

Theorem 8. *For each n, the integrality gap of each element of the G_n family is at least* $\frac{10}{9} > 1$.

Proof. [6] proved that, for a graph that has a structure and an arrangement of the cost of the edges $\boldsymbol{c}$ such as the one just presented, we have

- $\text{STSP}(\boldsymbol{c}) = 4 + 2\left(\frac{1}{a} + \frac{1}{b} + \frac{1}{c}\right)$
- $\text{SSEP}(\boldsymbol{c}) = 3 + 2\left(\frac{1}{a} + \frac{1}{b} + \frac{1}{c}\right)$

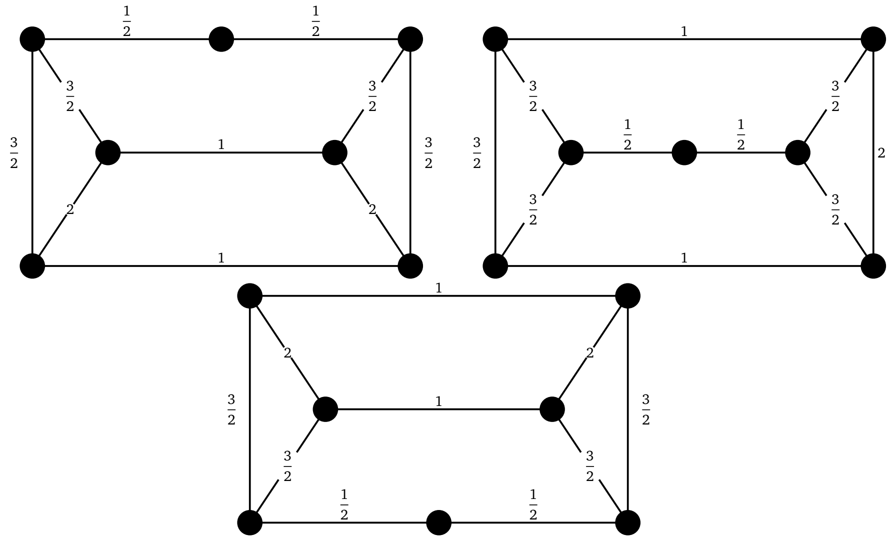


Figure 2.3. The metric completion of the graphs shown in this Figure constitutes the elements of P_7.

where a, b, c are the number of edges on the first, second and third path respectively. In our instances, $a = x + 1, b = y + 1, c = z + 1$ where $x + 2, y + 2, z + 2$ is the number of nodes for each path. As $x, y, z \geq 0$, it follows that $x + 1, y + 1, z + 1 \geq 1$ and thus $\mathrm{SSEP}(\boldsymbol{c}) \leq 3 + 2 * (1 + 1 + 1) = 9$. Furthermore, $x, y, z \leq n - 6$, thus $a, b, c \leq n - 5$. So

$$\mathrm{STSP}(\boldsymbol{c}) \geq 4 + \frac{2 \cdot 3}{n - 5}$$

As this is monotone decreasing in n, we get the maximum of this quantity in the minimum of n for which it is possible this configuration, that is $n = 6$. Thus, $\mathrm{STSP}(\boldsymbol{c}) \geq 10$, and this concludes our statement. □

Remark 9. For $n = 6$, we have $|P_6| = 1$, and thus the only chance is $x = y = z = n–6 = 0$, $a = b = c = 1$, and

$$\frac{\mathrm{STSP}(\boldsymbol{c})}{\mathrm{SSEP}(\boldsymbol{c})} = \frac{4 + 2 \cdot 3}{3 + 2 \cdot 3} = \frac{10}{9}$$

and thus, the lower bound found is tight.

Remark 10. For each n, each element of our family will have an integrality gap lower or equal to the one conjectured to be the highest possible in [6]. This has been proven by [71] (Lemma 6.2). All of our instances have the same fractional tour of the instances in [6], that is

$$x_e = \begin{cases} 1 & e \text{ is in one of the three paths} \\ \frac{1}{2} & e \text{ is one edge of the triangles} \\ 0 & \text{othrewise.} \end{cases}$$

2.2 Computational Results

This section is divided into three parts: first of all, we perform an extensive computational analysis of P_n for $6 \leq n \leq 30$; then, we provide some insights on how to obtain the potentially hardest instance from the P_n family. Lastly, we compare the computational hardness of such a family with the ones available in the literature.

2.2.1 Extensive computational analysis of P_n for $6 \leq n \leq 30$

For each $6 \leq n \leq 30$, for each instance in P_n, we run Concorde 10 times with different seeds. Then, we record the average runtime, the number of B&B nodes needed to prove the optimality of the solution, and the integrality gap.

Table 2.1 reports the collected average runtime with standard deviation, as well as the number of B&B nodes to prove optimality.

Interestingly, the hardest instance we found is not the one with the highest integrality gap (that is, as already discussed in Remark 10, the one provided by the closed formula of [6]). For instance, by looking at $n = 20$, the instance provided by [6] has a gap of 1.246, while we achieve the hardest instance, in terms of computational time, with a lower gap, namely 1.035.

Although we consider the instance that maximizes the average running time on ten independent runs, this instance is not significantly more difficult than the second more difficult instance. As an example, for $n = 20$, the hardest instance we found a running time of about 9.873 seconds, while the second hardest requires an average of 9.815 (and the differences among them are not statistically meaningful).

In each family, there are instances *easy* to solve: for instance, for $n = 20$, the easiest instance, that is $p_{5,4,4}$, requires 2 seconds to be solved to the optimum (average time on ten independent runs).

By looking at the second column of Table 2.1, it seems clear that a condition for a 3-path instance to be hard is to have a sort of "confusion" in the displacement of the nodes on the three paths. We try to visualize this concept as reported in Figure 2.2, where each colored dot represents a displacement of the nodes on the three paths, and thus an instance of P_n for three different values of n: the longer is the average runtime for an instance (x, y, z), the darker is the color.

We can see that the center of these triangles is relatively easy to solve, and this is associated with balanced instances, in the sense that the number of nodes in the internal paths varies little. On the contrary, the instances that are in the tips of the triangles are harder.

By observing Figure 2.2, the only recipe we have is to pick, from the family P_n, the triple (x, y, z) "close to the boundary" of the triangle to get an instance that results to be hard. Other than that, we do not have any suggestions.

n	(x,y,z)	$\alpha_{\text{STSP}}(\boldsymbol{c})$	runtime ± std	B&B nodes ± std
6	(0;0;0)	1.000	0.000 ± 0.000	0.000 ± 0.000
7	(0;0;1)	1.000	0.000± 0.000	0.000 ± 0.000
8	(0;0;2)	1.000	0.000 ± 0.000	0.000 ± 0.000
9	(0;0;3)	1.000	0.000± 0.000	0.000 ± 0.000
10	(2;2;0)	1.023	0.203 ± 0.115	1.000 ± 0.000
11	(3;1;1)	1.016	0.474 ± 0.264	1.000 ± 0.000
12	(4;1;1)	1.011	0.588 ± 0.370	1.800 ± 0.980
13	(1;4;2)	1.011	0.920 ± 0.258	3.200 ± 0.600
14	(1;3;4)	1.016	1.292 ± 0.449	2.800 ± 1.077
15	(4;3;2)	1.020	2.256 ± 0.618	4.600 ± 1.200
16	(4;2;4)	1.027	3.025 ± 1.271	4.200 ± 1.833
17	(4;2;5)	1.031	3.715 ± 0.799	7.000 ± 2.683
18	(6;2;4)	1.026	4.294 ± 1.181	10.400 ± 4.294
19	(6;5;2)	1.027	5.536 ± 1.088	10.400 ± 2.835
20	(1;7;6)	1.035	9.873 ± 2.248	10.000 ± 1.844
21	(8;6;1)	1.034	12.911 ± 3.050	16.200 ± 5.741
22	(1;8;7)	1.048	14.833 ± 5.328	16.600 ± 4.883
23	(12;1;4)	1.037	19.167 ± 7.015	33.400 ± 12.027
24	(2;4;12)	1.063	24.729 ± 5.545	46.000 ± 12.369
25	(16;2;1)	1.030	27.466 ± 7.190	62.800 ± 22.207
26	(1;15;4)	1.039	32.939 ± 4.563	65.000 ± 17.088
27	(0;2;19)	1.025	47.212 ± 12.220	107.800 ± 34.954
28	(16;4;2)	1.052	59.782 ± 20.667	101.800 ± 37.844
29	(2;18;3)	1.038	90.167 ± 26.047	186.800 ± 65.204
30	(0;22;2)	1.027	82.208 ± 29.203	259.200 ± 113.301

Table 2.1. Computational times with Concorde for elements of the P_n family that show to be the most difficult among all: Configuration, integrality gap, number of nodes, runtime (average over 10 runs with 10 different seeds), number of nodes in B&B.

2.2.2 A new parametric family of Metric STSP

From our experimental observation, we observe that, for each $n \geq 13$, a configuration of type $(0, 2, n-8)$ always leads to hard-to-solve instance. We call the instances of the P_n family with this configuration P_n^*. We fit a linear regression on the runtimes P_n^*, resulting in

$$t = 10^{0.130n-2.009} \tag{2.1}$$

Figure 2.4 shows the linear regression on the time of the P_n families against all the others presented in Section 1.3.1. Note that the family we propose is the second hardest

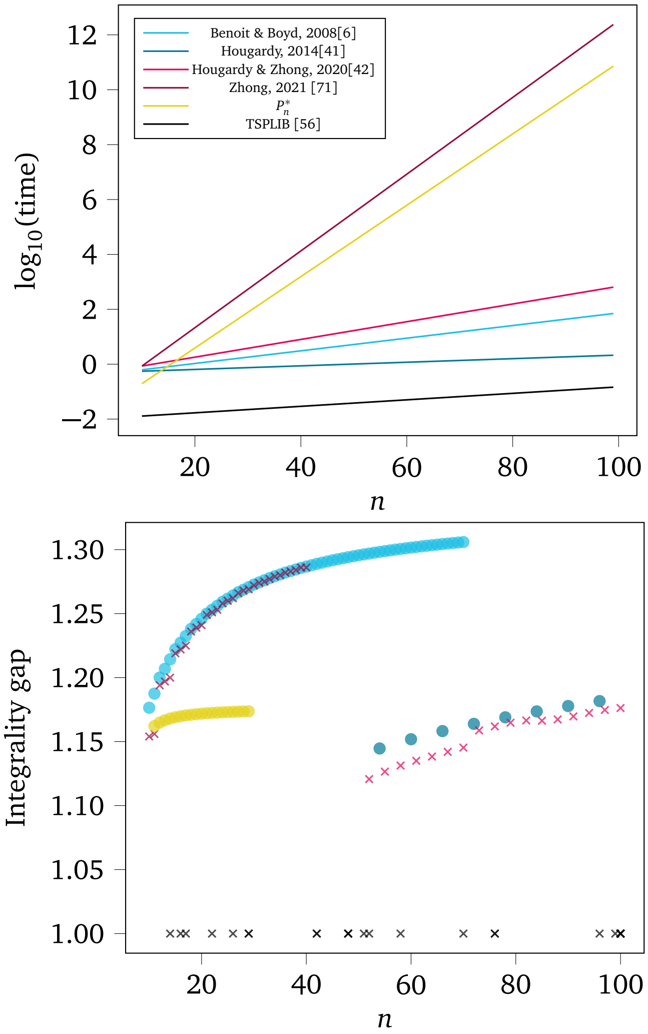


Figure 2.4. Top: Linear regression on the computational times (in seconds) of instance families with high integrality gap published in the literature. Bottom: Integrality gap of these instances. We observe that a high integrality gap is not associated with computational complexity.

family available in the literature.

On another hand, Figure 2.4 shows the trend of the integrality gap. We note that, on the contrary one may expect, that the instances we propose have a low integrality gap. Furthermore, The family of instances proposed in [6] have the highest integrality gap, even though the runtime is among the smallest.

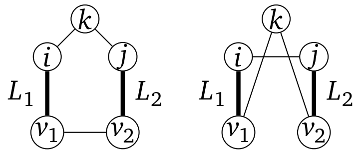


Figure 2.5. Two different valid solutions that differ in the way vertices i, j, k, v_1, v_2 are connected. Thick lines represent a path of nodes.

2.3 A family of optimality-based valid inequalities providing speed up in a naive implementation.

As we are considering Metric STSP on n nodes, we know that equation (1.10) holds. We can imagine that for some i, j, k such inequality is actually binding, namely

$$c_{ij} = c_{ik} + c_{jk}.$$

Note that for any triple of vertices (i, j, k) such that c_{ij}, c_{ik}, c_{jk} are non-zeros, at most one of the three corresponding triangle inequalities can be tight. To prove this, assume that we have both $c_{ij} = c_{ik} + c_{jk}$ and $c_{jk} = c_{ik} + c_{ij}$. Then

$$c_{ij} = c_{ik} + c_{jk} = c_{ik} + c_{ik} + c_{ij},$$

that would imply $c_{ij} = 0$. However, this is infeasible. Figure 2.5 shows two different valid solutions for an STSP on n nodes, that differ in the way vertices i, j, k, v_1, v_2 are connected. Thick lines represent a path of nodes, namely L_1 and L_2 are a sequence of nodes common among the two solutions. To provide further clarification, if there are n nodes in the instance, the tours depicted in Figure 2.5 involve all of these nodes. Specifically, five nodes, namely i, j, k, v_1, v_2, are explicitly mentioned, while L_1 and L_2 represent paths that collectively include $n-5$ nodes.

What distinguishes these configurations is the manner in which nodes and paths are interconnected. In the left graph, the edge $\{i, j\}$ is absent, whereas the edge $\{v_1, v_2\}$ is present.

The question then is: which of these two solutions will not be optimal, knowing that some triangle inequalities involving these nodes are binding and others are not?

This idea can be formalized in the following theorem.

Theorem 11. *Let $G = (V, E)$ be a complete weighted graph on n nodes leading to a STSP instance. Let $i, j, k, v_1, v_2 \in V$ be nodes such that*

$$c_{ij} = c_{jk} + c_{kj} \text{ and } c_{v_1 v_2} < c_{kv_1} + c_{kv_2}.$$

let $\mathbf{x}^1$ a tour represented on the left of Figure 2.5, namely having $x_{ik} = x_{jk} = x_{v_1 v_2} = 1$ and $\mathbf{x}^2$ the tour represented on the right of Figure 2.5, namely have $x_{ij} = x_{v_1 k} = x_{kv_2} = 1$. Then

$$\mathbf{c}^T \mathbf{x}^1 < \mathbf{c}^T \mathbf{x}^2.$$

Proof. Let $c(L_i)$ the cost of path L_i. Then

$$\begin{aligned}
\boldsymbol{c}^T \boldsymbol{x}^1 &= c_{ik} + c_{jk} + c(L_1) + c(L_2) + c_{v_1 v_2} \\
&= c_{ij} + c(L_1) + c(L_2) + c_{v_1 v_2} \\
&< c_{ij} + c(L_1) + c(L_2) + c_{v_1 k} + c_{v_2 k} \\
&= \boldsymbol{c}^T \boldsymbol{x}^2.
\end{aligned}$$

Where the first equality holds by definition, the second equality holds thanks to the binding triangle inequality on i, j, k, and the last inequality holds thanks to the non-binding triangle inequality $c_{v_1 v_2} < c_{k v_1} + c_{k v_2}$. □

The statement detailed above gives us insights on how to cut off non-optimal tours. More specifically if $x_{ij} = 1$ we do not want the tour to traverse the edges $\{v1, k\}$ and $\{v2, k\}$. If this were the case, we would already be able to construct a tour with a strictly lower cost, and therefore, we would be examining a redundant solution.

This implication can be expressed with the following logical constraints:

$$\begin{aligned}
x_{ij} &\leq (1 - x_{v_1 k}) \\
x_{ij} &\leq (1 - x_{v_2 k}).
\end{aligned}$$

At this point, it becomes interesting to evaluate the impact of adding these constraints. To do so, we compared different solution methodologies for the STSP. First of all, we note that adding these cuts will remove suboptimal solutions, and hence we can add as many as we want - potentially all of them. However, is not clear the impact they may have. Our conjecture, which is confirmed above, is that adding all of them will slow down the optimization procedure, increasing the size of the problem.

From a computational perspective, we first use Gurobi [38] and the callback technique to iteratively add subtours elimination constraints.

Then, we add all quintuples of nodes (i, j, k, v_1, v_2) that satisfy the conditions described in Theorem 11 and add the associated cuts. Since these cuts do not compromise the feasibility region, we can add as many as we want. In particular, among the valid cuts, we randomly add 1 of them and then different percentages: 10%, 30%, 50%, and 80%. Lastly, we add all of them.

Our goal is to quantify the solver's computational time and the number of times the callbacks are invoked. Table 2.2 reports these results for n = 10, 20, 25, 30, 25 on the elements of the P_n^* family.

We observe that the added cuts have a strong impact on reaching the optimal solution. We also notice that adding these cuts consistently reduces the number of callback invocations required by Gurobi to reach optimality, and thus reduces the number of subtour elimination constraints needed. Furthermore, we observe that the ideal number of

n	Percentage of cuts added	Number of subt. elim. added	Gurobi runtime	n	Percentage of cuts added	Number of subt. elim. added	Gurobi runtime
10	0 %	352	0.144	25	0 %	457999	38.837
	10%	**311**	**0.018**		10%	**668**	**1.521**
	30%	398	0.019		30%	920	2.918
	50%	399	0.023		50%	1135	4.184
	80%	328	0.029		80%	1365	5.718
	100%	402	0.034		100%	1485	4.380
20	0 %	26587	1.137	30	0 %	29892296	2010.549
	10%	**421**	**0.305**		10%	**1114**	**4.584**
	30%	480	0.599		30%	1802	10.039
	50%	536	0.900		50%	2192	13.292
	80%	616	1.282		80%	2790	18.509
	100%	643	1.196		100%	3137	16.068

Table 2.2. Solving instances of P_n^*, using Gurobi callback, adding different percentages of cuts to the model. We record the amount of cuts we add (either the number or the percentage) and the number of times the Gurobi callback function is called.

these cuts to add appears to be 10% of the total. Note that this kind of cut has an impact only when the original computational time is long. Otherwise, they only carry a new layer of complexity when they are applied as they add more nonimpactful constraints. Table 2.3 shows the impact of applying the same procedure to instances of the TSPLIB. In this case, we observe that they are not beneficial. Lastly, Table 2.4 shows the impact of applying the same procedure to the instances introduced by [71], namely the only instances harder than the one we propose. In this case, we observe two phenomena. First of all an implementation using only Gurobi results more effective in terms of computational time with respect to Concorde - Compare the computational times described in Table 2.4 and the linear regression presented in (1.18) ; Second, we note the same pattern as observed for the P_n^* instances, namely that the cuts provide a speed up and reduce the number of callbacks needed. Even in this case, the suitable amount of cuts to add results in 10%.

2.4 Conclusion

In this chapter, we examined the main issues related to the difficulty of the STSP. We create a family of instances by resembling the procedure described in [6] and changing the costs on the edges. We observe that the hardest instances are hence associated with low costs on some edges and higher costs on others. Contrary to what one might expect, there does not seem to be a clear correlation between a high IG and the compu-

Instance name	Percentage of cuts added	Number of subt. elim. added	Gurobi runtime	Instance name	Number of cuts added	Percentage of subt. elim. added	Gurobi runtime
burma14	0 %	**115**	**0.003**	ulysses16	0	**134**	**0.005**
	10%	283	0.004		10%	193	0.005
	30%	282	0.004		30%	193	0.006
	50%	282	0.005		50%	193	0.006
	80%	282	0.005		80%	193	0.007
	100%	282	0.006		100 %	193	0.007
gr17	0	**143**	**0.006**	ulysses22	0	**154**	**0.013**
	10%	276	0.009		10%	290	0.019
	30%	276	0.009		30%	290	0.022
	50%	276	0.010		50%	295	0.024
	80%	276	0.011		80%	300	0.028
	100 %	279	0.012		100%	300	0.029
fri26	0	**131**	**0.009**	bayg29	0	**285**	**0.029**
	10%	339	0.085		10%	325	0.065
	30%	359	0.120		30%	340	0.084
	50%	383	0.167		50%	353	0.109
	80%	435	0.268		80%	375	0.167
	100%	445	0.262		100%	393	0.165
bays29	0	**424**	**0.017**				
	10%	773	0.290				
	30%	1029	0.523				
	50%	997	0.788				
	80%	698	1.281				
	100%	1.115					

Table 2.3. Optimizing instances of TSPLIB with less than 30 nodes using Gurobi callback, adding also different percentages of cuts to the model.

tational complexity of an instance. In particular, in this chapter, we proposed a family of instances that has a low integrality gap, despite being the second most difficult family to solve among the ones available in the literature. Subsequently, we investigated the properties of the instances we proposed, noting that they are highly structured. Specifically, it often occurred that instances had many binding triangular inequalities, and for this reason, we introduced an argument based on optimality that does not act on the feasibility region but removes suboptimal solutions. We implemented it in a naive implementation of an STSP solver, namely the ILP implementation proposed by DFJ, using Gurobi callbacks to separate the subtour elimination constraints. We then observed that this routine significantly decreases the computational time for instances where a lot of binding triangle inequalities are available, namely in the case of the P_n^* instances and the Rectilinear 3-dimensional instances introduced by [71]. On the contrary, it has no impact on other cost configurations, for instance when attacking the TSPLIB. From these preliminary studies, we draw two conclusions: First of all, the mathematical properties of the cost vectors may suggest the best strategy to solve a STSP instance. Some properties will be discussed and revised in Chapter 4. Secondly, we have observed that difficult instances in the literature are scarce and highly structured, namely there exist one or

n	Percentage of cuts added	Number of subt. elim. added	Gurobi runtime	n	Percentage of cuts added	Number of subt. elim. added	Gurobi runtime
	0	410	0.191		0	96280	7.816
	10%	387	0.015		10%	**1368**	**1.066**
	30%	256	0.013		30%	1401	1.949
10	50%	256	0.014	25	50%	1715	2.784
	80%	257	0.018		80%	2074	3.883
	100%	257	0.019		100%	2022	3.281
	0	12737	1.095		0	1135342	51.127
	10%	**693**	**0.400**		10%	**2992**	**3.494**
	30%	704	0.525		30%	3700	7.210
20	50%	767	0.682	30	50%	3996	9.192
	80%	826	0.936		80%	5047	12.632
	100%	904	0.919		100%	4762	11.074

Table 2.4. Solving instances of the of the family introduced in [71] using Gurobi callback, adding also a different percentage of cuts to the model.

two parameters determining the cost on the edges and the number of nodes. Even the P_n^* family obeys this rule. For this reason, we retain that the first step is to generate challenging instances for the STSP (and the TSP more in general) in order to better understand its properties, and this will be the purpose of the next chapter.

Chapter 3

On the generation of Metric STSP instances with a large integrality gap by branch-and-cut

Although the family presented in the previous section presents instances that are actually hard to solve, it is still built of highly structured instances with a predefined cost displacement. More specifically, the P_n family we have introduced has an optimal tour that can be computed in constant time, as the optimum is known. The same holds for the instances present in the literature. The convex-hull-and-line TSP instances mentioned in [41] can be effectively solved in polynomial time using the algorithm presented by Deineko and coauthors. [21]. The problems described in [42] can be resolved using the algorithm outlined in [59], which operates within polynomial time. As demonstrated by the author in [71], the instances discussed there can be solved in linear time. Hence, in the literature, there is a lack of hard-to-solve instances with no structure. We present an integer programming problem, referred to as the "Integer Heuristic-OPT (IH-OPT)" problem, which yields a Metric TSP $\boldsymbol{c}$ having a large value of $\alpha(\boldsymbol{c})$. In our problem, the decision variables are the elements of the symmetric cost matrix $\boldsymbol{C}$, while the constraints are derived from the intersection of the metric cone [48] with an exponential number of inequalities, namely one inequality for each STSP tour. Our problem takes as input a vertex of P^n_{SSEP} and hence returns a metric cost vector $\boldsymbol{c}$. Consider the following pipeline, also represented in the flow chart of Figure 3.1: we pick any STSP instance $\boldsymbol{c}$ and we solve the SSEP problem on it. If we do it with the simplex method, the optimal solution is attained at a vertex of P^n_{SSEP} Then, we feed IH-OPT with such vertex, obtaining an optimal solution $\boldsymbol{c}'$. We have computational evidence that the instance associated with $\boldsymbol{c}'$ is harder to solve than the one associated with $\boldsymbol{c}$. Hence, we retain appropriate to call this procedure a "complexification" procedure: such procedure is described in Figure 3.1 and will be better explained through the chapter.

As a by-product of this chapter, we introduce the Hard-TSPLIB, which comprises 41

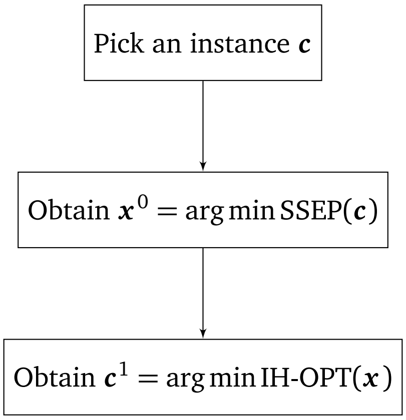


Figure 3.1. Flow chart of the so-called "complexifixation" procedure

small TSP instances (i.e., with $n \leq 76$) that prove to be highly challenging for Concorde in terms of both runtime and the number of branch-and-bound nodes.

Note that this chapter is widely based on [65], published on Mathematical Programming Computation.

3.1 Background material

Let us start by recalling that

$$\alpha_{STSP}^n := \max_{\boldsymbol{c}\geq 0 \text{ is metric on } n \text{ vertices}} \alpha_{STSP}(\boldsymbol{c})$$

We can thus write

$$\alpha_{\mathrm{STSP}}^n = \max_{\boldsymbol{c}\geq 0 \text{ is metric}} \alpha(\boldsymbol{c})$$

Following the idea presented in [6], we can divide the cost vector $\boldsymbol{c}$ by the optimum tour value $\mathrm{STSP}(\boldsymbol{c})$, obtaining a new cost vector $\boldsymbol{c}'$ that still satisfies the triangle inequalities, and which leaves $\alpha_{\mathrm{STSP}}(\boldsymbol{c})$ unchanged. Hence, we can restrict ourselves to metric cost vectors $\boldsymbol{c}$ such that $\mathrm{STSP}(\boldsymbol{c}) = 1$, and we can transform the maximization problem into a minimization:

$$\frac{1}{\alpha_{\mathrm{STSP}}^n} = \min_{\substack{\boldsymbol{c}\geq 0 \text{ is metric,} \\ \mathrm{STSP}(\boldsymbol{c})=1}} \mathrm{SSEP}(\boldsymbol{c}). \tag{3.1}$$

Problem (3.1) can be formulated as a mixed integer quadratic problem, where the decision variables are both the cost vector $\boldsymbol{c}$ and the incidence vector $\boldsymbol{x}$ of vertices of P_{SSEP}^n. Unfortunately, despite the recent improvements in the implementation of commercial optimization solvers, the quadratic model is intractable even for small values of n. In [6], the authors propose a clever idea for bypassing the quadratic model using the vertex representation of P^n_{SSEP}. That is, they represent P^n_{SSEP} as the convex combinations of its vertices $\{\bar{\boldsymbol{x}}^{(1)}, \dots, \bar{\boldsymbol{x}}^{(t)}\}$, which are finitely, although exponentially, many. Then, for each vertex $\bar{\boldsymbol{x}}^{(h)}$ of P^n_{SSEP}, they define an LP problem, called OPT($\bar{\boldsymbol{x}}^{(h)}$), having an exponential number of variables and constraints. Theoretical results in [6, 9] guarantee that only a subset of vertices is necessary to compute the integrality gap. By solving the OPT($\bar{\boldsymbol{x}}^{(h)}$) subproblem on each vertex of the previously mentioned subset, they were able to compute α^n_{STSP} for $n \leq 10$ in [6], and for $n \leq 12$ in [9].

Given the complete undirected graph on n nodes $K_n = (V, E)$, we introduce one vector $\bar{\boldsymbol{z}} \in \mathbb{R}^{|E|}$ for each π permutation of nodes, such that

$$\bar{z}_{ij} = \begin{cases} 1 & \text{if } \pi(i) = j \text{ or } \pi(j) = i, \\ 0 & \text{otherwise.} \end{cases}$$

Namely, we have a vector for each integer solution of the STSP, namely for each tour.

Let $\mathcal{T}_n$ be the collection of the incidence vectors $\bar{\boldsymbol{z}} \in \mathbb{R}^{|E|}$ of all the possible tours of K_n. Given a vertex $\bar{\boldsymbol{x}}^{(h)}$ of P^n_{SSEP}, the OPT($\bar{\boldsymbol{x}}^{(h)}$) problem is defined in [6] as

$$\text{OPT}(\bar{\boldsymbol{x}}^{(h)}) = \min \sum_{\{i,j\} \in E} \bar{x}^{(h)}_{ij} c_{ij} \tag{3.2}$$

$$\text{s.t.} \sum_{\{i,j\} \in E} \bar{z}_{ij} c_{ij} \geq 1 \qquad \forall \bar{\boldsymbol{z}} \in \mathcal{T}_n \tag{3.3}$$

$$c_{ij} \leq c_{ik} + c_{jk} \qquad \forall i, j, k \in V \tag{3.4}$$

$$c_{ij} \geq 0 \qquad \forall \{i, j\} \in E \tag{3.5}$$

$$y_i + y_j - u_{ij} + \sum_{S \in \mathcal{S}_{ij}} d_S \leq c_{ij} \qquad \forall \{i, j\} \in E \tag{3.6}$$

$$u_{ij} \geq 0 \qquad \forall \{i, j\} \in E \tag{3.7}$$

$$d_S \geq 0 \qquad \forall S \in \mathcal{S} \tag{3.8}$$

$$y_i + y_j - u_{ij} + \sum_{S \in \mathcal{S}_{ij}} d_S = c_{ij} \qquad \forall \{i, j\} \in E \text{ s.t.} \bar{x}^{(h)}_{ij} > 0 \tag{3.9}$$

$$u_{ij} = 0 \qquad \forall \{i, j\} \in E \text{ s.t } \bar{x}^{(h)}_{ij} < 1 \tag{3.10}$$

$$d_S = 0 \qquad \forall S \in \mathcal{S} \text{ s.t } \sum_{ij \in \delta(S)} \bar{x}^{(h)}_{ij} > 2. \tag{3.11}$$

Constraints (3.3) ensure that the optimal solution $\boldsymbol{c}^*$ of OPT($\bar{x}^{(h)}$) for every $\bar{\boldsymbol{x}}^{(h)}$ is such that $STSP(\boldsymbol{c}^*) = 1$, as discussed in [6]. Herein, we call the inequalities (3.3) the *TSP constraints*. Constraints (3.4) and (3.5) ensure that the cost vectors represent a semimetric. Constraints (3.6)–(3.8) are the dual constraints associated to the dual problem of (1.6)–(1.8) and (1.12). Constraints (3.9)–(3.11) ensure that the vertex $\bar{\boldsymbol{x}}^{(h)}$ remains the optimal solution of the SSEP. If we denote by $\mathcal{C}^*$ the set of the optimal solutions of OPT($\bar{\boldsymbol{x}}^{(h)}$), the dual slackness constraints are introduced to guarantee

$$\arg\min \text{SEP}(\boldsymbol{c}^*) = \boldsymbol{x}^{(h)}, \quad \forall\ \boldsymbol{c}^* \in \mathcal{C}^*. \tag{3.12}$$

Note that, in [6, 9], the authors observed that, for $6 \leq n \leq 12$, $|\mathscr{C}^*| = 1$.

Hence, given the list of the vertices $\{\bar{\boldsymbol{x}}^{(1)}, \ldots, \bar{\boldsymbol{x}}^{(t)}\}$ vertices of P^n_{SSEP}, the integrality gap α^n_{STSP} of K_n is computed by solving

$$\frac{1}{\alpha^n_{\text{STSP}}} = \min_{h=1,\ldots,t} \text{OPT}(\bar{\boldsymbol{x}}^{(h)}) \qquad \Rightarrow \qquad \alpha^n_{\text{STSP}} = \max_{h=1,\ldots,t} \frac{1}{\text{OPT}(\bar{\boldsymbol{x}}^{(h)})}. \tag{3.13}$$

Part of the original contribution presented in [6] is to prove that we need to consider only a subset of the t vertices of P_{SEP}, namely, such vertices that support graphs satisfy certain properties. We refer the reader to [6] for the details.

In the following section, we modify the single problem $\text{OPT}(\bar{\boldsymbol{x}}^{(h)})$ in order to introduce a new NP-hard problem that we use to generate metric TSP instances with a large integrality gap.

3.2 The Integer Heuristic-OPT Problem

The goal of our work is to devise an efficient computational procedure to generate instances with a large integrality gap for the Metric STSP. We are not interested in computing the exact value α^n_{STSP} for a fixed n as in [6], but we focus on finding a heuristic solution to problem (3.13). Notice that problem (3.2)–(3.8) has an exponential number of variables due to the dual variables d_S. Furthermore, the number of constraints (3.3) is equal to the number of tours, that is $\frac{(n-1)!}{2}$, and the number of triangular inequalities constraints (3.4) is $O(n^3)$. In practice, the exact solution of $\text{OPT}(\bar{\boldsymbol{x}}^{(h)})$ is intractable even for small values of n. Note that the computation of α^n_{STSP} for $n = 12$ required around 24 days [9]. In order to solve problem (3.13) heuristically, we introduce a new problem, called *Heuristic-OPT (H-OPT)*, which is related to problem $\text{OPT}(\bar{\boldsymbol{x}}^{(h)})$, but can be solved for larger values of n. The two key ideas for introducing the new problem are:

(i) To generate a Metric TSP instance with a sufficiently large integrality gap it is unnecessary to enumerate all vertices $\mathscr{X}^n_{\text{SSEP}}$. We can sample a subset of vertices and take the instance providing the largest integrality gap.

(ii) Since we do not perform exhaustive vertex enumeration, it is unnecessary to impose the complementary slackness constraints (3.6)–(3.8) to force that a given vertex $\bar{\boldsymbol{x}}^{(h)}$ remains the same vertex of P^n_{SSEP}. Hence, we can remove all the variables and constraints related to the slackness conditions.

For these two reasons, given a vertex $\bar{\boldsymbol{x}}^{(h)} \in P^n_{\text{SSEP}}$, we define the following LP problem:

$$\text{H-OPT}(\bar{\boldsymbol{x}}^{(h)}) := \min \quad \sum_{\{i,j\}\in E} \bar{x}_{ij}^{(h)} c_{ij} \tag{3.14}$$

$$\text{s.t.} \quad \sum_{\{i,j\}\in E} \bar{z}_{ij} c_{ij} \geq 1 \qquad \forall \bar{\boldsymbol{z}} \in \mathcal{T}_n \tag{3.15}$$

$$c_{ij} \leq c_{ik} + c_{jk} \qquad \forall i,j,k \in V \tag{3.16}$$

$$c_{ij} \geq 0 \qquad \forall \{i,j\} \in E. \tag{3.17}$$

In practice, we have relaxed the problem OPT($\bar{\boldsymbol{x}}^{(h)}$) by removing constraints (3.6)–(3.8). Notice that in H-OPT we have only $|E|$ cost variables, $\frac{(n-1)!}{2}$ TSP constraints (3.15) (one for each tour), and $O(n^3)$ triangles inequalities (3.16) that define the metric cone [48]. We can solve this LP problem by cutting planes, by separating both families of constraints. A critical point in the solution of the H-OPT problem by branch-and-cut is the separation of (maximally violated) TSP constraints (3.15). More specifically, the *separation problem* SP is defined as follows. Given a cost vector $\bar{\boldsymbol{c}} \in \mathbb{R}_+^{|E|}$, we look for a tour whose incidence vector $\bar{\boldsymbol{z}}$ verify the following:

$$\sum_{\{i,j\}\in E} \bar{z}_{ij} \bar{c}_{ij} < 1. \tag{3.18}$$

Notice that any tour that satisfies the previous relation gives a violated TSP constraint. However, to prove that no tour violates the TSP constraint, we need to verify the following:

$$\min_{\boldsymbol{z}\in\mathcal{T}_n} \left\{ \sum_{\{i,j\}\in E} z_{ij} \bar{c}_{ij} \right\} \geq 1. \tag{3.19}$$

Thus, to add a new STSP constraint, we solve a STSP instance for a specific cost vector $\bar{\boldsymbol{c}}$. In our implementation, we separate STSP constraints by first solving the TSP instance given by $\boldsymbol{c}^*$ using the LK-H heuristic [39, 40], and whenever the heuristic fails to find a violated tour, we solve (3.19) by embedding Concorde in the code. Section 3.4.1 describes the details of our implementation. Clearly, the STSP constraints make this problem challenging, as stated in the following lemma.

Lemma 2. *The H-OPT problem is NP-hard.*

First of all, we prove the following Lemma.

Lemma 3. *Let $n = |V|$. If H-OPT can be solved in $n^{O(1)}$ time, then the separation problem SP can be solved in $n^{O(1)}$ time.*

Proof sketch. Let P be the polyhedron defined by equations (3.15)–(3.17). The proof is implicit in the proof of a well-known theorem of Grötschel et al. [36] which states that for well-described polyhedron P the optimization problem can be solved in polynomial time if and only if the separation problem SP can be solved in polynomial time. Note that the requirement of well-described polyhedral P is only used for obtaining a time complexity polynomial in the input size of the problem. For example, if the length L of the input needed to describe the polyhedral P does not satisfy $n \leq L$, then even if the separation could be resolved in time $n^{O(1)}$ we could not claim that the separation problem SP can be resolved in polynomial time.

To see how the proof of this lemma is implicit in [36], it is sufficient to note that in [36] it is not asked to provide the input size as tight as possible. Actually, it is possible to make the encoding dimension L of a polyhedron P greater than n by adding, for example, a string with n zeros. This ensures $n \leq L$, and the claim follows. $\square$

Proof of Lemma 2. The claim follows by showing that the Hamiltonian Cycle problem (HC) can be solved in polynomial time by an oracle machine with an oracle for H-OPT. More precisely, we reduce in polynomial time HC to the separation problem SP of H-OPT. Then, by Lemma 3 an oracle machine for H-OPT that takes $n^{O(1)}$ time implies that the separation problem SP (and therefore HC) can be solved in $n^{O(1)}$ time.

Consider the following sets:

$$\begin{aligned} M &= \{\mathbf{c} \in \mathbf{R}^m \mid c_e > 0, \forall e \in [m]\, c_{ij} \leq c_{ik} + c_{jk}\ \forall i, j, k \in [n]\}, && (3.20)\\ TSP &= \{\mathbf{c} \in \mathbf{R}^m \mid, \mathbf{c}^T \boldsymbol{x} \geq 1\ \forall \boldsymbol{x} \in \mathscr{T}\}, && (3.21)\\ P &= M \cap TSP. && (3.22) \end{aligned}$$

Let $G = (V, E)$ be an undirected graph on n nodes, that is $|V| = n$. This graph defines an instance of the HC problem, namely the decision problem that searches for a cycle in a graph. We can perform a standard polynomial time reduction to get a Metric TSP and then normalize costs as follows:

$$\hat{c}_e = \begin{cases} (1 - \frac{\epsilon}{2})/n & e \in E, \\ (2 - \epsilon)/n & \text{otherwise}, \end{cases} \tag{3.23}$$

where $1 > \beta(n) > \epsilon > 0$, and the definition of $\beta(n)$ will be clarified later in the proof of this theorem.

Note that this instance of the TSP is metric: identity and symmetry are obvious, and the triangle inequalities can be verified case-by-case. Note also that, the graph contains a Hamiltonian cycle if and only if the optimal solution of the TSP is $1 - \frac{\epsilon}{2} < 1$, and thus such cost vector is in $M \setminus TSP$. On the opposite side, if the graph does not admit aHamiltonian cycle, then the TSP solution must contain at least one edge of length $(2-\epsilon)/n$. Then, the value of the optimal tour would be at least

$$\frac{2-\epsilon}{n} + \frac{(n-1)(1-\frac{\epsilon}{2})}{n} = \frac{1}{n} + 1 - \frac{\epsilon}{n}\left(\frac{1}{2} + \frac{n}{2}\right). \tag{3.24}$$

We observe that if $\varepsilon < \dfrac{2}{1+n}$, then (3.24) is greater than 1. Thus, we can set $\beta(n) = \dfrac{2}{(n+1)}$. Note that from now on we have proved that a graph $G = (V, E)$ on n nodes admits a Hamiltonian cycle if and only if the solution of the associated TSP with cost vector $\hat{\boldsymbol{c}}$ is less than 1. By putting everything together, we see that the Hamiltonian Cycle problem (HC) can be solved in polynomial time by an oracle machine with an oracle for H-OPT as follows:

1. Let $G = (V, E)$ be an undirected graph on n nodes.

2. Use equation (3.23) to obtain a TSP instance with cost vector $\hat{\boldsymbol{c}}$.

3. By Lemma 3, an oracle machine for H-OPT that takes $n^{O(1)}$ time implies that we can decide whether $\hat{\boldsymbol{c}} \in M$ in $n^{O(1)}$ time. If $\hat{\boldsymbol{c}} \in M$ then G does not admit a Hamiltonian cycle. Otherwise, it admits a Hamiltonian cycle.

□

We remark that our proof only yields nto NP-hardness under Turing reductions, as we have shown that there is an NP-complete decision problem, namely HC, that can be Turing-reduced to H-OPT. However, this also implies "hardness" for our problem, as yan existence of a polynomial time algorithm for H-OPT would imply $P = NP$. Note also that since we have removed from our problem the dual slackness constraints, the relation (6.10) is not necessarily satisfied when $\mathbf{c}^*$ is the optimal solution of H-OPT($\bar{\boldsymbol{x}}^{(h)}$). However, we can prove the following lemma which states that the solution of the H-OPT($\bar{\boldsymbol{x}}^{(h)}$) problem provides a cost vector $\boldsymbol{c}^*$ corresponding to a TSP instance with an integrality gap greater than or equal to any instance yielding $\bar{\boldsymbol{x}}^{(h)}$.

Lemma 4. *Let us consider a STSP instance $\mathbf{c}_0$ such that STSP($\mathbf{c_0}$) $= 1$, and let $\bar{\boldsymbol{x}}^{(0)}$ be an optimal solution of SSEP($\mathbf{c_0}$). We define a second STSP instance by the cost vector*

$$\boldsymbol{c}_1 = \arg\min \textit{H-OPT}(\bar{\boldsymbol{x}}^{(0)}).$$

Then, the following relation holds

$$\frac{\textit{STSP}(\boldsymbol{c_1})}{\textit{SSEP}(\boldsymbol{c_1}))} \geq \frac{\textit{STSP}(\mathbf{c_0})}{\textit{SSEP}(\mathbf{c_0})}. \tag{3.25}$$

Proof. Since $\bar{\boldsymbol{x}}^{(0)}$ is a feasible solution of the SSEP, we have that

$$\text{SSEP}(\boldsymbol{c_1}) \leq \boldsymbol{c}_1^T \bar{\boldsymbol{x}}^{(0)}$$

By definition of H-OPT($\bar{\boldsymbol{x}}^{(h)}$), we have $\boldsymbol{c}_1$ is the cost vector that realizes the minimum of $\boldsymbol{c}^T \bar{\boldsymbol{x}}^{(h)}$ among all the metric vectors such that STSP($\boldsymbol{c}$) $= 1$. By hypothesis, STSP($\boldsymbol{c_0}$) $=$

1 and this implies, $\boldsymbol{c}_0 \boldsymbol{z} \geq 1$ for all $\boldsymbol{z}$ 0-1 incidence vector. Thus, $\boldsymbol{c}_0$ is feasible for H-OPT($\bar{\boldsymbol{x}}^{(0)}$) and it holds

$$\boldsymbol{c}_1^T \bar{\boldsymbol{x}}^{(0)} \leq \boldsymbol{c}_0^T \bar{\boldsymbol{x}}^{(0)} = \mathrm{SSEP}(\boldsymbol{c_0}),$$

where the last equation holds by definition. Thus,

$$\frac{\mathrm{STSP}(\boldsymbol{c_0})}{\mathrm{SSEP}(\boldsymbol{c_0})} = \frac{1}{\mathrm{SSEP}(\boldsymbol{c_0})} \leq \frac{1}{\boldsymbol{c}_1^T \bar{\boldsymbol{x}}^{(0)}} \leq \frac{1}{\mathrm{SSEP}(\boldsymbol{c_1})} = \frac{\mathrm{STSP}(\boldsymbol{c_1})}{\mathrm{SSEP}(\boldsymbol{c_1})}.$$

□

From large integrality gaps to hard instances. The STSP solver Concorde [2] and the LK-H heuristic that we use for separating TSP constraints [39] can handle only integer costs. However, the H-OPT problem generates fractional cost vectors.

Hence, we need to devise a method to transform the fractional costs into integer values, by, for example, multiplying for a large constant τ and then rounding to the nearest integer, that is, $c_{ij} = \mathrm{round}(\tau c^*_{ij})$. Using this cost transformation, we cannot guarantee that the integrality gap for c_{ij} remains the same of c^*_{ij}. The TSP constraints (3.15) have a right-hand side equal to 1 because we have divided the cost vector $\boldsymbol{c}$ by the minimum tour length STSP($\boldsymbol{c}$). If we divide the cost vector by the quantity $\left(\frac{\mathrm{STSP}(\boldsymbol{c})}{\Delta}\right)$, where $\Delta > 0$ is a large positive constant, we have to change the right hand side of (3.15) to Δ, while still getting an equivalent LP problem. If Δ is large enough (see Section 3.4.1), we can also add the integrality constraint on the variable c_{ij}. In practice, in order to generate integer cost vectors, we have introduced the following ILP problem:

$$\begin{aligned}
\text{IH-OPT}(\boldsymbol{x}^{(h)}) := \min \quad & \sum_{\{i,j\}\in E} \bar{x}^{(h)}_{ij} c_{ij} && && (3.26)\\
\text{s.t.} \quad & (3.16), (3.17) && && (3.27)\\
& \sum_{\{i,j\}\in E} \bar{z}_{ij} c_{ij} \geq \Delta && \forall \bar{\boldsymbol{z}} \in \mathscr{T}_n && (3.28)\\
& c_{ij} \text{ integer} && \forall \{i,j\} \in E. && (3.29)
\end{aligned}$$

Clearly, IH-OPT is very challenging, as it is an integer program with as many constraints as STSP tours plus the number of triangle inequalities. We have computational evidence that the integrality gap of the integer problem could be smaller than those obtained by solving H-OPT. Surprisingly, our computational results show that the TSP instances obtained while solving the IH-OPT problem are very challenging for Concorde. In the next section, we show how we use the IH-OPT problem to search for very challenging instances for Concorde.

3.3 A sampling procedure for generating hard instances

A standard procedure for searching heuristically for (suboptimal) solutions of an optimization problem is based on uniformly sampling points (i.e., solutions) of the feasible region [7]. In our context, to find a heuristic solution for problem (3.13), we could sample a fixed number of vertices of P^n_{SSEP} and retain the vertex yielding the largest integrality gap. However, directly sampling the vertices of the P^n_{SSEP} is time-consuming, due to its exponential number of subtour constraints. We could instead easily sample random cost vectors and generate vertices of P^n_{SSEP} by solving directly problem (1.6)–(1.8) plus the bound constraint $0 \le x_e \le 1,\ e \in E$. A possibility for generating random cost vectors consists of generating r random points in an Euclidean space and then computing all the pairwise distances using a given distance (e.g., a distance induced by the Minkowski norm). As observed from our preliminary tests, this procedure leads very often to an integral, and hence useless, vertex of P^n_{ASEP}. In practice, this procedure takes a long time before returning a fractional vertex of P^n_{SSEP}. Furthermore, it only samples instances of the Euclidean TSP, implicitly excluding some remarkable metric TSP instances, such as the ones provided in [6]. To generate a random cost vector, we have designed a different approach. First, we sample a random point within the metric polytope [48] using the *hit-and-run algorithm* [62], which generates a random metric STSP instances $\boldsymbol{c} \in \mathbb{R}^{|E|}_{+}$. Second, we get a (random) vertex $\bar{\boldsymbol{x}}^{(h)}$ by solving SSEP($\boldsymbol{c}$) via the simplex algorithm. Since the cost vector is a uniformly random point of the metric polytope, we expect that $\bar{\boldsymbol{x}}^{(h)}$ is a random vertex of P^n_{SSEP}. We also expect a good variety among the sampled vertices: for instance, with $n = 15$, after the sampling of 999 vertices, it took only less than three seconds to find one vertex both non-integer and not yet sampled. In the next paragraphs, we briefly review the hit-and-run algorithm, and we detail how we use the sampling procedure to generate hard metric STSP instances.

The hit-and-run algorithm The hit-and-run algorithm [62] is designed to sample from a bounded set P uniformly. The basic steps of the algorithm are:

1. Pick a point $\boldsymbol{x}_k \in P \subset \mathbb{R}^m$.
2. Generate a random direction $\boldsymbol{d}_k$ uniformly distributed over the unitary hypersphere centered in $\boldsymbol{x}_k$.
3. Generate a random point $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \lambda \boldsymbol{d}_k$ uniformly distributed over the *line set* $\{\boldsymbol{x} \in P \mid \boldsymbol{x} = \boldsymbol{x}_k + \lambda \boldsymbol{d}_k,\ \lambda \in \mathbb{R}\}$.
4. If a stopping criterion is met, stop (e.g., terminate after a fixed number of iterations). Otherwise, $\boldsymbol{x}_k \leftarrow \boldsymbol{x}_{k+1}$ and repeat from Step 2.

The points sampled with this procedure converge in total variation to a uniform distribution, as proven in [62]. The time required to have a sample that effectively approximates the uniform distribution is polynomial in the dimension, as proven in [51]. Later, a modification of this algorithm to sample a set of points that converges to an arbitrary target distribution was introduced in [58]. The interested reader can find an in-depth review of hit-and-run algorithms in [70].

Algorithm 1: Sampling vertices of the P_{SEP} by using the metric polytope.

Input: $m = |E|$, the size of the TSP instance

Input: r, the required number of vertices

Output: R, a collection of vertices of P_{SEP}

1 $R \leftarrow \emptyset$

2 **while** $|R| < r$ **do**

3     $\boldsymbol{c} \leftarrow HitAndRun(P_{MET})$

4     $\boldsymbol{x} \leftarrow \arg\min\{SEP(\boldsymbol{c})\}$

5   **if** $\boldsymbol{x}$ *is fractional* **and** $\boldsymbol{x} \notin R$ **then**

6         $R \leftarrow R \cup \{\boldsymbol{x}\}$

7   **end**

8 **end**

9 **return** R

The random sampling algorithm Note that the constraints (3.4)–(3.5) define the metric cone:

$$C_{\text{MET}} := \{\boldsymbol{c} \in \mathbb{R}_{+}^{|E|} \mid c_{ij} - c_{ik} - c_{jk} \leq 0, \forall i, j, k \in V\}. \tag{3.30}$$

If we add the perimeter inequality (also known as the *homogeneous* triangle inequality) we get the metric polytope

$$P_{\text{MET}} := C_{\text{MET}} \cap \{c_{ij} + c_{ik} + c_{jk} \leq 2, \forall i, j, k \in V\}. \tag{3.31}$$

Note that for every metric $\boldsymbol{c} \in C_{\text{MET}}$, there exist an γ such that $\gamma\boldsymbol{c} \in P_{\text{MET}}$ [48]. For this reason, without loss of generality, we can sample from P_{MET} instead of C_{MET}. Sampling from P_{MET} instead of C_{MET} guarantees to remain in the framework described in [51], since P_{MET} is a convex compact set and, hence, a convex body.

Algorithm 1 presents our procedure for sampling metric STSP instances from the set P_{MET}. Given the size of the metric space $m = |E|$ and the number of required sampled points r, the algorithm initialized in Step 1 the empty set R. Then, until the number of sampled points is equal to r, steps 3–7 are iterated. Step 3 generates a uniformly distributed random cost vector $\boldsymbol{c}$ from the open set P_{MET} using the hit-and-run algorithm. Step 4 generates an optimal vertex $\boldsymbol{x}$ for SEP($\boldsymbol{c}$), $\boldsymbol{x} \in P_{\text{SSEP}}^{n}$. If the vertex $\boldsymbol{x}$ is fractional and does not belong to R, it is added to R. Finally, the procedure returns the set R of r vertices of P_{SSEP}^{n}.

3.4 Generating the Hard-TSPLIB

The objective of our computational experiments is to generate the Hard-TSPLIB, a collection of small metric STSP instances having a large integrality gap and are challenging for the Concorde solver. The instances of the Hard-TSPLIB are generated by using a branch-and-cut solver for problem IH-OPT($\bar{\boldsymbol{x}}^{(h)}$).

In the following paragraphs, first, we present the hard instances generated starting from the TSPLIB. Second, we present the hard instances generated using the random sampling procedure presented in Section 3.3. Third, we compare the runtime of Concorde for solving our hard instances with the runtime required for solving the *Rectilinear 3-Dimensional instances* [71]. In this work, the author compares the introduced instances with other works, namely [56, 6, 41, 42], showing that author's instances are computationally harder. Thus, we compare ourselves with [71] and deduce other comparisons from the context. Then, we visually analyze the structure of the hard instances we have generated. Finally, we discuss the implementation details of our algorithms.

3.4.1 Implementation details

Our computational procedure for generating hard metric STSP instances has two core algorithms:

1. The branch-and-cut algorithm for solving problem IH-OPT($\bar{\boldsymbol{x}}^{(h)}$).

2. The sampling procedure from the metric polytope.

In the following paragraphs, we describe our implementation of the two algorithms.

Solving IH-OPT by branch-and-cut. The problem IH-OPT($\bar{\boldsymbol{x}}^{(h)}$) is solved by branch-and-cut by using the Gurobi commercial solver and using the C++ programming language. We compile our code with the GNU C++ compiler v8.3, with the following flags `-O2 -D_REENTRANT -m64 -ffast-math -DNDEBUG -Wall -march=native`. In our implementation, we dynamically add both the triangle inequalities and the STSP constraints, in order to keep the core LP problem as small as possible. Note that a complete enumeration of triangle inequalities could exhaust the memory of a standard computer for medium values of n.

For the separation of triangular inequalities, we follow the strategies used in [35]: at each iteration, we add a fixed number k of the most violated triangular inequalities, until no more violated triangle inequalities exist. The separation of triangle inequalities is carried over both during the solution of the LP relaxation and when encountering a new incumbent integer solution.

The separation of STSP constraints is more challenging since it corresponds to the solution of an STSP instance, as shown in (3.19). As we only need a violated cut, we first separate the STSP constraints heuristically by solving the STSP instance using the Lin-Kernigan local search procedure [49], as implemented in Concorde. Note that the LK-H heuristic only handles integer costs. Hence, we use the support of Gurobi for lazy constraints, which allows running our separation procedure only on incumbent integer solutions. Whenever the LK-H heuristic returns an STSP solution whose incidence vector

$\boldsymbol{z}$ satisfies $\sum_{\{i,j\}\in E} z_{ij}c_{ij} \geq \Delta$, with Δ as described in Section 3.2, we run a second exact STSP algorithm. For the exact separation of STSP constraints, we use Concorde compiled using CPLEX 12.8 as an LP solver, by using the default parameter settings.

The optimal solution of the LK-H heuristic is given as warm start to the STSP branch-and-cut implementation.

Notice that before starting the solution of the integer problem IH-OPT($\bar{\boldsymbol{x}}^{(h)}$), we solve its LP relaxation H-OPT($\bar{\boldsymbol{x}}^{(h)}$) via cutting planes by only separating the STSP constraints with the LK-H heuristic (see, for instance, the computational results in Table 3.3). Once the LK-H heuristic does not find any STSP violated constraint, we stop, and we collect all the triangle inequalities violated by no more than a threshold $\tau = 0.05$, and all the generated STSP constraints, to initialize the first pool of cut for our branch-and-cut algorithm.

Sampling the metric polytope by hit-and-run. The sampling procedure described in Algorithm 1 is implemented in Python 3.8.2. For the hit-and-run algorithm in Step 3, we have used the implementation provided by [29], which is based on the original algorithm introduced in [62]. The solution of the SEP problem in Step 4 is implemented using the python wrapper of Gurobi. Since the sampling procedure is very fast, we did not port this procedure to C++.

Parameters tuning. In our implementation, we had only to decide a value for the parameter Δ in (3.28). If we only consider the H-OPT formulation, namely the formulation without the integer costs, we can imagine multiplying each cost for a fixed quantity ω and obtaining a tour that costs $\omega\Delta$. However, once we move to integer costs, the parameter Δ becomes more important because it is strongly related to the number of values that can be taken as cost coefficients. For instance, if $\Delta = n$, the solution having all c_{ij} equal to 1 is optimal. This does not lead to an hard-to-solve instance. On the contrary, if Δ is too big, we miss such "degeneracy" of the costs on some edges. In practice, we have observed that for different values of Δ we get STSP instances of different (runtime) difficulty. Table 3.1 shows the computational results for solving IH-OPT over sampled STSP instances using different values of Δ. Since the hardest instances were generated while using $\Delta = 1000$, we fix this value in all of the tests reported in this paper. In future work, we plan to investigate further the impact of the parameter Δ.

3.4.2 Generating hard instances from the TSPLIB

The TSPLIB contains 20 instances with less than 76 nodes. Only 13 of them have a fractional solution for the SSEP (i.e., the integrality gap is greater than 1). For 12 of these 13 instances, we have generated a corresponding hard instance by solving H-OPT($\bar{\boldsymbol{x}}^{(h)}$), where $\bar{\boldsymbol{x}}^{(h)}$ is the fractional solution of SSEP solved via the simplex algorithm.

Table 3.1. Impact of the parameter Δ on the instances generated by IH-OPT.

	$\Delta = 100$	$\Delta = 1000$	$\Delta = 10000$
gr24, hard			
Integrality gap	1.146	1.220	1.220
avg. Concorde mean time	3.263	16.300	10.770
std. dev Concorde time	0.977	2.000	1.924
bayg29, hard			
Integrality gap	1.159	1.186	1.187
avg. Concorde mean time	4.157	82.500	32.850
std. dev Concorde time	1.271	17.600	9.200
bays29, hard			
Integrality gap	1.197	1.229	1.228
avg. Concorde mean time	11.108	64.200	47.392
std. dev Concorde time	1.790	18.200	16.137

If we denote by $\boldsymbol{c}_0$ the cost vector of the TSPLIB instance, and by $\boldsymbol{c}^*$ the optimal solution of H-OPT($\bar{\boldsymbol{x}}^{(h)}$), by Lemma 4 we have

$$\frac{\text{STSP}(\boldsymbol{c}^*)}{\text{SSEP}(\boldsymbol{c}^*)} \geq \frac{\text{STSP}(\boldsymbol{c}_0)}{\text{SSEP}(\boldsymbol{c}_0)},$$

that is, the STSP instance $\boldsymbol{c}^*$ has an integrality gap larger than or equal to $\boldsymbol{c}_0$. If instead, we solve the problem IH-OPT($\bar{\boldsymbol{x}}^{(h)}$), we cannot guarantee the previous relation, but in practice, we get more challenging instances with almost the same integrality gap. For this reason, all the following results are obtained by solving the IH-OPT problem. Table 3.2 reports the detailed results for the generation of hard instances from the TSPLIB. The table first reports the name and the dimension $n = |V|$ of the original instance. The third and fourth columns report the integrality gap of $\boldsymbol{c}_0$ (easy instance) and $\boldsymbol{c}^*$ (hard instance). In the remaining six columns, the table shows the average runtime in seconds (with the standard deviations), and the average number of branch and bound nodes for solving with Concorde first the TSPLIB instance, and later the corresponding Hard-TSPLIB instance. The averages are computed over 10 independent runs of Concorde, using 10 different seeds. Finally, the last column reports whether the optimal SSEP solution of $\boldsymbol{c}_0$ is the same optimal solution of $\boldsymbol{c}^*$.

The results of Table 3.2 show that the small TSPLIB instances have a very small integrality gap and are extremely easy for Concorde. They are solved within a fraction of second at the root node of the branch-and-cut tree. On the contrary, the Hard-TSPLIB have a significantly larger integrality gap, and they require, on average, several seconds (or up to several hours) to be solved to optimality by Concorde. As expected, a larger integrality gap at the root node implies a larger branch-and-cut tree, that is, a larger

Table 3.2. Hard-TSPLIB instances generated from the TSPLIB. The columns give the instance name, number of nodes $|V|$, the integrality gap for the TSPLIB instance $\boldsymbol{c}_0$ and the HardTSPLIB $\boldsymbol{c}^*$; average runtime (and standard deviation) and number of BC nodes over 5 independent runs of Concorde, compiled with CPLEX 12.8. The last column reports if equation (6.10) holds. The last three instances marked with (*) are solved only once due to the large running time. The instance brazil58 reached a timeout of 24 hours.

		$\alpha_n(\boldsymbol{c})$		TSPLIB - $\boldsymbol{c}_0$			HARDTSPLIB - $\boldsymbol{c}^*$			Does Eq (6.10)
name	$\|V\|$	$\boldsymbol{c}_0$	$\boldsymbol{c}^*$	runtime	stddev	BC n.	runtime	stddev	BC n.	hold?
gr24	24	1.000	1.220	0.028	0.004	1	16.3	2.0	33.0	✗
bayg29	29	1.001	1.186	0.030	0.004	1	82.5	17.6	158.6	✗
bays29	29	1.003	1.229	0.030	0.004	1	64.2	18.2	106.2	✗
dantzing42	42	1.003	1.150	0.031	0.005	1	66.4	51.7	123.0	✓
swiss42	42	1.001	1.140	0.031	0.005	1	1466.3	1122.5	369.8	✗
att48	48	1.002	1.135	0.031	0.005	1	1204.4	444.6	2876.2	✗
gr48	48	1.018	1.222	0.031	0.005	1	97.4	18.3	132.6	✗
hk48	48	1.001	1.215	0.031	0.005	1	185.0	55.9	265.8	✗
eil51	51	1.008	1.285	0.032	0.006	1	743.0	148.5	1253.8	✗
brazil58	58	1.002	1.163	0.033	0.005	1	*86400.0*	(*)	*55139*	✗
st70	70	1.006	1.280	0.033	0.005	1	37759.1	(*)	28227	✗
pr76	76	1.029	1.282	0.035	0.007	1	63181.6	(*)	37459	✓

number of nodes (column 'BC n.'). However, this is not always true, because, among the three instances with 48 nodes (att48, gr48, and hk48), the instance with the smaller integrality gap requires the largest number of branch-and-cut nodes. Indeed, visiting a larger search tree implies a longer runtime. We remark that the instance brazil58_hard is not solved by Concorde within a timeout of 24 hours.

Table 3.3 and Table 3.4 report the computational results for generating the Hard-TSPLIB instances while solving H-OPT($\bar{\boldsymbol{x}}^{(h)}$) and IH-OPT($\bar{\boldsymbol{x}}^{(h)}$), respectively. For each instance, the table reports the number of cuts generated and the runtime for the triangle inequalities separation (*trian.*), the TSP constraints separated by the LKH heuristic (*LKH-cuts*), and the TSP constraints separated by Concorde (*TSP-cuts*). For the solution of IH-OPT($\bar{\boldsymbol{x}}^{(h)}$), the table gives also the total number of branch-and-bound nodes, the lower bounds (LB), and the upper bounds (UB): when the LB and UB are equal the instance is solved to optimality. Concerning the separation algorithms, the violated triangle inequalities are identified in a very short time, and they have almost no impact on the overall runtime. The heuristic separation of TSP constraints using the LKH heuristic is very effective, but the runtime begins to be important. The exact separation of TSP constraints is one of the two runtime bottlenecks for the generation of hard instances. For example, for the instance swiss42, most of the time is spent on the exact separation of a TSP cut. Finally, notice that for the instances pr76, eil76, rat99, kroB100, kroC100, the solution by branch-and-cut of IH-OPT($\bar{\boldsymbol{x}}^{(h)}$) hits the time limit of 24 hours (86400 seconds). Among those instances, only for pr76, we generate a hard TSP instance;

Table 3.3. Computational results for the heuristic solution of the LP problem H-OPT($\bar{\boldsymbol{x}}^{(h)}$) by cutting planes. At this stage, we only separate the TSP cuts with the LK-H heuristic. Columns 3 and 4 report the number of triangular inequality and TSP constraints added. Columns 5 and 6 report the runtime for separating those inequalities. The last column reports the overall runtime in seconds.

		CUTS		RUNTIME		
name	$\|V\|$	trian.	LK-H	trian.	LK-H	total
gr24	24	1500	61	0.00	1.1	2.2
bayg29	29	2394	66	0.00	3.1	6.5
bays29	29	2318	70	0.00	2.5	5.2
dantzig42	42	6515	88	0.00	7.2	38.7
swiss42	42	5083	87	0.00	15.0	46.5
gr48	48	10011	182	0.00	20.7	93.6
hk48	48	7494	177	0.00	42.2	78.7
eil51	51	7358	123	0.00	25.8	69.1
brazil58	58	9271	136	0.00	85.9	287.5
st70	70	21914	411	0.00	116.7	1593.4
eil76	76	27885	418	0.00	206.3	2684.5
pr76	76	26233	473	0.00	141.7	4716.2
rat99	99	61300	757	0.00	848.0	190715.4
kroB100	100	60300	883	0.00	841.5	162504.0
kroC100	100	62501	1084	0.01	757.4	131369.2

in all other cases, we were not able to find an integer cost vector satisfying all triangle inequalities and all TSP constraints, that is, an optimal integer solution for problem IH-OPT($\bar{\boldsymbol{x}}^{(h)}$).

3.4.3 Generating hard instances by sampling

We have also generated a collection of instances for the Hard-TSPLIB by using the random sampling procedure discussed in Section 3.3. First, we run Algorithm 1 to generate a random set R of vertices of P^n_{SSEP}, for a fixed size n of the TSP. When generating a random vertex $\bar{\boldsymbol{x}}^{(h)}$ in Algorithm 1, we also store the cost vector $\boldsymbol{c}_0^{(h)}$ sampled from the metric cone which yields the vertex $\bar{\boldsymbol{x}}^{(h)}$. Hence, we can compute the integrality gap of the initial *easy* STSP instances. Later, for each random vertex $\bar{\boldsymbol{x}}^{(h)}$ in R, we solve the IH-OPT($\bar{\boldsymbol{x}}^{(h)}$) problem to get an instance with a larger integrality gap.

Table 3.5 reports the three hardest instances we are able to provide for each $n \in \{10, 15, 20, 25, 30, 35, 40\}$, generating 10 random vertices for each value of n. The table reports first the integrality gap of the sampled cost vector $\boldsymbol{c}_0$, of the optimal cost vector $\boldsymbol{c}^*$ obtained after solving IH-OPT($\bar{\boldsymbol{x}}^{(h)}$), and the gap α^*_n conjectured in [6], that is, to

Table 3.4. Computational results for the solution of the integer problem IH-OPT($\bar{\boldsymbol{x}}^{(h)}$) by branch-and-cut. Columns 3, 4, and 5 report the number of cuts added for each type. Columns 6, 7, and 8 report the runtime of the separation algorithms for triangular, LKH, and exact TSP cuts. Column 9 reports the runtime and column 10 the number of branch-and-bound nodes required. The last two columns gives the lower (LB) and upper bounds (UB).

		CUTS			RUNTIME						
name	$\|V\|$	trian.	LK-H	TSP	trian.	LK-H	TSP	total	BC n.	LB	UB
gr24	24	866	25	1	0.00	0.5	60.6	68.2	936	820.0	820.0
bayg29	29	591	15	1	0.00	1.1	505.5	508.0	281	843.0	843.0
bays29	29	930	33	1	0.00	1.3	238.9	242.1	740	814.0	814.0
dantzig42	42	1750	82	3	0.00	11.8	542.9	586.0	2801	869.5	869.5
swiss42	42	2997	25	1	0.00	7.9	9683.0	9723.2	1643	877.0	877.0
gr48	48	53310	431	4	0.00	97.9	10369.5	20085.4	110488	818.5	818.5
hk48	48	10310	329	3	0.00	89.4	11.9	678.6	14469	823.0	823.0
eil51	51	7709	163	4	0.00	40.0	14230.2	14851.8	8301	778.5	778.5
brazil58	58	81	2	1	0.00	1.8	8314.9	8324.6	1	858.0	858.0
st70	70	62803	386	3	0.01	128.9	12740.9	44798.8	110663	779.5	779.5
pr76	76	57598	969	1	0.26	532.6	9527.8	86000.0	172533	*778.5*	*780.0*
eil76	76	107629	631	0	0.18	351.7	-	86000.0	255729	*780.0*	-
rat99	99	33369	30	0	3.07	37.4	-	86000.0	28542	*814.5*	-
kroB100	100	65429	62	0	2.81	72.7	-	86000.0	38288	*769.0*	-
kroC100	100	93370	26	0	2.02	24.3	-	86000.0	40141	*775.5*	-

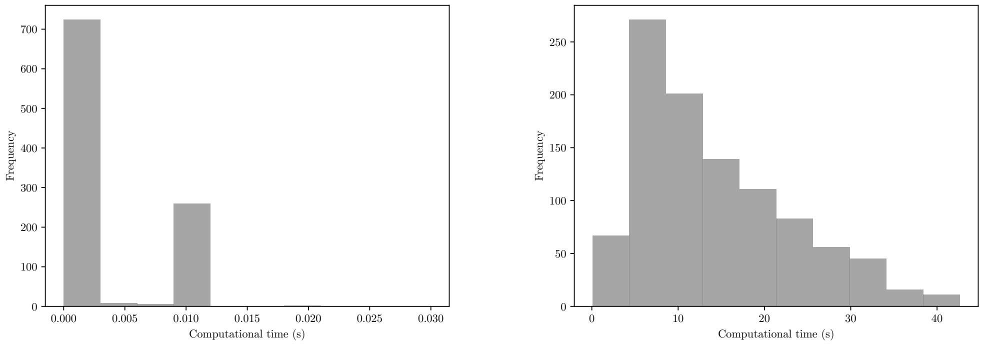


Figure 3.2. Left: distribution of the computational times for $n = 20$ of 1000 sampled TSP instances. Right: distribution of the computational times for $n = 20$ of 1000 instances obtained by the IH-OPT procedure

the best of our knowledge, the highest integrality gap available in the literature. Then, in the remaining columns, the table reports the average runtime (with the standard deviation) and the average number of BC nodes for solving the instances to optimality using Concorde, first for the instance $\boldsymbol{c}_0$ (random cost vector) and then for $\boldsymbol{c}^*$ (the optimal solution of IH-OPT). Similarly to the results of the Hard-TSPLIB, Table 3.5 shows that we are able to generate very hard instances by solving the IH-OPT problem.

We run a second experiment to study the effect of generating 1000 random instances for $n = 20$. For each random instance $\boldsymbol{c}_0^{(h)}$, with $h = 1, \ldots, 1000$, we measure the average solution runtime of Concorde. Then, using the corresponding vertex $\bar{\boldsymbol{x}}^{(h)}$ and solving IH-OPT($\bar{\boldsymbol{x}}^{(h)}$), we generate the harder instance, and we measure the runtime again. Figure 3.2(Left) shows the runtime distribution for the random instances $\boldsymbol{c}_0^{(h)}$, while Figure 3.2(Right) shows the runtime distribution for $\boldsymbol{c}^*$.

In the left plot, we have a runtime close to zero: we barely get close to 0.030 seconds. On the other hand, for the bottom plot, we have a mean runtime of 10 seconds, with a maximum of around 50 seconds. In practice, if we sample a large number of vertices we are able to get very challenging small instances.

Table 3.6 reports the average runtime (in seconds) to solve the IH-OPT problem on 1000 different vertices sampled for each n from 10 to 20. The solution to this problem exhibits a very large runtime variability. For instance, with $n = 20$, the minimum runtime is of 2 seconds, while the maximum is of 3445 seconds, that is, three orders of magnitude larger. In addition, we have fitted a linear regression on the $\log_{10}$ of the runtime, getting the following equation

$$t = 10^{0.146 \cdot n - 1.86}. \tag{3.32}$$

Using this regression, we try to predict the runtime for each n. For example, with $n = 40$, we would expect a runtime of around 3 hours. However, we have not observed

Table 3.5. Results for the Hard-TSP instances generated by sampling.

	Integrality Gap - $\alpha_n(\boldsymbol{c})$			Initial instance - $\boldsymbol{c}_0$			Hard instance - $\boldsymbol{c}^*$		
$\lvert V \rvert$	$\boldsymbol{c}_0$	$\boldsymbol{c}^*$	α_n^* [6]	runtime	stdev	BC n.	runtime	std.dev	BC n.
10	1.007	1.153	1.176	0.00	0	1	0.29	0.06	5.2
10	1.002	1.153	1.176	0.00	0	1	0.23	0.06	3.8
10	1.001	1.157	1.176	0.00	0	1	0.20	0.07	3.4
15	1.005	1.172	1.222	0.00	0	1	4.45	1.17	12.8
15	1.018	1.171	1.222	0.00	0	1	1.68	0.48	1.2
15	1.007	1.170	1.222	0.00	0	1	3.45	1.29	3.0
20	1.007	1.179	1.246	0.01	0	1	30.76	7.14	66.0
20	1.009	1.212	1.246	0.01	0	1	8.86	2.15	9.2
20	1.011	1.217	1.246	0.00	0	1	18.91	7.51	51.2
25	1.022	1.244	1.262	0.01	0	1	140.43	32.58	383.2
25	1.020	1.256	1.262	0.01	0	1	138.66	13.31	385.2
25	1.011	1.244	1.262	0.01	0	1	137.42	24.74	390.8
30	1.002	1.002	1.273	0.01	0	1	195.12	195.13	578.00
30	1.012	1.245	1.273	0.01	0	1	483.10	96.62	1426.8
30	1.000	1.263	1.273	0.01	0	1	156.55	21.13	454.4
35	1.013	1.263	1.281	0.01	0	1	2498.79	639.69	6565.0
35	1.018	1.276	1.281	0.02	0	1	1804.68	391.62	5193.5
35	1.012	1.244	1.281	0.01	0	1	1466.32	265.40	3962.1
40	1.002	1.210	1.287	0.01	0	1	224.85	102.18	514.0
40	1.005	1.280	1.287	0.01	0	1	5889.35	823.20	13212.4
40	1.011	1.278	1.287	0.02	0	1	6279.78	874.02	14261.0

any significant correlation between the runtime for solving the IH-OPT problem and the average runtime for solving the hard instance with Concorde.

3.4.4 A comparison with the 3D-Rectilinear instances

To the best of our knowledge, the current hardest instances of the literature are the Rectilinear 3D instances provided by Zhong [71]. In this work, the author proposes a family of 3-paths instances with a different configuration of nodes on the three paths and suggests the node configuration that makes them hard to solve for Concorde. For this reason, we tried to generate, using our computational procedure, metric TSP instances as hard as those 3D Rectilinear instances. We have tried the following two strategies.

(a) We massively sampled the metric polytope for small n with the algorithm described in Section 3.3 and then apply IH-OPT to the correspondent vertices, until we found an instance with an average runtime competitive with the 3D-Rectilinear instances. We call this found instance $\boldsymbol{c}_n^S$.

Table 3.6. Average computational time and standard deviation of 1000 IH-OPT on different $\boldsymbol{x}$ at a fixed n number of nodes.

n	avg. time	std. dev	min runtime	max runtime
10	0.63	0.39	0.29	6.86
11	0.51	0.22	0.21	1.98
12	0.74	0.33	0.31	2.84
13	1.19	0.64	0.45	9.08
15	2.40	1.84	0.63	26.97
16	3.38	2.40	1.00	27.03
17	5.82	6.41	1.29	100.14
18	9.37	22.13	1.81	622.00
19	15.63	107.89	2.15	3396.61
20	15.63	109.43	2.24	3445.62

(b) We tried to use our computational procedure to generate hard instances starting from the optimal solution of the instances provided by Zhong [71]. Let $\boldsymbol{c}_n^R$ the cost vector of a 3D rectilinear instance with n nodes [71], and let $\boldsymbol{x}_n^R$ the corresponding optimal solution of SEP($\boldsymbol{c}_n^R$). Similarly, let $\boldsymbol{c}_n^{IH} = \arg\min \text{IH-OPT}(\boldsymbol{x}_n^R)$ and $\boldsymbol{x}_n^{IH} = \arg\min \text{SEP}(\boldsymbol{c}_n^{IH})$.

We compared the average runtime of solving TSP($\boldsymbol{c}_n^R$), TSP($\boldsymbol{c}_n^{IH}$), and TSP($\boldsymbol{c}_n^S$). We have observed from our preliminary tests that Concorde has great runtime variance in solving the same instance: See, for example, the standard deviation values in Table 3.2 and Table 3.5, where we performed 10 independent runs on each instance, using every time a different seed. This variability is typical of Mixed Integer Programming and branch and cut algorithms (see e.g. [28], [50]). For this reason, the runtime comparison instance by instance is meaningless unless the difference is of at least one order of magnitude. Thus, we present the runtime comparison using linear regression on the logarithm ($\log_{10}$) of the runtime measured in seconds (again, using 10 independent runs for each instance).

Figure 3.3 shows the three regression lines for comparing the runtime for solving (i) the 3D rectilinear instances, marked with R, (ii) the instances produced using the sampling procedure mentioned above in point (a), marked with S, and (iii) the instances used with the procedure described in the point (b), marked with IH. The three runtimes are denoted by t_n^R, t_n^S, t_n^{IH}, respectively. The fitted regression lines are the following:

$$t_n^R = 10^{0.168\,n-2.209} \tag{3.33}$$

$$t_n^S = 10^{0.171\,n-2.157} \tag{3.34}$$

$$t_n^{IH} = 10^{0.137\,n-1.794}. \tag{3.35}$$

By looking at the regression, the instances that we have found by massively sampling the space are a bit harder than the Rectilinear 3-D instance. On the contrary, the instances

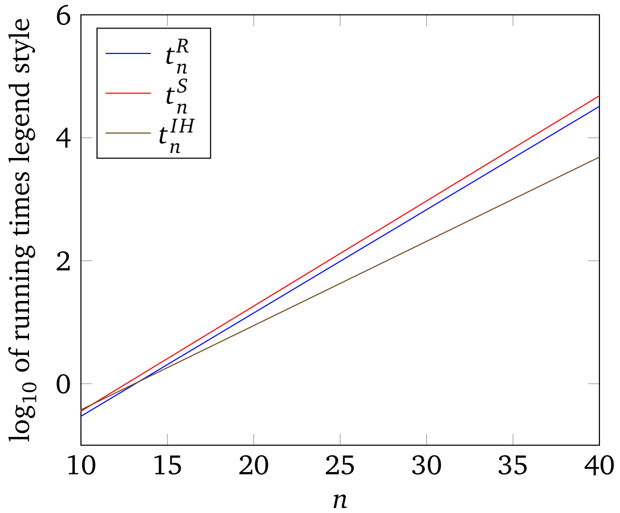


Figure 3.3. Linear regression for the runtime t_n^R, t_n^S, and t_n^{IH}, corresponding to 3D rectilinear instances, sampled instances, and instances which are solution of IH-OPT($\boldsymbol{x}_n^R$), respectively.

directly obtained from vertices are easier. Interestingly, in the logarithm scale, the three instances have all the same order of magnitude: by looking at Figure 3.3 it is possible to observe that the three families are competitive to each other, with the differences mostly related to the variability of Concorde. Noteworthy, by observing the structure of costs of the instances created by hand as [71] with respect to the one obtained by the two heuristic procedures, we note less regularity. This fact will be discussed widely in the next subsection. In terms of the integrality gap, we do not notice any significant evidence. For $n = 12$, the instance we sample has an integrality gap of 1.164 while the Rectilinear 3-D instance has 1.193 and the two runtimes are competitive. On the contrary, for $n = 20$, we sample an instance with an integrality gap of 1.242, while the Rectilinear 3-D instance has 1.240, and yet the two instances are competitive. Lastly, we run the computation as reported by Zhong, namely by multiplying the instances by 1000 and rounding to the nearest integer. We verified that this procedure might lead to non-metric instances as, due to rounding errors, triangular inequalities might not be satisfied.

3.4.5 Structure of small hard instances

The hard instances recently introduced in [42] and [71] are characterized, by construction, by half-integer solutions for SEP solution, and by support graphs having two triangles where each edge has a weight equal to $\frac{1}{2}$. Hence, we have looked at the structure of the support graph of our hard instances. For some instances, such as bays29 and eil51, the optimal vertex moves from a complicated structure to a 3-path configuration with two triangles having $\frac{1}{2}$-vertices. For the instance gr48, in the original version, the vertices have entries of values in the set $\{0.0, 0.25, 0.5, 0.75, 1.0\}$, while in the corresponding hard version, they have entries in the set $\{0.0, 0.5, 1.0\}$.

In addition, for two small sampled instances with $n = 15$ (s_{15}) and $n = 20$ (s_{20}), we studied "by hand" the cost structure of the support graph of the optimal solution of SEP. The motivation of this study is to investigate if the hard instances share some common cost patterns and/or structures of the support graph. Figures 3.4, and 3.5 show the support graph of the *easy* and *hard* version of the two instances, where the easy refer to the original cost vector $\mathbf{c}_0$ with its SEP vertex solution, and the hard to the optimal solution $\mathbf{c}^*$ of IH-OPT. In the support graphs, the purple edges correspond to the solution of SEP having value $x_e = \frac{1}{2}$, while the gray edges correspond to $x_e = 1$. The missing edges have $x_e = 0$. The label of each edge gives its cost in the TSP instance.

We observe that the main difference between the easy and hard instances is in the pattern of the edge length. In the easy instances, the edge costs look randomly distributed, while in the hard instances obtained after solving IH-OPT, there is a clear cost pattern. The edges lying along the same path have nearly the same cost, while the dashed edges connecting two distinct paths generally have a cost nearly equal to the sum of the costs of a single edge on a path. In addition, the overall length of the 3-paths are almost equal: for instance, in s_{15} (Figure 3.4), in the easy version, we have on each path a sum of, respectively, 707, 48, 1633, while in the hard one, we find 144, 147, 147. Finally, notice that since in IH-OPT we have removed the slackness constraints introduced in OPT_h, we do not have any guarantee that the optimal solution for the SEP associated to $\mathbf{c}^0$ and $\mathbf{c}^*$ is the same. However, in s_{15} we have the $\mathbf{x}^0 = \mathbf{x}^*$, while for s_{20} we get $\mathbf{x}^* \neq \mathbf{x}^0$, as shown in Figure 3.5.

3.5 Conclusion

In this Chapter, we have introduced a computational procedure to generate metric TSP instances that have large integrality gaps and are challenging for Concorde, the state-of-the-art STSP solver. As a by-product, we have introduced the Hard-TSPLIB, a collection of small but challenging metric TSP instances, which are not generated explicitly exploiting specific cost structures, as in [42, 71]. Notice that, to the best of our knowledge, all the hard instances from the literature have half-integer optimal SEP solutions. On the contrary, the instances of the Hard-TSPLIB have a larger variety of fractional optimal vertices (i.e., they are not only half integers). We expect our new instances will serve as a benchmark for designing new exact and heuristic methods for solving the TSP problem.

Curiously, we have observed that the most challenging instances generated using our computational procedure have regular cost patterns, with several edges sharing the same costs, and several paths on the support graphs having the same length. These types of cost patterns are in common with the manually-generated hard instances recently

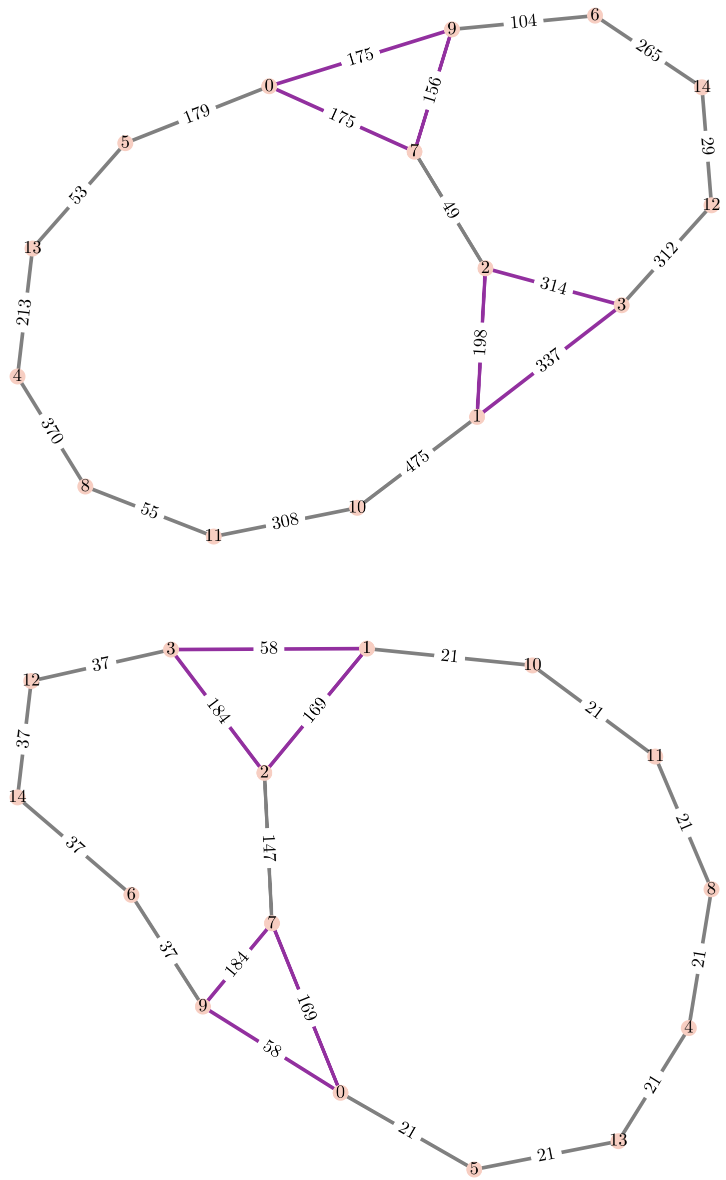


Figure 3.4. Easy (top) and hard (bottom) support graph associated to one of the sampled vertex with $n = 15$. The cost value on the edges of the support graph is also provided. Gray edges have a value 1, while purple edges have values $\frac{1}{2}$

introduced in [42] and [71], and also in the ones we provided in the previous chapter. Hence, we believe that our Hard-TSPLIB instances could help further studies in the cost structures of TSP instances. Part of this study was conducted during my PhD and is at

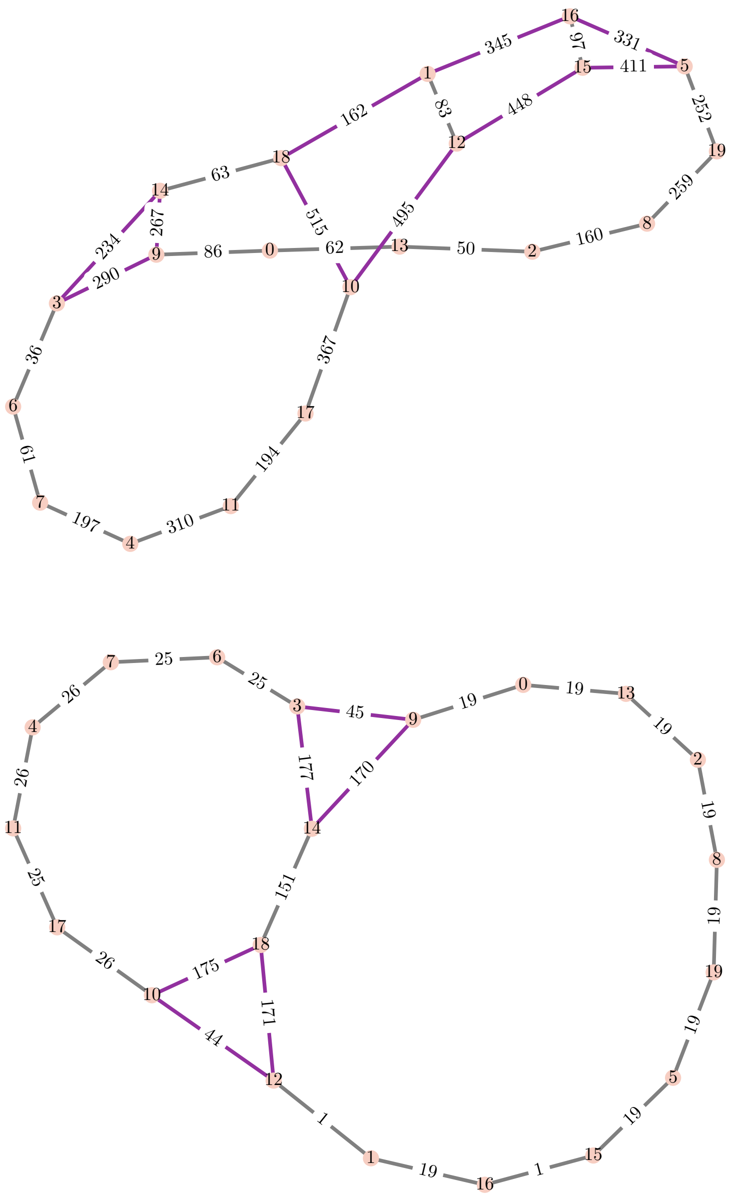


Figure 3.5. Easy (top) and hard (bottom) support graph associated to one of the sampled vertex with $n = 20$. The cost value on the edges of the support graph is also provided. Gray edges have a value 1, while purple edges have values $\frac{1}{2}$

the core of the next chapter.

Chapter 4

Hardness of STSP instances: a computational study

Section 3.4.5 raises an interesting research question: *which are the common traits among hard-to-solve instances?* This chapter is a tentative answer to this question. Here, we propose to classify the empirical hardness of metric STSP instances by using four features that depend only on the distribution of values in the cost matrix.

4.1 Background material

Several works in the literature have studied the features that make TSP instances hard to solve in practice for a given class of algorithms. We describe next the main works that consider empirical hardness for exact algorithms, but there is also a scientific interest in hardness for heuristics (e.g., see [64, 63]). In [14], the authors observe a clear phase transition from easy to hard TSP instances related to the standard deviation of the values c_{ij} in the cost matrix $\boldsymbol{C}$. In [33], the phase transition is related to a tour value ℓ and the area A occupied by n nodes, according to the ratio $\frac{\ell}{\sqrt{n \cdot A}}$. Note that this ratio is only reliable when ℓ is the optimal tour (or an almost optimal tour) and when the cities are defined as points in a metric space where the notion of "area" is given (e.g., the Euclidean space in 2D). Unfortunately, the notion of area is undefined for general metric spaces. In [52], the authors report computational experiments with instances generated with a random uniform distribution and an exponential random distribution: the two types of distributions lead to TSP instances with two different levels of hardness. Interestingly, the level of empirical hardness is defined in terms of the dimension of the branch and bound tree. Along the same line of research, two families of TSP instances are introduced in [25], where the empirical hardness is related to the amount of perturbation added to the cost matrix: a moderate amount of perturbation makes an instance very hard, while no perturbation and a significant perturbation reduce the complexity. Instead of classifying instances according to features such as the number of branch and bound nodes

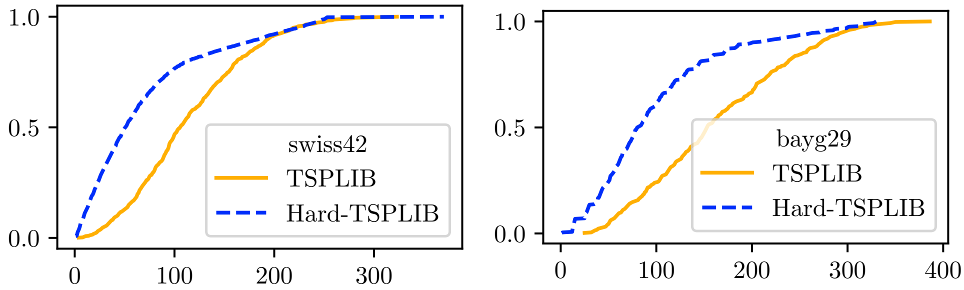


Figure 4.1. Empirical cumulative distribution of edge cost values of two instances of the TSPLIB and the Hard-TSPLIB.

or the runtime, a Random Forest is used in [11] to classify instances from the TSPLIB as either easy or hard. The main idea is to predict the quantity $\frac{\ell}{\sqrt{n \cdot A}}$ and to distinguish the instances between easy and hard according to their distance to the transition phase conjectured in [33]. Unfortunately, with this approach, there are instances such as the gr120 from the TSPLIB which is classified as hard, even if it is solved by Concorde in less than a second.

4.2 Cost Features of the Metric STSPs

In this section, we introduce the features of the cost vector $\boldsymbol{c}$ we use to predict the empirical hardness of metric TSP instances. The first two features we consider were already used in the literature: the standard deviation is used in [33] and [11], while the skewness is used in [11]. The third feature is the Gini index [34], a well-known score in statistics, which we use for the first time to predict the empirical hardness of TSP instances. The fourth feature we introduce is the *tight triangle inequality index*, which gives the percentage of all possible triangle inequalities in the cost matrix that are satisfied with equality.

In the following paragraphs, we describe in detail the four features used in the context of the metric TSP instances.

Standard Deviation. In our work, we consider the *normalized standard deviation* of the cost vector $\boldsymbol{c}$, which is defined as follows

$$\mathrm{SD}(\boldsymbol{c}) := \sqrt{\frac{1}{|E|} \sum_{\{i,j\} \in E} \left(\frac{c_{ij}}{\max(\boldsymbol{c})} - \bar{c} \right)^2}, \quad \text{where } \bar{c} = \frac{1}{|E|} \sum_{\{i,j\} \in E} \frac{c_{ij}}{\max(\boldsymbol{c})}. \tag{4.1}$$

Note that due to the normalization factor, we have $0 \leq \mathrm{SD}(\boldsymbol{c}) \leq 1$. The standard deviation of the cost matrix $\boldsymbol{C}$ was already successfully explored for asymmetric TSP instances in [14], where the author reports a phase transition related to this measure: both small

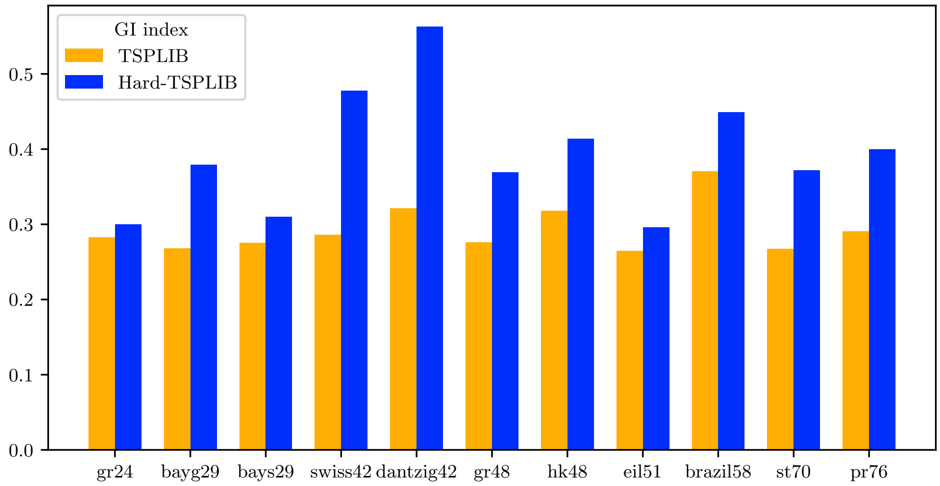


Figure 4.2. Gini index–GI($\boldsymbol{c}$) of 11 instances of the TSPLIB, along with the corresponding Hard-TSPLIB instances.

and large standard deviations lead to easy instances, while values in the middle lead to hard instances. The motivation given by the author is that whenever the standard deviation is high, only the low-cost tail of the cost distribution is considered to include an edge in an optimal tour, and thus this reduces the complexity. On the other hand, a small value of the standard deviation means that the costs are nearly all equal, and thus, it is easy to find a low-cost (optimal) tour.

Skeweness. The sample skewness is computed as the Fisher-Pearson coefficient of skewness of the cost vector:

$$\mathrm{SK}(\boldsymbol{c}) := \frac{m_3}{m_2^{\frac{3}{2}}}, \quad \text{where } m_k = \frac{1}{|E|} \sum_{\{i,j\} \in E} (c_{ij} - \overline{c})^k. \tag{4.2}$$

The skewness measures the displacement of the weights in a distribution: for unimodal distributions, $SK < 0$ indicates that the tail is on the left side of the distribution, while $SK \geq 0$ that the tail is on the right. Our computational results show that hard instances are associated with right-skewed distributions.

Figure 4.1 shows the empirical cumulative distribution of the values in the cost vector $\boldsymbol{c}$ for four instances. We can observe that the cumulative distributions associated with the Hard-TSPLIB instances have edge cost values concentrated toward low values. Note that having a lot of small costs close to a few higher ones is a commonly believed source of hardness for branch-and-bound algorithms.

Gini Index. The Gini index is a measure of statistical dispersion that quantifies the inequality among values of a frequency distribution. This index was introduced by Gini

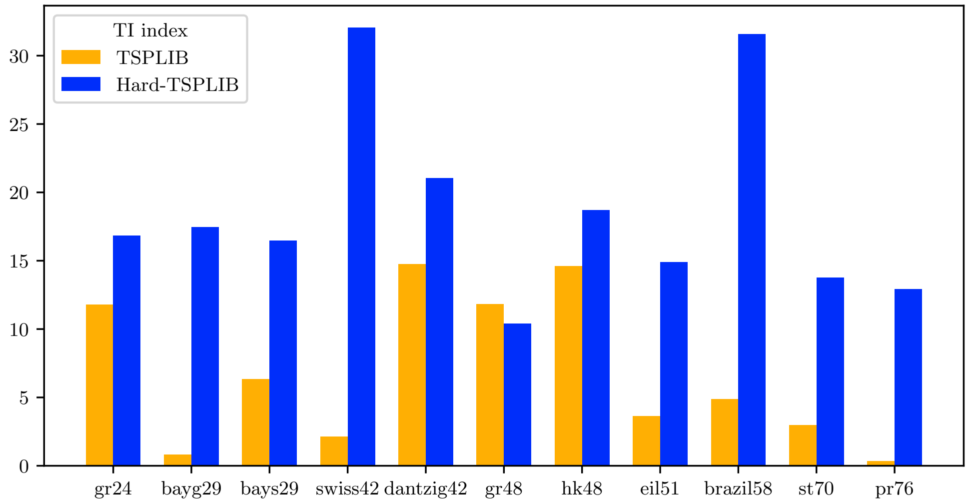


Figure 4.3. Tight triangle inequality index–TI($\boldsymbol{c}$) of 11 instances of the TSPLIB, along with the corresponding Hard-TSPLIB instances.

to measure the income (or wealth) inequality within a population [34]. When the values of the distribution are non-negative, the Gini index ranges in the interval $[0,1]$, where 0 means perfect equality and 1 represents perfect inequality. We use the Gini index to measure the dispersion (or inequality) of the cost vector values:

$$\mathrm{GI}(\boldsymbol{c}) := \frac{1}{2\,\bar{c}\,|E|^2} \sum_{\{i,j\}\in E} \sum_{\{v,w\}\in E} \left| c_{ij} - c_{vw} \right|. \tag{4.3}$$

Figure 4.2 compares the GI of the instances of the TSPLIB with those of the Hard-TSPLIB. We note that hard instances always have a larger value of this index, and this motivates its introduction as a feature of our predictor.

Tight Triangle Inequality Index. Given a Metric TSP instance with the associated cost vector $\boldsymbol{c}$, the *Tight triangle inequality Index* (TI for short) is defined as the number of triangle inequalities satisfied with equality with respect to all possible triangle inequalities.

$$\mathrm{TI}(\boldsymbol{c}) := \frac{\sum_{i=1}^{n} \sum_{j=i+1}^{n} \sum_{k=j+1}^{n} \mathbf{1}_{c_{ij}=c_{ik}+c_{jk}} + \mathbf{1}_{c_{ik}=c_{ij}+c_{jk}} + \mathbf{1}_{c_{jk}=c_{ij}+c_{ik}}}{3\cdot\binom{n}{3}}. \tag{4.4}$$

According to Section 2.3, for any triple of vertices (i,j,k), at most one of the three corresponding triangle inequalities can be tight. Thus, we have

$$0 \leq \mathrm{TI}(\boldsymbol{c}) \leq \frac{1}{3}.$$

Figure 4.3 compares $\mathrm{TI}(\boldsymbol{c})$ instances of the TSPLIB with the corresponding instances in the Hard-TSPLIB introduced in the previous chapter. In all these instances, except for gr48, $\text{TI}(\boldsymbol{c})$ is higher, and for instances swiss42 and brazil58, is close to the upper bound of $33.\overline{3}\%$. Note that, according to Table 3.2, it is not possible to solve brazil58 using Concorde within a time limit of 24h. Furthermore, we observed in Section 3.4.5 that hard-to-solve instances are highly structured, with multiple costs of the same weights and connected edges with the same sum with respect to the weight. Our idea is hence that $\text{TI}(\boldsymbol{c})$ should encode this high regularity of the costs.

Another reason to associate the empirical hardness of TSP instances with high values of TI is that any triple (i, j, k) satisfied with equality $c_{ij} = c_{ik} + c_{jk}$ makes the associated vector $\boldsymbol{c}$ to lay on a face of the metric polytope as defined in (3.31). Being on a face means being close to the outside P_{MET}, that is, being close to non-metric instances. Non-metric instances are known to be computationally harder than metric ones. Note that in the general TSP constant approximation is not possible in polynomial time. Otherwise, the Hamiltonian cycle would be solvable in polynomial time, and therefore $\text{P} = \text{NP}$ (e.g., see Theorem 9.10 page 226 in [22]).

4.3 Learning Decision Trees

Our goal is to design a decision tree that can infer a relation among the features $\text{SD}(\boldsymbol{c})$, $\text{SK}(\boldsymbol{c})$, $\text{GI}(\boldsymbol{c})$, and $\text{TI}(\boldsymbol{c})$ versus the hardness label (0=easy, 1=hard). Indeed, we want to evaluate if these four features can correctly detect hard instances. We selected decision trees mainly because they are interpretable. However, instead of computing a single decision tree, we perform extensive numerical tests to assess the expected performance of a decision tree trained with a small subset of every possible instance, and we report the average performance. Each decision tree is trained using the `scikit-learn` library [54] with default parameter values.

In the following paragraphs, we describe the datasets we have generated to run our experiments using the code complexification procedure designed in the previous chapter, and we detail the prediction scores used to evaluate each fitted decision tree.

Dataset. To create the dataset we proceed as follows: First, we sample 5 000 metric TSP instances using the *hit-and-run* algorithm [62]. We repeat this procedure for $n \in \{20, 25, 30, 35, 40\}$. Second, we apply the complexification procedure presented in Chapter 3 to all the instances with a time limit of 7 days. However, when the time limit expires, we are unable to guarantee a complete conversion of a sampled instance to its optimal hard version. Third, we solve both easy and hard instances using Concorde, with 10 different seeds, and we collect the number of Branch & Bound nodes (B&B) and the total runtime.

We decided to label as *easy* all the instances that take an average number of B&B nodes equal to or less than the 5-percentile of the complexified instances. In this way, we label as *easy* also the easiest instances of the complexified set, and if, by chance, we sample an instance that requires a lot of B&B nodes to be solved to the optimum, this is

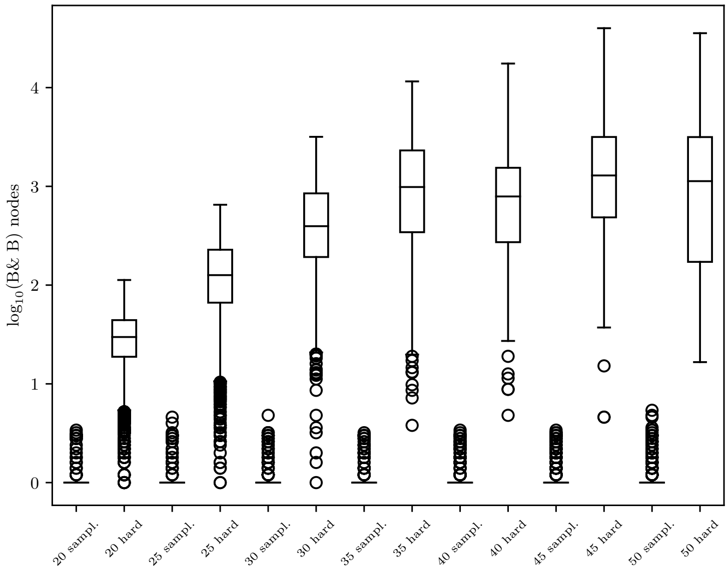


Figure 4.4. Boxplots of the distribution of numbers of B&B nodes required by Concorde to solve the TSP instances to optimality. Note that the time reduces the impact on the number of nodes for instances with $n \geq 30$.

hence labeled as hard. To have an idea of the meaning of such a parameter, Figure 4.4 shows the distribution of the number of B&B nodes in $\log_{10}$ scale of the instances we used for training and testing the decision tree. More specifically, we report the number of B&B nodes for $n \in \{20, 25, 30, 35, 40, 45, 50\}$ of the sampled instances, and the hard version of each sampled instance, using the Hit & Run algorithm as done in the previous chapter. Note that sampled instances require always less than 10 nodes, while hard instances need thousands. The decrease in $n = 50$ is just apparent: the really hard instances are not solved to optimality within the time limit.

To build a fair training set, we should know the true distribution of hard metric TSP instances within the metric TSP instances space. As this is unknown, we test five different percentage p of hard instances in each dataset, that is, $p \in \{0.01, 0.05, 0.1, 0.2\}$. For each n, for each p, and for each $i \in \{1, \ldots, 10\}$, we build a dataset D^i_{np} containing 2 000 instances. Each dataset contains $100\,p\,\%$ of hard instances, and the remaining are easy. To train a decision tree, we divide each dataset D^i_{np} into training and test sets, with a test size of 33%. Finally, we train a binary decision tree for each dataset D^i_{np}. Although we know that the standard procedure for machine learning tasks is to use k-fold cross-validation, we decided to proceed as discussed to better control each dataset dimension and the percentage of contained hard instances.

Prediction score: Matthew Correlation Coefficient. To evaluate the performance of each decision tree, we use the Matthew Correlation Coefficient (MCC), which is defined as follows. Given the confusion matrix and two classes, positive P and negative N, MCC is computed as

$$\text{MCC} := \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}},$$

where T stands for true (correctly classified) and F stands for false (misclassified). Note that $-1 \le \text{MCC} \le 1$, where -1 means perfect discordance, 1 means perfect accordance, and 0 means that predictions are random.

The motivation for using MCC for unbalanced datasets instead of Accuracy or F1-score is that with the latter a naive predictor can achieve a high score by simply classifying all the instances as the more frequent class (i.e., "easy" in this case). For in-depth motivations supporting the use of MCC for unbalanced datasets, we refer the interested reader to [15].

Implementation details. To parallelize the computations as much as possible for our tests, we limit the execution of each job to 16 GB of RAM, and we reserve one core for the sampling procedure, ten cores for the complexification procedure, and one core for running Concorde on each instance. The code used for the complexification procedure is the one described in the previous chapter, while the remaining pipeline is implemented in Python.

4.4 Computational Results

We have designed two types of experiments to validate the impact of the four features introduced in Section 2 in predicting the empirical hardness of metric TSP instances. The first class of tests targets instances of a fixed size n, while the second class of tests is independent of n. Note that all of our cost features, except SK, range between 0 and 1, and, thus, we expect them to be independent with respect to the number of nodes n of the TSP instance.

Decision Trees for fixed n. For each sub-sampled dataset $D^i_{n,p}$, we perform the following steps: (i) we split the dataset into training and test sets, (ii) we fit it with a binary decision tree, and (iii) we collect the MCC prediction scores on the test set. Finally, we average over the 10 decision trees indexed by i for each pair of n and p. Interestingly, we observe that the three-depth distribution decreases with the increase of the number of nodes. Figure 4.5 shows the distribution of the depth of 40 decision trees we train for each number of nodes n and each percentage p, hence exhibiting such trend.

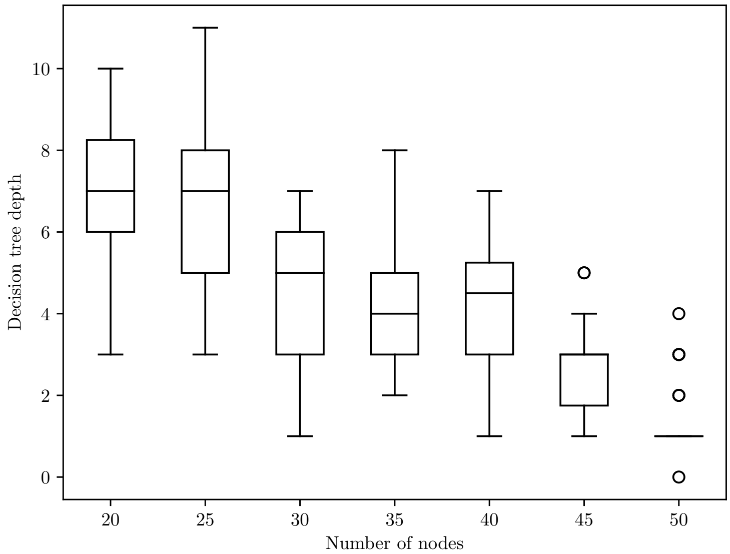


Figure 4.5. Boxplots of the decision tree depth for size n of the instances.

Table 4.1 shows the results of our trained decision tree on the sub-sampled datasets D^i_{np} with four different combinations of predictors. First of all, we study the impact of using only SD($\boldsymbol{c}$) and SK($\boldsymbol{c}$). These features were already used to predict hardness in [11], together with other predictors, such as the maximum and minimum distance between nodes, mean, and kurtosis. Once adding GI($\boldsymbol{c}$), we observe a small improvement. On the contrary, considering SD($\boldsymbol{c}$), SK($\boldsymbol{c}$), and TI($\boldsymbol{c}$) leads to better predictions, suggesting that indeed the tight index represents an important feature of hard instances. In any case, considering all four predictors together yields the best performances in terms of MCC. Moreover, the larger the percentage of hard instances in the test set, the better the decision tree predicts the empirical hardness.

Figure 4.7 shows a decision tree that was trained for an instance with $n = 40$ and $p = 10\%$. The blue leaf nodes are those corresponding to the "hard" class, while the yellow nodes correspond to "easy" instances. Note that a first level is defined by the TI: if this score is close to zero, instances are more likely to be classified as easy. Note also that hard instances have higher TI (e.g., see node 7). Note that, as already observed, the complexification procedure is time-consuming, and the instances we have generated are hard to solve. Thus, when $n = 45, 50$, we were unable to provide enough hard-to-solve instances to build datasets with specific percentages of the hard-to-solve instances. More specifically, for $n = 45$, we were able to generate and run Concorde 10 times for 266 instances, while with $n = 50$, only for 75. Thus, to perform our analysis, we reduce the size of the sub-sampled datasets, setting it equal to 1250 when $n = 45$ and 350 when $n = 50$. Then, we repeat the pipeline described in Section 4.3 and Section 4.4, obtaining Table 4.2, which is similar to Table 4.1. Although the datasets have smaller sizes, we

Table 4.1. MCC prediction scores: mean and standard deviations (in parenthesis). Each entry gives the average over 10 datasets for the corresponding n and p; we highlight in bold the best results for each group.

Features	n	$p = 0.01$	$p = 0.05$	$p = 0.10$	$p = 0.20$
SD+SK	20	0.223 (0.17)	0.400 (0.08)	0.517 (0.05)	0.550 (0.04)
SD+SK+GI	20	0.355 (0.19)	0.498 (0.05)	0.568 (0.05)	0.630 (0.03)
SD+SK+TI	20	0.612 (0.12)	0.834 (0.05)	0.850 (0.03)	0.917 (0.02)
SD+SK+GI+TI	20	**0.750** (0.13)	**0.868** (0.05)	**0.888** (0.02)	**0.930** (0.01)
SD+SK	25	0.256 (0.15)	0.519 (0.07)	0.641 (0.07)	0.650 (0.04)
SD+SK+GI	25	0.384 (0.10)	0.604 (0.07)	0.712 (0.05)	0.728 (0.05)
SD+SK+TI	25	0.505 (0.23)	0.829 (0.04)	0.887 (0.02)	0.912 (0.02)
SD+SK+GI+TI	25	**0.694** (0.22)	**0.882** (0.02)	**0.919** (0.03)	**0.922** (0.03)
SD+SK	30	0.447 (0.25)	0.624 (0.07)	0.719 (0.03)	0.794 (0.03)
SD+SK+GI	30	0.660 (0.19)	0.778 (0.08)	0.807 (0.03)	0.878 (0.02)
SD+SK+TI	30	0.851 (0.12)	0.960 (0.03)	0.962 (0.02)	**0.972** (0.01)
SD+SK+GI+TI	30	**0.875** (0.09)	**0.961** (0.03)	**0.965** (0.01)	0.970 (0.02)
SD+SK	35	0.462 (0.19)	0.712 (0.08)	0.756 (0.03)	0.835 (0.03)
SD+SK+GI	35	0.654 (0.20)	0.842 (0.07)	0.876 (0.03)	0.923 (0.02)
SD+SK+TI	35	0.822 (0.13)	0.964 (0.03)	0.968 (0.01)	0.977 (0.01)
SD+SK+GI+TI	35	**0.844** (0.14)	**0.967** (0.02)	**0.970** (0.02)	**0.982** (0.01)
SD+SK	40	0.377 (0.15)	0.644 (0.04)	0.734 (0.04)	0.759 (0.02)
SD+SK+GI	40	0.594 (0.13)	0.766 (0.06)	0.852 (0.05)	0.875 (0.02)
SD+SK+TI	40	0.791 (0.15)	0.937 (0.03)	**0.967** (0.02)	0.976 (0.01)
SD+SK+GI+TI	40	**0.815** (0.12)	**0.953** (0.03)	0.966 (0.02)	**0.977** (0.01)

draw the same conclusions as in Section 4.4. Note that when $n = 50$, the performances decrease, both globally and when compared to the usage of the four scores together with SD + SK + TI. We motivate this fact by the small dimension of the training set (only 230 instances).

Global Decision Tree. For the second class of tests, which are independent of n, we collected all the instances of the D^i_{np} datasets, including instances of size $n = 45$ and $n = 50$. Namely, we define a new dataset D as

$$D = \bigcup_{n,i,p} D^i_{np}.$$

where the three indices range as follows: $n \in \{20, 25, 30, 35, 40, 45, 50\}$, $i = \{1, 2, \dots, 10\}$, and $p \in \{0.01, 0.05, 0.1, 0.2\}$.

Overall, our global dataset contains 45 392 instances, among which 11 307 have been classified as hard. Using this large dataset, we train a single *global* decision tree with

Table 4.2. MCC prediction scores: mean and standard deviations (in parenthesis) for $n = 45, 50$. Each entry gives the average over 10 datasets for the corresponding n and p; we highlight in bold the best results for each group.

Features	n	$p = 0.01$	$p = 0.05$	$p = 0.10$	$p = 0.20$
SD+SK	45	0.532 (0.31)	0.745 (0.07)	0.776 (0.02)	0.800 (0.04)
SD+SK+GI	45	0.589 (0.32)	0.830 (0.05)	0.889 (0.04)	0.926 (0.02)
SD+SK+TI	45	**0.854** (0.13)	**0.970** (0.03)	**0.978** (0.01)	0.986 (0.01)
SD+SK+GI+TI	45	0.823 (0.12)	**0.970** (0.03)	0.977 (0.01)	**0.988** (0.01)
SD+SK	50	0.370 (0.46)	0.715 (0.14)	0.789 (0.10)	0.834 (0.05)
SD+SK+GI	50	0.269 (0.42)	0.765 (0.28)	0.894 (0.06)	0.941 (0.04)
SD+SK+TI	50	0.600 (0.49)	**0.949** (0.07)	**0.973** (0.04)	**0.995** (0.01)
SD+SK+GI+TI	50	**0.670** (0.44)	**0.949** (0.07)	0.965 (0.07)	0.992 (0.01)

`scikit-learn` using all the default parameters except the tree depth which is limited to 5, to make the trained decision tree easier to interpret. By running 10-fold cross-validation, we obtain an average MCC of 0.964. Thus, we retain our scores to be appropriate in distinguishing between hard and easy instances whenever the easy instances are sampled randomly for the metric cone.

To further asses the capability of our scores in distinguishing easy and hard instances, we use our trained DT to predict a different test set. To this purpose, we used another collection of instances available in the literature and described in Chapter 1. We have collected all these published instances that have less than 100 nodes, for a total of 282 instances. Using the number fo B&B nodes of Concorde as discriminator criteria, we labeled as "easy" the instances proposed in [6, 41, 56], and as "hard" all the instances in [42, 71] and the Hard-TSPLIB we have introduced in Chapter 3.

Table 4.3 reports the results of our global classification tree for the 282 published instances. First of all, we note a decrease in the performances of the classifier with respect to the one tested with the original dataset. More specifically, the MCC is 0.076, meaning no correlation.

Looking more deeply at the values obtained, we observe that our global decision tree is better at predicting hard instances than easy instances. We conjecture three reasons for explaining this observation. The first reason is that our training set only contains easy instances that come from a random sampling procedure, which are thus completely unstructured. On the contrary, the instances proposed in [6, 41] are highly structured, but they are classified as easy for our criteria. Indeed, our global decision tree is unable to capture some features of a regular easy instance. The second reason is that our training set contains instances with at most 50 nodes, while the easy instances of the test set have mostly more than 50 nodes. The third reason for explaining the results of Table 4.3 might be an implicit consequence of the original motivations of this work: We are focused on

Table 4.3. Results on published instances achieved by the global decision tree.

Family	Hard?	Correctly predicted
TSPLIB	No	4/26
Hougardy, 2014	No	1/17
Benoit & Boyd, 2008	No	0/91
Hougardy & Zhong 2020	Yes	17/17
Zhong, 2021	Yes	90/91
Hard-TSPLIB, 2021	Yes	39/40

cost features that can predict the empirical hardness of an instance, disregarding the features of easy instances. Hence, we did not study easy instances carefully, and we did not try to understand *why* they are easy. However, the effort to understand hardness is more compelling: knowing the true sources of empirical hardness may lead to the design of ad-hoc algorithms for each specific feature. In addition, it might help to prove or disprove conjectures on the integrality gap of certain polytopes, such as, for instance, the $\frac{4}{3}$ integrality gap conjecture for the subtour elimination polytope. Lastly, we are aware that supervised learning methods, as DTs are, are good at detecting procedural approaches. However, we believe that the difficulty in generalizing of our procedure is mainly due to the three reasons already discussed. As a matter of fact, our classifier correctly predicts the instances in [42, 71], which are generated by other researchers and with a procedure different than those presented in Chapter 1. The global decision tree is reported in Figure 4.6. We colored leaf nodes classified as easy with orange, while the ones classified as hard with blue. For each node, we reported a label (node number), the impurity of the node (imp.), and, for the non-leaf-nodes, the splitting attribute: The leftmost node means "less or equal", while the rightmost one means "greater". As an example, considering node 13: the splitting value on GI means that in node 14, GI ≤ 0.434, while in node 21, GI > 0.434. Blue nodes are the ones voting for "hard", while easy nodes are the ones voting for "easy". Note that a first level is defined by the TI: if this score is close to zero, instances are more likely to be classified as easy. Note also that most of our conjectures are confirmed: hard instances tend to have higher TI (see, e.g. nodes 27, 39), non-negative SK (see node 17, where greater of -0.034 is required, and thus most of the instances have non-negative SK), high GI (see, e.g nodes 21 and children). In some nodes, also the phase transition of SD is confirmed [14] (see, e.g nodes 19, 20, 31, 38, and 39).

4.5 Conclusion

In this Chapter, we have classified the empirical hardness of the metric TSP with a binary decision tree trained using features that depend exclusively on the distribution of values in the cost vector. The decision trees trained for a given fixed size n show statistically

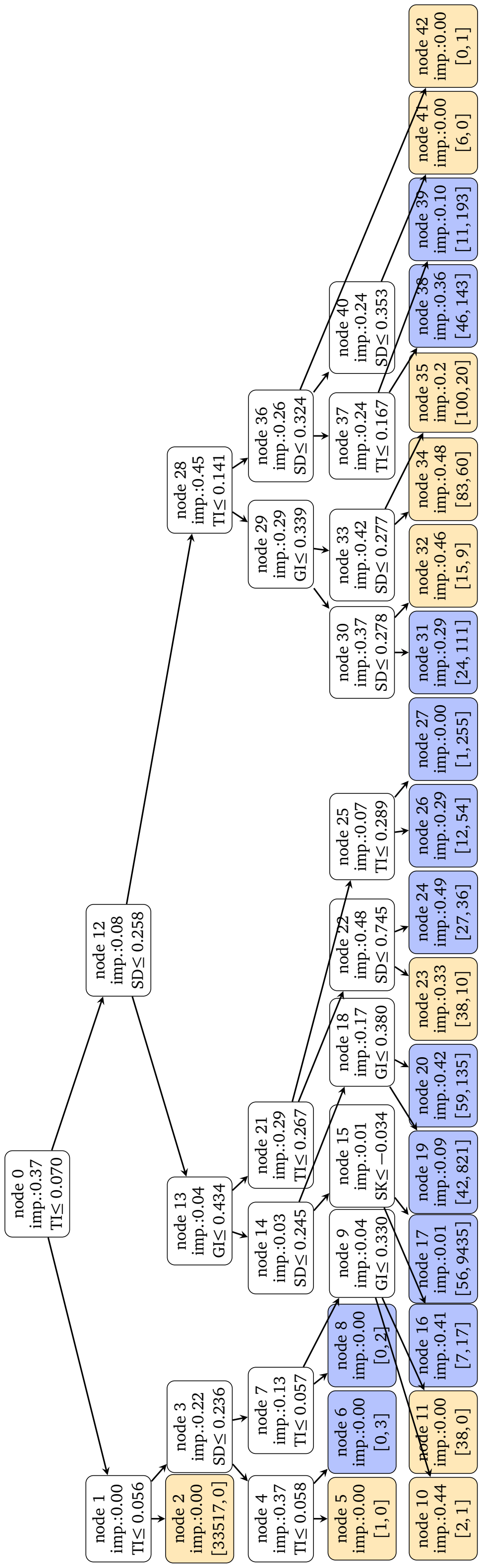


Figure 4.6. Global decision tree trained on all the available instances.

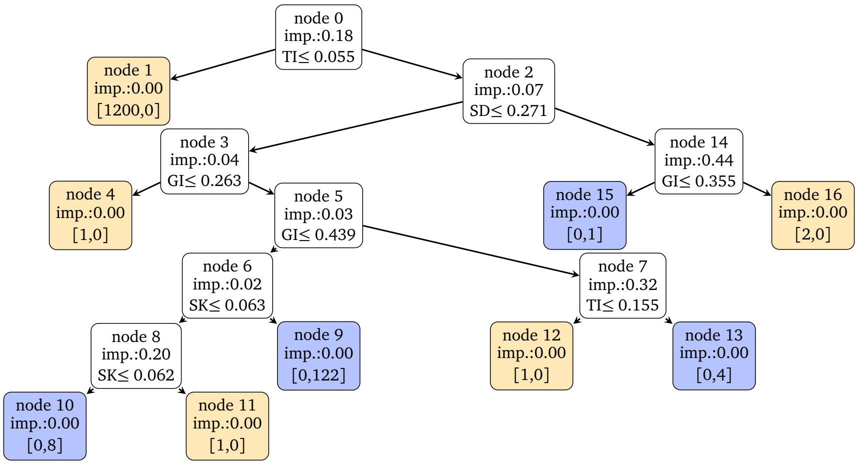


Figure 4.7. Decision tree with $n = 40$ and $p = 10\%$.

outstanding performance for the MCC prediction score. We also trained a global decision tree independent of n by putting all the instances (sampled and complexified) together. We test it in two different ways. First, we do standard 10-fold cross-validation, achieving good performances on average. Then, we test the global decision tree in trying to predict different families of instances published in the literature. In this task, our decision tree performs poorly, only predicting correctly the hard instances.

Chapter 5

A branching algorithm for solving STSP

In this chapter, we introduce a novel branching strategy to solve the STSP by addressing a sequence of minimum-weight 2-matching (MW2M) problems. The strength of our approach is the possibility to tackle this problem in polynomial time. This result implies that the minimum-weight triangle-free 2-factor problem is fixed-parameter tractable with respect to the number of triangles, but whether it is polynomial or NP-hard remains an open problem. Although the algorithm we propose has a different paradigm compared to standard B&C algorithms, the complexity of the instances we have introduced remains the same. Note that in this chapter we do not require the instances to satisfy the triangle inequality.

5.1 Background material

First of all, we start by re-stating some known definitions in graph theory.

Definition 10 (Cycle cover)**.** Given a graph G, a cycle cover of $G = (V, E)$ is a set $Z \subset E$ such that the subgraph $G' = (V, Z)$ consists of cycles only.

If the graph is weighted, it is an interesting problem to find the minimum cost cycle cover for a graph. One important aspect is to consider cycle covers that do not contain small cycles, aiming for cycles that are as large as possible. However, solving this problem becomes computationally difficult as it is equivalent to the NP-complete Hamiltonian cycle problem, which is known for its complexity. Note that, for undirected graphs, the smallest possible cycle is a triangle. Hence, it becomes of interest to study the following problem:

Definition 11 (minimum-weight triangle-free 2-factor problem)**.** Given a graph $G = (V, E)$, find a minimum cost cycle cover without triangles, i.e., cycles of length three.

In [66], where was first introduced the problem, the author proved some results about the complexity of finding the cheapest cycle covering without cycles of length less or equal to a certain k for different values of k. For instance, in [66] it is proved that it is NP-complete to find a cycle cover without cycles of length 4 when each node of the graph has degree 3. It is instead in P to compute a cycle cover without cycles of length 3 when each node of the graph has degree 3. The ultimate goal was to locate the cycle length whose avoidance causes the jump from polynomial time to NP-completeness in an arbitrary graph. Where such a jump is located it's still an open problem [61, 47].

5.2 A branching strategy for solving the STSP

Consider now the solution of the STSP using the DFJ formulation. As already stated in the introduction, the most effective solution procedure is to first solve the MW2M problem, and then separate iteratively all the subtour elimination constraints. Despite the number of subtour elimination constraints being exponential, in practice, only a few are needed to prove optimality. Table 2.3 in Chapter 2, in particular, shows that for simple instances, such as those presented in TSPLIB, the number of imposed constraints is low. Furthermore, efficient polynomial-time algorithms exist for finding the maximal violated subtour elimination constraint, and imposing it time by time makes the solution globally very fast. According to [18], this formulation works very well in practice, and this is due to essentially two reasons. Note that there exists a polynomial time algorithm for solving the MW2M problem [23], whose complexity is $O(nm)$, where $m = |E|$. As already done through this manuscript, we now focus on a complete graph, where $m = n(n-1) = O(n^2)$, making the complexity of the MW2M equal to $O(n^3)$. Consider now the problem of finding the MW2M plus just one subtour elimination constraint, denoted by $\boldsymbol{x}(\delta(S)) \geq 2$. Satisfying such constraint requires that at least two edges cross the cut from set S to the complement set S^c. Hence, the question is: Which nodes are involved in the transition from set S to S^c? As this transition point is not known beforehand, we create a subproblem for each vertex in $S = \{s_1, s_2, \dots, s_f\}$. Specifically, for each subproblem i, we require one or more edges to cross from vertex s_i to S. Let

$$\boldsymbol{x}(\delta(s_i, S)) := \sum_{\{i,j\} \in E \mid i = s_i, j \notin S} x_{ij}. \tag{5.1}$$

Asking that one or more edges cross from s_i to S^c is hence equivalent to enforce

$$\boldsymbol{x}(\delta(s_i, S)) \geq 1, \tag{5.2}$$

Where $\delta(s, S)$ stands for $\{\{i, j\} \in V \mid i = s \text{ and } j \notin S\}$. As we want to partition the space of the feasible solutions, we enforce that the vertices of the set $\{s_j, j < i\}$, are forbidden from neighboring non-subtour vertices. Hence, at each iteration, we partition the feasible space using subproblems, similarly to what is done in classical branch and bound, where the space is progressively partitioned and leaf nodes can be pruned due to infeasibility, bounding, or being proven optimal. Figure 5.1 shows the branching tree

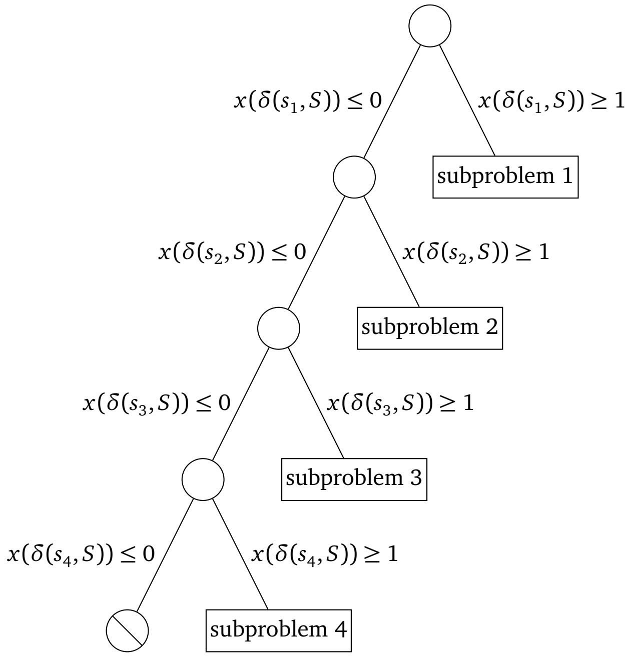


Figure 5.1. Expanded view of branching rule for $S = \{s_1, s_2, s_3, s_4\}$.

for a set of cardinality 4, where the space has been partitioned in

$$\boldsymbol{x}(\delta(s,S)) \leq 0 \qquad \boldsymbol{x}(\delta(s,S)) \geq 1.$$

Note that, thanks to the non negativity constraints, $\boldsymbol{x}(\delta(s,S)) \leq 0$ is equivalent to $\boldsymbol{x}(\delta(s,S)) = 0$.

5.2.1 Generating 2-matching problems equivalent to solving MW2M with subtour elimination constraints.

We introduce a sequence of graph operations that enable solving each subproblem as an MW2M problem in each node. The nodes requiring $\boldsymbol{x}(\delta(v,S)) \leq 0$ can be handled easily - we can simply remove the edges $\{\{i,j\} \in E \mid i = v, j \in S\}$. We will call this procedure *"edge deletion"*. For the $x(\delta(v,S)) \geq 1$ branch, we propose a technique that we called *"paw-splitting"*. This operation is similar to the commonly used node-splitting operation, but it involves connecting the two sides using a paw graph, as depicted in Figure 5.2. This gadget is built of the original vertex plus 3 new "dummy" vertices: v_L, v_M, v_R. Any edges originally extending from $S \setminus \{v\}$ to v are rerouted to v_L instead at the same cost. Last, we add the edge $\{v_R, v\}$. In total, this adds three new vertices and four zero-cost edges. The underlying idea behind this gadget derives from the trivial observation that a subtour elimination constraint can be satisfied in two different ways:

(a) We cross from set S to S^c and then continue in S^c, and then go back in S.

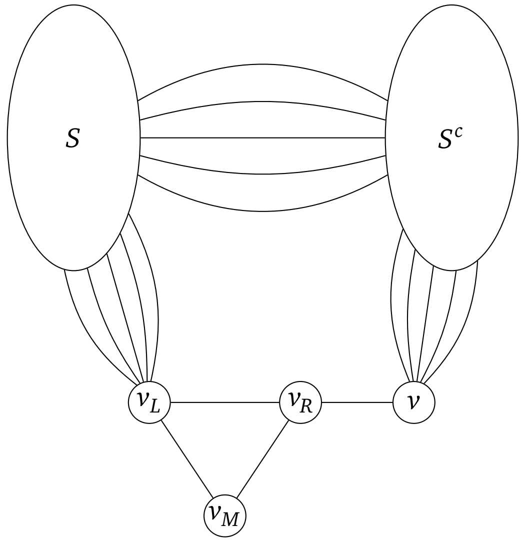


Figure 5.2. A *paw-splitting* gadget to impose $x(\delta(v,S)) \geq 1$.

(b) We cross from set S to S^c and then we go immediately back in S.

Point (a) is associated with leaving set S through node v, traversing the triangle $v_L \to v_M \to v_R$, and then re-entering node v, ultimately ending up in S^c.

On the other hand, point (b) can be satisfied by having two edges directly connecting from v to S^c. Each feasible solution of this second type generates a subtour with cost 0, involving nodes v_L, v_M, v_R, which does not impact the overall cost and can be discarded during the procedure of recovering the optimal solution.

To recover a MW2M with the imposed subtour elimination constraints satisfied for the original graph, simply take shortcuts through the new vertices and discard any cycles that consist solely of new vertices. First of all, the following Lemma holds.

Lemma 5. *Every 2-matching of $G = (V,E)$ that satisfies $\boldsymbol{x}(\delta(S)) \geq 2$, $S \subseteq V$, can be mapped to one of the following graphs: (i) G', obtained by applying paw-splitting to G trough nodes of S; (ii) G'', obtained by edge deletion to G. The costs of the two solutions are the same.*

Proof. Let $\boldsymbol{x}$ be the 2-matching on the graph G. If $\boldsymbol{x}(\delta(S)) \geq 2$ holds, then there exist $v,u \in S$ and $w_1,w_2 \in S^c$ such that $x_{vw_1} = x_{u,w_2} = 1$ for some $w_1,w_2 \in S^c$. Thanks to the degree constraints, we also have $x_{vz_1} = x_{u,z_2}$, where z_i belong to either S or S^c. If one among $z_1,z_2 \in S$, then such 2-matching can be obtained using the paw-gadget on v (respectively on u). If both $z_1,z_2 \in S^c$, then this can be obtained using edge-deletion on either u or v. The cost is trivially the same, as all the edges we add are at zero cost. □

Lemma 6. *If graph G' is obtained by applying edge deletion and/or paw-splitting recursively to G, then short-cutting a 2-matching of G' gives a 2-matching of G of the same cost.*

Proof. Let $G_0, G_1, \ldots, G_q$ be the sequence of graphs obtained from repeated edge deletions and/or paw-splitting, where $G_0 = G$ and $G_q = G'$. We prove that for each $j = 0, 1, 2, \ldots, q$, short-cutting a 2-matching of G_j gives a 2-matching of G_0 of the same cost. The statement clearly holds for $j = 0$. For induction purposes, suppose that the claim holds for some j with $0 \leq j < q$. We show the claim holds for G_{j+1}. In the first case, suppose that G_{j+1} is obtained by applying edge deletion to G_j. Then, any 2-matching of G_{j+1} is also a 2-matching of G_j, as G_{j+1} is a subgraph of G_j. So, the short-cut of the 2-matching of G_{j+1} is the same as the short-cut of the 2-matching of G_j, which is a 2-matching of G_0 by the induction hypothesis. In the other case, suppose that G_{j+1} is obtained by applying paw-splitting to G_j. The difference between G_j and G_{j+1} is the paw-splitting gadget. The nodes of the gadget can be involved in the solution in two different ways, as all the degree constraints must be satisfied. In one case, the gadget is traversed $v \to v_R \to v_M \to v_L \to w \in S^c$, and the cost of traversing this gadget is equivalent to the cost of traversing $v \to w$, as all the remaining costs on the edges are 0. As shortcutting this procedure leads to a 2-matching on G_j, the statement holds by induction. On the other hand, nodes v_R, v_M, v_L build a loop by themselves. In this case, discarding such a loop is at 0 cost and the 2-matching only involves nodes from G_j. $\square$

5.2.2 Algorithm for Given Subtour Constraints

Let's focus on the task of finding a minimum weight 2-matching with subtour elimination constraints with the minimum weight in the graph $G = (V, E)$. The matching's weight is determined by the costs $\boldsymbol{c}$ associated with the edges, and it must satisfy the subtour constraints imposed by given vertex subsets $S_1, S_2, \ldots, S_k$. To solve this problem, we propose a recursive algorithm called `ralg`. To use this algorithm, input the graph G and the collection of subsets $\mathscr{S} = S_1, S_2, \ldots, S_k$.

`ralg`$(G, \mathscr{S})$:

- if $\mathscr{S} = \emptyset$, return the cost of a minimum-weight 2-matching of G (or ∞ if infeasible)
- select a vertex subset $S = \{s_1, s_2, \ldots, s_f\}$ from the collection $\mathscr{S}$
- from G create subproblems $G_1, G_2, \ldots, G_f$ using the paw gadget and edge deletions
- return $\min\{\texttt{ralg}(G_j, \mathscr{S} \setminus \{S\}) \mid 1 \leq j \leq f\}$

We remark that the last subproblem G_f does not need the gadget, as we can simply remove the edges between s_f and the rest of S.

Theorem 12. *The algorithm* `ralg` *is correct and solves $|S_1||S_2| \cdots |S_k|$ minimum-weight 2-matching problems, each having at most $n+3k$ vertices and $m+4k$ edges, with a complexity of $k! \cdot O(n^4 + k^2)$.*

Proof. First, we consider the number of 2-matching problems. By the recursive nature of the algorithm, the bottom of the search tree has $|S_1||S_2|\cdots|S_k|$ leaf nodes in which a min-weight 2-matching problem is solved (when $\mathscr{S} = \emptyset$). The depth is k, and each application of the gadget adds 3 vertices and 4 edges, so the leaf nodes of the search tree have graphs with (up to) $3k$ more vertices and $4k$ more edges.

Now, let us consider the correctness. By recursively applying `ralg`, we recursively apply edge-deletion and paw gadget. Thus, the MW2M on the final graph is a feasible 2-matching for the original graph, thanks to Lemma 6. Let $\boldsymbol{w}$ such solution and z_w its cost. We claim that short-cutting $\boldsymbol{w}$ leads to an optimal solution, namely the original graph did not contain any 2-matching satisfying all the subtour elimination constraints that cost strictly less. If this was the case, consider such solution $\boldsymbol{x}$ at a cost z_x. By contradiction, it follows that $z_x < z_w$. Consider now and a MW2M $\boldsymbol{y}$ on the original graph at a cost z_y. By comparing $\boldsymbol{x}$ and $\boldsymbol{y}$, we can build from $\boldsymbol{y}$ a sequence of problems by paw-gadgets and edge deletion, where the final has a cost z_x. Let $\boldsymbol{x}'$ such solution. Note that $\boldsymbol{x}'$ is in the list we check in the recursive procedure of `ralg`, hence should have been returned at a certain point. But this is not the case, as it costs z_x, which costs strictly less than z_w, that is the final solution of `ralg`. Contradiction.

Regarding the complexity, In the base case where $\mathscr{S} = \emptyset$, the algorithm returns the cost of a minimum-weight 2-matching of G. This has a complexity of $O((3k+n)(4k+m)) = O(k^2+n^3)$ in the worst case of the complete graph, where the graphs have $O(n^2)$ edges. n each recursive call, the algorithm selects a vertex subset S from $\mathscr{S}$ and creates subproblems $G_1, G_2, \ldots, G_f$ using the paw gadget and edge deletions. The number of subproblems created depends on the size of S and the specific construction of the paw gadget. However, since S is a subset of the vertices, its size is at most n, so the number of subproblems is also at most n. Note that creating such subproblems can be done in constant time. The algorithm then recursively calls ralg on each of the subproblems and takes the minimum of the returned values. This results in a recursive tree structure, where each level corresponds to one vertex subset S and each node has f children (f depends on S and is at most n). Therefore, by denoting with $C(n,k)$ the complexity of the algorithm on the whole collection of sets, we have that

$$C(n,k) = kC(n,k-1) + nO(k^2+n^3)$$

as we solve a minimum weight 2-matching for each element of S, that has at most cardinality $n-3 = O(n)$. Hence, the whole complexity is

$$k!O(n^4+k^2).$$

□

Corollary 1. The minimum-weight triangle-free 2-matching problem is fixed-parameter tractable

Proof. Simply run the algorithm described by the previous theorem considering $\{S_1, S_2, \ldots, S_k\}$ of cardinality 3. □

5.2.3 Algorithm for TSP

Here, we provide a combinatorial branch-and-bound algorithm `cbb` for solving the STSP. Similar to the recursive algorithm, it uses the gadget and edge deletions to branch on subtours. However, it detects and imposes violated subtour constraints during the execution of the algorithm. Another change is that the final subproblem can be omitted, per an earlier observation, since we seek only STSP tours.

`cbb`(G):

- initialize $\mathcal{G} \leftarrow \{G\}$ and $z \leftarrow +\infty$
- while $\mathcal{G} \neq \emptyset$ do

  1. select and remove a subproblem G from $\mathcal{G}$
  2. find a minimum-weight 2-matching $\boldsymbol{x}$ of G, let z its cost.
  3. if the cost $\boldsymbol{c}^T\boldsymbol{x} \geq z$, then continue
  4. find the connected components (cycles) $C_1, C_2, \ldots, C_q$ of $\boldsymbol{x}$
  5. identify their original vertices, $V_j \leftarrow V_0 \cap V(C_j)$ for $j \in \{1, \ldots, q\}$
  6. if there is a unique nonempty V_j, then update $z \leftarrow \boldsymbol{c}^T\boldsymbol{x}$ and continue
  7. find the smallest nonempty $V_j = \{s_1, s_2, \ldots, s_f\}$ to branch on
  8. using the gadget, create subproblems $G_1, G_2, \ldots, G_{f-1}$
  9. $\mathcal{G} \leftarrow \mathcal{G} \cup \{G_1, G_2, \ldots, G_{f-1}\}$

- return z and the shortcut of $\boldsymbol{x}$.

In the terminology of branch-and-bound, line 1 is node selection, line 2 provides a (combinatorial) bound, line 3 prunes by bound, line 6 prunes by feasibility, and lines 7-9 give the disjunction/branching.

Theorem 13. *The algorithm* `cbb` *correctly solves the TSP in* $kO\left(n^3 + nk^2 + n^2x + k^3\right)$*, where k is the number of subtour needed.*

Proof. First, let's prove the complexity statement. As in the previous theorem, for each S_j we have to solve $O(n)$ minimum weight 2-matching. However, in this case, for each iteration, we have to find connected components ($O(n+m+7k)$)) and the minimum in a list having each time at most $\lfloor (n+3k)/3 \rfloor$ elements (As this is the maximum number of connected components). Hence we have

$$kO(n+3k)[O((n+3k)(m+4k)) + O(n+m+7k) + O(n)].$$

By doing the computation, we have

$$kO\left(n^3 + nk^2 + n^2x + k^3\right).$$

Regarding correctness, we can perform the same argument as in Theorem 12. Any feasible solution of `cbb` is hence a feasible STSP for the original graph and it costs the minimum. Otherwise, it is possible to recover a solution leading to one of the G_j explored in `cbb` leading to a strictly smaller cost. □

Remark 14. We want to clarify that this is not a polynomial algorithm for the STSP. Note that the theorem states a result that is polynomial in n and k. Although in practice just a few subtour elimination constraints are needed, in theory, $k = O(2^n)$.

5.3 Computational analysis

In this section, we assess the performances of the algorithm we propose "in practice". We would like to note that our claim is not to assert that our algorithm performs better than the state-of-the-art solver Concorde, as when comparing different algorithms, we are not comparing the algorithms "themselves", but rather their implementations. Note that the core of this chapter is not to focus on an efficient implementation. Rather, we aim to verify whether instances that prove to be complex for a B&C paradigm are also complex for our paradigm.

In particular, the following tests serve to answer three research questions:

1. Is the complexity of an instance of the STSP related to the algorithm used to solve it, or is it intrinsic to the instance itself? Are the instances that result empirically hard for Concorde even hard in general or just for Concorde? For example, Concorde uses several valid inequalities besides linear relaxation, which are not considered in our algorithm. To be more precise, our algorithm only relies on the integral solution of the 2-matching, which can be solved in polynomial time using [23]. In practice, an efficient implementation of such an algorithm was not available in the literature. For this reason, we opted to use our own implementation of Gurobi to solve the 2-matching. Note that this is insufficient on its own to justify the difference in the approaches, as Concorde utilizes several other routines. Nevertheless, our preliminary results indicate that instances challenging for Concorde face a bottleneck in solving the linear relaxation. Therefore, we conjecture that a procedure not relying on routines implemented in Concorde may exhibit different performances

2. Let
$$\alpha^{2M}(\boldsymbol{c}) := \frac{\text{STSP}(\boldsymbol{c})}{\text{2M}(\boldsymbol{c})}. \tag{5.3}$$
We call this value *2-matching gap*. Intuitively, it is natural to conjecture that if $\alpha^{2M}(\boldsymbol{c})$ is small, our algorithm requires fewer iterations, as the initial relaxation is already very close to the optimal solution, namely a tour. We have already verified that such a relationship between the integrality gap with respect to LP relaxation and the complexity of an instance is not present.

3. Furthermore, we are interested in understanding how much the graph and the number of subproblems on which we iteratively solve the 2-matchings grows as the iterations increase, and whether this could be a hurdle to consider in case of future development involving an efficient implementation of the proposed algorithm.

Implementation details. The branching algorithm has been implemented in Python, using Gurobi as an optimization solver. At each iteration, we branch and the smallest subtour elimination constraint we find. We ignore, namely we do not branch, on subtour elimination constraints related to the paw-gadget nodes.

Results We perform two different tests: one on the TSPLIB and the other on the HardT-SPLIB we have introduced in Chapter 3. Before running these experiments, we sorted the instances from TSPLIB and Hard-TSPLIB by the number of nodes and then solved them one by one, retrieving for each instance: the number of nodes in the resulting graph - the one leading to the optimal solution after shortcutting- the number of subproblems that were explored, the value of $\alpha^{2M}(\boldsymbol{c})$, and the runtime. Table 5.1 and 5.2 report these results.

Regarding the TSPLIB, we observed that when applying our procedure to the brazil58 instance, the number of subproblems to handle becomes too large, and the computational resources we allocated, as described in the introduction, are not sufficient. First of all, it is interesting to note that there is no clear relation between $\alpha^{2M}(\boldsymbol{c})$ and the empirical hardness. For instance, gr48 and hk48 have 48 nodes and a similar value of $\alpha^{2M}(\boldsymbol{c})$. However, att48 requires 10 times the computational time of gr48. The reason lies in the number of subproblems explored. The more subproblems are needed to prove optimality and the bigger the number of nodes, the more complex is one instance with respect to this procedure.

The observation with which we started this section is confirmed: the instances of the Hard-TSPLIB remain difficult even for this approach. For example, for n = 16, instance 16038 proves to be much more challenging than ulysses16, both in terms of computational times and subproblems to explore. We also observe that in the Hard-TSPLIB, the phenomenon of out-of-memory still occurs, but with instances that have a much smaller size. In particular, instances with 30 nodes cannot be solved with the computational resources we have allocated.

5.4 Conclusion

In this chapter, a new branching strategy for solving STSP has been proposed. The strategy involves solving a sequence of MW2M problems on a sequence of extended graphs, incorporating subtour elimination constraints. To do so, we introduce some operations such as edge deletion and paw-gadget extension. As a consequence, the

Instance	n	$\overline{n}$	# subp.	$\alpha^{2M}(\boldsymbol{c})$	Runtime
burma14	14	45	23	1.107	0.566
ulysses16	16	326	31	1.122	2.643
gr17	17	94	26	1.238	1.335
ulysses22	22	8704	46	1.138	107.854
fri26	26	130	41	1.061	4.251
bayg29	29	379	41	1.040	15.687
bays29	29	106	38	1.037	4.251
swiss42	42	137	57	1.038	10.050
dantzig42	42	777	57	1.082	85.857
hk48	48	2982	72	1.024	296.219
att48	48	27944	78	1.054	2899.279
gr48	48	2327	78	1.050	307.969
eil51	51	66	60	1.017	9.675
berlin52	52	46	61	1.053	6.267

Table 5.1. Instances from TSPLIB tested with the branching procedure described in this chapter. Instance name, number of nodes, number of nodes in the final graph on which the 2-matching is solved to obtain the optimal tour, Number of subproblems, $\alpha^{2M}(\boldsymbol{c})$, and runtime.

minimum-weight triangle-free problem is fixed-parameter tractable. The proposed algorithm presents a different paradigm compared to standard branch and bound algorithms, hence, it is interesting to assess the impact of such an algorithm on the hard-to-solve instances we have introduced. First, we observe that instances challenging for Concorde remain challenging when employing this branching strategy, suggesting that hardness is an inherent characteristic of these instances.

Second, mirroring our previous investigations into the integrality gap, we noticed that the 2-matching gap, which quantifies the difference between the problem's value at the root node of our branching tree and the optimal value, might somehow correlate with the degree of difficulty. However, as previously discussed, this is not always true.

Third, we note a substantial increase in the number of subproblems and the dimension of the graph when extended with gadgets (See for instance the fourth columns of Table 5.1 5.2: this is the number of 2-matching problems to evaluate). This presents a noteworthy challenge when considering the design of an engineered implementation of this algorithm.

Instance	n	$\overline{n}$	# subp.	$\alpha^{2M}(\boldsymbol{c})$	Runtime
10001	10	50	13	1.152	0.421
10007	10	26	16	1.101	0.324
10008	10	53	13	1.155	0.439
10010	10	16	13	1.155	0.232
11675	11	86	14	1.181	0.539
12290	12	168	18	1.129	3.766
14850	14	718	17	1.199	8.562
15002	15	1735	27	1.111	39.084
15005	15	1244	24	1.133	40.172
15007	15	1514	27	1.093	65.225
16038	16	1435	22	1.211	53.645
20004	20	28896	32	1.217	1507.004
20007	20	23696	29	1.142	978.879
20009	20	34959	56	1.129	2050.439
20012	20	28013	32	1.129	1452.976
gr24	24	414195	42	1.316	12365.229
25001	25	207212	100	1.152	14955.405
25004	25	207810	85	1.152	15406.681
25006	25	92907	40	1.196	7602.673

Table 5.2. Instances from Hard-TSPLIB introduced in Chapter 3 tested with the branching procedure described in this chapter. Instance name, number of nodes, number of nodes in the final graph on which the 2-matching is solved to obtain the optimal tour, Number of subproblems, $\alpha^{2M}(\boldsymbol{c})$, and runtime.

Chapter 6

Computing the Exact Integrality Gap of small Asymmetric Traveling Salesman Problem via Vertex Analysis

This chapter has two main contributions. First, we provide lower bounds for the integrality gap for the Metric ATSP for $n \leq 18$ by the vertices of the polytope P^n_{ASEP}, revising the strategy already adopted in [24]. For $n \leq 15$, we recover the same results as known in the literature. We combine a pivoting-based algorithm and an inductive strategy, that generates vertices from P^n_{ASEP} to P^{n+1}_{ASEP}, in order to obtain as many classes of vertices as possible. Our pivoting-based strategy results in being more effective with respect to what is provided in [24], but as it is time-consuming, we can only use it for $n \leq 10$. As a by-product of such computation, we generate Metric ATSP instances. Second, we also analyze them in terms of computational complexity, looking for connections between a high integrality gap and the computational hardness of the instance. We show that our procedure also exhibits some hard-to-solve instances.

This Chapter is organized as follows: in Section 6.1 we explain our methodology starting by revising background material. In Section 6.2 the symmetries of the polytope P^n_{ASEP} are explored. In particular, a group of automorphisms is identified, which enables us to explicitly generate the orbits of the vertices, that is the classes of isomorphic extreme points. Section 6.3 revises the basis of the pivoting algorithm and illustrates how we will use it in our pipeline for vertices generation. Section 6.4 illustrates a technique that generates vertices of P^{n+1}_{ASEP} from vertices of P^n_{ASEP}. Section 6.6 discusses the results obtained. This chapter is part of a broader work that we will publish soon, once further details are finalized.

6.1 Background Material

In this section, we revise the technique presented in [24], which is the starting point of our work. Based on the idea firstly introduced in [6] for the symmetric TSP and in [24] for the asymmetric TSP, we can divide the cost vector $\boldsymbol{c}$ by the optimal value ATSP($\boldsymbol{c}$). This normalization provides a new cost vector $\boldsymbol{c}'$ that maintains the triangle inequalities and preserves the value of the integrality gap, namely $\alpha(\boldsymbol{c}) = \alpha(\boldsymbol{c}')$. Therefore, it becomes possible to confine ourselves to metric cost vectors $\boldsymbol{c}$ for which ATSP($\boldsymbol{c}$) $= 1$. Consequently, we can convert the original problem of maximization into a problem of minimization:

$$\frac{1}{\alpha_n} = \min_{\substack{\boldsymbol{c}\geq 0 \text{ is metric,} \\ \text{ATSP}(\boldsymbol{c})=1}} \text{ASEP}(\boldsymbol{c}) = \min_{\substack{\boldsymbol{c}\geq 0 \text{ is metric,} \\ \text{ATSP}(\boldsymbol{c})=1}} \min_{\boldsymbol{x}\in P^n_{\text{ASEP}}} \boldsymbol{c}^T \boldsymbol{x}. \tag{6.1}$$

[24] enumerates the vertices of P^n_{ASEP} to find a solution to (6.1). That is, they represent P^n_{ASEP} as the convex combinations of its vertices which are finitely many. Let us denote with $\mathscr{X}^n_{\text{ASEP}}$ the set of all vertices of P^n_{ASEP}. They claim that it is possible to compute the exact integrality gap α_n by solving one LP for each vertex of $\mathscr{X}^n_{\text{ASEP}}$. This is a direct consequence of the fact that there exists, for each LP, at least an optimal solution attained at one vertex.

The authors called this problem $Gap(\boldsymbol{x})$, and showed that

$$\alpha_{\text{ATSP}} = \max_{\boldsymbol{x}\in\mathscr{X}^n_{\text{ASEP}}} \frac{1}{\text{Gap}(\boldsymbol{x})}.$$

Let $\mathscr{T}_n$ be the collection of the incidence vectors $\bar{\boldsymbol{z}} \in \mathbb{R}^{|A|}$ of all possible tours of A.

Intuitively, we want to find a cost vector $\boldsymbol{x}$, which minimizes the value of ASEP($\boldsymbol{c}$) and keeps the value of ATSP($\boldsymbol{c}$) fixed to 1. We also want the optimal solution of ASEP($\boldsymbol{c}$) to be attained precisely at the vertex we provide as input. Thus, given a vertex $\overline{\boldsymbol{x}}$ of P^n_{ASEP}, the $Gap(\overline{\boldsymbol{x}})$ problem is defined in [24] as

$$\min \quad \sum_{i=1}^{n} \sum_{j=1,j\neq i}^{n} \overline{x}_{ij} c_{ij} \tag{6.2}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} \sum_{j=1,j\neq i}^{n} \overline{x}_{ij} z_{ij} c_{ij} \geq 1 \qquad \forall \boldsymbol{z} \in \mathscr{T}_n \tag{6.3}$$

$$c_{ij} \leq c_{ik} + c_{jk} \qquad \forall i,j,k \in V \tag{6.4}$$

$$c_{ij} \geq 0 \qquad \forall (i,j) \in A \tag{6.5}$$

$$c_{ij} - y_i^{\text{out}} - y_j^{\text{in}} - \sum_{S\in\mathscr{S}(\overline{\boldsymbol{x}})} (d_S \mid (i,j) \in \delta(S)) \geq 0 \qquad \forall (i,j) \in A \backslash A(\overline{\boldsymbol{x}}) \tag{6.6}$$

$$c_{ij} - y_i^{\text{out}} - y_j^{\text{in}} - \sum_{S\in\mathscr{S}(\overline{\boldsymbol{x}})} (d_S \mid (i,j) \in \delta(S)) = 0 \qquad \forall (i,j) \in A(\overline{\boldsymbol{x}}) \tag{6.7}$$

$$d_S \geq 0 \qquad \forall S \in \mathscr{S}(\overline{\boldsymbol{x}}) \tag{6.8}$$

Where

$$A(\boldsymbol{x}) = \{(i,j) \in A \text{ such that } x_{ij} > 0\}$$

and

$$\mathscr{S}(\boldsymbol{x}) = \{S \subset A \text{ such that } \boldsymbol{x}(\delta(S)) = 1 \text{ and } 2 \leq |S| \leq n-2\}. \tag{6.9}$$

Constraints (6.3) ensure that the optimal solution $\boldsymbol{c}^*$ of Gap($\boldsymbol{x}$) for every $\boldsymbol{x}$ is such that ATSP($\boldsymbol{c}^*$) = 1. Constraints (6.4) and (6.5) ensure that the costs are non-negative and the triangle inequality holds. Note that the case of $c_{ij} = 0$ is allowed. Constraints (6.6)–(6.8) ensure that

$$\boldsymbol{x}^* \in \arg\min_{\boldsymbol{x} \in P^n_{\text{ASEP}}} \boldsymbol{c}^*\boldsymbol{x} \iff \boldsymbol{c}^* \in \arg\min_{\boldsymbol{c},\boldsymbol{y},\boldsymbol{d} \text{ satisfy}(6.3)\text{——}(6.8)} Gap(\boldsymbol{x}^*). \tag{6.10}$$

Thus, the arg min $\boldsymbol{c}^*$ of the program $Gap(\boldsymbol{x}^*)$ is such that, once ASEP($\boldsymbol{c}^*$) is solved, the minimum is attained precisely at $\boldsymbol{x}^*$. We refer the interested reader to [24] for further details. What is of interest to our extent is this program as a way to compute the exact value of the integrality gap attained by all the ATSP instances having as optimal ASEP solution the input vertex.

6.2 Symmetry group of P^n_{ASEP}

A basic issue arising at this point is the nature of the symmetries of the polytope P^n_{ASEP}. First of all, we recall that ATSP can be modeled as a problem on a directed weighted graph $D_n = (V_n, A_n)$. By observing constraints (1.2) – (1.4) and (1.11), which define P^n_{ASEP}, two important considerations can be made: the first is that the costs c_{ij} do not appear in the constraints. In other terms, the feasible solutions do not depend on the weights on the arc A_n. The second consideration is that, due to the structure of the constraints, they are not affected by a permutation of the node indices. More precisely, any relabeling of the nodes $i \in V_n$ induces just an internal permutation of the constraint groups (1.2), (1.3) and (1.4), which leaves the feasible region unchanged. Consequently, it is possible to identify a group of automorphisms of the polytope P^n_{ASEP} induced by the symmetric group S_n of permutations of the nodes $i \in V_n$.

In order to formulate the automorphisms of $\mathscr{X}^n_{\text{ASEP}}$ explicitly, it is useful to convert the feasible solutions into matrix form. More precisely, we can rewrite any feasible solution $\boldsymbol{x} \in P^n_{\text{ASEP}}$ as a matrix $\boldsymbol{X} \in [0,1]^{n\times n}$, with the corresponding components x_{ij} for $i \neq j$ and setting $x_{ij} = 0$ for $i = j$.

Now, consider the symmetric group of permutations on n elements S_n. Let $\pi \in S_n$, such that $\pi : i \mapsto \pi(i)$, be a permutation of the nodes $i \in V_n$. For any feasible solution $\boldsymbol{x} \in P^n_{\text{ASEP}}$, with matrix representation $\boldsymbol{X}$, it is possible to generate a new feasible solution $\boldsymbol{x}'$, expressed by a matrix $\boldsymbol{X}'$, by applying the permutation π to the nodes of the graph. It must hold

$$x'_{\pi(i)\pi(j)} = x_{ij} \quad \forall i,j \in V_n.$$

This means that $\boldsymbol{X'}$ is obtained by permuting rows and columns of $\boldsymbol{X}$ according to π. Let us define the permutation matrix $\boldsymbol{P}_\pi = (p_{ij})$ associated to $\pi \in S_n$ as follows

$$p_{ij} = \begin{cases} 1 & \text{if } i = \pi(j) \\ 0 & \text{otherwise.} \end{cases}$$

It is easy to see that in the product $\boldsymbol{P}_\pi \boldsymbol{X}$, the permutation matrix $\boldsymbol{P}_\pi$ permutes the rows of $\boldsymbol{X}$ according to π. Since also the columns of $\boldsymbol{X}$ have to be permuted, we apply $\boldsymbol{P}_\pi$ to the transpose of $\boldsymbol{P}_\pi \boldsymbol{X}$. Thus

$$\boldsymbol{X}' = \left(\boldsymbol{P}_\pi(\boldsymbol{P}_\pi \boldsymbol{X})^T\right)^T = \left(\boldsymbol{P}_\pi \boldsymbol{X}^T \boldsymbol{P}_\pi^T\right)^T = \boldsymbol{P}_\pi \boldsymbol{X} \boldsymbol{P}_\pi^T.$$

In conclusion

$$\boldsymbol{X}' = \boldsymbol{P}_\pi \boldsymbol{X} \boldsymbol{P}_\pi^T.$$

A notion that we will widely use is the *isomorphism between digraphs*. Hence, let us recall some helpful definitions.

Definition 12 (Support digraph)**.** Let $\boldsymbol{x} \in \mathscr{X}^n_{\text{ASEP}}$. The weighted support digraph of the vertex $\boldsymbol{x}$ is the graph $H(\boldsymbol{x})$ defined on the set of nodes V_n, having an arc (i,j) with the weight x_{ij}, if and only if $x_{ij} > 0$.

Definition 13 (Isomorphism between weighted graphs)**.** Two graphs $D = (V, A_D)$ and $F = (V, A_F)$ on n nodes are isomorphic if there exists a permutation π of V such that $(u,v) \in A_D \iff (\pi(u), \pi(v)) \in A_F$. Furthermore, the weights of the edges must satisfy $c_{\pi(u)\pi(v)} = c_{uv}, \forall u, v \in V_n$.

Example 15. Consider $\boldsymbol{x} \in P^4_{\text{ASEP}}$ defined by

$$\boldsymbol{x} = \left(\frac{1}{2}, 0, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, 0, \frac{1}{2}, 0, \frac{1}{2}, 0, \frac{1}{2}, \frac{1}{2}\right)^T.$$

Its matrix version is hence

$$\boldsymbol{X} = \begin{pmatrix} 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 \end{pmatrix}.$$

This feasible solution is associated with the support graph in Figure 6.1, left. Let $\pi = (1,2,3,4)$ be the permutation such that $\pi(1) = 2,\ \pi(2) = 3,\ \pi(3) = 4,\ \pi(4) = 1.$ Thus, in this case

$$\boldsymbol{P}_\pi = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

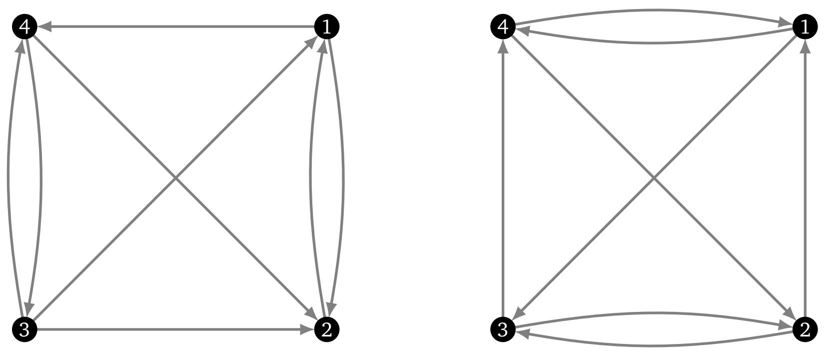


Figure 6.1. Two isomorphic vertices obtained via vertex permutation. Each arc is weighted $\frac{1}{2}$.

and we obtain

$$X' = P_\pi X P_\pi^T = \begin{pmatrix} 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \end{pmatrix},$$

which corresponds to the graph in Figure 6.1 on the right.

Consider now the action of the symmetric group S_n on the polytope P^n_{ASEP}. Let be, for a given $\pi \in S_n$

$$\begin{aligned} g_\pi : P^n_{\text{ASEP}} &\to P^n_{\text{ASEP}} \\ X &\mapsto P_\pi X P_\pi^T \end{aligned}$$

With a slight abuse of notation, we will write $g_\pi(x)$ and $g_\pi(X)$, interchangeably.

Consider the set of all the feasible permutations of elements of P^n_{ASEP}, namely

$$G(n) := \{g_\pi \mid \pi \in S_n\}.$$

Then, the following holds.

Theorem 16.

1. *The set $G(n)$ is a group.*

2. *X is a vertex of $P^n_{ASEP} \iff g_\pi(X)$ is a vertex of P^n_{ASEP} for all X.*

3. *The support graph of X is isomorphic to the support graph of $g_\pi(X)$.*

Proof. Let $X' := g_\pi(X)$.

1. We observe that $G(n)$ inherits the properties of S_n. More specifically:

   - The identity of $G(n)$ is defined by g_{id} where id is the identity of S_n: $\boldsymbol{P}_{\pi_{id}}$ is the identity matrix, and thus $\boldsymbol{X}' = \boldsymbol{X}$.
   - The inverse element of $g_\pi \in G(n)$, namely g_π^{-1} is obtained through $g_{\pi^{-1}}$, since that $\boldsymbol{P}_{\pi^{-1}} = \boldsymbol{P}_\pi^{-1}$.
   - To show the associativity, we have to show that

   $$\left(g_{\pi_1} g_{\pi_2}\right) g_{\pi_3} = g_{\pi_1} \left(g_{\pi_2} g_{\pi_3}\right) \qquad \forall \boldsymbol{X},\ \forall \pi_1, \pi_2, \pi_3 \in S_n.$$

   This is a direct consequence of the associativity of matrix products and the transposition of the product of matrices.

2. Proposition 4.3.1 in [24] proves an equivalent result. In summary, as $\boldsymbol{x}$ is a vertex, is a unique solution of a set of equalities. By applying π to the indices of the coefficients of such equalities, we obtain a set of valid equalities for $g_\pi(\boldsymbol{x})$.

3. As $g_\pi(\boldsymbol{X})$ is mapping from $\boldsymbol{X}$ through π, we have that the support graph remains unchanged.

□

Claim 3 of Theorem 16 states that the classes of vertices with isomorphic support graphs, as introduced in [24] are equivalent to the orbits of the action of the group $G(n)$ on $\mathscr{X}^n_{\text{ASEP}}$. Note that the action is well-defined for any point in P^n_{ASEP}, but for our purposes, it is enough to focus on the set of vertices $\mathscr{X}^n_{\text{ASEP}}$. Let G be a group that acts on a set S and let $s \in S$.

Definition 14 (Orbit)**.** Let G be a group and denote with g its elements; Let S be a set and denote with $s \in S$ it's element. Let $g \cdot s$ the action of the element $g \in G$ on element $s \in S$. The set

$$O_s = \{g \cdot s \in S \mid g \in G\}$$

is the *orbit* of the element s.

Definition 15 (Stabilizer)**.** Consider the same notation of the previous definition, The set

$$G_s := \{g \in G \mid g \cdot s = s\}$$

is the *stabilizer* of G with respect to $s \in S$.

This is a subgroup of G. In our framework, since $G(n)$ is a group of automorphisms of the polytope P^n_{ASEP}, we define the orbit of a feasible solution $\boldsymbol{x} \in P^n_{\text{ASEP}}$ as the set

$$O_{\boldsymbol{x}} = \{g_\pi(\boldsymbol{x}) \mid g_\pi \in G(n)\},$$

and its stabilizer as

$$G_{\boldsymbol{x}} = \{g_\pi \in G(n) \mid g_\pi(\boldsymbol{x}) = \boldsymbol{x}\}.$$

We observe that the orbits have different lengths. More specifically, by the so-called *orbit-stabilizer theorem* we have the relation

$$|G(n)| = |G_{\boldsymbol{x}}||O_{\boldsymbol{x}}|.$$

The result regarding the stabilizer being a group, as well as the orbit-stabilizer theorem, can be found by the interested reader in [3].

Note that $|G(n)| = n!$. Furthermore, we know that the cardinality of the stabilizer divides the cardinality of the group since the stabilizer is a subgroup of $G(n)$. A relevant point for the geometry of P^n_{ASEP} is the size of the orbits of its vertices. A trivial upper bound for the size of each orbit is given by $|G(n)| = n!$. On the other hand, we observe that the shortest orbit is always the orbit of the integer vertices, namely one of the tours. As we have precisely $(n-1)!$ tours, we can derive the following

Lemma 7. *For each $\boldsymbol{x}$ an integer vertex, $|G_{\boldsymbol{x}}| = n$.*

Proof. We have that there are $(n-1)!$ tours, all belonging to the same orbit. Hence, thanks to the orbit-stabilizer theorem

$$|G_{\boldsymbol{x}}| = \frac{|G(n)|}{O_{|\boldsymbol{x}|}} = \frac{n!}{(n-1)!} = n.$$

□

Note that we claim that the stabilizer of the tour is the biggest possible. Therefore, our result suggests that $1 \leq |G_{\boldsymbol{x}}| \leq n$. Hence, this would imply

$$(n-1)! \leq |O_{\boldsymbol{x}}| \leq n!.$$

This means that, in general, orbits are large, and many vertices belong to the same orbit.

Remark 17. Note that vertices belonging to the same orbit have the same integrality gap. This follows directly from Proposition 4.3.1 in [24]: If $\boldsymbol{x}' \in O_{\boldsymbol{x}}$, then there exists $g_\pi \in G(n)$, such that $\boldsymbol{x}' = g_\pi(\boldsymbol{x})$. Let $\boldsymbol{c}^* \in \arg\min Gap(\boldsymbol{x})$.

$$Gap(\boldsymbol{x}) = \boldsymbol{c}^{*T}\boldsymbol{x} = g_\pi(\boldsymbol{c}^{*T})g_\pi(\boldsymbol{x}) = Gap(\boldsymbol{x}').$$

Where the first equality holds by definition; the second holds because applying a permutation to both entries of the scalar product does not change the value of the scalar product; the last equality is because if there existed $\boldsymbol{c}'$ such that $\boldsymbol{c}'^T\boldsymbol{x}' < \boldsymbol{c}^{*^T}\boldsymbol{x}$, then $g_\pi^{-1}(\boldsymbol{c}')$ would lead to a solution of minor cost w.r.t the one provided by $\boldsymbol{c}^{*^T}$, making this latter non-optimal.

6.3 Pivoting moves between vertices

The construction presented so far allows us to explore the orbits, that is given a vertex $\boldsymbol{x} \in \mathcal{X}^n_{\text{ASEP}}$, we can generate all the vertices in its orbit $O_{\boldsymbol{x}}$. In this section, we use some strategies yet introduced for the design of the Simplex Method [20] to recover vertices from other orbits. The idea is similar to one presented in [24]. Instead of pivoting only on the vertices with unique cobases, we pivot on all of them. Unlike the approach adopted in [24], in the case of $n = 6$, we can recover all vertices using pivoting moves. However, this approach exhibits high computational complexity and therefore can only be exploited for a very small number of nodes, particularly until $n \leq 10$.

6.3.1 Standard form polyhedron for P^n_{ASEP}

Note that P^n_{ASEP} is defined through equality and inequality constraints. Thus, we add *slack variables* to recover the polytope in standard form. In our case, we need one slack variable for each constraint (1.4). Let n_S be the number of subtour elimination constraints. The number n_S can be determined by counting the number of sets S such that $2 \leq |S| \leq n-2$, namely $n_S = 2^n - 2n - 2$. Let $\boldsymbol{x} \in \mathbb{R}^{m+n_S}$ and let $\overline{\boldsymbol{A}} \in \mathbb{N}^{(2n+n_S)\times(m+n_S)}$ and $\overline{\boldsymbol{b}} \in \mathbb{N}^{(2n+n_S)}$ defined by the following set of equality constraints:

$$\text{subject to} \quad \sum_{i=1}^{n} x_{ij} = 1, \qquad j = 1, \ldots, n \tag{6.11}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad i = 1, \ldots, n \tag{6.12}$$

$$\sum_{i \in \delta(S)} x_{ij} - x_S = 1, \qquad \forall S \subset V \text{ such that } 2 \leq |S| \leq n-2. \tag{6.13}$$

We define the standard form polyhedron of P^n_{ASEP} as

$$\overline{P}^n_{\scriptscriptstyle ASEP} = \{\boldsymbol{x} \in \mathbb{R}^{m+n_S} \text{ such that } \overline{\boldsymbol{A}}\boldsymbol{x} = \overline{\boldsymbol{b}} \text{ and } \boldsymbol{x} \geq 0\}. \tag{6.14}$$

Lemma 8. *Let $r(n) := rank(\overline{\boldsymbol{A}})$. Then, it holds $r(n) = 2n + n_S - 1$.*

Proof. As it has been yet observed in [5] we have precisely $2n - 1$ linearly independent constraints among the degree constraints (6.11)–(6.12). Each subtour elimination constraint has a -1 at the column $m + i$, where $i \in \{1, \ldots, n_S\}$. As it is the only non-zero entry in the $(m+i)^{th}$ column of $\boldsymbol{A}$, we have that all such constraints are linearly independent (and also independent from the degree constraints). □

6.3.2 Pivoting strategy

Consider a linear program (LP) in the standard form

$$\begin{array}{ll} \max & \boldsymbol{c}^T\boldsymbol{x} \\ \text{s.t} & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \geq 0. \end{array} \tag{6.15}$$

The simplex method starts with an initial feasible solution and iteratively improves it by performing pivoting steps. A pivot element is selected to determine the direction of improvement, and variables are exchanged to increase the objective function value. This process continues until no further improvement is possible, resulting in an optimal solution.

In our framework, the objective function is disregarded, and an exploration of neighboring vertices is performed. The pivoting step involves selecting a variable from the set of basic variables and letting it enter the base, while another variable is forced to exit. We have observed that all the vertices we found are degenerate, hence the number of non-zero entries is strictly less than $r(n)$. As a consequence, more bases are feasible for a given vertex $\boldsymbol{x}$ of P^n_{ASEP}.

Hence, given a vertex, we explore all possible feasible bases. Such bases contain all the non-zero entries by definition, but they are not enough to be a stand-alone basis, as they are always strictly less than $r(n)$. Hence, we include also all possible combinations of zeros to saturate the base's dimension. More specifically, variables in basis should be chosen among the $(i, j) \in A$ such that $x_{ij} = 0$ and the $2 \leq |S| \leq n-2$ such that $x_S = 0$. Thus we apply the primal simplex method by considering all the possible feasible bases. For each vertex of $\overline{P}^n_{ASEP}$, let $n_{\overline{Z}}(\boldsymbol{x})$ the number of non-zero entries of $\boldsymbol{x}$. Each of these variables is forced to remain base. Let hence $n_Z(\boldsymbol{x})$ the number of zeros of $\boldsymbol{x}$. It holds

$$n_Z(\boldsymbol{x}) + n_{\overline{Z}}(\boldsymbol{x}) = m \qquad m = |A|.$$

To clarify this notation consider the integer vertex representing a tour for $n = 4$, that is $1 \to 2 \to 3 \to 4 \to 1$. In terms of arc variables, 4 are non-zero, while 8 are zeros. Here $r(n) = 13$. Let $\mathscr{S} := \{\{1,2\},\{1,3\},\{1,4\},\{2,3\},\{2,4\},\{3,4\}\}$. We have

$$x_{\{\{1,2\}} = x_{\{1,4\}} = x_{\{2,3\}} = x_{\{3,4\}} = 0 \qquad x_{\{1,3\}} = x_{\{2,4\}}.$$

Hence,

$$n_Z(\boldsymbol{x}) = 12 \qquad n_{\overline{Z}}(\boldsymbol{x}) = 8.$$

In conclusion, the remaining $r(n)-n_{\overline{Z}}(\boldsymbol{x})$ entries should be chosen among the entries equal to zero. Thus we need to test

$$\binom{n_Z(\boldsymbol{x})}{r(n) - n_{\overline{Z}}(\boldsymbol{x})}$$

different combinations. Vertices obtained through a pivoting step are called adjacent and share a number of tight and linearly independent constraints equal to the dimension of the space minus one. Thus, given $\boldsymbol{x}$ vertex of $\overline{P}^n_{ASEP}$, we can list all the elements of $\mathscr{N}(\boldsymbol{x})$ through a pivoting step. More formally:

Definition 16 (Neighborhood)**.** We call *neighborhood of a vertex* $\boldsymbol{x}$, and denote it with $\mathcal{N}(\boldsymbol{x})$ the set of all the vertices adjacent to it, that is, the set of all the vertices that can be obtained through a pivoting step from $\boldsymbol{x}$.

Note that exhaustively listing the $\mathcal{N}(\boldsymbol{x})$ for a given $\boldsymbol{x}$ is infeasible in practice. For instance, for $n = 6$, each tour has 42 zeros out of 80 variables, and $r(6) = 61$. It holds

$$\binom{42}{61-(80-42)} = \binom{42}{23} = 446,775,310,800$$

combinations to explore. With only the purpose of obtaining as many representatives of vertices as possible, we can limit the number of iterations to a certain number or time limit. In this way, we aim to balance between *exploitation* of a single vertex, that is how much we are interested in the neighborhood of a vertex and *exploration* of the full polytope, by moving from the neighborhood of one vertex to another. Hence, given a time limit T, our function

$$\mathcal{N}'(\boldsymbol{x}) = \text{Pivoting}(\boldsymbol{x}, T) \tag{6.16}$$

returns only a subset of $\mathcal{N}(\boldsymbol{x})$, namely the adjacent vertices that the strategy is capable of finding within a time limit. More details follow in Section 6.5.

Another interesting result that allows us to reduce the number of vertices we want to pivot on is the following:

Lemma 9. *Let* $\boldsymbol{x}^1 \in \mathcal{X}^n_{ASEP}$ *and* $\boldsymbol{x}^2 \in O_{\boldsymbol{x}^1}$*, where* $\boldsymbol{x}^2 = g_\pi(\boldsymbol{x}^1)$*. Hence,* $\boldsymbol{x}^3 \in \mathcal{N}(\boldsymbol{x}^1)$ *if and only if* $\boldsymbol{x}^4 \in \mathcal{N}(\boldsymbol{x}^2)$*, where* $\boldsymbol{x}^4 := g_\pi(\boldsymbol{x}^3)$*,*

Proof. First, we use the transformation π to sort the rows of the constraint matrix, by mapping each degree constraint associated with i to $\pi(i)$, and each subtour elimination constraint associated with $S = \{s_1, \ldots, s_f\}$ to $\pi(S) := \{s_1, \ldots, s_f\}$ and each non-negative constraint accordingly. With a slight abuse of notation, we will call $\pi(i)$ the mapping of the i^{th} constraint. Note that $\boldsymbol{x}^3$ shares exactly $m-1$ linearly independent and tight constraints with $\boldsymbol{x}^1$. $i_1, \ldots, i_{m-1}, i^1_m$ are this set of constraints of $\boldsymbol{x}^1$ and $i_1, \ldots, i_{m-1}, i^3_m$ is the one of $\boldsymbol{x}^3$, then, $\pi(i_1), \ldots, \pi(i_{m-1}), \pi\left(i^1_m\right)$ is associated to $\boldsymbol{x}^2$ and $\pi(i_1), \ldots, \pi(i_{m-1}), \pi\left(i^3_m\right)$ is hence associated with $\boldsymbol{x}^4$. Thus, $\boldsymbol{x}^4 \in \mathcal{N}(\boldsymbol{x}^2)$. □

Hence, if we have two isomorphic vertices, it is enough to pivot only one of them, as the second will have the same neighborhood (up to a transformation g_π).

Let $\boldsymbol{y_1}, \ldots, \boldsymbol{y_{n_x}} \in \overline{P}^n_{ASEP}$ that can be obtained from $\boldsymbol{x} \in P^n_{\text{ASEP}}$ trough a pivoting step and such that $\boldsymbol{y_k} \neq \boldsymbol{x}$ for all $k \in \{1, \ldots, n_x\}$ (Non degenerate pivoting). Then, there exists k such that $\boldsymbol{x}$ and $\boldsymbol{y}_k$ belong to two different orbits. Note that two vertices belong to the same orbits if and only if their support graphs are isomorphic. Assume that all the neighbors of a vertex $\boldsymbol{x}$ belong to the same orbits of $\boldsymbol{x}$. Thus, all they have isomorphic support graphs. Inductively, also all the neighbors of the neighbors of $\boldsymbol{x}$ will belong to the same orbit of $\boldsymbol{x}$. As we can reach every vertex through a finite number of subsequent pivoting steps, we have that all the vertices have isomorphic support graphs, that is all the vertices have a support graph isomorphic to a Hamiltonian cycle. This would imply that $\overline{P}^n_{ASEP}$ is an integer polytope, but that is not true. Another result derived from Lemma 9 is related to the so-called *polyhedral graph*.

Definition 17 (Polyhedral graph)**.** A polyhedral graph is an undirected graph formed from the vertices and edges of a convex polyhedron.

Let $G_{P^n_{\text{ASEP}}}$ be the polyhedral graph of P^n_{ASEP}. It follows directly from Lemma 9 the following:

Corollary 2. For all $\pi \in S_n$, $g_\pi \in G(n)$ is an automorphism of $G_{P^n_{\text{ASEP}}}$ that is an adjacency preserving bijection on the set of vertices

6.4 Computing vertices of P^{n+1}_{ASEP} from vertices of P^n_{ASEP}.

We specialize some results in [24] to obtain a strategy for generating vertices starting from P^n_{ASEP} to P^{n+1}_{ASEP}.

Let us start by recalling some definitions:

Definition 18 ([24])**.** Let $S \subset V_k$ and $\boldsymbol{x} \in P^k_{\text{ASEP}}$. If

$$\boldsymbol{x}(\delta(S)) := \sum_{(i,j)\in\delta(S)} x_e = 1$$

Then, S is called a *thight set* (t.s)

Definition 19 ([24])**.** Let $S \subset V_k$ be a t.s and $\boldsymbol{x} \in P^{k+|S|-1}_{\text{ASEP}}$. Then we can collapse the set S in node w as follows:

$$(\boldsymbol{x} \downarrow_w (S))_{uv} = \begin{cases} \sum_{s\in S} x_{us} & v = w \\ \sum_{s\in S} x_{sv} & u = w \\ x_{uv} & \text{otherwise} \end{cases}$$

Lemma 10 ([24], Prop. 3.3.1)**.** *Let $\boldsymbol{x} \in P^k_{ASEP}$ and let $S \subset V_k$ t.s of $\boldsymbol{x}$. Then $\boldsymbol{x} \downarrow_w (S)$ belongs to $P^{k-|S|+1}_{ASEP}$*

From our preliminary observation, we notice that a lot of vertices, especially the ones associated with high values of integrality gap, contain a basic structure that we named λ-loop. Figure 6.2 shows an example of such a structure. It consists of two arcs of weight λ and $1-\lambda$, with $0 < \lambda < 1$, involving two nodes.

More formally:

Definition 20 (λ-loop)**.** Let $\boldsymbol{x}$ vertex of P^n_{ASEP} and let $v_1, v_2 \in V_n$ such that $x_{v_1v_2} = \lambda$ and $x_{v_2v_1} = 1-\lambda$. We say that $\boldsymbol{x}$ *contains a λ-loop $\overleftrightarrow{v_1v_2}$*.

We implement a strategy for generating vertices from P^n_{ASEP} of P^{n+1}_{ASEP} that we named *λ-loop break*.

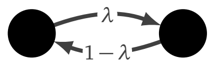


Figure 6.2. A λ-loop

(a) $\boldsymbol{x}$ (b) $\boldsymbol{x} \uparrow^{v_3} (v_1, v_2)$

Figure 6.3. From $\boldsymbol{x} \in P^n_{ASEP}$ to $\boldsymbol{x} \uparrow^{v_3} (v_1, v_2) \in P^{n+1}_{ASEP}$. Here, $\overline{\lambda} = 1 - \lambda$.

6.4.1 λ-loop break

Definition 21. Let $\boldsymbol{x}$ vertex of P^n_{ASEP} that contains a λ-loop $\overleftrightarrow{v_1 v_2}$. Then, we can define

$$(\boldsymbol{x} \uparrow^{v_3} (v_1, v_2))_{uv} = \begin{cases} \lambda & uv \in \{v_1 v_3,\ v_3 v_2\} \\ 1-\lambda & uv \in \{v_3 v_1,\ v_2 v_3\} \\ 0 & uv \in \{v_1 v_2,\ v_2 v_1\} \\ x_{uv} & \text{otherwise} \end{cases}$$

Figure 6.3 show an example where $\lambda = \frac{1}{2}$. The underlying idea comes from the observation that vertices tend to have several loops in their support graphs. See for instance Figure 6.6.

Note that the following holds:

Lemma 11. *In the configuration of Definition 21, $S = \{v_1, v_3\}$ is a tight set for $\boldsymbol{x} \uparrow^{v_3}$ (v_1, v_2).*

Proof. Let $\boldsymbol{x}' = \boldsymbol{x} \uparrow^{v_3} (v_1, v_2)$

$$\boldsymbol{x}'(\delta(S)) = \sum_{\substack{b \in v \\ b \neq v_3}} x_{v_1 b} + \sum_{\substack{b \in r \\ b \neq v_1}} x_{v_3 b}$$

given that

$$\sum_{b\in V} x_{v_1 b} = 1 \Rightarrow \sum_{\substack{b\in V\\ b\neq v_3}} x_{v_1 b} = 1 - x_{v_1 v_3} = 1-\lambda$$
$$\sum_{b\in V} x_{v_3 b} = 1 \Rightarrow \sum_{\substack{b\in V\\ b\neq v_1}} x_{v_3 b} = 1 - x_{v_3 v_1} = 1-(1-\lambda) = \lambda$$
$$\Rightarrow \boldsymbol{x}'(\delta(S)) = 1-\lambda+\lambda = 1$$

□

By using results from [24] we are thus able to retrieve the following:

Lemma 12. *If $\boldsymbol{x}$ is a vertex of P^n_{ASEP} such that contains a λ-loop $\overleftrightarrow{v_1 v_2}$ then $\boldsymbol{x} \uparrow^{v_3} (v_1, v_2)$ is a vertex of P^{n+1}_{ASEP}.*

Proof. Let $\boldsymbol{x}' := \boldsymbol{x} \uparrow^{v_3} (v_1, v_2)$. Clearly is feasible. To show that is a vertex, assume by contradiction that there exists $\boldsymbol{y}', \boldsymbol{z}' \in P^{n+1}_{\text{ASEP}}$ and $\mu \in (0,1)$ s.t

$$\boldsymbol{x}' = \mu \boldsymbol{y}' + (1-\mu)\boldsymbol{z}'.$$

Let $S = \{v_1, v_3\}$. Note that

$$\boldsymbol{x}'(\delta(S)) = 1 = \mu \boldsymbol{y}'(\delta(S)) + (1-\mu)\boldsymbol{z}'(\delta(S)) \tag{6.17}$$

As $\boldsymbol{y}', \boldsymbol{z}'$ are feasible, we have $\boldsymbol{y}'(\delta(S)) \geq 1$ and $\boldsymbol{z}'(\delta(S)) \geq 1$. As the equality should hold, we have then $\boldsymbol{z}'(\delta(S)) = \boldsymbol{y}'(\delta(S)) = 1$. Thus, are well defined

$$\boldsymbol{y} :=\downarrow_{v_1} \boldsymbol{y}'(S) \quad \boldsymbol{z} :=\downarrow_{v_1} \boldsymbol{z}'(S).$$

Thanks to Lemma 10, $\boldsymbol{z}, \boldsymbol{y} \in P^n_{\text{ASEP}}$. Clearly

$$\boldsymbol{x} = \mu \boldsymbol{y} + (1-\mu)\boldsymbol{z}.$$

This can be verified entry by entry. As an example, consider the case $e = uv_1$:

$$\begin{aligned}
&\mu y_{uv_1} + (1-\mu) z_{uv_1} = \\
&\mu\left(y'_{uv_1} + y'_{uv_3}\right) + (1-\mu)\left(z'_{uv_1} + z'_{uv_3}\right) = \\
&\left[\mu y'_{uv_1} + (1-\mu) z'_{uv_1}\right] + \left[\mu y'_{uv_3} + (1-\mu) z'_{uv_3}\right] = \\
&x'_{uv_1} + x'_{uv_3} = x_{uv_1}
\end{aligned}$$

Thus $\boldsymbol{x}$ is not a vertex of P^n_{ASEP}. Hence, we get a contradiction. □

Remark 18. The reasons why we introduce this kind of structure are twofold: firstly, as yet noticed, we have observed that loop structures are very common in vertices with a high integrality gap. Secondly, we have observed that starting from a vertex of P^n_{ASEP} and obtaining vertices of P^{n+1}_{ASEP} by breaking loops, the integrality gap always increases. See Section 6.6.1 for evidence on this topic.

Remark 19. A similar idea has been explored in [24]. In this work, the author proposes an operation called *2-jack insertion*, where a node satisfying certain hypotheses is replaced with a λ-loop using $\lambda = \frac{1}{2}$. Note that two adjacent λ-loops satisfy such conditions, and hence they use this strategy to iteratively insert one loop between two loops in vertices having a large integrality gap. Our strategy is hence more general with respect to the one they propose, as we deal not only with half-integer vertices but with all types of vertices. Hence, our computational results on the integrality gap will provide lower bounds that are at least as strong as the ones they provide, as the vertices they have used are in the set we explore.

6.5 An explore-exploit algorithm for the vertices exploration

In this section, we describe how we utilized the pivoting procedure and the extension procedure to enumerate the vertices. Our procedure is iterative, namely, it continues to iterate vertex-by-vertex, exploring each time the neighborhood of each vertex. The input parameters are:

- M, the maximum number of iterations of the algorithm, equivalent to the maximum number of vertices we pivot on.
- T_{tot}, timelimit for the whole iterations.
- T_{it}, timelimit for the single iteration.

Parameters T_{it} balance the tradeoff between *exploration* and *exploitation*. Small values of T_{it} allow for exploring only a few elements adjacent to a given vertex $\boldsymbol{x}$ and quickly moving on to the next neighboring vertex to be explored. On the other hand, high values of T_{it} insist heavily on the neighborhood of a vertex $\boldsymbol{x}$. We continue to iterate the procedure until either the timelimit T_{tot} is hit or the maximum number M of iteration is reached. The whole procedure can be described in words as follows. Starting from $n = 5$ and the full collection of non-isomorphic vertices of P^5_{ASEP} that we can exhaustively generate using the software PORTA [16] :

1. We apply the procedure of λ-loop break obtaining vertices of P^{n+1}_{ASEP}.
2. We complete each of these vertices with the slack variables.
3. We order all the vertices found in this way by the number of zeros, from the one with the fewest zeros to the one with the most zeros.

4. Starting from the first one, we begin to apply the Pivoting strategy as described in Section 6.3 and equation 6.16. Specifically, we enumerate all possible combinations of variables that can form a feasible basis and attempt to include each of the nonbasic variables in the basis.

5. When we reach the time limit T_{it}, we have a subset of adjacent vertices denoted as $\mathcal{N}'(\boldsymbol{x}) \subseteq \mathcal{N}(\boldsymbol{x})$. At this point, two things can occur:

   (a) $\mathcal{N}'(\boldsymbol{x}) = \emptyset$: in this case, we move on to the next vertex in the ordered list and start again from step 4.

   (b) $\mathcal{N}'(\boldsymbol{x}) \neq \emptyset$: in this case, we take each vertex from this set and add it to the input list - if it is not isomorphic to any of the previous ones - in order to maintain the list sorted by the number of zeros. Then, we start again from step 4.

6. We continue iteratively until either T_{tot} or M are reached.

Note that, thanks to Lemma 9, we do not need to pivot twice on isomorphic vertices, as they share the same neighborhood.

We named this algorithm “explore/exploit” for two reasons: the parameter T_{it} governs the exploitation of a single vertex. For small values of T_{it}, we do not focus much on the neighborhood of a single vertex, instead preferring to pivot on multiple vertices. However, for large values of T_{it}, we continue to build a single $\mathcal{N}(\boldsymbol{x})$, having less time to explore different areas of P^n_{ASEP}.

For a formal description of the proposed procedure see Algorithm 2. The Pivoting function mentioned in line 9 of Algorithm 2 is the one of Equation (6.16).

Implementation details. The pivoting algorithm has been implemented in C++. We use Eigen [37] to deal with the matrices and MPFR [30] to do the operation related to it in multiple precision, as we have observed that most of the matrices obtained as submatrices of the constraint matrix for $\overline{P}^n_{ASEP}$ are ill-conditioned. The gap has been computed implementing the linear program (6.2)–(6.8) using the commercial software Gurobi [38]

Parameter setting. By using notation of Algorithm 1, we set $M = 1000,\ T_{tot} = 24h,\ T_{it} = 1h$. Note that if the number of combinations to explore is less or equal to 5000, we do the full exploration of the Neighborhood, regardless of the runtime (Hence $T_{it} = \infty$).

6.6 Computational analysis

6.6.1 The impact of the loop-breaking procedure

Starting from one representative of the vertices of $n = 6$, we expand all the loops by obtaining vertices of $n = 7$. For $n = 6$, we expand all the loops by obtaining 149 orbits

Algorithm 2: Generating vertices of P^n_{ASEP} trough the explore-exploit algorithm.

Input: n, Number of nodes of the ATSP instance under study
Input: $R := \{\boldsymbol{x}_i\}_{i \in I}$, list of vertices available for $n-1$
Input: M, Maximum number of iteration of algorithm
Input: T_{tot}, timelimit for the whole iteration
Input: T_{it}, timelimit for the single iteration
Output: R', a collection of non isomorphic vertices of P^n_{ASEP}

$R' = \text{Extend}(R)$
$R' = \text{SortByNumberOfZeros}(R')$
$i = 0$
$\boldsymbol{x}_0 = R'[i]$
$Ps = []$
while *$i < M$ and time $< T_{tot}$* **do**
    if *$\boldsymbol{x}_0$ is not isomorphic to any vertex in Ps* **then**
        $Ps.\text{append}(\boldsymbol{x}_0)$
        $\mathcal{N}'(\boldsymbol{x}_0) = \text{Pivoting}(\boldsymbol{x}_0, T_{it})$
        for $\boldsymbol{y} \in \mathcal{N}(\boldsymbol{x}_0)$ **do**
            if *$\boldsymbol{y}$ is not isomorphic to any vertex in R'* **then**
                $R'.\text{insert}(\boldsymbol{y})$
            end
        end
        $i{+}{+}$
        $Ps.\text{append}(R'[i])$
    end
end
return R'

for $n = 7$. As already mentioned in Section 6.4, this procedure leads to vertices always having an integrality gap greater than or equal to the starting one. See Figure 6.4 for a representation of this phenomenon.

6.6.2 Neighborhood exploration

This section is devoted to studying the neighborhood of some vertices and deducing local properties. First of all, from our preliminary test, we state the following conjecture:

Conjecture 1. For each $\boldsymbol{x}$ vertex, $\mathcal{N}(\boldsymbol{x})$ contains at least one tour.

If this conjecture was true, solely pivoting on the integer vertex would lead to the full V-description of the polytope: by using Lemma 9 the neighborhood of an integer vertex will contain at least one representative of each orbit. We can then list all the vertices by fully describing each orbit. In the next paragraphs, we study the neighborhoods of

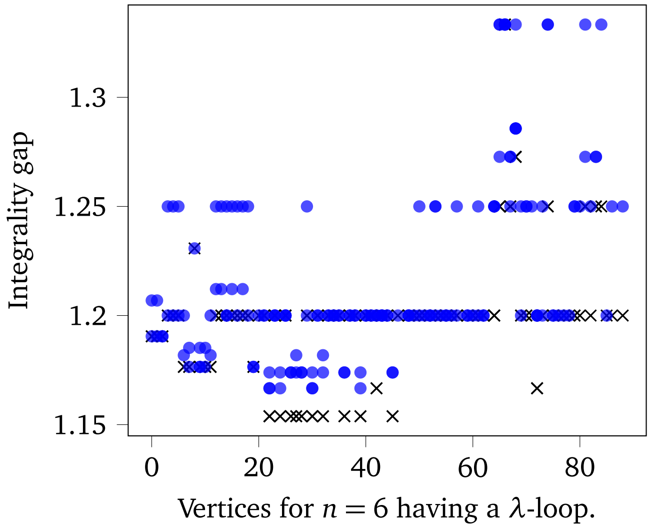


Figure 6.4. Value of the integrality gap of for $n = 6$ (black crosses) and values of all the integrality gap obtained from that vertex for $n = 7$ (blue dots). The operation of extending λ-loops leads to instances having an integrality gap greater than or equal to the one associated with the vertex.

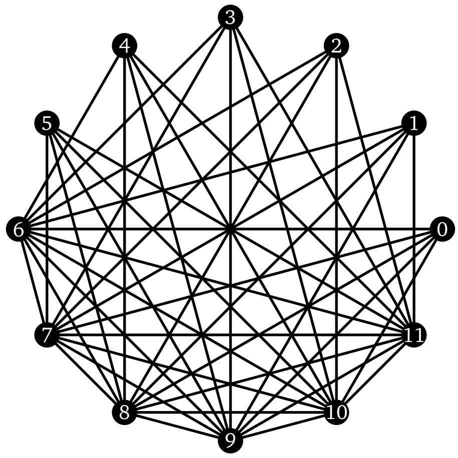


Figure 6.5. Polyhedral graph for $G_{P^4_{\text{ASEP}}}$

vertices for $n = 4, 5$. Then, we focus on $n = 6$ and hence study the impact of our algorithm.

For $n = 4$, all the vertices have been enumerated in [24]. More specifically, we have 12 vertices in total and two orbits. Figure 6.5 show the polyhedral graph of P^4_{ASEP}. Nodes from 0 to 5 represent the non-integer vertices, while nodes from 6 to 11 represent the tours. We can observe that each non-integer vertex has an integer adjacent; Interestingly, each tour is connected to all the vertices but one.

$\lvert O_x \rvert$	Components		Frequencies		IG
6	0	$\frac{1}{2}$	4	8	1.2
6	0	1	8	4	1.0

Table 6.1. Orbit structure for $n = 4$. Columns: cardinality of the orbit, type of components, frequency of each component, and integrality gap attained at the elements of that orbit.

Vertex	$\lvert O_x \rvert$	# tight sets	$\lvert \mathcal{N}(\boldsymbol{x}) \rvert$
(a)	120	4	11
(b)	60	4	20
(c)	60	6	28
(d)	120	4	23
(e)	24	10	148

Table 6.2. Main feature of orbits in $n = 5$. The label of the first column refers to Figure 6.6.

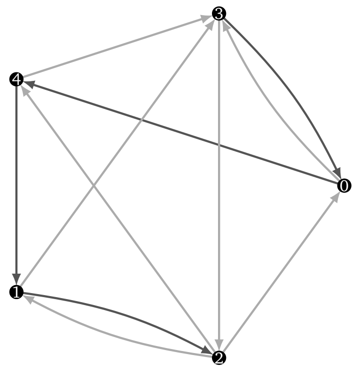
Table 6.1 reports the structure of the orbit we found for $n = 4$. As there is only one half-integer solution, the maximum integrality gap is hence attained at that vertex.

For $n = 5$, we recover 384 vertices, as already done in [24]. A representation of these vertices can be found in Figure 6.6. The one attaining the highest integrality gap is vertex (c). Interestingly, it does not look very different from vertex (b). Both of them contains to $\frac{1}{2}$-loops and have only $\frac{1}{2}$ entries. By looking at Table 6.3, we understand that each of these vertices has the same number of non-zeros and zeros entries. The main differences lie in the number of tight sets (see Definition 18). Tight sets are associated with slack variables $\overline{x}_S = 0$. In particular, vertex (c) has 6 TS, while vertex (b) has only 4. We choose a representative for each class and study its neighborhood. This can be done w.l.o.g thanks to Lemma 9. As expected, the number of neighbors is related to the degeneracy of the vertex, that is number of zeros among both arc and slack variables.

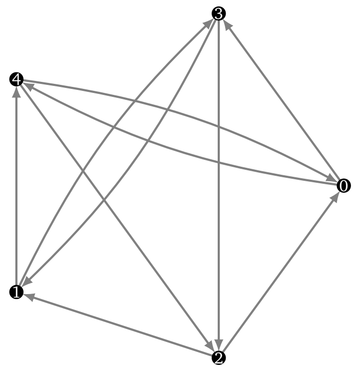
Figure 6.7 reports a compressed version of the polyhedral graph P^5_{ASEP}. In this case, we merge all the vertices belonging to the same orbits in the same node, in order to only observe the adjacency pattern. Note that such a graph is less symmetric w.r.t to the one for $n = 4$. Interestingly, each vertex has at least one adjacent representative for each equivalence class. Furthermore, The vertices (d) and (e) also have a representative of themselves among the neighbors.

In the case of $n = 6$, we have 90 orbits and thus any visual representation would be meaningless. Thus, we show step by step how the algorithm described in Section 6.3.2 works in practice. First, we need a starting vertex. To do so, we recover it using the breaking loop procedure from $n = 5$.

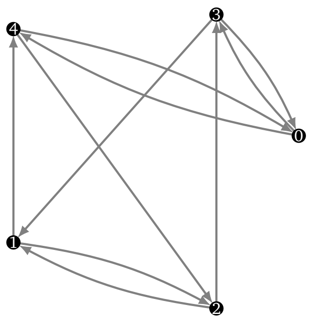
This gives us 6 non-isomorphic vertices and, among them, we choose the one with



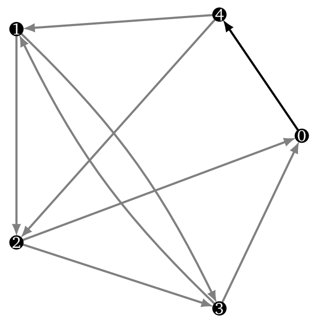
(a) Vertex with $\frac{1}{3}$ and $\frac{2}{3}$ entries.



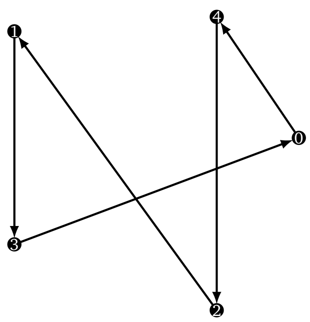
(b) Vertex with $\frac{1}{2}$ entries only.



(c) Vertex with $\frac{1}{2}$ entries only.



(d) Vertex with $\frac{1}{2}$ and 1 entries only.



(e) A tour.

Figure 6.6. Support graph of the 5 representatives of each isomorphism class. The edges are colored according to the value. The darker, the higher (That is, black means $x_a = 1$ and white $x_a = 0$).

the smallest number of zeros. By observing the behavior of our algorithm we see that, although the solutions we found with the “breaking loops” procedure are associated with high integrality gaps, they have a lot of zeros and it’s hard to explore the full neighbor-

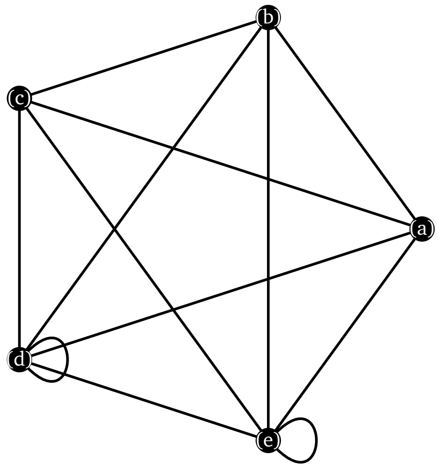


Figure 6.7. Compressed version of the polyhedral graph $G_{P^5_{\mathrm{ASEP}}}$.

$\lvert O_x \rvert$	Components			Frequencies			IG
120	0	$\frac{1}{3}$	$\frac{2}{3}$	9	7	4	1.20
60	0	$\frac{1}{2}$		10	10		1.20
60	0	$\frac{1}{2}$		10	10		1.25
120	0	1	$\frac{1}{2}$	11	1	8	1.20
24	0	1		15	5		1.00

Table 6.3. Orbit structure for $n = 5$. Columns: cardinality of the orbit, type of components, frequency of each component, and integrality gap attained at the elements of that orbit.

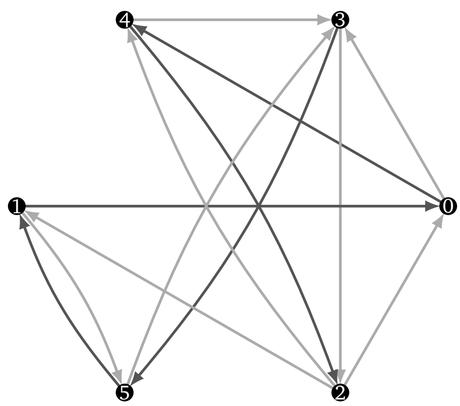


Figure 6.8. Starting point for the Pivoting algorithm for $n = 6$.

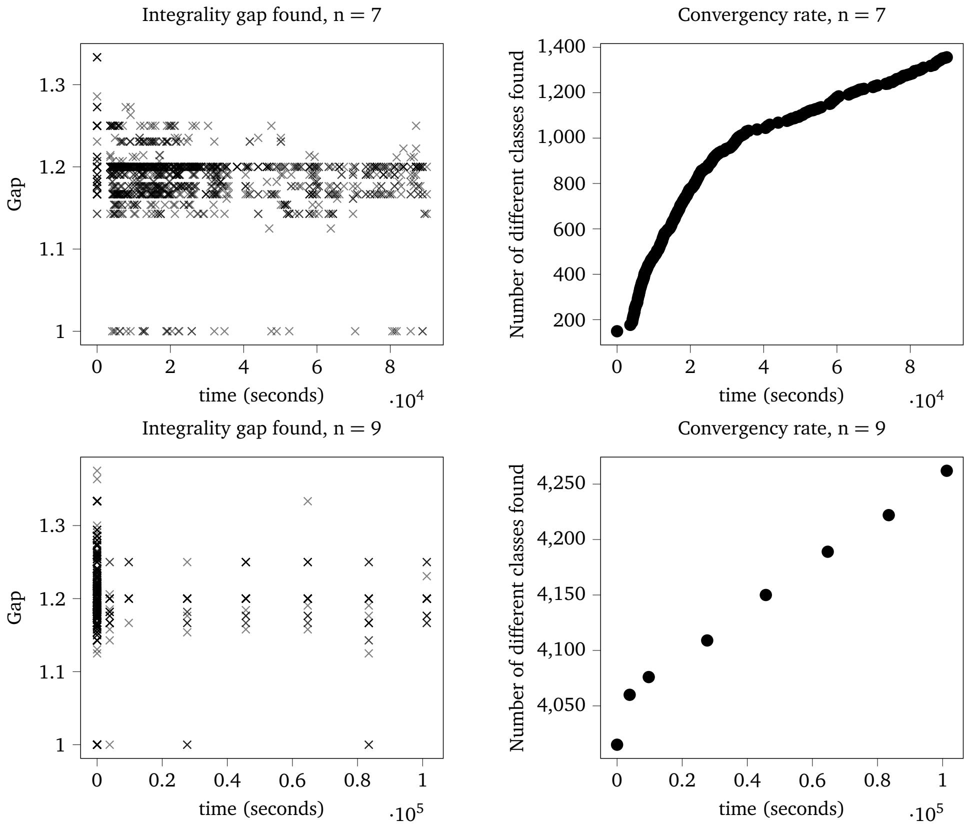


Figure 6.9. Left: time vs integrality gap found. The darker the $\times$, the bigger the number of vertices found leading to that integrality gap. We note that the highest integrality gap is found at the very beginning of the procedure. Right: Time vs. Number of classes of isomorphism. We note that for $n = 7$ the algorithm continuously finds new and new vertices; for $n = 9$, new vertices are more and more rare.

hood of them. However, such points are the starting point of our procedure. Figure 6.8 reports its support graph. Such vertex has entries equal to $\frac{k}{3}$, $k \in \{0, \ldots, 5\}$. We start by pivoting from this vertex and collecting all its neighbors. Then, among its neighbors, we move to the one having the smaller number of zeros, and we proceed as follows. Differently from the pivoting pipeline presented in [24], we can recover at least one representative for each orbit.

6.6.3 Results of the explore-exploit procedure

Figure 6.9 shows how our algorithm performs in practice for $n = 7, 9$. In terms of number of new vertices found, we observe a different trend: first of all, for $n = 7$, a lot of vertices are found at the very first iterations and the trend seems exponential. After a certain amount of time, the trend becomes linear. From $n = 9$, we note also that the impact of the

n	Before Pivoting				After Pivoting			
	Number of orbits	α_n^{LB}	Number of orbits with α_n^{LB}	Non Half-integer?	Number of orbits	α_n^{LB}	Number of orbits with α_n^{LB}	Non Half-integer?
7	149	4/3	5	No	1356	4/3	5	No
8	2097	4/3	39	Yes	2837	4/3	41	Yes
9	3476	11/8	1	No	4262	11/8	1	No
10	5641	7/5	2	No	5796	7/5	2	No

Table 6.4. Results of the explore-exploit algorithm for $6 \leq n \leq 10$.

pivoting becomes less and less relevant while n increases. Note that, for all n, the vertex with the highest gap is found by solely splitting the λ-loops of vertices of P_{ASEP}^{n-1}. The trend shows also that, empirically, the relevant vertices for the integrality gap are found at the very first iteration of the algorithms. Lastly, we observe that, for all n either the gaps are close to 1, or the gaps are strictly greater than 1.1. This may underlay some particular structure of the non-relevant vertices. Table 6.4 reports the information we retrieve with our algorithm. The first 3 columns are related to the information we have before starting the pivoting procedure, while the last three columns are after. For each group, we report α_n^{LB}, namely the highest integrality gap we found that serves as a lower bound for the highest integrality gap at all, at a given number of nodes. Interestingly, our algorithm is capable of finding the vertex with the highest integrality gap at the very first iteration, by simply expanding from vertices of one fewer n number of nodes. Interestingly, for $n = 8$ differently from what has been done in [24], we found 4 orbits with non-half-integer vertices attaining a maximal integrality gap. This could be a weak indication that the conjecture that vertices with maximum integrality gap are half-integer is not necessarily true. For $n = 10$, no additional information is provided in the manuscript of [24]. We can generate 2 half-integer vertices with a maximal integrality gap. Table 6.4 reports in detail the obtained values of the integrality gap we obtain for each number of nodes between $6 \leq n \leq 10$.

By observing both Table 6.4 and Figure 6.9 we observe that with the increasing of n, the impact of the pivoting is less and less effective. Unfortunately, starting from $n = 11$ on, it is impossible to find any new vertices using the pivoting within the time limit of 24h. Hence, from $n \geq 11$, we decide to abandon the pivoting part, and only obtain vertices using the extending procedure from $n = 10$.

However, to the best of our knowledge, this is the first time that non-half-integer vertices are explored for $n \geq 10$.

Table 6.5 reports all the results for $n \geq 11$. Unfortunately, we were not able to find any vertices having an integrality gap greater than the one found by [24] only exploring half-integer vertices. Note that, for each number of nodes, more than 90% of the explored vertices are non half integer.

n	Number of orbits	α_n^{LB}	Number of orbits with α_n^{LB}	Non Half-integer?
11	7389	10/7	1	No
12	9133	56/39	1	No
13	11010	13/9	1	No
14	13025	100/69	1	No
15	15181	11/16	1	No
16	17473	156/107	1	No
17	19902	19/13	1	No
18	22470	224/153	1	No
19	25172	22/15	1	No

Table 6.5. Results of the explore-exploit algorithm for $n \geq 11$.

This enforces the conjecture that the instances with a high integrality gap should be searched among the half-integer. Interestingly, the instances found having an odd number of nodes are the ones introduced by [24] to provide a family of ATSP instances having an integrality gap asymptotically tending to 1.5. When, instead we have an even number of nodes, the instances we found are new.

6.6.4 Integrality gap and computational hardness

Several studies in the literature have examined the empirical hardness of the Symmetric Travelling Salesman Problem concerning the integrality gap, such as [42, 71]. Also Chapters 2 and 3 deal with this issue. However, to the best of our knowledge, no such studies have been conducted on the ATSP. Hence, we pick all the instances we generate by computing the integrality gap, and we assess their computational complexity. As already discussed in Section 1.3.2, recent studies [27, 57] suggest that Concorde remains the most efficient method for solving ATSP. It is important to note that Concorde can only handle symmetric nonnegative costs, but it is possible to transform any ATSP into an STSP by doubling the number of nodes using the method proposed in [44] and the observation in Section 1.3.2. Hence, transform each ATSP instance we wound as argmin of problem $Gap(\boldsymbol{x})$, and we solve each instance using Concorde and record the computational time and the integrality gap. Remarkably, the resulting instances exhibit a reduced integrality gap. Figure 6.10 reports this information for $n = 9$.

In terms of computational complexity, we are hence able to retrieve some hard-to-solve instances. More specifically, for each n number of nodes, we apply the procedure yet described, hence obtaining an STSP with $2n$ nodes. We do this procedure for each instance $\boldsymbol{c}$ obtained as the optimal value of the linear programming (6.2)–(6.8). Then, we run Concorde with 10 different seeds on each instance, picking the one leading to

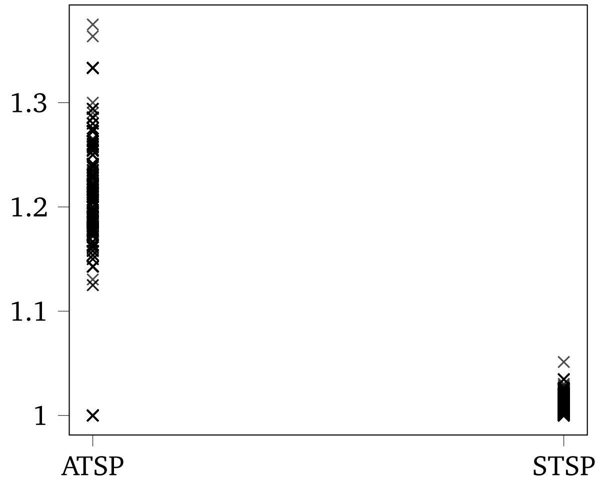


Figure 6.10. Distribution of the integrality gap from the Asymmetric version to the symmetric one through the application of the Jonker-Volgenant duplication procedure for $n = 9$

the highest computational time. We then compare the runtime with the one achieved by running the instances of the TSPLIB having the same number of nodes. Table 6.6 reports this information, providing both the integrality gap and the runtime for one instance for each number of nodes. We also report the p-value obtained performing the Wilcoxon signed-rank test [67], to assess the null hypothesis that two paired samples, which are related or dependent, are drawn from identical distributions. As the p-value is always 0, we can claim that the higher computational time is not obtained by chance.

By comparing Table 3.2 and 3.5 with Table 6.6, we notice that the instances we obtain are actually hard to solve, namely they are competitive with the instances produced with the H-OPT procedure introduced in Chapter 3. As an example, the instance derived from doubling the node of an STSP with 12 nodes is as difficult as the one obtained from gr24 using the procedure described in Chapter 3.

6.7 Conclusion and Future works

In this chapter, we started with the work of [24] and made some extensions. Specifically, we extended their pivoting strategy in a way that allowed us to recover all vertices for $n = 6$, while their method worked for all isomorphism classes except 3. To achieve this, we studied the symmetries of P^n_{ASEP} to reduce the number of vertices on which to perform pivoting.

We developed new strategies for generating vertices of P^{n+1}_{ASEP} based on those of P^n_{ASEP}. Our strategy builds upon the one generated by [24], but works in a more general framework. In particular, we explored vertices that are not half-integer for the first time to

n	2n	$\alpha(\boldsymbol{c})$ for the ATSP	$\alpha(\boldsymbol{c})$ for the STSP	runtime	TSPLIB instance	$\alpha(\boldsymbol{c})$	runtime	p-value
6	12	1.200	1.000	0.179 ± 0.008	N.A	N.A	N.A	N.A
7	14	1.176	1.000	1.222 ± 0.054	burma14	1.000	0.001 ± 0.000	0.000
8	16	1.154	1.002	3.687 ± 0.193	ulysses16	1.000	0.028 ± 0.000	0.000
9	18	1.197	1.004	6.454 ± 0.180	N.A	N.A	N.A	N.A
10	20	1.174	1.010	9.300 ± 0.202	N.A	N.A	N.A	N.A
11	22	1.207	1.006	14.27 ± 0.316	ulysses22	1.000	0.062 ± 0.000	0.000
12	24	1.224	1.007	27.242 ± 0.673	gr24	1.000	0.028 ± 0.000	0.000
13	26	1.255	1.005	56.325 ± 1.628	fri26	1.0	0.013 ± 0.000	0.000
14	28	1.258	1.005	104.115 ± 3.008	N.A	N.A	N.A	N.A
15	30	1.257	1.011	101.619 ± 3.195	N.A	N.A	N.A	N.A
16	32	1.257	1.005	325.497 ± 7.311	N.A	N.A	N.A	N.A
17	34	1.258	1.004	587.210 ± 18.451	N.A	N.A	N.A	N.A

Table 6.6. Reesults of 10 runs of Concorde with 10 different seeds. Columns represent the number of nodes of the ATSP and the number of nodes after the procedure of making it an STSP, the integrality gap of the original asymmetric instance and the one of the STSP, the mean of the runtimes (± standard deviation) and the same values for an instance of the TSPLIB having $2n$ number of nodes. Lastly, the p-value is the result of the Wilcoxon test as described in the manuscript. N.A means that an instance with $2n$ nodes in the TSPLIB does not exist.

understand the structure of vertices with high integrality gaps. Our research was motivated by the observation that for n = 8, there exist 4 non-half-integer vertices that achieve the maximum IG. However, for higher n, the maximum integrality gap is always generated by a single half-integer vertex.

In conclusion, we examined the computational aspects of the instances generated as a by-product. We attack some ATSP instances with Concorde after making them symmetric has been observed as the state-of-the-art approach in exact methods. We observed that some instances with high but not maximum integrality gaps are very challenging from a computational perspective.

As we have already mentioned, this is a preliminary work and some theoretical results are missing, in particular, we aim to formally prove the maximality of the stabilizer of the integer vertices and that the λ-loop break procedure always leads to an integrality gap greater than or equal to the starting vertex.

Chapter 7

Conclusions

In conclusion, this thesis explores various aspects of the TSP in two variants, symmetric and asymmetric, focusing on computational hardness and algorithmic approaches.

In Chapter 2, we introduced a new family of Metric TSP instances that are particularly challenging for Concorde, a state-of-the-art TSP solver. We demonstrated that these instances are significantly harder than those found in existing libraries such as TSPLIB, as well as the instances introduced by [42]. Additionally, we found similarities between our instances and the rectilinear TSP introduced by [71]. We showed that the presence of a large integrality gap does not necessarily indicate computational hardness for exact methods. We also derived a set of inequalities based on the cost vector of the TSP, which are valid for optimal solutions. We integrate such inequalities in a naive implementation of a TSP solver, showing that they provide speed-up in hard-to-solve instances. Overall, our findings highlight the significance of our work in understanding and addressing the challenges posed by certain classes of TSP instances.

Chapter 3 focuses on the development of a complexification procedure for generating challenging STSP instances. Starting from an input instance, we construct STSP instances with integrality gaps that are larger than or equal to the gaps of the original instances. We implemented a branch-and-cut algorithm to handle the large number of constraints involved in the procedure. We hence show that such procedure is NP-Hard. Our computational results demonstrate the effectiveness of this method in generating instances that are challenging for Concorde. Additionally, we created the Hard-TSPLIB, a library consisting of 41 small metric TSP instances with large integrality gaps, which serve as valuable benchmarks for evaluating TSP solvers.

In Chapter 4, we introduced a classification approach to assess the empirical hardness of metric STSP instances. By utilizing four features that capture the distribution of cost coefficients in the cost matrix, we developed a decision tree model that accurately predicts the hardness of TSP instances. The features include standard deviation, skewness, Gini index, and a novel statistic called the tight triangle inequality index. Our extensive dataset of over 45,000 instances, collected from literature and randomly generated, enabled us to train and evaluate the model effectively. The high prediction score, measured by the Matthew Correlation Coefficient, indicates the reliability of our classification approach, although some limitations are discussed and highlighted.

Chapter 5 explores the complexity of solving the STSP using the formulation proposed by DFJ. Starting from the evidence that the minimum-weight 2-matching problem with zero subtour constraints in this formulation is polynomial-time solvable, we extended this result by demonstrating that the mw2m problem with a constant number of subtour elimination constraints can be solved by reducing it to a sequence of mw2m problems. Consequently, the STSP with the DFJ formulation can be solved using this algorithm. Our findings have implications for the fixed-parameter tractability of the minimum-weight triangle-free 2-factor problem, which remains an open problem in the field. Furthermore, we question whether the complexity of an instance is an intrinsic property independent of the algorithm used, and our results on TSPLIB and Hard-TSPLIB confirm this to be the case.

Finally, in Chapter 6, we shift our focus to the metric case of the Asymmetric TSP (ATSP) where the costs satisfy the triangle inequality. We investigate the integrality gap of the ATSP and provide lower bounds for instances with up to 21 cities. Notably, our analysis includes instances with $15 \leq n \leq 21$, which have not been previously studied. To identify the fractional vertices of the subtour elimination polytope, we employ a combination of a pivoting-based algorithm that exploits symmetries and an inductive strategy for generating vertices of higher dimensions. Computational experiments reveal that instances generated using this approach exhibit computational hardness, highlighting the presence of challenging instances within the procedure.

Overall, this thesis contributes to the understanding of computational hardness and algorithmic approaches for solving the TSP and its variants. The findings have implications for the development of improved algorithms and the creation of benchmark instances for evaluating TSP solvers.

Starting from this work, we see some possible extensions.

- **Further investigation of challenging TSP instances.** While this thesis has identified and generated challenging instances for the STSP and ATSP, there is still room for exploring other families of instances. Investigating different problem characteristics, such as additional constraints or structural properties, could lead to the discovery of even more challenging instances that push the boundaries of current algorithms.

- **Integration of machine learning techniques.** The classification approach presented in Chapter 4 demonstrates the potential of using machine learning techniques to predict the hardness of TSP instances based on specific features. Expanding on this work, more advanced machine learning models could be explored, leveraging larger datasets and incorporating additional features to enhance the accuracy of hardness prediction.

- **Development of tailored algorithms.** The identification of hard TSP instances in this thesis opens up opportunities for developing specialized algorithms that are specifically designed to tackle these challenging scenarios. By understanding the unique characteristics of these instances, novel algorithmic approaches could be devised to improve the efficiency and effectiveness of solving them. A preliminary result in this direction is Theorem 11 in Chapter 2, but further advancements can be achieved

- **Investigation of other combinatorial optimization problems.** The research conducted in this thesis primarily focuses on the TSP and its variants. However, the methodologies and insights gained can be extended to other combinatorial optimization problems as well. Exploring the hardness and algorithmic approaches for related problems, such as the Vehicle Routing Problem (VRP) or the Traveling Salesman Problem with Time Windows (TSPTW), could be a promising avenue for future research.

- **Theoretical advancements.** This thesis touches on several theoretical aspects, including complexity analysis, fixed-parameter tractability, and polyhedral results. Further theoretical investigations could delve deeper into the structural properties of TSP. We have already obtained some preliminary results in both the computational complexity of STSP with a subset of subtour elimination constraints and the generation of vertices of the polytope P^{n+2}_{ASEP} from P^{n}_{ASEP}.

Acronyms

ASEP Asymmetric Subtour Eliminarion Problem. (page 4)

ATSP Asymmetric Travelling Salesman Problem. (page 3)

DFJ Dantzig - Fulkerson - Johnson. (page 1)

HC Hamiltonian Cycle. (page 1)

IG Integrality Gap. (page 5)

ILP Integer Linear Programming. (page 1)

MW2M Minimum-Weight 2-Matching. (page 14)

SEP Subtour Eliminarion Problem. (page 4)

SSEP Symmetric Subtour Eliminarion Problem. (page 4)

STPS Symmetric Travelling Salesman Problem. (page 2)

Bibliography

[1] David Applegate, Robert Bixby, William Cook, and Vasek Chvátal. On the solution of traveling salesman problems. 1998.

[2] David L Applegate, Robert E Bixby, Vasek Chvatal, and William J Cook. *The traveling salesman problem: a computational study*. Princeton university press, 2006.

[3] Michael Artin. *Algebra (Classic Version) (Pearson Modern Classics for Advanced Mathematics Series)*. Pearson, 2nd edition, 2006.

[4] Arash Asadpour, Michel X Goemans, Aleksander Mądry, Shayan Oveis Gharan, and Amin Saberi. An o (log n/log log n)-approximation algorithm for the asymmetric traveling salesman problem. *Operations Research*, 65(4):1043–1061, 2017.

[5] ML Balinski and Andrew Russakoff. On the assignment polytope. *Siam Review*, 16 (4):516–525, 1974.

[6] Genevieve Benoit and Sylvia Boyd. Finding the exact integrality gap for small traveling salesman problems. *Mathematics of Operations Research*, 33(4):921–931, 2008.

[7] Dimitris Bertsimas and Santosh Vempala. Solving convex programs by random walks. *Journal of the ACM (JACM)*, 51(4):540–556, 2004.

[8] Sylvia Boyd and Robert Carr. Finding low cost TSP and 2-matching solutions using certain half-integer subtour vertices. *Discrete Optimization*, 8(4):525–539, 2011.

[9] Sylvia Boyd and Paul Elliott-Magwood. Structure of the Extreme Points of the Subtour Elimination Polytope of the STSP (Combinatorial Optimization and Discrete Algorithms). *RIMS Kokyuroku Bessatsu*, 23:33–47, 2010.

[10] Sylvia Boyd, René Sitters, Suzanne van der Ster, and Leen Stougie. TSP on cubic and subcubic graphs. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 65–77. Springer, 2011.

[11] Miguel Cárdenas-Montes. Predicting hardness of travelling salesman problem instances. In *Conference of the Spanish Association for Artificial Intelligence*, pages 68–78. Springer, 2016.

[12] Robert Carr and Santosh Vempala. On the held-karp relaxation for the asymmetric and symmetric traveling salesman problems. *Mathematical Programming*, 100(3): 569–587, 2004.

[13] Moses Charikar, Michel X Goemans, and Howard Karloff. On the integrality ratio for asymmetric tsp. In *45th Annual IEEE Symposium on Foundations of Computer Science*, pages 101–107. IEEE, 2004.

[14] Peter C Cheeseman, Bob Kanefsky, and William M Taylor. Where the really hard problems are. In *International Joint Conferences on Artificial Intelligence*, volume 91, pages 331–337, 1991.

[15] Davide Chicco and Giuseppe Jurman. The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC genomics*, 21(1):1–13, 2020.

[16] Thomas Christof. Porta. *http://www. iwr. uni-heidelberg. de/groups/comopt/software/PORTA/index. html*, 2009.

[17] Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976.

[18] Michele Conforti, Gérard Cornuéjols, Giacomo Zambelli, et al. *Integer programming*, volume 271. Springer, 2014.

[19] George Dantzig, Ray Fulkerson, and Selmer Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, 2(4):393–410, 1954.

[20] George B Dantzig. Programming in a linear structure. *Washington, DC*, 1948.

[21] Vladimir G Deineko, Rene Van Dal, and Günter Rote. The convex-hull-and-line traveling salesman problem: A solvable case. *Information Processing Letters*, 51 (3):141–148, 1994.

[22] E. Demaine, W. Gasarch, and M. Hajiaghayi. *Computational Intractability: A Guide to Algorithmic Lower Bounds*. MIT Press, 2022.

[23] Jack Edmonds. Maximum matching and a polyhedron with 0, 1-vertices. *Journal of research of the National Bureau of Standards B*, 69(125-130):55–56, 1965.

[24] Paul Elliott-Magwood. *The integrality gap of the asymmetric travelling salesman problem*. PhD thesis, University of Ottawa (Canada), 2008.

[25] Thomas Fischer, Thomas Stützle, Holger Hoos, and Peter Merz. An analysis of the hardness of tsp instances for two high performance algorithms. In *Proceedings of the Sixth Metaheuristics International Conference*, pages 361–367, 2005.

[26] Matteo Fischetti and Paolo Toth. A polyhedral approach to the asymmetric traveling salesman problem. *Management Science*, 43(11):1520–1536, 1997.

[27] Matteo Fischetti, Andrea Lodi, and Paolo Toth. Exact methods for the asymmetric traveling salesman problem, 2007.

[28] Matteo Fischetti, Andrea Lodi, Michele Monaci, Domenico Salvagnin, and Andrea Tramontani. Improving branch-and-cut performance by random sampling. *Mathematical Programming Computation*, 8(1):113–132, 2016.

[29] Francesc Font-Clos. fontclos/hitandrun: Initial release, 2021. URL `https://doi.org/10.5281/zenodo.4906246`.

[30] Laurent Fousse, Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, and Paul Zimmermann. Mpfr: A multiple-precision binary floating-point library with correct rounding. *ACM Transactions on Mathematical Software (TOMS)*, 33(2):13–es, 2007.

[31] Luca Maria Gambardella, Stefano Gualandi, Monaldo Mastrolilli, and Eleonora Vercesi. Predicting the empirical hardness of metric tsp instances. In *Workshop on Operation Research and Data Science in Public Services*, pages 1–16. Springer, 2022.

[32] Michael R Garey and David S Johnson. Computers and intractability. *A Guide to the*, 1979.

[33] Ian P Gent and Toby Walsh. The TSP phase transition. *Artificial Intelligence*, 88 (1-2):349–358, 1996.

[34] Corrado Gini. *Variabilità e mutabilità: contributo allo studio delle distribuzioni e delle relazioni statistiche.[Fasc. I.].* Tipogr. di P. Cuppini, 1912.

[35] Martin Grötschel and Yoshiko Wakabayashi. A cutting plane algorithm for a clustering problem. *Mathematical Programming*, 45(1):59–96, 1989.

[36] Martin Grötschel, Làszlo Lovàsz, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169 – 197, 1981.

[37] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[38] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2022.

[39] Keld Helsgaun. An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European journal of operational research*, 126(1):106–130, 2000.

[40] Keld Helsgaun. General k-opt submoves for the Lin-Kernighan TSP heuristic. *Mathematical Programming Computation*, 1(2):119–163, 2009.

[41] Stefan Hougardy. On the integrality ratio of the subtour LP for Euclidean TSP. *Operations Research Letters*, 42(8):495 – 499, 2014. ISSN 0167-6377. doi: https://doi.org/10.1016/j.orl.2014.08.009. URL http://www.sciencedirect.com/science/article/pii/S0167637714001205.

[42] Stefan Hougardy and Xianghui Zhong. Hard to solve instances of the Euclidean Traveling Salesman Problem. *Mathematical Programming Computation*, pages 1–24, 2020.

[43] David S. Johnson and Lyle A. McGeoch. *Experimental Analysis of Heuristics for the STSP*, pages 369–443. Springer US, Boston, MA, 2007. ISBN 978-0-306-48213-7. doi: 10.1007/0-306-48213-4_9. URL https://doi.org/10.1007/0-306-48213-4_9.

[44] Roy Jonker and Ton Volgenant. Transforming asymmetric into symmetric traveling salesman problems. *Operations Research Letters*, 2(4):161–163, 1983.

[45] Anna R Karlin, Nathan Klein, and Shayan Oveis Gharan. A (slightly) improved approximation algorithm for metric tsp. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 32–45, 2021.

[46] Richard M. Karp. Reducibility among combinatorial problems. In R. E. Miller, J. W. Thatcher, and J. D. Bohlinger, editors, *Complexity of Computer Computations*, pages 85–103. Plenum, New York, 1972. ISBN 978-1-4684-2003-6. doi: 10.1007/978-1-4684-2001-2_9.

[47] Yusuke Kobayashi. Weighted triangle-free 2-matching problem with edge-disjoint forbidden triangles. *Mathematical Programming*, 192(1):675–702, 2022.

[48] Monique Laurent. Graphic vertices of the metric polytope. *Discrete Mathematics*, 151(1-3):131–153, 1996.

[49] Shen Lin and Brian W Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973.

[50] Andrea Lodi and Andrea Tramontani. Performance variability in mixed-integer programming. In *Theory driven by influential applications*, pages 1–12. INFORMS, 2013.

[51] László Lovász. Hit-and-run mixes fast. *Mathematical programming*, 86(3):443–461, 1999.

[52] María A Osorio and David Pinto. Hard and easy to solve tsp instances. *XXX Aniversario del Programa Educativo de Computación, BUAP*, pages 26–30, 2003.

[53] Christos H Papadimitriou. The Euclidean travelling salesman problem is NP-complete. *Theoretical computer science*, 4(3):237–244, 1977.

[54] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[55] Gerhard Reinelt. TSPLIB—A traveling salesman problem library. *ORSA journal on computing*, 3(4):376–384, 1991.

[56] Gerhard Reinelt. TSPLIB - A Traveling Salesman Problem Library. *INFORMS Journal on Computing*, 3(4):376–384, November 1991. doi: 10.1287/ijoc.3.4.376. URL `https://ideas.repec.org/a/inm/orijoc/v3y1991i4p376-384.html`.

[57] Roberto Roberti and Paolo Toth. Models and algorithms for the asymmetric traveling salesman problem: an experimental comparison. *EURO Journal on Transportation and Logistics*, 1(1-2):113–133, 2012.

[58] H Edwin Romeijn and Robert L Smith. Simulated annealing for constrained global optimization. *Journal of Global Optimization*, 5(2):101–126, 1994.

[59] J Hyam Rubinstein, Doreen A Thomas, and Nicholas C Wormald. A polynomial algorithm for a constrained traveling salesman problem. *Networks: An International Journal*, 38(2):68–75, 2001.

[60] Sartaj Sahni and Teofilo Gonzalez. P-complete approximation problems. *Journal of the ACM (JACM)*, 23(3):555–565, 1976.

[61] Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer, 2003.

[62] Robert L Smith. Efficient Monte Carlo procedures for generating points uniformly distributed over bounded regions. *Operations Research*, 32(6):1296–1308, 1984.

[63] Kate Smith-Miles, Jano van Hemert, and Xin Yu Lim. Understanding TSP difficulty by learning from evolved instances. In *International Conference on Learning and Intelligent Optimization*, pages 266–280. Springer, 2010.

[64] Jano I van Hemert and Neil B Urquhart. Phase transition properties of clustered travelling salesman problem instances generated with evolutionary computation. In *International Conference on Parallel Problem Solving from Nature*, pages 151–160. Springer, 2004.

[65] Eleonora Vercesi, Stefano Gualandi, Monaldo Mastrolilli, and Luca Maria Gambardella. On the generation of metric tsp instances with a large integrality gap by branch-and-cut. *Mathematical Programming Computation*, 15(2):389–416, 2023.

[66] Oliver Vornberger. Easy and hard cycle covers. Technical report, Technical report, Universität/Gesamthochschule Paderborn, 1980.

[67] Frank Wilcoxon. Individual comparisons by ranking methods. In *Breakthroughs in Statistics: Methodology and Distribution*, pages 196–202. Springer, 1992.

[68] David Paul Williamson. Analysis of the Held-Karp heuristic for the traveling salesman problem. Master's thesis, Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science, 1990.

[69] Laurence A Wolsey. Heuristic analysis, linear programming and branch and bound. *Combinatorial Optimization II*, pages 121–134, 1980.

[70] Zelda B Zabinsky, Robert L Smith, SI Gass, and MC Fu. Hit-and-run methods. *Encyclopedia of Operations Research & Management Science*, pages 721–729, 2013.

[71] Xianghui Zhong. Lower Bounds on the Integraliy Ratio of the Subtour LP for the Traveling Salesman Problem. *arXiv preprint arXiv:2102.04765*, 2021.

Acknowledgements

First and foremost, I want to express my heartfelt gratitude to my advisor, Professor Stefano Gualandi, for being an incredibly example of dedication, motivation, and passion. Your constant support, knowing that you always believed in me, pushing me to embrace every challenge, and entrusting me with even the most intricate parts of the project have been a source of strength for me. Thank you for have been so supportive in all my choices, for introducing me to all your collaborators and allowing them to become mine as well, and for providing me with valuable connections among your contacts. You are an example and a figure whom I will draw inspiration from in my future career.

I want to express my gratitude to all my collaborators. In particular, I want to thank (in alphabetical order and not by importance): J. Barta, A.M Bernardelli, A. Buchanan, D. Duma, L. Gambardella, G. Guidetti, M. Mastrolilli, G. Toscani, thank you for giving me the opportunity to work with you. Each of you has taught me something, and I hope that working with me has been as enjoyable for you as it has been for me working with you.

I would like to thank all my colleagues, both in Switzerland and in Pavia, for making the hours of study and work a little lighter. In particular, I want to sincerely thank my research group, that is an example of genuine and selfless exchange of information and support.

Thanks to my family, always present and constantly supportive. It's wonderful to return to my small village and know that I will always find your smiles, happy to welcome me.

Thank you to all my friends. "The lone wolf dies, but the pack survives," and if I have come this far, it's because my pack, sometimes physically distant, is always close to my heart.