ON THE GENERATION OF METRIC TSP INSTANCES WITH A LARGE INTEGRALITY GAP BY BRANCH-AND-CUT

Eleonora Vercesi
Department of Mathematics
University of Pavia
eleonora.vercesi01@universitadipavia.it

Stefano Gualandi
Department of Mathematics
University of Pavia
stefano.gualandi@unipv.it

Monaldo Mastrolilli
IDISA, Lugano
monaldo.mastrolilli@idsia.ch

Luca Maria Gambardella
Faculty of Informatics, USI, IDSIA - USI-SUPSI, Lugano
luca.gambardella@idsia.ch

September 10, 2021

ABSTRACT

This paper introduces a computational method for generating metric Travelling Salesperson Problem (TSP) instances having a large integrality gap. The method is based on the solution of an NP-hard problem, called IH-OPT, that takes in input a fractional solution of the Subtour Elimination Problem (SEP) on a TSP instance and compute a TSP instance having an integrality gap larger than or equal to the integrality gap of the first instance. The decision variables of IH-OPT are the entries of the TSP cost matrix, and the constraints are defined by the intersection of the metric cone with an exponential number of inequalities, one for each possible tour. Given the very large number of constraints, we have implemented a branch-and-cut algorithm for solving IH-OPT. 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. If we solve the IH-OPT problem for every sampled vertex using our branch-and-cut algorithm, we can select the generated TSP instance (i.e., cost vector), yielding the longest 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 which have a large integrality gap and are challenging in terms of runtime for Concorde.

Keywords Integer Programming · Integrality gap analysis · Branch-and-cut · Metric Traveling Salesman Problem

1 Introduction

The Branch-and-Cut (B&C) algorithms are powerful tools to solve NP-hard problems. A core component of these algorithms is the solution of several Linear Programming (LP) relaxations that appear while searching for an integral optimal solution. Given an instance of an ILP model, its *integrality gap* for one linear relaxation is the ratio between the optimal solution of the integer problem and the corresponding LP relaxation. In practice, this is a measure of how much the relaxation is good to approximate the original problem. As a rule of thumb, given an Integer Linear Programming (ILP) model for an NP-hard problem, the larger is the integrality gap of the initial LP relaxation, the longer is the runtime for a BC algorithm to find the optimal solution. In practice, a large integrality gap at the root node very often implies that a great number of branch-and-bound nodes must be visited before proving the optimality of an integral solution. For this reason, a significant effort in designing efficient B&C algorithms is spent looking for tight LP relaxations [25].

Let us consider, for instance, the Travelling Salesman Problem (TSP): Given a set of n nodes $V = \{1, \ldots, n\}$, a square matrix $\boldsymbol{C} \in \mathbb{R}_+^{n \times n}$ whose entries c_{ij} represent the cost of going from node i to node j, we have to find the cyclic

permutation π of V such that the total cost $\sum_{i \in V} c_{i,\pi_i}$ is minimum. If the matrix $\mathbf{C}$ is (i) *symmetric*, that is, $c_{ij} = c_{ji}$ for all $i, j \in V$, (ii) satisfies the *triangle inequalities* $c_{ij} + c_{jk} \geq c_{ik}, \forall i, j, k \in V$, and (iii) $c_{ij} = 0$ if and only if $i = j$, then the matrix $\mathbf{C}$ defines a *metric* on V and gives a *Metric TSP* instance. Indeed, there exist several ILP models to solve the TSP [1, 8, 24, 26]. However, the most successful exact method is the branch-and-cut algorithm introduced by Padberg and Rinaldi [28], which includes as a core component the solution of an LP relaxation based on the subtour elimination constraints [27]. This LP relaxation is known in the literature as the *Subtour Elimination Problem (SEP)*. Nowadays, the state-of-the-art exact software for the TSP is Concorde, a solver which implements a sophisticated B&C algorithm and which includes the separation of several facet-defining inequalities [1]. Concorde has a track record in solving the instances from the TSPLIB [29], a wide set of TSP benchmark instances. Note that most of the TSPLIB instances have a small integrality gap for the SEP. The largest integrality gap is equal to 1.095, achieved by the ts225 instance, which was specifically designed to foil TSP software. Except for that one instance, D.S. Johnson [19] is not aware of any testbed instance with a percentage integrality gap exceeding 1.03.

While studying theoretically the performance of the Held-Karp lower bound for the TSP, which is equivalent to the optimal lower bound of the SEP, Wolsey proved in [34] that the integrality gap for SEP is at most $\frac{3}{2}$. Later, in his master thesis, Williamson conjectured that for metric TSP the integrality gap of subtour elimination relaxation is equal to $\frac{4}{3}$ [33]. So far, this conjecture is only proved for a very special class of instances [5]. For cubic graphs, we know that $\frac{4}{3}$ is an upper bound for the integrality gap [7]. In [2, 6], Boyd and Benoit have computed the exact integrality gap for every $n \leq 12$, and, thus, they have verified by exhaustive enumeration the conjecture for all Metric TSP instances with at most 12 nodes. In addition, Benoit and Boyd [2] have introduced a new class of graphs having an integrality gap that asymptotically converges to $\frac{4}{3}$ as n tends to infinity. To the best of our knowledge, nobody has been able to find, for a given number on nodes, and instance with an higher integrality gap than the one they proposed. Curiously, we have computationally verified that those instances are not challenging for Concorde. More recently, new families of instances with an integrality gap that asymptotically converges to $\frac{4}{3}$ were introduced in [18] by exploiting Tetrahedron configurations, and in [17, 36] by exploiting three paths configurations. However, all those families of instances are solvable in polynomial time: the instances of [17] are convex-hull-and-line TSP instances and they are solvable in polynomial time with the algorithm proposed by Denieko et al. [9]. The instances of [18] can be solved using the polynomial time algorithm introduced in [31]. The instances of [36] can be solved in linear time, as shown by the author himself. Nevertheless, the TSP instances introduced in [36] are remarkable: in practice, they are extremely challenging for Concorde.

In this paper, we introduce a new challenging NP-hard problem, herein called the *Integer Heuristic-OPT (IH-OPT)* problem, whose optimal solution provides a Metric TSP instance with a large integrality gap. Our approach is purely computational and differently from [17, 18, 36], we only require that the generated instance is a metric. Such instances are not restricted and strongly structured as those from [17, 18, 36]. The decision variables of our problem are the entries of the symmetric cost matrix $\mathbf{C}$, while the constraints are defined by the intersection of the metric cone [20] with an exponential number of inequalities, one for each possible permutation of V, that is, one for each possible solution of the TSP. For example, starting from an instance C_0 of the TSPLIB, taking an optimal solution of the LP relaxation of the corresponding subtour elimination model, and by solving the IH-OPT problem, we can generate a new TSP instance C^* having an integrality gap larger than or equal to the integrality gap of the original instance C_0 (e.g., see Table 1). In addition, by integrating the solution of the IH-OPT problem into a sampling procedure, we can generate several TSP instances that have a large integrality gap and are challenging in terms of runtime for Concorde (e.g., see Table 4). As a by-product of our work, we introduce the Hard-TSPLIB, a collection of 41 small TSP instances (i.e., with $n \leq 76$) which are very challenging for Concorde in terms of runtime and number of branch-and-bound nodes.

The outline of this paper is as follows. In Section 2, we review the background material and we fix the notation. Section 3 formally introduces the *IH-OPT* problem and discusses how it is related to previous works. Section 4 presents the sampling procedure that we use to generate TSP instances which are hard for Concorde in terms of runtime. In Section 5, we present our extensive computational results and describe how we generated the instances that we have included in the Hard-TSPLIB. Finally, in Section 6, we conclude with a discussion on future works.

2   Background material

In this section, we review the main formulation for the symmetric TSP and we introduce the Subtour Elimination polytope. We formally define the integrality gap for the SEP, presenting the work of [2], which is the foundation of our work.

An instance of the symmetric TSP can be completely defined by the symmetric matrix $\mathbf{C}$. Otherwise, we can define a TSP instance using a complete undirected graph $K_n = (V, E)$ along with a cost vector $\mathbf{c} \in \mathbb{R}_+^{|E|}$, which is given by

the upper triangular matrix of C (without the diagonal). While in this paper we focus on the general *Metric TSP*, and we denote it only with TSP, as defined in Section 1, two special cases that are relevant to compare our work with the literature:

1. The *Euclidean TSP*, where the input is a collection of n points in $\mathbb{R}^d$, whose reciprocal distances are computed using the Euclidean norm. These are the type of TSP instances used in [17, 18], with $d = 2$.

2. The *Rectilinear TSP*, where the input is again a collection of n points in $\mathbb{R}^d$, but where the Manhattan norm (or city block or L^1 norm) is used to compute the distances. These are the type of TSP instances used in [36], with $d = 3$.

In both cases, using the collection of points given in input, and the corresponding distance function, it is possible to define a complete graph K_n with the set of nodes $V = \{1, \ldots, n\}$ and the set of edges $E = V \times V$, with the corresponding cost vector $c \in \mathbb{R}_+^{|E|}$. In the following, we use $\delta(S)$ with $S \subset V$, to denote the set of weighted edges $e = \{v, w\}$ with $v, w \in S$ and $v \neq w$. We use later the two following collection of subsets of vertices: $\mathscr{S} := \{S \mid S \subset V, 3 \leq |S| \leq n - 3\}$ and $\mathscr{S}_{ij} := \{S \mid S \in \mathscr{S}, \{i, j\} \in \delta(S)\}$.

Given a connected graph $G = (V, E)$ and the cost vector $c \in \mathbb{R}_+^{|E|}$, the Subtour Elimination Problem, originally proposed in [8], is formulates as follows:

$$\min \quad \sum_{e \in E} c_e x_e \tag{1}$$

$$\text{s.t.} \quad \sum_{e \in \delta(\{v\})} x_e = 2 \quad \forall v \in V \quad \text{(Degree Constraints)} \tag{2}$$

$$\sum_{e \in \delta(S)} x_e \geq 2 \quad \forall S \in \mathscr{S} \quad \text{(Subtour Elimination Constraints)} \tag{3}$$

$$0 \leq x_e \leq 1 \quad \forall e \in E \quad \text{(Bound Constraints)} \tag{4}$$

$$x_e \text{ integer} \quad \forall e \in E. \tag{5}$$

The decision variable x_e is equal to 1 if the edge $e = \{v, w\}$ is part of an optimal tour. The objective function (1) minimizes the overall tour length. Constraints (2) state that each node v must have two incident edges. The Subtour Elimination Constraints (3) force the cut set $\delta(S)$ of every proper subset S of V to contain at least two edges.

If we relax the integrality constraints (5), we can define the polytope of the SEP as

$$P_{SEP} := \{x \in \mathbb{R}^{|E|+} \mid (2), (3), (4)\} \tag{6}$$

If we optimize the objective function (1) over P_{SEP}, we get exactly the Subtour Elimination Problem.

Let us consider the complete graph K_n. Let us denote by $TOUR(c)$ the optimal value of the TSP instance defined by the cost vector c, and by $SUBT(c)$ the lower bound given by the optimal value of SEP minimized over the cost vector c. Similarly to [2], we denote by α_n the integrality gap of SEP for K_n, that is, the largest possible ratio between $TOUR(c)$ and $SUBT(c)$:

$$\alpha_n = \max_{c \text{ is a metric}} \frac{TOUR(c)}{SUBT(c)}. \tag{7}$$

For $n \leq 5$, we have that $\alpha_n = 1$. The exact value for $6 \leq n \leq 12$ was computed in [2, 6]. Williamson's conjecture states that $\alpha_n \leq \frac{4}{3}$ for any n [33]. The families of instances introduced in [17, 18, 36] have all the properties that $\alpha_n < \frac{4}{3}$ and $\lim_{n \to \infty} \alpha_n = \frac{4}{3}$. However, the largest integrality gap for the instances in the TSPLIB is equal to 1.095 [29], which is largely less than $\frac{4}{3} = 1.\overline{333}$.

Following the idea presented in [2], we can divide the cost vector c by the optimum tour value $TOUR(c)$, obtaining a new cost vector c' that still satisfies the triangle inequalities, and which leaves α_n unchanged. Hence, we can restrict ourselves to metric cost vectors c such that $TOUR(c) = 1$, and we can transform the maximization problem into a minimization:

$$\frac{1}{\alpha_n} = \min_{\substack{c \text{ is a metric} \\ TOUR(c)=1 \text{ for } c}} SUBT(c). \tag{8}$$

Problem (8) can be formulated as an integer quadratic problem, where the decision variables are both the cost vector c and the incidence vector x of vertices of P_{SEP}. Unfortunately, despite the recent improvements in the implementation of commercial optimization solvers, the quadratic model is intractable even for small value of n. In [2], the authors propose a clever idea for bypassing the quadratic model using the vertex representation of SEP. That is, they represent

P_{SEP} as the convex combinations of its vertices $\{\bar{\boldsymbol{x}}^{(1)}, \dots, \bar{\boldsymbol{x}}^{(t)}\}$, which are finitely many. Then, for each vertex $\bar{\boldsymbol{x}}^{(h)}$ of P_{SEP}, they define an LP problem, called OPT_h, having an exponential number of variables and constraints. Theoretical results in [2, 6] guarantee that only a subset of vertices is necessary to compute the integrality gap. By solving the OPT_h subproblem on each vertex of the previous mentioned subset, they were able to compute α_n for $n \leq 10$ in [2], and for $n \leq 12$ in [6].

Given the complete graph $K_n = (V, E)$ and a permutation π of its vertices V, we define the incidence vector of a tour as $\bar{\boldsymbol{z}} \in \mathbb{R}^{|E|}$ such that $\bar{z}_{ij} = 1$ if $\pi_i = j$, and $\bar{z}_{ij} = 0$, otherwise. Let $\mathscr{T}_n$ 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_{SEP}, the OPT_h problem is defined in [2] as

$$OPT\boldsymbol{x}^{(h)} = \min \sum_{\{i,j\} \in E} \bar{x}_{ij}^{(h)} c_{ij} \tag{9}$$

$$\text{s.t.} \quad \sum_{\{i,j\} \in E} \bar{z}_{ij} c_{ij} \geq 1 \qquad \forall \bar{\boldsymbol{z}} \in \mathscr{T}_n \tag{10}$$

$$c_{ij} \leq c_{ik} + c_{jk} \qquad \forall i, j, k \in V \tag{11}$$

$$c_{ij} \geq 0 \qquad \forall \{i,j\} \in E \tag{12}$$

$$y_i + y_j - u_{ij} + \sum_{S \in \mathscr{S}_{ij}} d_S \leq c_{ij} \qquad \forall \{i,j\} \in E \tag{13}$$

$$u_{ij} \geq 0 \qquad \forall \{i,j\} \in E \tag{14}$$

$$d_S \geq 0 \qquad \forall S \in \mathscr{S}. \tag{15}$$

Constraints (10) ensure that the optimal solution $\boldsymbol{c}^*$ of $OPT\boldsymbol{x}^{(h)}$ for every $\boldsymbol{x}^{(h)}$ is such that $TSP(\boldsymbol{c}^*) = 1$, as discussed in [2]. Herein, we call the inequalities (10) the *TSP constraints*. Constraints (11) and (12) ensure that the cost vectors represent a semi-metric. Constraints (13)–(15) are the dual constraints associated to the dual problem of (1)–(4). To ensure that the vertex $\bar{\boldsymbol{x}}^{(h)}$ remains the optimal solution of the SEP, the inequality constraints are replaced with equality in (13) when $\bar{x}_{ij}^{(h)} > 0$, in (14) when $\bar{x}_{ij}^{(h)} < 1$, and in (15) when $\sum_{ij \in \delta(S)} \bar{x}_{ij}^{(h)} > 2$. If we denote by $\boldsymbol{c}^*$ the optimal solution of $OPT(\boldsymbol{x}^{(h)})$, the dual slackness constraints are introduced to guarantee

$$\arg\min SEP(\boldsymbol{c}^*) = \boldsymbol{x}^{(h)}. \tag{16}$$

Hence, given the list of the vertices $\{\bar{\boldsymbol{x}}^{(1)}, \dots, \bar{\boldsymbol{x}}^{(t)}\}$ of P_{SEP}, the integrality gap α_n of K_n is computed by solving

$$\frac{1}{\alpha_n} = \min_{h=1,\dots,t} OPT_h \qquad \Rightarrow \qquad \alpha_n = \max_{h=1,\dots,t} \frac{1}{OPT_h}. \tag{17}$$

Part of the original contribution presented in [2] is to prove that we need to consider only a subset of the t vertices of P_{SEP}, namely, such vertices which support graphs satisfy certain properties. We refer the reader to [2] for the details.

In the following section, we modify the single problem OPT_h in order to introduce a new NP-hard problem that we use to generate metric TSP instances with a large integrality gap.

3 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 TSP. We are not interested in computing the exact value α_n for a fixed n as in [2], but we focus on finding a heuristic solution to problem (17).

Notice that problem (9)–(15) has an exponential number of variables due to the dual variables d_S. Furthermore, the number of constraints (10) is equal to the number of tours, that is $\frac{(n-1)!}{2}$, and the number of triangular inequalities constraints (11) is $O(n^3)$. In practice, the exact solution of OPT_h is intractable even for small values of n. Note that the computation of α_n for $n = 12$ required around 24 days [6]. In order to solve problem (17) heuristically, we introduce a new problem, called *Heuristic-OPT (H-OPT)*, which is related to problem OPT_h, but which can be solved for larger values of n. The two key ideas for introducing the new problem are:

(i) To generate a TSP instance with a sufficiently large integrality gap is unnecessary to enumerate all vertices $\{\bar{\boldsymbol{x}}^{(1)}, \dots, \bar{\boldsymbol{x}}^{(t)}\}$ of the SEP polytope. We can sample a subset of vertices and take the instance providing the largest integrality gap.

4

(ii) Since we do not perform exhaustive vertex enumeration, it is unnecessary to impose the complementary slackness constraints (13)-(15) to force that a given vertex $\bar{\boldsymbol{x}}^{(h)}$ remains the same vertex of P_{SEP}. Hence, we can remove all the variables and constraints related to the slackness conditions.

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

$$\text{H-OPT}(\bar{\boldsymbol{x}}^{(h)}) := \min \sum_{\{i,j\} \in E} \bar{x}_{ij}^{(h)} c_{ij} \tag{18}$$

$$\text{s.t.} \sum_{\{i,j\} \in E} \bar{z}_{ij} c_{ij} \geq 1 \qquad \forall \bar{\boldsymbol{z}} \in \mathscr{T}_n \tag{19}$$

$$c_{ij} \leq c_{ik} + c_{jk} \qquad \forall i,j,k \in V \tag{20}$$

$$c_{ij} \geq 0 \qquad \forall \{i,j\} \in E. \tag{21}$$

In practice, we have relaxed the problem OPT_h by removing constraints (13)–(15). Notice that in H-OPT we have only $|E|$ cost variables, $\frac{(n-1)!}{2}$ TSP constraints (19) (one for each tour), and $O(n^3)$ triangles inequalities (20) that define the metric cone [20]. 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 (19). 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:

$$\sum_{\{i,j\} \in E} \bar{z}_{ij} \bar{c}_{ij} < 1. \tag{22}$$

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

$$\min_{\boldsymbol{z} \in \mathscr{T}_n} \left\{ \sum_{\{i,j\} \in E} z_{ij} \bar{c}_{ij} \right\} \geq 1. \tag{23}$$

Thus, to add a new TSP constraint, we solve a TSP instance for a specific cost vector $\bar{\boldsymbol{c}}$. In our implementation, we separate TSP constraints by first solving the TSP instance given by $\boldsymbol{c}^*$ using the LK-H heuristic [15, 16], and whenever the heuristic fails to find a violated tour, we solve (23) using our exact TSP branch-and-cut algorithm based on the Gurobi commercial solver. Section 5.5 describes the details of our implementation. Clearly, the TSP constraints makes this problem challenging, as stated in the following lemma.

Lemma 3.1. *The H-OPT problem is NP-hard.*

Proof. Given a cost vector $\boldsymbol{c} \in \mathbb{R}_+^{|E|}$, the separation of the TSP constraints (19) requires the solution of a TSP instance, and, hence, it is NP-hard. As a consequence, for the equivalence between separation and optimization (e.g., see [14]), the solution of the LP problem H-OPT is also NP-Hard. $\square$ $\square$

Since we have removed from our problem the dual slackness constraints, the relation (16) is not necessarily satisfied when $\boldsymbol{c}^*$ 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 3.2. *Let us consider a TSP instance $\boldsymbol{c}_0$ such that $TOUR(\boldsymbol{c}_0) = 1$, and let $\bar{\boldsymbol{x}}^{(0)}$ be the optimal solution of $SEP(\boldsymbol{c}_0)$. We define a second TSP instance by the cost vector*

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

Then, the following relation holds

$$\frac{TOUR(\boldsymbol{c}_1)}{SUBT(\boldsymbol{c}_1)} \geq \frac{TOUR(\boldsymbol{c}_0)}{SUBT(\boldsymbol{c}_0)}. \tag{24}$$

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

$$SUBT(\boldsymbol{c}_1) \leq \boldsymbol{c}_1^T \bar{\boldsymbol{x}}^{(0)}$$

5

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 $TOUR(\boldsymbol{c}) = 1$. By hypothesis, $TOUR(\boldsymbol{c}_0) = 1$ 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($\bar{\boldsymbol{x}}^{(h)}$) and it holds

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

where the last equation hold by definition. Thus,

$$\frac{TOUR(\boldsymbol{c}_0)}{SUBT(\boldsymbol{c}_0)} = \frac{1}{SUBT(\boldsymbol{c}_0)} \leq \frac{1}{\boldsymbol{c}_1^T\bar{\boldsymbol{x}}^{(0)}} \leq \frac{1}{SUBT(\boldsymbol{c}_1)} = \frac{TOUR(\boldsymbol{c}_1)}{SUBT(\boldsymbol{c}_1)}.$$

$\square$ $\square$

From large integrality gaps to hard instances.   The TSP solver Concorde [1] and the LK-H heuristic that we use for separating TSP constraints [15] 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, $c_{ij} = \text{round}(\tau c_{ij}^*)$. Using this cost transformation, we cannot guarantee that the integrality gap for c_{ij} remains the same of c_{ij}^*.

The TSP constraints (19) have a right hand side equal to 1 because we have divided the cost vector $\boldsymbol{c}$ by the minimum tour length $TOUR(\boldsymbol{c})$. If we divide the cost vector by the quantity $\left(\frac{TOUR(\boldsymbol{c})}{\Delta}\right)$, where $\Delta > 0$ is large positive constant, we have to change the right hand side of (19) to Δ, while still getting an equivalent LP problem. If Δ is large enough (see Section 5.5), 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:

$$\text{IH-OPT}(\boldsymbol{x}^{(h)}) := \min \sum_{\{i,j\}\in E} \bar{x}_{ij}^{(h)} c_{ij} \tag{25}$$

$$\text{s.t.} \quad (20), (21) \tag{26}$$

$$\sum_{\{i,j\}\in E} \bar{z}_{ij} c_{ij} \geq \Delta \qquad\qquad \forall\bar{\boldsymbol{z}} \in \mathcal{T}_n \tag{27}$$

$$c_{ij} \text{ integer} \qquad\qquad \forall\{i,j\} \in E. \tag{28}$$

Clearly, IH-OPT is again an NP-hard problem, since it has H-OPT as the LP relaxation. We can prove 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.

4   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 [4]. In our context, to find a heuristic solution for problem (17), we could sample a fixed number of vertices of P_{SEP} and retain the vertex yielding the minimum value for IH-OPT, that is, yielding the largest integrality gap. However, directly sampling over the vertices of the SEP polytope is impracticable, due to its exponential number of subtour constraints. We could instead easily sample random cost vectors and generate vertices of the SEP polytope by solving directly problem (1)–(4).

A possibility for generating random cost vectors consists of generating n random points in a 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_{SEP}. In practice, this procedure takes a long time before returning a fractional vertex of the SEP polytope. Furthermore, it only samples over instances of the Euclidean TSP, implicitly excluding some remarkable metric TSP instances, such as the ones provided in [2].

To generate a random cost vector, we have designed a different approach. First, we sample a random point within the metric polytope [20] using the *hit-and-run* algorithm [32], which generates a random metric TSP instances $\boldsymbol{c} \in \mathbb{R}_+^{|E|}$. Second, we get a (random) vertex $\bar{\boldsymbol{x}}^{(h)}$ by solving SEP($\boldsymbol{c}$) 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 the SEP polytope. 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 TSP instances.

Algorithm 1: Sampling vertices of the SEP polytope 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 the SEP polytope
$R \leftarrow \emptyset$
while $|R| < r$ **do**
    $\boldsymbol{c} \leftarrow HitAndRun(\hat{P}_{MET})$
    $\boldsymbol{x} \leftarrow \arg\min\{SEP(\boldsymbol{c})\}$
    if $\boldsymbol{x}$ *is fractional* **and** $\boldsymbol{x} \notin R$ **then**
        $R \leftarrow R \cup \{\boldsymbol{x}\}$
    end
end
return R

The hit-and-run algorithm   The hit-and-run algorithm is a polynomial-time algorithm to uniformly sample from a bounded open set P, which was introduced in [32]. 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 hyper-sphere 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 [32]. The convergence rate of the hit-and-run algorithm is polynomial, as proved in [23]. Later, a modification of this algorithm to sample points that converge to an arbitrary target distribution was introduced in [30]. The interested reader can find an in-depth review of hit-and-run algorithms in [35].

The random sampling algorithm   As noted in Section 2, the constraints (11)–(12) defines the metric cone:

$$C_{MET} := \{\boldsymbol{c} \in \mathbb{R}_+^{|E|} \mid c_{ij} - c_{ik} - c_{jk} \leq 0, \forall i, j, k \in V\}. \tag{29}$$

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

$$P_{MET} := C_{MET} \cap \{c_{ij} + c_{ik} + c_{jk} \leq 2, \forall i, j, k \in V\}. \tag{30}$$

Note that for every metric $\boldsymbol{c} \in C_{MET}$, there exist an α such that $\alpha \boldsymbol{c} \in P_{MET}$ [20]. For this reason, without loss of generality, we can sample from P_{MET} instead of C_{MET}.

Since we use the hit-and-run algorithm, which requires a bounded *open* set, we define the *open metric polytope* as

$$\hat{P}_{MET} := \{\boldsymbol{c} \in \mathbb{R}_+^{|E|} \mid c_{ij} > 0, \ c_{ij} - c_{ik} - c_{jk} < 0, \ c_{ij} + c_{ik} + c_{jk} < 2, \forall i, j, k \in V\}. \tag{31}$$

Algorithm 1 presents our procedure for sampling metric TSP instances from the open set $\hat{P}_{MET}$. 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 $\hat{P}_{MET}$ using the hit-and-run algorithm. Step 4 generate the optimal vertex $\boldsymbol{x}$ for the polytope of SEP($\boldsymbol{c}$). 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 the SEP polytope of dimension m.

5   Generating the `Hard-TSPLIB`

The objective of our computational experiments is to generate the `Hard-TSPLIB`, a collection of small metric TSP instances having a large integrality gap and challenging for the Concorde solver. The instances of the `Hard-TSPLIB` are generated by using a branch-and-cut solver for problem IH-OPT($\bar{\boldsymbol{x}}^{(h)}$) with the parameter Δ set to 1000. Discussion about this choice will be provided in Section 5.5.

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 4. Third, we compare the

runtime of Concorde for solving our hard instances with the runtime required for other challenging instances from the recent literature. Then, we visually analyze the structure of the hard instances we have generated. Finally, we discuss the implementation details of our algorithms.

Note that all the computations described in the following paragraphs are executed on a single node of an HPC cluster running Cento OS, having an Intel CPU with 32 physical cores working at 2.1 GHz, and 64 GB of RAM. The code was compiled with the GNU C++ compiler v8.3, with all optimization flags enabled.

5.1   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 SEP (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 SEP solved via the simplex algorithm.

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 3.2, we have

$$\frac{TOUR(\boldsymbol{c}^*)}{SUBT(\boldsymbol{c}^*)} \geq \frac{TOUR(\boldsymbol{c}_0)}{SUBT(\boldsymbol{c}_0)},$$

that is, the TSP 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 1 reports the detailed results for the generation of hard instance from the `TSPLIB`. The table reports first the name and the dimension $n = |V|$ of the original instance. In the third and fourth columns, it reports 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 report whether the optimal SEP solution of $\boldsymbol{c}_0$ is the same optimal solution of $\boldsymbol{c}^*$.

The results of Table 1 show clearly that the small `TSPLIB` instances have a very small integrality gap and are extremely easy for Concorde. They are solved within a fraction of seconds 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 number of nodes (column 'BC n.'). Visiting a larger search tree implies in general a longer runtime. 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 longer runtime. We remark that the instance `brazil58_hard` is not solved by Concorde within a timeout of 24 hours.

Table 2 and Table 3 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 in 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 do generate an hard TSP instances; in all other cases, we were not able to find an integer cost vector satisfying all triangles inequalities and all TSP constraints, that is, an optimal integer solution for problem IH-OPT($\bar{\boldsymbol{x}}^{(h)}$).

5.2   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 4. First, we run Algorithm 1 to generate a random set R of vertices of the SEP polytope, for a fixed size n of the TSP. When generating a random vertex $\bar{\boldsymbol{x}}^{(h)}$ in Algorithm 1, we store also 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* TSP

Table 1: `Hard-TSPLIB` instances generated from the `TSPLIB`. The columns gives the instance name, number of nodes $|V|$, the integrality gap α_n for the TSPLIB instance c_0 and the HardTSPLIB c^*; average runtime (and standard deviation) and number of BC nodes over 5 independent runs of Concorde, compiled with CPLEX 12.8. The last columns reports if equation (16) 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.

name	$|V|$	α_n c_0	c^*	TSPLIB - c_0 runtime	stddev	BC n.	HARDTSPLIB - c^* runtime	stddev	BC n.	Does Eq (16) 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	✓

Table 2: Computational results for the heuristic solution of the LP problem H-OPT($\bar{x}^{(h)}$) 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.

name	$|V|$	CUTS trian.	LK-H	RUNTIME 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

instances. Later, for each random vertex $\bar{x}^{(h)}$ in R, we solve the IH-OPT($\bar{x}^{(h)}$) problem to get an instance with a larger integrality gap.

Table 4 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 report first the integrality gap of the sampled cost vector c_0, of the optimal cost vector c^* obtained after solving IH-OPT($\bar{x}^{(h)}$), and the gap α_n^* conjectured in [2], that is, to 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 c_0 (random cost vector) and then for c^* (the optimal solution of IH-OPT). Similarly to the results of the `Hard-TSPLIB`, the result of Table 4 shows that indeed we are able to generate very hard instances by solving the IH-OPT problem.

Table 3: 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).

name	$|V|$	CUTS trian.	LK-H	TSP	RUNTIME 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*	-

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 again the runtime. Figure 1(a) shows the runtime distribution for the random instances $\boldsymbol{c}_0^{(h)}$, while Figure 1(b) 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 right plot, we have mean runtime of 10 seconds, with a maximum of around 50 seconds. In practice, if we sample a very large number of vertices we are be able to get very challenging small instances.

Table 5 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

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

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

5.3   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 [36]. 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. However, as noted by the author himself, these instances are polynomially solvable [36]. 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 4 and then apply IH-OPT to the correspondent vertices, until we find an instance with an average runtime competitive with the 3D-Rectilinear instances. We call this found instance $\boldsymbol{c}_n^S$.

(b) We tried to use our computational procedure to generate hard instances starting from the optimal solution of the instances provided by Zhong [36]. Let $\boldsymbol{c}_n^R$ the cost vector of a 3D rectilinear instance with n nodes [36], 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 $\boldsymbol{x}_n^{IH} = \arg\min \text{SEP}(\boldsymbol{c}_n^{IH})$.

Table 4: Results for the `Hard-TSP` instances generated by sampling.

$|V|$	Integrality Gap - α_n			Initial instance - c_0			Hard instance - c^*		
	c_0	c^*	α_n^* [2]	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

Table 5: Average computational time and standard deviation of 1000 IH-OPT on different 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

We compared the average runtime of solving TSP(c_n^R), TSP(c_n^{IH}), and TSP(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 1 and Table 4, where we performed 10 independent runs on each instance, using every time a different seed. Actually, this variability is typical of Mixed Integer Programming and branch and cut algorithm (see e.g [11], [22]). 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 2 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, marked with S (a), and (iii)
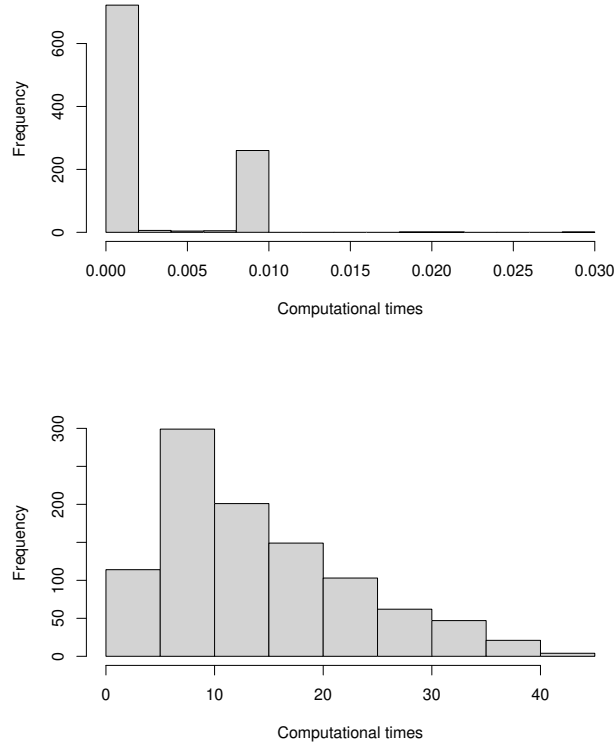
Figure 1: Top: distribution of the computational times for $n = 20$ of 1000 sampled TSP instances. Bottom: distribution of the computational times for $n = 20$ of 1000 instances obtained by the IH-OPT procedure.

the instances used with the procedure described in the point (b), marked with IH. The three runtime are denoted by t_n^R, t_n^S, t_n^{IH}, respectively. The fitted regression lines are the following:

$$t_n^R = 10^{0.168n-2.209}, \quad t_n^S = 10^{0.171n-2.157}, \quad t_n^{IH} = 10^{0.137n-1.794}.$$

By looking at the regression, the instances that we have found by massively sampling the space are a bit harder with the Rectilinear 3-D instance. On the contrary, the instances directly obtained from vertices result easier. Interestingly, in logarithm scale, the three instances have all the same order of magnitude: by looking at the boxplots in Figure 2 it is possible to observe that there three families are actually 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 [36] with respect to the one obtained by the two heuristic procedures, we note less regularity. This fact will be discussed wider in the next subsection. In terms of 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 note that the we run the computation as reported by Zhong, namely by multiply the instances by 1000 and round to the nearest integer. We have a verified that this procedure might lead to non-metric instances as, due to rounding error, triangular inequalities might be not satisfied.

5.4 Structure of small hard instances

The hard instances recently introduced in [18] and [36] 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, have entries in the set $\{0.0, 0.5, 1.0\}$.
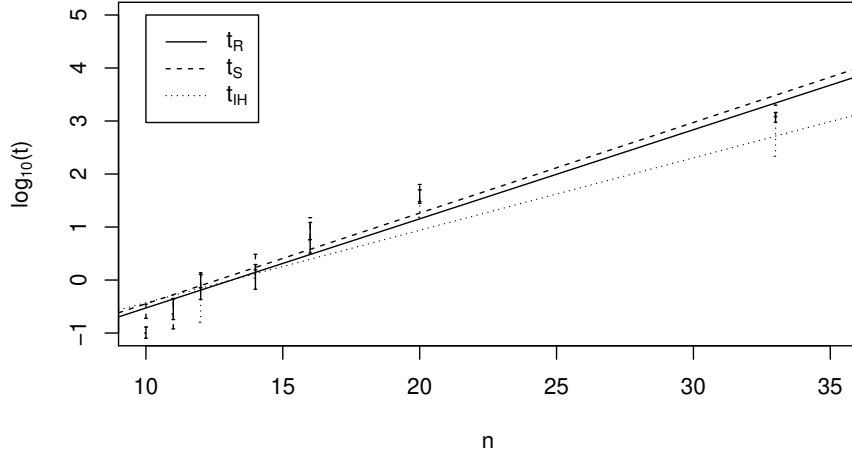
Figure 2: Linear regression for the runtime t_n^R, t_n^S, and t_n^{IH}, corresponding to 3D rectilinear instances, sampled instances, and instances which are solution of IH-OPT($\boldsymbol{x}_n^R$), respectively.

In addition, for three small instances, namely `gr24` and two 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. We selected these 3 instances since they are challenging for Concorde. The motivation of this study is to investigate if the hard instance share some common cost patterns and/or structures of the support graph. Figures 3, 4, and 5 show the support graph of the *easy* and *hard* version of the two instances, where the easy refer to the original cost vector $\boldsymbol{c}_0$ with its SEP vertex solution, and the hard to the optimal solution $\boldsymbol{c}^*$ of IH-OPT. In the support graphs, the dotted edges correspond to solution of SEP having value $x_e = \frac{1}{2}$, while the solid 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 looks 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), 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 $\boldsymbol{c}^0$ and $\boldsymbol{c}^*$ is the same. However, in s_{15} we have the $\boldsymbol{x}^0 = \boldsymbol{x}^*$, while for s_{20} and `gr24` we get $\boldsymbol{x}^* \neq \boldsymbol{x}^0$, as shown in Figure 4 and Figure 5.

5.5   Implementation details

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

1. The branch-and-cut algorithm for solving problem IH-OPT($\bar{\boldsymbol{x}}^{(h)}$) (see Section 3).
2. The sampling procedure from the metric polytope (see Section 4).

In the following paragraphs, we first described 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. In our implementation, we add dynamically both the triangles inequalities and the TSP constraints, in order to keep the core LP problem as small as possible. Note that yet a complete enumeration of triangle inequalities could exhaust the memory of a standard computer for medium values of n.
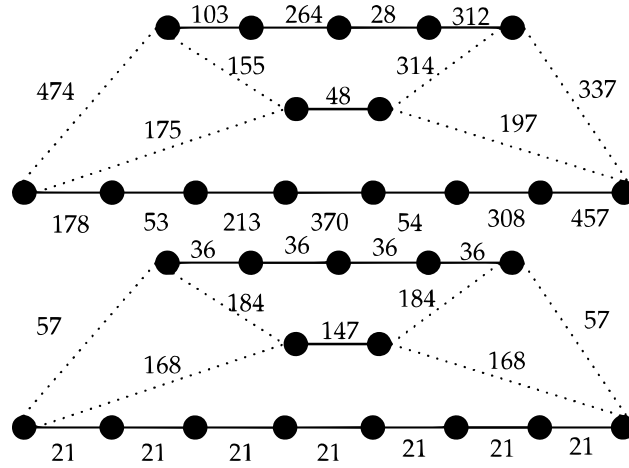
13

Figure 3: Easy (top) and hard (bottom) support graph associated to one of the sampled vertex with $n = 15$. The cost value on the edge of the support graph is also provided.
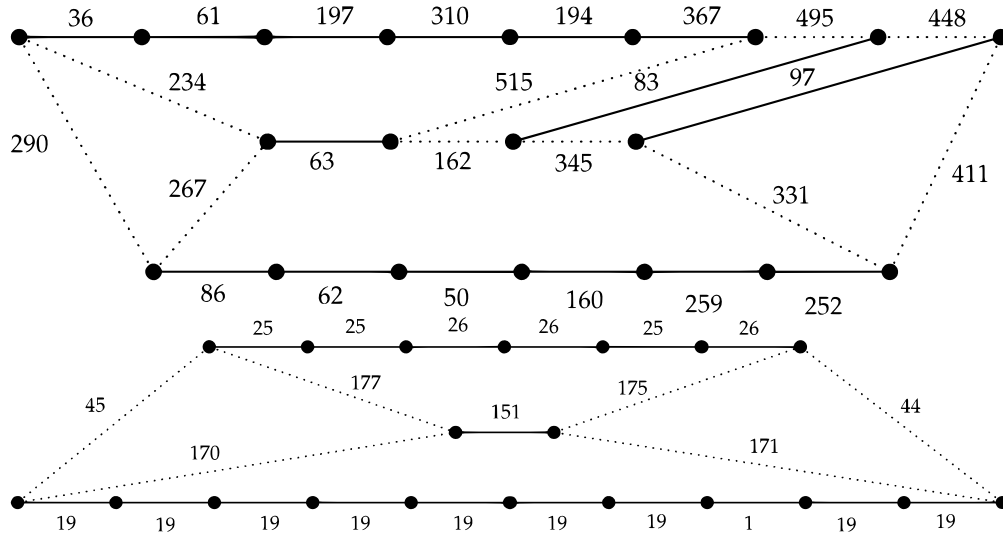

Figure 4: Easy (top) and hard (bottom) support graph associated to one of the sampled vertex with $n = 20$. The cost value on the edge of the support graph is also provided.

For the separation of triangular inequalities, we follow the strategies used in [13]: 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 TSP constraints is definitely more challenging, since it corresponds to the solution of a TSP instance, as shown in (23). As we only need a violated cut, we first separate the TSP constraints heuristically by solving the TSP instance using the Lin-Kernigan local search procedure [21], as implemented in Concorde[1]. Note that the LK-H heuristic only handle integer costs. Hence, we use the support of Gurobi for lazy constraints, which allows to run our separation procedure only on incumbent integer solutions. Whenever the LK-H heuristic finds returns a TSP solutions whose incidence vector z satisfies $z_{ij} c_{ij} \geq \Delta$, with Δ as described in Section 3, we run a second exact TSP algorithm. For the exact separation of TSP constraints we use Concorde compiled using CPLEX 12.8 as LP solver, by using the default parameter settings. The optimal solution of the LK-H heuristic is given as warm start to the TSP branch-and-cut implementation.

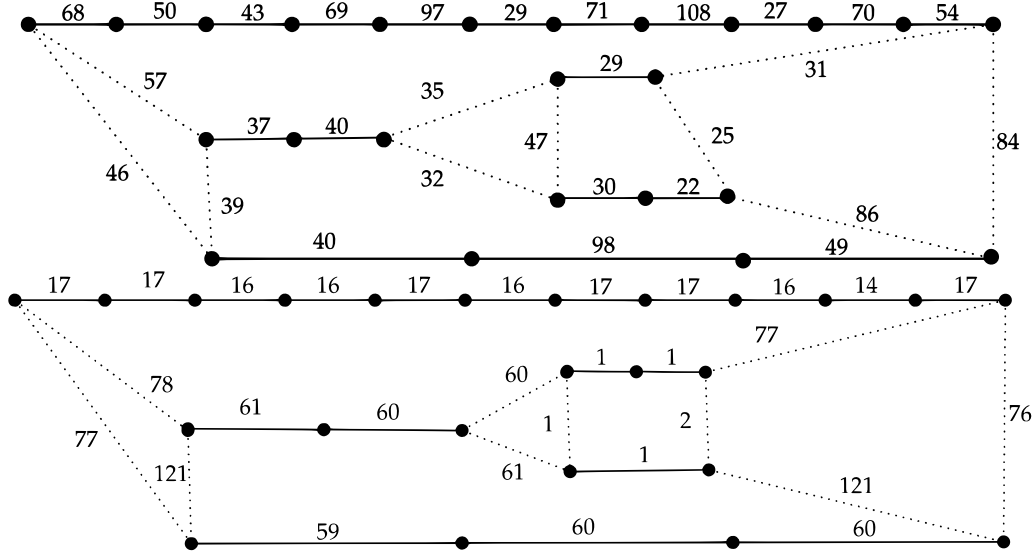[1]http://www.math.uwaterloo.ca/tsp/concorde/downloads/codes/src/co031219.tgz.

Figure 5: Easy (top) and hard (bottom) support graph associated to the instance gr24 of the TSPLIB. The cost value on the edge of the support graph is also provided.

Table 6: 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	16.3	10.77
std. dev concorde time	0.977	2.0	1.924
bayg29, hard			
Integrality gap	1.159	1.186	1.187
avg. concorde mean time	4.157	82.5	32.85
std. dev concorde time	1.271	17.6	9.200
bays29, hard			
Integrality gap	1.197	1.229	1.228
avg. concorde mean time	11.108	64.2	47.392
std. dev concorde time	1.790	18.2	16.137

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 TSP constraints with the LK-H heuristic (see, for instance, the computational results in Table 2). Once the LK-H heuristic does not find any TSP violated constraints, we stop, and we collect all the triangle inequalities violated by no more than a threshold $\tau = 0.05$, and all the generated TSP 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 at Step 3, we have used the implementation provided by [12], which is based on the original algorithm introduced in [32]. 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 (27). In theory, any value of $\Delta \geq n$ should be equivalently good. In practice, we have observed that for different values of Δ we get TSP instances of different (runtime) difficulty. Table 6 shows the computational results for solving IH-OPT over sampled TSP instances using different value of Δ. Since the hardest instances were generated while using $\Delta = 1000$, we fix this value in all of tests reported in this paper. As a future work, we plan to investigate further the impact of the parameter Δ.

6   Conclusions

In this work, we have introduced a computational procedure to generate metric TSP instances which have large integrality gaps and are challenging for Concorde, the state-of-the-art TSP 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 [18, 36]. 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 hope that 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 introduced in [18] and [36]. Hence, we believe that our `Hard-TSPLIB` instances could help further studies in the cost structures of TSP instances.

We emphasize that our framework is really general and can be applied to any combinatorial optimization problem. There are several combinatorial optimization problems that have metric costs, such as, for instance, the metric Steiner tree [3] or the shortest Euclidean minimum spanning tree [10]. The main difference would be to change the TSP constraints into different inequalities for the corresponding combinatorial objects.

Acknowledgements

References

[1] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook. *The traveling salesman problem: a computational study*. Princeton university press, 2006.

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

[3] M. Bern and P. Plassmann. The Steiner problem with edge lengths 1 and 2. *Information Processing Letters*, 32(4): 171–176, 1989.

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

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

[6] S. Boyd and P. 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.

[7] S. Boyd, R. Sitters, S. van der Ster, and L. Stougie. TSP on and subcubic graphs. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 65–77. Springer, 2011.

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

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

[10] M. Fischetti, H. W. Hamacher, K. Jørnsten, and F. Maffioli. Weighted k-cardinality trees: Complexity and polyhedral structure. *Networks*, 24(1):11–21, 1994.

[11] M. Fischetti, A. Lodi, M. Monaci, D. Salvagnin, and A. Tramontani. Improving branch-and-cut performance by random sampling. *Mathematical Programming Computation*, 8(1):113–132, 2016.

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

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

[14] M. Grötschel, L. Lovàsz, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2. Springer-Verlag, 1988.

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

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

[17] S. 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.

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

[19] D. S. Johnson and L. A. McGeoch. Experimental analysis of heuristics for the STSP. In *The traveling salesman problem and its variations*, pages 369–443. Springer, 2007.

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

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

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

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

[24] C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, 7(4):326–329, 1960.

[25] J. E. Mitchell. Branch-and-cut algorithms for combinatorial optimization problems. *Handbook of applied optimization*, 1:65–77, 2002.

[26] A. Orman and H. P. Williams. A survey of different integer programming formulations of the travelling salesman problem. In *Optimisation, econometric and financial analysis*, pages 91–104. Springer, 2007.

[27] M. Padberg and G. Rinaldi. Facet identification for the symmetric traveling salesman polytope. *Mathematical programming*, 47(1):219–257, 1990.

[28] M. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 33(1):60–100, 1991.

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

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

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

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

[33] D. P. 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.

[34] L. A. Wolsey. Heuristic analysis, linear programming and branch and bound. In *Combinatorial Optimization II*, pages 121–134. Springer, 1980.

[35] Z. B. Zabinsky, R. L. Smith, S. Gass, and M. Fu. Hit-and-run methods. *Encyclopedia of Operations Research & Management Science*, pages 721–729, 2013.

[36] X. Zhong. Lower Bounds on the Integraliy Ratio of the Subtour LP for the Traveling Salesman Problem. *arXiv preprint arXiv:2102.04765*, 2021.