

Vehicle Routing in Theory and Practice

DISSERTATION
ZUR
ERLANGUNG DES DOKTORGRADES (DR. RER. NAT.)
DER
MATHEMATISCH-NATURWISSENSCHAFTLICHEN FAKULTÄT
DER
RHEINISCHEN FRIEDRICH-WILHELMS-UNIVERSITÄT BONN

VORGELEGT VON
JANNIS BLAUTH
AUS
AHLEN

BONN, APRIL 2024

Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der
Rheinischen Friedrich-Wilhelms-Universität Bonn

Erstgutachter: Prof. Dr. Jens Vygen
Zweitgutachterin: Prof. Dr. Vera Traub

Tag der Promotion: 12.07.2024
Erscheinungsjahr: 2024

Acknowledgments

First and foremost, I would like to thank my advisor Prof. Dr. Jens Vygen for his excellent support and outstanding supervision. Discussions with Jens are always enriching and it is truly remarkable how much time he devotes to supporting his students, including me. It is a great pleasure to work with him.

Next, I want to say special thanks to Dr. Dirk Müller who has been working on the BonnTour code since the very beginning. He is always willing to help and I have learned a lot from him, especially in terms of coding.

I also want to thank my other co-authors Antonia Ellerbrock, Prof. Dr. Stephan Held, Dr. Nathan Klein, Dr. Martin Nägele, Niklas Schlomberg, Prof. Dr. Vera Traub, and Thorben Tröbst. Working with each of them is a great pleasure.

Additionally, I want to thank my colleague Luise Puhlmann and our student assistants who complete our BonnTour team. It is always a lot of fun to work with them.

Special thanks also go to our industry partner Greenplan. Thanks to them and our cooperation, BonnTour is used successfully in various practical scenarios. I would particularly like to mention Dr. Clemens Beckmann and Florian Merget, as well as the mathematical team of Greenplan.

Moreover, I would like to thank Susanne Armbruster, Antonia Ellerbrock, Dr. Dirk Müller, Dr. Martin Nägele, Luise Puhlmann, and especially Prof. Dr. Jens Vygen for their helpful feedback on early versions of this thesis.

Last but not least, I would like to thank all my other colleagues at the institute for the pleasant and inspiring working atmosphere, and our system administration and secretary's office for their steady help. I also want to thank my family and friends for their support, especially Verena Küther for having my back.

Contents

1	Introduction	1
1.1	Outline and contributions	1
1.1.1	Improved approximation guarantees for TSP generalizations . .	2
1.1.2	Vehicle routing in practice	5
1.2	Preliminaries	6
1.2.1	Bounding the running time of algorithms	6
1.2.2	Linear programming	6
1.2.3	Graph theory	7
1.2.3.1	Walks and paths	8
1.2.3.2	Cuts and s - t -cuts	8
1.2.3.3	Trees and spanning trees	8
1.2.3.4	Matchings and Q -joins	8
1.2.4	The traveling salesperson problem	9
I	Approximation algorithms for generalizations of the traveling salesperson problem	11
2	Improved guarantees for the prize-collecting TSP	13
2.1	Introduction	13
2.1.1	The prize-collecting TSP	13
2.1.2	Prior work on PCTSP	14
2.1.3	Our results and contributions	15
2.1.4	Related problems	16
2.2	Overcoming the weaknesses of threshold rounding	18
2.2.1	Our results and contributions	18
2.2.2	Review of the classical threshold rounding algorithm	19
2.2.3	A refined tour construction	20
2.2.4	Choosing the right threshold	25
2.2.5	Sampling walks and derandomization	27
2.2.5.1	Proof of Lemma 10	28
2.2.5.2	A deterministic selection of walks	30
2.2.5.3	Deterministic selection of a threshold	33
2.2.6	Proof of Theorem 13	34

2.3	Simple 2-approximations for PCTSP and PCST through decompositions	36
2.4	Pruning the trees for cheaper parity correction	39
2.4.1	Our algorithm	39
2.4.2	A $(1 + \sqrt{5})/2$ -approximation guarantee	42
2.4.3	Getting below 1.6	45
2.5	From splitting off to tree decompositions	48
3	Improved guarantees for capacitated vehicle routing	53
3.1	Introduction	53
3.1.1	Outline	55
3.1.2	Related work	56
3.2	Overview of our algorithm	58
3.3	Reduction to vehicle routing with target groups	62
3.4	LP-based approach to vehicle routing with target groups	65
3.4.1	The linear program	66
3.4.2	Existence of a cheap LP solution (Proof of Lemma 44)	69
3.4.3	Rounding LP solutions (Proof of Lemma 43)	71
3.4.4	Improving the approximation ratio for regret-bounded vehicle routing	74
3.5	Comments on integrality ratios	75
II	Solving real-world vehicle routing problems: The BonnTour algorithm	77
4	Introduction	79
4.1	Problem description	80
4.2	Outline and contributions	80
4.3	Related work	82
5	Incorporating time-dependent travel times	85
5.1	Basic operations on arrival time functions	85
5.2	Evaluating and updating tours	89
5.2.1	Maintaining arrival time functions of tours	89
5.2.2	Tour scheduling	92
6	A local-search based heuristic for real-world vehicle routing problems	95
6.1	Constructing and improving solutions	95
6.1.1	Preprocessing the street map and computing fastest paths	96
6.1.2	Preprocessing items	97
6.1.3	Constructing tours	98
6.1.4	Post-optimization	99
6.1.5	Random walk	101
6.2	Detailed model description	103
6.2.1	Breaks	104

6.2.2	Multiple pickup and delivery locations	104
6.2.3	Compact tours	105
7	Experimental results	107
7.1	Results on standard vehicle routing benchmarks	107
7.2	New benchmarks with time-dependent travel times	109
7.3	Results on new benchmarks	111
7.4	Demonstrating selected BonnTour features	118
7.4.1	Multiple pickup and delivery locations	118
7.4.2	Compact tours	120
	Summary	123
	Bibliography	125

Chapter 1

Introduction

1.1 Outline and contributions

In this thesis, we study combinatorial optimization problems arising in logistics. Due to the steadily increasing amount of goods transferred each single day in Europe and all over the world, there is great economic interest in finding high-quality solutions to these problems in reasonable computation time. We focus on optimization problems that arise, for example, in last mile delivery and can be summarized under the name *vehicle routing*.

We tackle these problems from two different points of view. In the first part of this thesis, we study their computational complexity. All problems we look at are generalizations of the famous *traveling salesperson problem*: a driver needs to visit certain customers (e.g., to deliver goods) and return to the starting point at the end of the tour. The goal is to compute a tour of minimum length (or cost). Since this problem is NP-hard, there is no *efficient* algorithm for computing an optimum solution unless $P = NP$. (We call an algorithm efficient if it has polynomial running time, cf. [Section 1.2.1](#).) For NP-hard problems, one main focus of research lies in finding algorithms that give certain guarantees on the cost of the computed solution. An α -*approximation algorithm* for a combinatorial minimization problem is a polynomial-time algorithm that guarantees that the cost of the returned solution is at most α times the cost of an optimum solution. α is called *approximation ratio* or *approximation guarantee*.

In this thesis, we provide improved approximation guarantees for two well-known natural generalizations of the traveling salesperson problem. Both variants are of high relevance in logistics. In the first variant, known as the *prize-collecting traveling salesperson problem*, we are allowed to reject customer requests, which comes at customer-specific penalties. The goal is then to optimize the sum of tour length and total penalty for rejected inquiries. Almost all real-world problems in logistics are actually prize-collecting with the option to reject requests or delay them to the next day which comes at a loss of income and customer satisfaction. The customer-specific penalties can be used to model these losses.

The second variant of the traveling salesperson problem for which we provide improved approximation guarantees is the *capacitated vehicle routing problem*. In many

applications, the goods that need to be delivered to customers exceed the capacity of a single vehicle. Hence, we need to distribute the goods to several vehicles and compute an efficient route for each of these such that the sum of tour lengths is minimized.

In the second part of this thesis, we study vehicle routing problems from a practical point of view. On typical real-world instances, the approximation algorithms discussed in the first part of this thesis are not practical, neither in terms of solution quality nor in terms of running time. Moreover, we need to satisfy various additional constraints such as time windows, working time limits, limited heterogeneous vehicle fleets, and many more. Hence, we discuss heuristics for which we cannot provide any worst-case guarantees, but which turn out to perform well on realistic instances. More precisely, we present an algorithm called *BonnTour* that covers a wide class of vehicle routing problems and provides high-quality solutions in reasonable computation time. On numerous benchmarks, the cost of the computed solution comes close to the optimum or the best known. The *BonnTour* algorithm is developed in cooperation with our industry partner [Greenplan](https://greenplan.de)¹, and is used on a daily basis to solve real-world vehicle routing problems for various companies.

The model covered by *BonnTour* takes into account that the travel time from one location to another can vary a lot over the day due to congestion and time-dependent road restrictions. Most vehicle routing algorithms approximate travel times by constant estimates. However, evaluating the thereby obtained solutions with respect to the actual (time-dependent) travel times often leads to time window violations and overtime. Tour plans computed by *BonnTour* guarantee to satisfy all these constraints and are therefore much more robust in their execution.

In the following, we discuss the main contributions of this thesis in more detail. Please refer to [Section 1.2](#) for basic definitions and notation that will be used repeatedly.

1.1.1 Improved approximation guarantees for **TSP** generalizations

In the first part of this thesis, we study two well-known generalizations of the traveling salesperson problem, **TSP** for short. In the classical interpretation of the **TSP**, there is a salesperson who needs to visit a set of cities V and wants to minimize the length of their tour. Following this interpretation, in prize-collecting **TSP**, the salesperson can decide whether or not to include each city besides the starting one in their tour at the cost of a city-dependent penalty. We study this problem in [Chapter 2](#). We provide two new algorithms, both beating the previous best known approximation guarantee for this problem. As one can see in [Table 1.1](#), the algorithms discussed in this thesis significantly reduce the gap between the approximability of prize-collecting **TSP** and **TSP**. It is still open whether this gap can (almost) be closed. Due to the presented progress, this possibility feels less out of reach. In this thesis, we obtain an approximation guarantee of 1.599.

The first of the two new algorithms is discussed in [Section 2.2](#) and is joint work with Martin Nägele [[BN23](#)]. We refine the threshold rounding approach by Bienstock, Goemans, Simchi-Levi, and Williamson [[BGSW93](#)]. In the classical threshold rounding approach, first, a natural linear programming relaxation of the prize-collecting **TSP** is

¹For more information on our partner company, see, for example, <https://greenplan.de>.

Algorithm			Approximation ratio	
			Black-box guarantee	Current value
Threshold rounding		[BSW93]	$1 + \beta \approx$	2.5
Primal-dual		[GW95]		2
Primal-dual & subset selection		[ABHK11]	$2 - (2 - \rho)^2 / (2 + \rho)^2 \approx$	1.979
Primal-dual & threshold rounding		[Goe09]	$1 / (1 - \frac{1}{\beta} e^{1-2/\beta}) \approx$	1.914
Component sampling	Section 2.2	[BN23]		1.774
Tree pruning	Section 2.4	[BKN24]		1.599

Table 1.1: Approximation algorithms for the prize-collecting TSP and their approximation guarantees. ρ denotes the best known approximation guarantee for the TSP, and β denotes the best known approximation guarantee for the TSP with respect to the optimum value of the well-known Held-Karp relaxation defined in Section 1.2.4. Both values are slightly below $3/2$ [KKO21; KKO22; KKO23; GKL23]. Even for $\rho = 1$ and $\beta = 4/3$ (it is well known that the integrality gap of the Held-Karp relaxation is at least $4/3$, see for example [Wil90]), both algorithms presented in this thesis remain superior to previous approaches. Besides the result by [ABHK11], all guarantees are with respect to the optimum value of the natural linear programming relaxation stated in Section 2.1.1, therefore implying the same upper bound on its integrality gap.

solved, which contains a y -variable for each city, and x -variables for edges connecting cities. In an integral solution, the y -variables tell which cities to visit, and the edge-variables yield the salesperson's tour. In a fractional solution, the y -variables yield for each city a number between 0 (we should not visit the city and pay the penalty instead) and 1 (we should definitely visit the city). Bienstock, Goemans, Simchi-Levi, and Williamson [BSW93] call a TSP algorithm in a black-box way on the set of cities for which this value is larger than some threshold. On a very high level, we refine their approach in the following way. By opening the black box, we are able to construct a tour on the set of cities exceeding the threshold, while visiting each city below the threshold with a certain probability. This reduces the penalties we have to pay in expectation. We maintain the same bound on the tour length as in classical threshold rounding. We achieve this by decomposing the x -part of an optimum LP solution into components that will serve as building blocks for tour construction. We sample a spanning tree in an auxiliary graph, where each edge corresponds to one component of the decomposition, and apply parity correction to the union of thereby sampled components. Choosing the threshold randomly with respect to a certain distribution yields a 1.774-approximation as stated in Table 1.1. We also present a deterministic variant of our algorithm.

With an even simpler approach, we further improve the approximation ratio to slightly below 1.6 in Section 2.4. This is joint work with Nathan Klein and Martin Nägele [BKN24]. Again, we apply decomposition techniques to an optimum LP solution (x, y) . We decompose the x -part of the LP solution into trees that all contain the city in which the salesperson starts (each tree-vertex corresponds to a city). Informally speaking, each city v is contained in a y_v -fraction of these trees. In order to get a good bound on the cost of parity correction, we prune each tree T so that afterwards each cut $\delta(S)$ with $|S \cap \text{odd}(T)|$ odd either contains three tree edges, or $x(\delta(S))$ is large (while paying manageable additional penalties). We use this to construct a point z of low cost in the dominant of the $\text{odd}(T)$ -join polytope by combining x and the incidence

vector of T . Note that $c^\top z$ constitutes an upper bound on the cost of parity correction (cf. [Section 1.2.3.4](#)). In the end, we return the best tour constructed this way.

The other generalization of the [TSP](#) we deal with in the first part of this thesis is the capacitated vehicle routing problem, [CVRP](#) for short. We study this problem in [Chapter 3](#). In this problem the salesperson has to ship goods of certain sizes to each city, where the goods are initially located at the starting one, often called *depot* in this setting. As the capacity of the vehicle used by the salesperson is limited, the salesperson has to return to the depot from time to time to reload the vehicle. After shipping all goods, the salesperson has to return to the depot. One often distinguishes between *splittable* and *unsplittable* capacitated vehicle routing. In splittable [CVRP](#), each good can be split to be shipped fractionally on several tours. In unsplittable [CVRP](#) though, all goods must be delivered in one piece. If not explicitly stated, we consider the unsplittable variant. The special case of unsplittable [CVRP](#) where all goods have the same size is called *unit-demand* capacitated vehicle routing. It is well known that this special case is more or less equivalent to the splittable variant (see, e.g., [\[Trö19\]](#)). More precisely, any α -approximation algorithm for splittable [CVRP](#) yields an α -approximation algorithm for the unit-demand case. Conversely, each approximation algorithm for the unit-demand case implies a pseudo-polynomial time algorithm for splittable [CVRP](#) with the same guarantee. Although the general reduction only gives pseudo-polynomial running time, the known approximation algorithms for the unit-demand case can be transformed into polynomial-time algorithms for splittable [CVRP](#) while preserving the approximation ratio.

For more than 30 years, the iterated tour partitioning heuristic by Haimovich and Rinnooy Kan [\[HR85\]](#) and Altinkemer and Gavish [\[AG87\]](#) persisted in yielding the best known approximation guarantees for the three [CVRP](#) variants. These were first beaten by an approach that was initiated in my master’s thesis: For all three variants, the approximation guarantee was improved by some small value in the order of 10^{-12} [\[Bla20\]](#). In [Chapter 3](#), we simplify and refine this approach in several places, pushing the improvement to be in the order of 10^{-3} . This is joint work with Vera Traub and Jens Vygen [\[BTV23\]](#). Our main new contribution consists of proposing a different strategy for solving a subproblem established in [\[Bla20\]](#). In this subproblem we are asked to find a set of paths starting in the depot and ending in certain groups of cities such that all cities are visited by some path. We leverage an algorithm for *regret-bounded vehicle routing* by Friggstad and Swamy [\[FS17\]](#) to compute a good solution to this subproblem. By improving their algorithm in one place, we get as a by-product improved approximation guarantees for regret-bounded vehicle routing and the *school bus problem*. (These problems are introduced in [Chapter 3](#).) More precisely, we improve the approximation guarantees from 15 and 16 to 10 and 11, respectively.

For unsplittable [CVRP](#), the approximation ratio was recently further improved by Friggstad, Mousavi, Rahgoshay, and Salavatipour [\[FMR22\]](#). They remark that calling their algorithm and ours and returning the better solution further improves the approximation ratio by some small constant. They comment that this constant can apparently be chosen in the order of 10^{-3} . Hence, for all three major [CVRP](#) variants, our algorithm is a building block of the state-of-the-art approximation algorithm.

1.1.2 Vehicle routing in practice

In the second part of this thesis, we present an algorithm called BonnTour that covers various aspects arising in real-world vehicle routing problems. Most of the presented content is based on joint work with Stephan Held, Dirk Müller, Niklas Schlömborg, Vera Traub, Thorben Tröbst, and Jens Vygen [BHMSTTV22]. Luise Puhlmann and our student assistants also make significant contributions to the BonnTour code.

As mentioned earlier, the BonnTour algorithm is developed in cooperation with Greenplan. The algorithm is used on a daily basis to solve real-world vehicle routing problems for various companies with different needs and requirements.²

One main aspect is that our algorithm incorporates travel time fluctuations caused by congestion. We work with time-dependent travel time estimates based on historical data. Time-dependent travel times are often modeled using piecewise linear arrival time functions (ATFs) that map the departure time at the starting point to the arrival time at the destination (see, e.g., [IGP03]). We require each ATF to be nondecreasing, a reasonable assumption since there is always the option to wait for a better departure time. A large fraction of the overall computation time of our algorithm is spent in performing basic operations on ATFs such as composing two or more ATFs or computing the point-wise minimum. In Chapter 5, we provide a comprehensive overview of fast algorithms for these problems and describe how to evaluate insertion and deletion operations in tours efficiently in the presence of time-dependent travel times.

In Chapter 6, we give a high-level overview of our approach. As input, we are given a set of items. Each item has a pickup time window, a pickup address, a delivery time window, and a delivery address. Recently, we added the possibility to pass several feasible pickup and delivery addresses per item, which can for example be used to model scenarios where identical copies of certain items might be available at several locations (depots or shops). In addition to the items, we are given a set of (possibly heterogeneous) vehicles. We guarantee to satisfy various constraints such as working time limits and break regulations. We optimize a fine-grained cost function that includes, for example, working time costs (possibly with overtime options at certain extra costs), costs for the covered distance, and setup costs for the vehicles. A detailed model description is given in Section 6.2. We compute a solution following a local-search based approach: After constructing an initial solution, we optimize this solution by a sequence of local improvements. A detailed description is given in Section 6.1.

We conclude the thesis with various experimental results in Chapter 7. We conducted experiments on several public benchmarks for capacitated vehicle routing and generalizations of this problem. Most of these benchmarks consider constant travel times. Despite the huge handicap that BonnTour is designed for a much more general model and the resulting running time overhead, we only get slightly worse results in reasonable computation time than state-of-the-art algorithms that were designed to perform best on these instances (cf. Section 7.1). For some existing benchmark instances with time-dependent travel times, we report new best known solutions.

Due to the lack of large-scale and realistic benchmarks for vehicle routing with time-

²See <https://greenplan.de> for a selection of partners relying on Greenplan and the BonnTour algorithm.

dependent travel times, we designed a new set of benchmark instances using real-world data that is publicly accessible. We describe how we generated our new benchmark set in [Section 7.2](#). As shown in [Section 7.3](#), computing solutions with constant estimates on the travel times and evaluating these solutions with respect to the actual time-dependent travel times either leads to a lot of time window violations (when working with average travel times) or to significantly higher costs (when using pessimistic travel time estimates). This quantifies the benefit of incorporating time-dependent travel times already during optimization.

Finally, in [Section 7.4](#), we demonstrate selected features of our algorithm, both on (modifications of) our new benchmark set, as well as on real-world instances from our industry partner [Greenplan](#).

1.2 Preliminaries

In this section, we introduce some basic definitions and notation that we will use repeatedly.

1.2.1 Bounding the running time of algorithms

Definition 1. Given two functions $f, g: \mathbb{Z}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$, we say that f is $O(g)$ if there are constants $c_1, c_2 \geq 0$ such that $f(x) \leq c_1 \cdot g(x) + c_2$ for all $x \in \mathbb{Z}_{\geq 0}$. We say that f is $\Omega(g)$ if g is $O(f)$. We say that f is $\Theta(g)$ if f is $O(g)$ and g is $O(f)$.

When talking about the running time of an algorithm, we consider the number of elementary steps until the algorithm terminates. Elementary steps are, e.g., variable access, variable assignment, or simple arithmetic operations like addition, subtraction, multiplication, division, or the comparison of two numbers. We say that an algorithm runs in *polynomial time* if there is an integer k such that for each rational input that can be stored on n bits (using binary representation of the numerators and denominators) the running time and the total memory consumption is $O(n^k)$. We call a polynomial-time algorithm *efficient*. An algorithm runs in *pseudo-polynomial time* if its running time is bounded by a polynomial in the input size and the largest input number.

1.2.2 Linear programming

Given an inequality system $Ax \geq b$ with $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$, its feasible region $P := \{x \in \mathbb{R}^n: Ax \geq b\}$ is called *polyhedron*. A bounded polyhedron is called *polytope*. Given a polyhedron P , a *vertex solution* (or basic solution) is a feasible solution $x \in P$ that cannot be written as convex combination of other feasible solutions. A polyhedron or polytope is called *integral* if all vertex solutions are integral.

In linear programming, we are asked to find a feasible solution $x \in P$ that minimizes a given linear cost function $c \in \mathbb{R}^n$, i.e.,

$$\min_{x \in P} c^\top x \quad (1.1)$$

(1.1) is called *linear program*, or LP for short. Grötschel, Lovász, and Schrijver [GLS81] showed that if the optimum value of (1.1) is bounded, an optimum vertex solution can be computed in polynomial time using the ellipsoid method if we can solve the *separation problem* in polynomial time. The separation problem is, for a point $z \in \mathbb{R}^n$, decide that $z \in P$ or find a separating hyperplane, i.e., a vector $a \in \mathbb{R}^n$ such that $a^\top z > a^\top x$ for each $x \in P$. Hence, if we can solve the separation problem in polynomial time and P is integral, we can find an optimum integral solution in polynomial time.

The *dual* of a linear program

$$\begin{aligned} \min \quad & c^\top x \\ & Ax \geq b \\ & x \geq 0 \end{aligned}$$

is the linear program

$$\begin{aligned} \max \quad & b^\top y \\ & A^\top y \leq c \\ & y \geq 0. \end{aligned}$$

Note that dualizing the dual LP results in the original (*primal*) linear program. It is well known that if the primal and its dual admit feasible solutions, then both linear programs are bounded and their optimum values are the same [Neu63; GKT51]. This property is known as *strong duality*.

A *linear programming relaxation* of a combinatorial optimization problem is an instance-dependent linear program such that for each instance, any feasible solution of the optimization problem corresponds to an integral point in the polyhedron of the linear program of the same cost, and vice versa. Consider a linear programming relaxation of a combinatorial minimization problem. The supremum (among all instances) of the ratio of the optimum solution cost and the LP value is called *integrality gap* of the linear programming relaxation.

1.2.3 Graph theory

An *undirected graph* on vertex set V is a pair $G = (V, E)$, where each *edge* $e \in E$ is an unordered pair of vertices, i.e., $e \in \binom{V}{2}$. A *directed graph* on vertex set V is a pair $G = (V, E)$, where each edge $e \in E$ is an ordered pair of vertices, i.e., $e = (v, w) \in V \times V$ with $v \neq w$. Moreover,

- an undirected or directed graph $G = (V, E)$ is called *complete* if $E = \binom{V}{2}$ or $E = \{(v, w) \in V \times V : v \neq w\}$, respectively.
- A *multigraph* is a graph where each edge is allowed to appear multiple times.
- A graph $H = (V_H, E_H)$ is called *subgraph* of $G = (V, E)$ if $V_H \subseteq V$ and $E_H \subseteq E$. We write $V[H] := V_H$ and $E[H] := E_H$.
- A *weighted graph* is a graph $G = (V, E)$ with edge lengths $c \in \mathbb{R}^E$.

For a weighted graph $G = (V, E)$ and $F \subseteq E$, we use $c(F) := \sum_{e \in F} c_e$. For $e \in E$, we write $c(e) := c(\{e\})$. In this thesis, if not explicitly specified, all graphs are undirected.

1.2.3.1 Walks and paths

A *walk* in a graph is a sequence of vertices such that two consecutive vertices are connected by an edge. A *closed walk* starts and ends in the same vertex. A walk is called *path* if no vertex appears more than once. Often, we identify walks and paths with the corresponding set of edges.

1.2.3.2 Cuts and s - t -cuts

Let $G = (V, E)$ be a graph and $S \subseteq V$. If G is undirected, we let

$$\delta(S) := \{e \in E : |e \cap S| = |e \setminus S| = 1\} .$$

$\delta(S)$ is called *cut* in G . For $v \in V$, we use $\delta(v) := \delta(\{v\})$. The cardinality of $\delta(v)$ is called *degree* of v . If G is directed, let $\delta^+(S) := \{(v, w) \in E : v \in S, w \notin S\}$ and $\delta^-(S) := \{(v, w) \in E : v \notin S, w \in S\}$. For $v \in V$, we use $\delta^+(v) := \delta^+(\{v\})$ and $\delta^-(v) := \delta^-(\{v\})$.

Given $s, t \in V$, an s - t cut is a cut $\delta(S)$ or a cut $\delta^+(S)$ (depending on whether G is directed) with $s \in S$ and $t \notin S$. A minimum s - t cut in a weighted directed or undirected graph can be computed in polynomial time (see, e.g., [EK72]).

1.2.3.3 Trees and spanning trees

Let $G = (V, E)$ be an undirected graph. G is *connected* if $\delta(S) \neq \emptyset$ for each nonempty $S \subsetneq V$. G is called *cycle* if G is connected and each vertex has degree two. A *tree* is a connected graph that does not contain a cycle, i.e., no subgraph is a cycle. A *spanning tree* in G is a subgraph of G on vertex set V that is a tree. It is well known that a minimum cost spanning tree in a weighted undirected graph G can be computed in polynomial time (see, e.g., [Kru56; Pri57]).

The *spanning tree polytope* $P_{\text{ST}}(G)$ over a graph $G = (V, E)$ is the convex hull of all incidence vectors of spanning trees in G , and can be described by

$$P_{\text{ST}}(G) = \{x \in \mathbb{R}_{\geq 0}^E : x(E) = |V| - 1, x(E[S]) \leq |S| - 1 \forall S \subsetneq V, S \neq \emptyset\}$$

as shown by Edmonds [Edm70].

1.2.3.4 Matchings and Q -joins

Let $G = (V, E)$ be an undirected graph. A *matching* is a set of edges with degree at most one at every vertex. For $Q \subseteq V$ with $|Q|$ even, a Q -*join* is a set of edges with odd degrees precisely at vertices in Q . A minimum cost Q -join in an undirected graph with nonnegative edge weights $c \in \mathbb{R}_{\geq 0}^E$ can be computed in polynomial time [EJ73].

The Q -*join polytope* over a graph $G = (V, E)$ is the convex hull of all incidence vectors of Q -joins in G . Edmonds and Johnson [EJ73] showed that the cost of a shortest Q -join is bounded by the cost $c^\top z$ of a point z that is feasible for the dominant of the Q -join polytope (also called *Q -join polyhedron*), which is given by

$$P_{Q\text{-join}}^\uparrow := \{x \in \mathbb{R}_{\geq 0}^E : x(\delta(S)) \geq 1 \forall S \subseteq V \text{ with } |S \cap Q| \text{ odd}\} .$$

1.2.4 The traveling salesperson problem

Most problems discussed in this thesis are generalizations of the (metric) traveling salesperson problem, which is one of the most fundamental problems in combinatorial optimization. In an instance of this problem, we are given a set V of n vertices along with their pairwise distances $c: V \times V \rightarrow \mathbb{R}_{\geq 0}$ that form a *metric*, i.e., the distances are symmetric, and for all vertices $u, v, w \in V$, the triangle inequality $c(u, w) \leq c(u, v) + c(v, w)$ is satisfied.³ The goal is to find a shortest possible cycle among those that contain all vertices and are referred to as *Hamiltonian*. In the classical interpretation, as mentioned earlier, there is a salesperson who needs to visit a set of cities V and wants to minimize the length of their tour. Formally, this problem can be stated as follows.

Metric Traveling Salesperson Problem (TSP): Given a complete undirected graph $G = (V, E)$ with metric edge lengths $c_e \geq 0$ for all $e \in E$, the task is to find a Hamiltonian cycle $C = (V, E_C)$ in G that minimizes $c(E_C)$.

The **TSP** is strongly NP-hard, as otherwise we could efficiently solve the Hamiltonian circuit problem, which is known to be NP-complete [Kar72]. More precisely, the **TSP** is NP-hard even if restricted to instances with $c_e \in \{1, 2\}$ for each $e \in E$. Furthermore, the problem is known to be APX-hard: Karpinski, Lampis, and Schmied [KLS15] have shown that the **TSP** is NP-hard to approximate within a factor of $1^{23}/122$. Hence, one main focus of research lies in finding good approximation algorithms for this problem. Despite extensive research, the $3/2$ -approximation algorithm of Christofides [Chr76] and Serdyukov [Ser87] (also see [Chr22; vS20]) persisted to be the best known for more than four decades, until a recent breakthrough of Karlin, Klein, and Oveis Gharan [KKO21; KKO23], who improved the factor to $3/2 - \varepsilon$ for some $\varepsilon > 10^{-36}$. Gurvits, Klein, and Leake [GKL23] provided a more streamlined proof and slightly improved the approximation ratio to $3/2 - 10^{-34}$.

The Christofides-Serdyukov-algorithm proceeds as follows. First, a minimum length spanning tree T in the input graph is computed. Second, a shortest $\text{odd}(T)$ -join J is added so that in the resulting multigraph each vertex has even degree.⁴ This step is known as *parity correction*. Then, an Eulerian walk is computed in this multigraph, i.e., a closed walk that contains each edge exactly once. Such an Eulerian walk exists because this multigraph is connected and each vertex has even degree. Finally, the returned Hamiltonian cycle is obtained via *shortcutting*: The vertices are visited in order of their first appearance in the Eulerian walk. Note that by the triangle inequality, the length of the cycle obtained via shortcutting can be bounded by the length of the Eulerian walk. Hence, to get bounds on the length of the constructed cycle, it is enough to bound the length of T and J .

³Some definitions of a metric additionally require $c(u, v) > 0$ for all $u \neq v$. Dropping these constraints is then called *semi-metric*. However, in our definition we only require nonnegativity.

⁴For a graph H , we denote by $\text{odd}(H)$ the set of all vertices of odd degree in H . Note that for metric cost functions and any $Q \subseteq V$ of even cardinality, there is always a minimum cost Q -join that is a matching.

1.2. Preliminaries

Wolsey [Wol80] proposed the following LP-based analysis: For any point x feasible for the *Held-Karp relaxation* of the TSP, whose feasible region is given by

$$P_{\text{HK}}(G) := \left\{ x \in \mathbb{R}_{\geq 0}^E : \begin{array}{ll} x(\delta(v)) = 2 & \forall v \in V \\ x(\delta(S)) \geq 2 & \forall S \subsetneq V, S \neq \emptyset \end{array} \right\},$$

one can prove that a shortest spanning tree on V has cost at most $c^\top x$, and a shortest odd(T)-join has cost at most $\frac{1}{2}c^\top x$, therefore proving a $3/2$ -approximation guarantee with respect to a shortest solution $x \in P_{\text{HK}}$.

For more information on the traveling salesperson problem, please refer to the book by Traub and Vygen [TV24] that will be published soon.

Part I

Approximation algorithms for generalizations of the traveling salesperson problem

Chapter 2

Improved guarantees for the prize-collecting TSP

2.1 Introduction

2.1.1 The prize-collecting TSP

In this chapter, we study a generalization of the traveling salesperson problem known as *prize-collecting TSP*, in which the salesperson can decide whether or not to include each city besides the starting one¹ in their tour at the cost of a city-dependent penalty. The prize-collecting TSP is a very natural variation of the TSP and can formally be stated as follows.

Prize-Collecting TSP (PCTSP): Given a complete undirected graph $G = (V, E)$ with metric edge lengths $c_e \geq 0$ for all $e \in E$, a root $r \in V$, and penalties $\pi_v \geq 0$ for all $v \in V \setminus \{r\}$, the task is to find a cycle $C = (V_C, E_C)$ in G that contains the root r and minimizes

$$\sum_{e \in E_C} c_e + \sum_{v \in V \setminus V_C} \pi_v .$$

In other words, for each vertex other than the root, we may decide between covering the vertex by the output cycle or dropping it at cost π_v , and the goal is to find a balance between the tour cost and incurred penalties so as to minimize the sum of the two. We remark that this may lead to degenerate cycles C that may contain only two vertices (or even the root only). Many real-world applications of TSP variants (e.g., in logistics) share the property that it is allowed to not visit some of the vertices at certain extra costs, and the PCTSP is one very natural way of modeling such degrees of freedom.

Clearly, the PCTSP is at least as hard as the TSP: Large enough penalties enforce visiting all vertices. As a consequence, the PCTSP is APX-hard, and one research focus

¹“Rooted” and unrooted versions are reducible to one another while preserving approximability, as noted in, e.g., [ABHK11]. Here, we always require a root vertex r .

2.1. Introduction

lies in finding good approximation algorithms. The first constant factor approximation was given by Bienstock, Goemans, Simchi-Levi, and Williamson [BGSW93] through a threshold rounding approach based on the following linear programming relaxation, where we use $\pi_r := 0$ for convenience:

$$\begin{aligned}
 \min \quad & \sum_{e \in E} c_e x_e + \sum_{v \in V} \pi_v (1 - y_v) \\
 \text{subject to} \quad & x(\delta(v)) = 2y_v \quad \forall v \in V \setminus \{r\} \\
 & x(\delta(r)) \leq 2 \\
 & x(\delta(S)) \geq 2y_v \quad \forall S \subseteq V \setminus \{r\}, v \in S \quad (\text{PCTSP LP relaxation}) \\
 & y_r = 1 \\
 & x_e \geq 0 \quad \forall e \in E \\
 & y_v \in [0, 1] \quad \forall v \in V.
 \end{aligned}$$

Here, the variables y_v can be interpreted to indicate the extent to which v is covered by a fractional solution. Note that we are using a version of the linear relaxation that includes the constraint $x(\delta(r)) \leq 2$. This does not change the optimal value, but, if $|V| \geq 2$, it guarantees that this **PCTSP LP relaxation** generalizes the classical Held-Karp relaxation for the **TSP**, which is recovered by setting $y_v = 1$ for all $v \in V$ (corresponding to the requirement that all vertices need to be fully connected to the root).

Note that the **PCTSP LP relaxation** can be solved in polynomial time through the ellipsoid method, as efficient separation over the constraints $x(\delta(S)) \geq 2y_v$ can be reduced to minimum r - v cut calculations for all $v \in V \setminus \{r\}$ in the graph with capacities given by x .

2.1.2 Prior work on the PCTSP

While Balas [Bal89] was the first to study prize-collecting variations of the **TSP**, the first constant-factor approximation algorithm for **PCTSP** was given by Bienstock, Goemans, Simchi-Levi, and Williamson [BGSW93] through a simple threshold rounding approach: Starting from a solution (x, y) of the **PCTSP LP relaxation**, the Christofides-Serdyukov algorithm is used to construct a tour on all vertices $v \in V$ with $y_v \geq 3/5$, giving an LP-relative $5/2$ -approximation. Goemans and Williamson [GW95] later obtained a 2-approximation through a classical primal-dual approach. More precisely, they showed how to compute a tree T with $c(E[T]) \leq c^\top x$ and $\pi(V \setminus V[T]) \leq \pi^\top (1 - y)$, so that doubling the tree yields the 2-approximation.

The approximation factor of 2 was first beaten by Archer, Bateni, Hajiaghayi, and Karloff [ABHK11]. As a black-box subroutine, they use an approximation algorithm for **TSP** which we assume has ratio ρ . They achieved a $2 - \left(\frac{2-\rho}{2+\rho}\right)^2$ approximation, which—for $\rho = 3/2$ —is approximately 1.979. Their algorithm runs the primal-dual algorithm of Goemans and Williamson and the ρ -approximation algorithm for **TSP** on a carefully selected node set, and outputs the better of the two tours. Goemans [Goe09] then observed that running both threshold rounding for different thresholds and the primal-dual algorithm, and choosing the best among the computed solutions yields an approximation guarantee of $1/(1 - \frac{1}{\beta} e^{1-2/\beta})$, where β denotes the approximation guarantee of an LP-relative approximation algorithm for **TSP** that is used in a black-box way. For $\beta = 3/2$,

the guarantee of Goemans equals approximately 1.914. Goemans was the first to exploit a randomized analysis of threshold rounding, in which the threshold γ is chosen from a specific distribution.

2.1.3 Our results and contributions

We provide two new approximation algorithms for **PCTSP** both beating the so far best known approximation guarantee of roughly 1.914 by Goemans [Goe09]. Both algorithms are based on decompositions of LP-solutions into trees, the key technical tool that allows for our improved guarantees. Whereas the decomposition result used for the second algorithm is similar to an existential result on packing branchings in a directed multigraph by Bang-Jensen, Frank, and Jackson [BFJ95, Theorem 2.6], the first algorithm uses some refined variation. We prove a common generalization of both statements. The proof is deferred to Section 2.5.

The first algorithm for **PCTSP** with an improved guarantee is discussed in Section 2.2 and is joint work with Martin Nägele (see [BN23] for the conference version of the paper). The algorithm refines the classical threshold rounding approach. For threshold γ , we sample a connected subgraph such that each vertex $v \in V$ with $y_v \geq \gamma$ is always contained in the vertex set of this subgraph, whereas each vertex $v \in V$ with $y_v < \gamma$ is contained with probability at least $\exp(-3y_v/4\gamma)$. Since we guarantee that each vertex below the threshold has even degree in this subgraph, parities can be corrected at the same cost as in classical threshold rounding, yielding an approximation guarantee of slightly below 1.774 through a randomized analysis. This guarantee beats those of the previously mentioned algorithms even for $\rho = 1$ and $\beta = 4/3$ (the integrality gap of the linear programming relaxation for **TSP** used in [Goe09], the Held-Karp relaxation, is at least $4/3$).

The decomposition used for the second algorithm is much simpler and can be interpreted as a distribution μ over a polynomial number of trees $\mathcal{T}$ rooted at r such that (i) $\mathbb{E}_{T \sim \mu} [c(E[T])] \leq c^\top x$ and (ii) $\mathbb{P}_{T \sim \mu} [v \in V[T]] = y_v$ for all $v \in V$.² In Section 2.3, as a warm up, we showcase that this decomposition immediately gives rise to simple 2-approximation algorithms for the **PCTSP** and the *prize-collecting Steiner tree problem*, which is introduced in Section 2.1.4. Then, in Section 2.4, we discuss the second proposed algorithm which yields the currently best known approximation guarantee for **PCTSP**. This is joint work with Nathan Klein and Martin Nägele (see [BKN24] for the conference version of the paper). After some preprocessing we sample a tree according to the distribution μ mentioned above. Then we perform a novel pruning step by removing all subtrees containing only vertices v with $y_v < \gamma$ for some threshold parameter γ . We apply parity correction to the thereby obtained tree. The pruning step is key to an improved bound on the cost of parity correction while paying a manageable additional amount of penalties. Trying different parameters and returning the best solution yields the desired guarantee.

Theorem 2. *There is an LP-relative 1.599-approximation algorithm for **PCTSP**.*

²For a tree T chosen from a distribution μ we write $T \sim \mu$.

It remains an open question whether there is an efficient algorithm for **PCTSP** that matches—in terms of the approximation factor—the $3/2$ -approximation for **TSP** by Christofides [Chr76] and Serdyukov [Ser87] (also see [Chr22; vS20]), or the current best known approximation guarantee for **TSP**, which is just slightly below $3/2$ [KKO21; KKO23; GKL23]. The ideal result for **PCTSP** would be an algorithm that given an α -approximation for **TSP** produces an α -approximation for **PCTSP** (or possibly an $(\alpha + \varepsilon)$ -approximation for every $\varepsilon > 0$). Such a result was recently shown for Path TSP [TVZ22], and as approximation algorithms for **PCTSP** begin to approach the threshold $3/2$, this possibility feels less out of reach.

2.1.4 Related problems

Besides being key to our improved approximation factors for **PCTSP**, a common special case of our decomposition result (cf. Section 2.5) and the one of Bang-Jensen, Frank, and Jackson [BFJ95] gives rise to simple algorithms that we describe in Section 2.3 with approximation guarantees matching those of the primal-dual approach of Goemans and Williamson [GW95], both for **PCTSP** and the related prize-collecting Steiner tree problem, which is the following.

Prize-Collecting Steiner Tree (PCST): Given a graph $G = (V, E)$ with edge lengths $c_e \geq 0$ for all $e \in E$, a root $r \in V$, and penalties $\pi_v \geq 0$ for all $v \in V \setminus \{r\}$, the task is to find a tree $T = (V_T, E_T)$ in G that contains the root r and minimizes

$$\sum_{e \in E_T} c_e + \sum_{v \in V \setminus V_T} \pi_v .$$

Choosing very large penalties for a subset of vertices (typically called terminals) that enforce visiting these vertices in each reasonably good solution, and choosing zero penalties for the remaining vertices results in the well-known Steiner tree problem.

The classical LP formulation for **PCST** is very similar to the **PCTSP LP relaxation**, except that the degree constraints are dropped, and the coefficients 2 on the right-hand side of the cut constraints are replaced by a 1. For **PCTSP** and **PCST**, the algorithms of Goemans and Williamson based on their primal-dual framework return cycles and trees, respectively, with length at most twice the x -cost of an optimum LP solution, and total penalty no more than once the y -cost of an optimum LP solution. Matching the resulting guarantee on the sum of both, we present simple 2-approximations for either of the problems. We note that the currently best known approximation factor for **PCST** is 1.79 [AGHJM24b], hence our proposed algorithm is inferior. One particular reason why our new techniques for **PCTSP** do not lead to an improved approximation guarantee for **PCST** is that the LP formulation for **PCST** has a rather weak integrality gap of 2. Still, we believe that our techniques enrich the study and may prove useful again also for **PCST**, potentially in the context of provably stronger LP relaxations for **PCST**. One natural option is to adapt the bidirected cut relaxation for the Steiner tree problem to **PCST**. However, it is a big open question whether the integrality gap of the bidirected cut relaxation is strictly less than 2 (see, e.g., [RV99; CDV08]).

The prize-collecting traveling salesperson problem discussed in this chapter can be considered the main variant among several problems of similar nature that emerged from work of Balas on a model for scheduling the daily operation of a steel rolling mill [Bal89]. In his original problem formulation, on top of the **PCTSP** definition used here, there was a prize money $w_v \geq 0$ for every $v \in V \setminus \{r\}$ that the traveling salesperson collects upon visiting v , and a constraint that a prescribed amount w_0 of total prize money should be collected. To distinguish from the **PCTSP** formulation used in this thesis, we refer to Balas' formulation as the *Quota PCTSP*, also in view of the special case of *Quota TSP* that is obtained by setting $\pi_v = 0$ for all $v \in V \setminus \{r\}$. Quota TSP admits a 5-approximation [ALM00], and by concatenating solutions of the Quota TSP and the **PCTSP** subproblem, a constant factor approximation for Quota PCTSP is readily obtained [ABLM07]. The special case of Quota TSP where $w_v = 1$ for each $v \in V \setminus \{r\}$ and $w_0 = k$ for an integer k is typically referred to as k -TSP in the literature, and admits a 2-approximation [Gar05], again assuming metric edge lengths as in all these variants. Other budget-driven versions of **PCTSP** are variants of the *Orienteering* problem, where an upper bound on the allowed traveling distance is given. The goal is to maximize the sum of all collected prizes. These problems are as well widely studied in terms of approximation algorithms [BCKLMM07; CKP12; PFFSW20; PFFSW22; DFPS22].

PCTSP was also studied in special metric spaces. For graph metrics in planar graphs, Bateni, Chekuri, Ene, Hajiaghayi, Korula, and Marx [BCEHKM11] show the existence of a PTAS. For Euclidean distances, a PTAS is known as well [CJJ20]. For the asymmetric version of **PCTSP**, Nguyen [Ngu13] provides a $\lceil \log(n) \rceil$ -approximation algorithm, where n denotes the number of vertices.

Prize-collecting variations are also popular with other network design problems, in particular the Steiner tree problem mentioned earlier. A further generalization of this problem is the *prize-collecting Steiner forest problem* (also called prize-collecting generalized Steiner tree problem), in which we are given penalties for vertex pairs that need to be paid in case the vertices are not connected in the output graph. By using a straightforward adaption of the threshold rounding approach from [Goe09], one can get a $1/(1 - e^{-1/2}) \approx 2.541$ -approximation [HJ06]. This also upper bounds the integrality ratio of the natural LP relaxation for this problem. Very recently, it was shown that the prize-collecting Steiner forest problem admits an approximation guarantee going beyond the integrality gap of this LP relaxation [AGHJM24a]. Interestingly, for the prize-collecting Steiner forest problem, it is known that the integrality gap is strictly larger than the one of the Steiner forest problem [KOPRSV17], indicating that prize-collecting aspects may in some cases make the problem strictly harder to approximate. To date, no such separation is known for **TSP** and **PCTSP**.

2.2 Overcoming the weaknesses of threshold rounding

In this section we discuss our first improved approximation algorithm for **PCTSP** which refines the classical threshold rounding algorithm, getting significantly closer to the best known approximation guarantee for **TSP**. As mentioned earlier, this is joint work with Martin Nägele [BN23].

Theorem 3. *There is an LP-relative 1.774-approximation algorithm for **PCTSP**.*

After giving a high-level overview of our algorithm in [Section 2.2.1](#), we briefly discuss the classical threshold rounding approach in [Section 2.2.2](#).

In [Section 2.2.3](#), we give a detailed overview of a randomized version of our new algorithm, including in particular our decomposition lemma, the key technical tool that allows for our improved approximation guarantee. As mentioned in [Section 2.1](#), the proof of this lemma is deferred to [Section 2.5](#).

In [Section 2.2.4](#), we discuss our threshold choice. [Section 2.2.5](#), besides providing details on a randomized step of our algorithm, shows how to transform our approach to a deterministic one at no qualitative loss. Finally, we conclude this section with a deferred technical proof in [Section 2.2.6](#).

2.2.1 Our results and contributions

Our algorithm is similar in spirit to the classical threshold rounding approach that was leveraged in [BSW93] and [Goe09], but overcomes one of its critical weaknesses: In threshold rounding, the two decisions of (i) which vertices to visit, and (ii) how to build the tour are taken independently. Concretely, based on an optimal solution (x^*, y^*) of the **PCTSP LP relaxation** and a threshold $\gamma \in (0, 1]$, the set $V_\gamma := \{v \in V : y_v^* \geq \gamma\}$ of vertices that will be visited by the returned tour is fixed, and any LP-relative **TSP** algorithm can be used to blindly build a tour on V_γ . In our algorithm, to the contrary, choosing which vertices to visit and the actual construction of the tour are carefully intertwined.

Similarly to the threshold rounding algorithm, we also maintain a core set V_γ of vertices that are guaranteed to be on the output tour. Instead of building a tour from edges in $\binom{V_\gamma}{2}$, though, the building blocks of our tour are given by a set $\mathcal{W}$ of walks on V having precisely their endpoints in V_γ and that are pre-computed based on the solution (x^*, y^*) of the **PCTSP LP relaxation**. Through a procedure based on randomized pipage rounding [AS04; Sri01; CCPV11], we sample a subset of $\mathcal{W}$ such that the resulting multigraph H is connected and spans all vertices in V_γ , and its cost can be bounded with respect to the LP solution. Crucially, through our construction, H will have odd degrees in V_γ only, hence the cost of constructing a tour based on H (by parity correction) is not more than in the classical threshold rounding approach. In comparison, the primal-dual approach by Goemans and Williamson [GW95] constructs a tree without taking care of the vertex parities and ends up with doubling the computed tree for parity correction. Finally, our randomized tour construction allows us to analyze probabilities for vertices in $V \setminus V_\gamma$ to appear on the tour, and thus get bounds on expected penalties incurred

for such vertices. We also note that derandomizing our construction is possible using a deterministic variant of pipage rounding.

Due to the refined tour construction, our approach as sketched above does not allow to use any LP-relative approximation algorithm for **TSP** as a black-box. Though, as remarked in [Section 2.1.3](#), even if an LP-relative $4/3$ -approximation for **TSP** (matching the current best lower bound on the integrality gap of the Held-Karp relaxation) was available, our approximation guarantee remains superior to the current best black-box algorithm by Goemans [[Goe09](#)]. (Goemans' algorithm would yield a 1.835-approximation in this case.) Furthermore, as mentioned in [Theorem 3](#), our analysis is still with respect to the optimal value of the **PCTSP LP relaxation**, thereby immediately giving the subsequent corollary.

Corollary 4. *The integrality gap of the **PCTSP LP relaxation** is less than 1.774.*

On the technical side, the main ingredient of our algorithm is the construction and composition of the set $\mathcal{W}$ of walks mentioned earlier. We obtain appropriate walks through a decomposition of the x -part of a solution (x, y) of the **PCTSP LP relaxation**. Our decomposition of $x \in \mathbb{R}_{\geq 0}^E$ can equivalently be seen as a packing result on the multigraph obtained through scaling up x to an integral vector, and interpreting x_e as an edge multiplicity for every $e \in E$. Undoubtedly, the most prominent such packing theorem is Edmonds' result on packing disjoint spanning arborescences [[Edm73](#)]: A digraph $D = (V, A)$ admits a packing of k spanning arborescences rooted at r if and only if it is rooted k -edge connected, i.e., all r - v cuts have size at least k for all $v \in V$. Due to the typically uniform connectivity requirement, such results may well be used to decompose solutions of the Held-Karp relaxation (see [[GW17](#)] for such an application). In order to decompose a solution of the **PCTSP LP relaxation**, where the variables y_v indicate the non-uniform connectivities to the root r , we generalize ideas behind one of few results with non-uniform connectivity requirements due to Bang-Jensen, Frank, and Jackson [[BFJ95](#), Theorem 2.6]. Concretely, we revise their splitting-off based construction to step away from decompositions into components that span the full subset of vertices with maximum connectivity (i.e., in our setting, those vertices with $y_v = 1$), and instead only anchor each of the components at exactly two vertices in this subset, which we call the *anchors* of the component. This anchoring has the advantage that we can split each component of the decomposition into a path connecting its anchors (its *backbone*) and trees incident to this path (its *limbs*). By deleting or doubling the limbs we get even degree on all vertices except the anchors, which will prove useful to bound the cost of parity correction as mentioned earlier.

2.2.2 Review of the classical threshold rounding algorithm

Bienstock, Goemans, Simchi-Levi, and Williamson [[BGSW93](#)] leveraged Wolsey's analysis [[Wol80](#)] of the Christofides-Serdyukov algorithm for the **TSP** for an analysis of their threshold rounding algorithm as follows. Starting from a solution (x, y) of the **PCTSP LP relaxation**, let $V_1 := \{v \in V : y_v = 1\}$. Through applying *splitting off* techniques to x (see [Section 2.5](#) for details), we can obtain a point $z \in P_{\text{HK}}(G_1)$ of total

2.2. Overcoming the weaknesses of threshold rounding

length at most $c^\top x$, where $G_1 = (V_1, E_1)$ is the complete graph on the vertex subset V_1 . Hence, Wolsey's analysis shows that we can get a cycle on V_1 of length at most $\frac{3}{2}c^\top z \leq \frac{3}{2}c^\top x$. Because the loss in the penalty term incurred by this cycle may be arbitrarily large compared to the LP penalty $\sum_{v \in V} (1 - y_v)\pi_v$, the idea of threshold rounding is to not directly apply the above reasoning to an optimal solution of the [PCTSP LP relaxation](#), but a scaled version thereof that can be obtained through the following lemma.

Lemma 5. *Let (x, y) be a feasible solution of the [PCTSP LP relaxation](#). For every $\lambda \geq 0$, we can efficiently obtain a feasible solution (x_λ, y_λ) of the [PCTSP LP relaxation](#) such that $c^\top x_\lambda \leq \lambda \cdot c^\top x$ and*

$$y_{\lambda,v} = \min\{1, \lambda y_v\} \quad \text{for all } v \in V.$$

Simply speaking, [Lemma 5](#) states that by investing a factor of λ more on the x -part of the solution, we can also boost the connectivities y by the same factor. We remark that [Lemma 5](#) was already proven in [\[BGSW93\]](#) for optimal solutions of the [PCST LP relaxation](#) using the parsimonious property introduced by Goemans and Bertsimas [\[GB93\]](#); we repeat it here for general feasible solutions (x, y) , and provide a short proof in [Section 2.5](#) for completeness. To complete the argument for threshold rounding started above, let (x^*, y^*) be an optimal solution of the [PCTSP LP relaxation](#), and for a threshold $\gamma \in (0, 1]$, let (x, y) be the solution obtained from (x^*, y^*) through [Lemma 5](#) with $\lambda = 1/\gamma$. We have seen that we can obtain a cycle $C = (V_C, E_C)$ of cost

$$c(E_C) \leq \frac{3}{2}c^\top x \leq \frac{3}{2\gamma} \cdot c^\top x^* \tag{2.1}$$

on $V_1 = \{v \in V : y_v = 1\} = \{v \in V : y_v^* \geq \gamma\}$. Additionally, we can now bound the penalty by $\pi(V \setminus V_C) = \sum_{v \notin V_1} \pi_v \leq \frac{1}{1-\gamma} \cdot \sum_{v \in V} (1 - y_v^*)\pi_v$. Choosing $\gamma = 3/5$ balances the coefficients $3/2\gamma$ and $1/(1-\gamma)$, and gives the upper bound of $5/2$ on the approximation factor, as shown in [\[BGSW93\]](#).

2.2.3 A refined tour construction

As already mentioned earlier, our refined tour construction allows us to include some vertices in the output cycle that are not contained in our core set V_1 while preserving the upper bound on the length of the cycle. To do so, we exploit a weak spot in the analysis of threshold rounding, namely the slack between the cost $c^\top z$ of the point $z \in P_{\text{HK}}(G_1)$ defined in [Section 2.2.2](#) and the cost $c^\top x$. Transforming x to $z \in P_{\text{HK}}(G_1)$ is done by splitting off operations. In a single such operation, for some $\delta > 0$, we decrease the value of x on two well-chosen edges $\{v, u\}$ and $\{v, w\}$ by δ , and increase the value on $\{u, w\}$ by δ . Thereby, the total cost decreases by $\delta c_{\{v,u\}} + \delta c_{\{v,w\}} - \delta c_{\{u,w\}}$, which is nonnegative because c satisfies the triangle inequality. Viewed differently, we may exploit a budget of $c_{\{v,u\}} + c_{\{v,w\}}$ instead of $c_{\{u,w\}}$ for a δ -fraction of the edge $\{u, w\}$ in z , i.e., if $\{u, w\}$ is chosen to be part of a tour, we can afford to replace it by the path $u-v-w$ with a certain probability, thereby gaining extra coverage of the vertex v while maintaining the total cost bound $c^\top x$ in expectation. This idea motivates backtracking

the splitting off operations done to reach z , so that we can assign a certain budget to every component of z . Doing so in a careful way results in our main technical lemma, which is [Lemma 7](#) below (a formal proof is deferred to [Section 2.5](#)). Before actually stating the lemma, let us discuss an assumption on feasible solutions of the [PCTSP LP relaxation](#) that we enforce repeatedly throughout this section and which will prove useful later on.

Assumption 6. *For the feasible solution (x, y) of the [PCTSP LP relaxation](#) and $V_1 := \{v \in V : y_v = 1\}$, there is an edge $e_0 \in \binom{V_1}{2}$ with $x_{e_0} \geq 1$.*

We remark that [Assumption 6](#) can be made without loss of generality. Indeed, we can leverage a trick used previously, for example by Karlin, Klein, and Oveis Gharan [[KKO21](#)], namely dividing the root node r and all incident edges into two (with penalty zero and equal edge costs, respectively), and adding an edge e_0 of length $c(e_0) = 0$ between the root and its copy. Note that an LP solution for the original instance can be transformed to an LP solution of the same value for the thereby obtained auxiliary instance by setting $y_v = 1$ for the copy of the root, $x_{e_0} = 2 - \frac{x(\delta(r))}{2}$, and distributing the x -weight on every edge incident to the root equally to its two copies in the auxiliary graph. (Note that $x_{e_0} > 1$ if $x(\delta(r)) < 2$.) Conversely, by shortcutting, we can transform a [PCTSP](#) solution in the auxiliary instance to a [PCTSP](#) solution of at most the same value in the original instance.

Lemma 7 (Decomposition lemma). *Let (x, y) be a feasible solution of the [PCTSP LP relaxation](#) satisfying [Assumption 6](#) with edge e_0 . We can construct in polynomial time a set $\mathcal{T}$ of trees and weights $\mu \in [0, 1]^{\mathcal{T}}$ with the following properties:*

- (i) *The solution x is a conic combination of the trees in $\mathcal{T}$ with weights μ and the edge e_0 , i.e.,*

$$x = \sum_{T \in \mathcal{T}} \mu_T \chi^{E[T]} + \chi^{e_0} .^3$$

- (ii) *For every $v \in V \setminus V_1$,*

$$\sum_{T \in \mathcal{T} : v \in V[T]} \mu_T = y_v .$$

- (iii) *For every $T \in \mathcal{T}$, we have $|V[T] \cap V_1| = 2$, and we call the vertices in $V[T] \cap V_1$ the anchors of T .*
- (iv) *For $T \in \mathcal{T}$, let $e_T := V[T] \cap V_1$ denote the edge joining the anchors of T , and let $G_1 := (V_1, E_1)$ be the multigraph with edge set $E_1 := \{e_0\} \cup \{e_T : T \in \mathcal{T}\}$. Then*

$$z := \sum_{T \in \mathcal{T}} \mu_T \chi^{e_T} + \chi^{e_0} \in P_{HK}(G_1) .$$

Note that [Lemma 7 \(ii\)](#) guarantees that the connectivity y_v of vertices $v \in V \setminus V_1$ is reflected by the (weighted) number of trees incident to v in our decomposition. The latter will be crucial to obtain bounds on the penalties paid by our solutions for vertices

³For a set $F \subseteq E$ of edges, we denote by $\chi^F \in \{0, 1\}^E$ the incidence vector of F . Moreover, for an edge e , we use $\chi^e := \chi^{\{e\}}$.

2.2. Overcoming the weaknesses of threshold rounding

in $V \setminus V_1$, and guaranteeing this property requires caution when constructing the trees in $\mathcal{T}$. We also remark that in our proof, the point $z \in P_{\text{HK}}(G_1)$ in Lemma 7 (iv) will be a point that can be obtained from x through splitting off operations. Figure 2.1 shows an example of our decomposition.

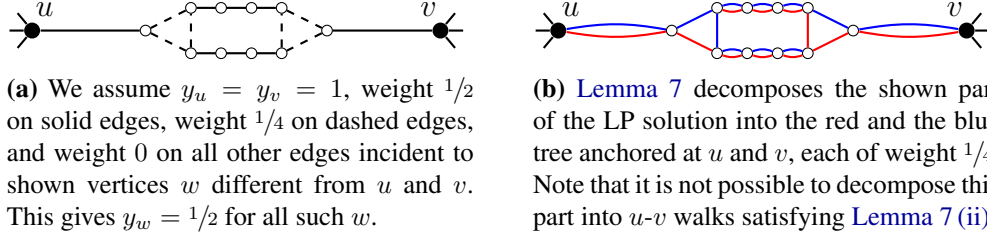


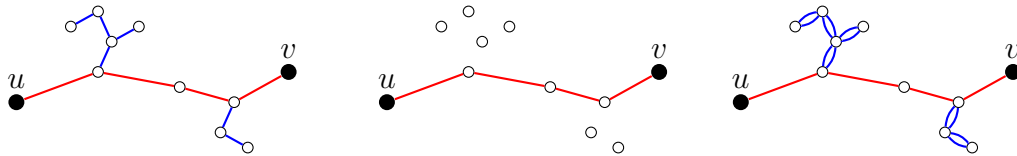
Figure 2.1: An excerpt of a solution of the PCTSP LP relaxation and its decomposition given by Lemma 7.

Our decomposition gives a more fine-grained analysis of the bound on the edge cost in threshold rounding than obtained in (2.1): We have

$$c(E_C) \leq \frac{3}{2} c^\top z = \frac{3}{2} \sum_{T \in \mathcal{T}} \mu_T c_{e_T} \leq \frac{3}{2} \sum_{T \in \mathcal{T}} \mu_T c(E[T]) \leq \frac{3}{2} c^\top x. \quad (2.2)$$

Here, in the middle inequality, we are bounding c_{e_T} by $c(E[T])$. If T was a path connecting the endpoints of e_T , we could exploit this as follows. Whenever e_T appears in a tour that we construct, replace e_T with T , and thereby avoid penalty at all vertices on the path without affecting the upper bound in (2.2). In the general case, in order to maintain a tour, we can only plug in the path in T connecting the endpoints of e_T . This path and its complement are important notions that we use repeatedly, hence we define the following (also see Figure 2.2 for an illustration).

Definition 8. Let T be a tree, and let V_1 be a vertex set such that $|V[T] \cap V_1| = 2$. The anchors of T in V_1 are the two vertices in $V[T] \cap V_1$. The backbone of T with respect to V_1 , denoted by $\text{backbone}_{V_1}(T)$, is the set of edges of T on the unique path connecting the two vertices in $V[T] \cap V_1$. The limbs of T with respect to V_1 , denoted by $\text{limbs}_{V_1}(T)$, is the set of edges of T that are not in the backbone of T with respect to V_1 .



(a) A tree T with anchors u and v , red backbone, and blue limbs.

(b) The two walks generated from T : One consisting of the backbone only (left), and one with two copies of the limbs added on top (right).

Figure 2.2: Anchors, backbone, and limb edges of a tree, and the two walks that can be constructed thereof.

Using this new notation, we can strengthen the bound in (2.2) to

$$c(E_C) \leq \frac{3}{2}c^\top z \leq \frac{3}{2} \sum_{T \in \mathcal{T}} \mu_T c(\text{backbone}_{V_1}(T)) . \quad (2.3)$$

At the above cost, we can build a tour on the vertices in V_1 and replace edges of that tour with the corresponding backbone paths. This extends the set of vertices that are covered beyond V_1 by those lying on the added backbones. Moreover, compared to the upper bound $\frac{3}{2}c^\top x$, we still have a budget of $\frac{3}{2} \sum_{T \in \mathcal{T}} \mu_T c(\text{limbs}_{V_1}(T))$ remaining. We can use this budget to add two copies of the limb edges of some trees to the cycle C in order to cover even more vertices. Note that by adding two copies, we make sure that the resulting edge set can again be shortcut to a cycle. Both of these observations contribute to saving penalties.

Illustrating how to translate the above intuition to actually improve coverage of vertices not in V_1 is easiest through exploiting randomization. Thus, we present a probabilistic approach here, and only later show that derandomization is possible. To build a basis for our tour, we exploit [Assumption 6](#) and the structure obtained thereby: [Lemma 7 \(iv\)](#) implies that $z - \chi^{e_0} = \sum_{T \in \mathcal{T}} \mu_T \chi^T$ is in the spanning tree polytope $P_{\text{ST}}(G_1)$ over G_1 (cf. [Section 1.2.3.3](#)). Being a matroid base polytope, we can exploit known marginal-preserving negatively correlated randomized rounding schemes in $P_{\text{ST}}(G_1)$, concretely *randomized pipage rounding* (see [Section 2.2.5](#) for details). Based on this, our approach is the following:

- (i) Sample a spanning tree of G_1 with marginals $z - \chi^{e_0}$, such that edges e_T appear with probability μ_T .
- (ii) Replace every sampled edge e_T by the path with edge set $\text{backbone}_{V_1}(T)$, and with probability $3/4$ (independently for all sampled e_T), add two copies of $\text{limbs}_{V_1}(T)$.
- (iii) Correct parities in the resulting multigraph using an appropriate join, and shortcut.

We remark that—instead of randomized pipage rounding—one could sample the spanning tree in [Step \(i\)](#) from a maximum entropy distribution and use the framework introduced by Karlin, Klein, and Oveis Gharan in their seminal $(1.5 - \varepsilon)$ -approximation algorithm for TSP [[KKO21](#); [KKO23](#)] to slightly reduce the cost of parity correction in [Step \(iii\)](#). However, the order of magnitude of this improvement (the authors show a bound of $\varepsilon > 10^{-36}$, which has been slightly improved to $\varepsilon > 10^{-34}$ in [[GKL23](#)]) is negligible in comparison with the loss that we incur through our analysis (see [Remark 13](#)). Additionally, derandomization of pipage rounding is rather straightforward in our setting using known tools (see [Section 2.2.5](#)).

Replacing each sampled tree edge e_T by the corresponding backbone together with parity correction in the resulting multigraph leads to precisely the bound on the total edge cost in (2.3), and adding limb edges probabilistically as in [Step \(ii\)](#) uses up the remaining budget $\frac{3}{2} \sum_{T \in \mathcal{T}} \mu_T c(\text{limbs}_{V_1}(T))$ in expectation. Adding two copies of $\text{limbs}_{V_1}(T)$ implies that the edge set we replace e_T by can always be cast as a walk connecting the two endpoints of e_T , hence vertices in $V \setminus V_1$ will always have even degree. Consequently, in [Step \(iii\)](#), parities only need to be corrected at vertices in V_1 , and hence at a cost that we can still bound by $\frac{1}{2}c^\top z$. Moreover, we can make use of the negative correlation property of the sampling procedure to prove bounds on the probability that a

2.2. Overcoming the weaknesses of threshold rounding

vertex $v \in V \setminus V_1$ is not covered by any of the used walks, and thereby obtain an upper bound on the expected penalty that we incur. (Note that in [Step \(iii\)](#), we may use a join that could possibly connect further vertices, but we do not exploit this in our analysis.) Summarizing [Steps \(i\)](#) and [\(ii\)](#) leads to the following lemma, whose formal proof we defer to [Section 2.2.5](#).

Lemma 9. *Let (x, y) be a feasible solution of the [PCTSP LP relaxation](#) satisfying [Assumption 6](#). Let $\mathcal{T}$ and $\mu \in [0, 1]^{\mathcal{T}}$ be a set of trees and weights with the properties guaranteed by [Lemma 7](#). We can efficiently sample a multigraph H on a subset of V with the following properties:*

- (i) *H is connected, spans $V_1 := \{v \in V : y_v = 1\}$, and all vertices in $V \setminus V_1$ have even degree.*
- (ii) *We have*

$$\begin{aligned} & \mathbb{E} [c(E[H]) + \pi(V \setminus V[H])] \\ & \leq \sum_{T \in \mathcal{T}} \mu_T \left(c(\text{backbone}_{V_1}(T)) + \frac{3}{2} \cdot c(\text{limbs}_{V_1}(T)) \right) + \sum_{v \in V \setminus V_1} \pi_v \exp \left(-\frac{3y_v}{4} \right). \end{aligned}$$

With this result at hand, we can state a randomized version of our new algorithm, [Algorithm 1](#). Note that the input only requires a feasible solution (x^*, y^*) of the [PCTSP LP relaxation](#), even though typically, we will call [Algorithm 1](#) on an optimal solution. Moreover, the choice of a suitable threshold $\gamma \in (0, 1]$ is left open for the moment; we show how to optimally exploit this remaining flexibility later on.

Algorithm 1: Our new randomized algorithm for [PCTSP](#).

Input: Feasible solution (x^*, y^*) of the [PCTSP LP relaxation](#), threshold $\gamma \in (0, 1]$.

1. Guarantee that [Assumption 6](#) is satisfied by modifying the instance accordingly if necessary.
2. Apply [Lemma 5](#) to (x^*, y^*) with $\lambda = 1/\gamma$ to obtain another feasible solution (x, y) .
3. Compute a set of trees $\mathcal{T}$ with weights $\mu \in [0, 1]^{\mathcal{T}}$ by applying [Lemma 7](#) to (x, y) .
4. Sample a multigraph H on V through [Lemma 9](#) applied to (x, y) and $(\mathcal{T}, \mu)$.
5. Compute a shortest odd(H)-join J .
6. Let C be a cycle obtained by shortcutting an Eulerian tour in $H \cup J$.

return C .

Using [Lemma 9](#), we get the following guarantees. We remark that compared to classical threshold rounding, the factor $3/2\gamma$ on the term $c^\top x^*$ is unchanged, while at the same time, we get clearly improved factors on the penalty side.

Theorem 10. *Let (x^*, y^*) be a feasible solution of the **PCTSP LP relaxation**. On input (x^*, y^*) and $\gamma \in (0, 1]$, **Algorithm 1** returns, in polynomial time, a cycle $C = (V_C, E_C)$ such that*

$$\mathbb{E}[c(E_C) + \pi(V \setminus V_C)] \leq \frac{3}{2\gamma} \cdot c^\top x^* + \sum_{v \in V: y_v^* < \gamma} \pi_v \cdot \exp\left(-\frac{3y_v^*}{4\gamma}\right). \quad (2.4)$$

Proof. By **Lemma 9**, the multigraph H constructed in **Algorithm 1** has odd degrees only at vertices in V_1 , hence we can also correct parities by an $\text{odd}(H)$ -join in the multigraph $G_1 = (V_1, E_1)$ with edge set $E_1 = \{e_0\} \cup \{e_T: T \in \mathcal{T}\}$. By **Item (iv)** of **Lemma 7**, the point $z = \sum_{T \in \mathcal{T}} \mu_T \chi^{e_T} + \chi^{e_0}$ is in $P_{\text{HK}}(G_1)$. Thus, following Wolsey's analysis, a shortest $\text{odd}(H)$ -join J in G_1 has cost at most

$$c(E[J]) \leq \frac{1}{2} c^\top z \leq \frac{1}{2} \sum_{T \in \mathcal{T}} \mu_T c(\text{backbone}_{V_1}(T)).$$

Note that J also is a shortest $\text{odd}(H)$ -join in the original complete graph G . Combining this with the bound of **Item (ii)** in **Lemma 9** immediately gives the claimed guarantee

$$\begin{aligned} \mathbb{E}[c(E_C) + \pi(V \setminus V_C)] &\leq \mathbb{E}[c(E[H]) + c(E[J]) + \pi(V \setminus V_C)] \\ &\leq \frac{3}{2} c^\top x + \sum_{v \in V \setminus V_1} \pi_v \exp\left(-\frac{3y_v}{4}\right) \\ &\leq \frac{3}{2\gamma} \cdot c^\top x^* + \sum_{v \in V: y_v^* < \gamma} \pi_v \exp\left(-\frac{3y_v^*}{4\gamma}\right). \end{aligned}$$

Here, in the last inequality, we use that by **Lemma 5**, we have $c^\top x \leq \frac{1}{\gamma} c^\top x^*$, and also $y_v = \frac{1}{\gamma} y_v^*$ for $v \in V \setminus V_1$ because $V \setminus V_1 = \{v \in V: y_v^* < \gamma\}$. To finish the proof, we note that indeed, all steps in **Algorithm 1** can be implemented in polynomial time. $\square$

2.2.4 Choosing the right threshold

Directly balancing the terms in **Theorem 10** by a proper choice of the threshold γ gives the subsequent corollary. We remark that the resulting approximation guarantee does not yet beat the previously best known one of roughly 1.914 by Goemans [**Goe09**], but it is significantly better than the standard analysis of threshold rounding, which gives a $5/2$ -approximation.

Corollary 11. *Given a feasible solution (x^*, y^*) of the **PCTSP LP relaxation** and $\gamma = 1/(1 + \frac{2}{3} \exp(-3/4)) \approx 0.761$, **Algorithm 1** returns in polynomial time a **PCTSP** solution of expected value at most $\alpha \cdot (c^\top x^* + \pi^\top(1 - y^*))$ with $\alpha = \frac{3}{2} + \exp(-3/4) < 1.973$.*

Proof. For general $\gamma \in (0, 1]$, we can bound the penalty term of the bound given in **Theorem 10** as follows:

$$\sum_{v \in V: y_v^* < \gamma} \pi_v \exp\left(-\frac{3y_v^*}{4\gamma}\right) \leq \max\left\{1, \frac{\exp(-3/4)}{1 - \gamma}\right\} \cdot \sum_{v \in V: y_v^* < \gamma} \pi_v (1 - y_v^*).$$

2.2. Overcoming the weaknesses of threshold rounding

Here we used that for $y \in [0, \gamma]$, we have $\exp(-\frac{3y}{4\gamma}) \leq \max\{1, \frac{\exp(-3/4)}{1-\gamma}\} \cdot (1-y)$. The latter can be derived by noting that the inequality holds for $y \in \{0, \gamma\}$, and that $y \mapsto \exp(-\frac{3y}{4\gamma})$ is convex in y . Thus, [Theorem 10](#) implies

$$\mathbb{E}[c(E_C) + \pi(V \setminus V_C)] \leq \max\left\{\frac{3}{2\gamma}, 1, \frac{\exp(-3/4)}{1-\gamma}\right\} \cdot \left(\sum_{e \in E} c_e x_e^* + \sum_{v \in V} \pi_v(1 - y_v^*)\right),$$

where the maximum expression is minimized for $\gamma = 1/(1 + \frac{2}{3}\exp(-3/4))$, giving the claimed approximation factor with respect to the objective value of the input solution (x^*, y^*) of the [PCTSP LP relaxation](#). $\square$

When choosing the threshold γ with respect to a specific distribution (similarly to Goemans [[Goe09](#)]), we get a significant improvement of the best known approximation ratio. We remark that in contrast to the classical threshold rounding algorithm, it is no longer optimal to choose γ with respect to a uniform distribution over some interval in $[0, 1]$.

Theorem 12. *Let $b = 0.6945$. Sampling γ from the interval $[b, 1]$ using a distribution with density $f(\gamma) \propto \exp(-b/\gamma)$, and starting [Algorithm 1](#) from a feasible solution (x^*, y^*) of the [PCTSP LP relaxation](#), we get in polynomial time a [PCTSP](#) solution of expected value at most $\alpha \cdot (c^\top x^* + \pi^\top(1 - y^*))$, where $\alpha < 1.774$.*

The proof of [Theorem 12](#) is postponed to [Section 2.2.6](#). We remark that in contrast to the classical threshold rounding approach, we do not profit from combining our algorithm with the primal-dual algorithm by Goemans and Williamson [[GW95](#)]. This makes our algorithm self-contained and easy to state.

As mentioned earlier, the [PCTSP LP relaxation](#) can be solved in polynomial time through the ellipsoid method. Thus, applying [Theorem 12](#) to an optimal solution (x^*, y^*) of the [PCTSP LP relaxation](#) directly implies [Theorem 3](#) if one allows randomization. In order to obtain a deterministic procedure, we show in [Section 2.2.5](#) how to derandomize the proposed algorithm. Note that based on [Theorem 12](#), there are two steps where we exploit randomization. First, we choose the threshold γ from some distribution. We will see that given an *optimal* LP-solution (x^*, y^*) of the [PCTSP LP relaxation](#), it suffices to try all thresholds in the set $\{y_v^* : v \in V\}$ to get the claimed bound (cf. [Section 2.2.5.3](#)). Second, the construction of the multigraph H in [Step 4](#) of [Algorithm 1](#) is based on randomized pipage rounding in the spanning tree polytope of an auxiliary graph. We show in [Section 2.2.5.2](#) that we can also use a deterministic version of pipage rounding to achieve the same guarantees. This will prove [Theorem 3](#).

We make the reader aware that the first step above, namely the derandomization of the threshold choice, was immediate in the setting of Goemans [[Goe09](#)], because the values $\gamma = y_v^*$ for $v \in V$ already result in all possible supports of the tours that can be obtained through a threshold, and tours were subsequently built in a black-box way. In our refined approach, this is no longer the case; other values of γ may result in different tours, and it requires some thought to recover the conclusion that the best solution is obtained for some $\gamma = y_v^*$.

Remark 13. *Computational experiments (through discretizing the distribution to sample γ from) indicate that even by choosing an optimal distribution for γ , one cannot push the above analysis to prove a bound on the approximation factor of value less than or equal to 1.773. Consequently, the distribution over γ that we propose in [Theorem 12](#) is very close to optimal for an analysis with respect to the bounds given by [Theorem 10](#). More precisely, for any distribution over γ and any $n \in \mathbb{N}$, the approximation guarantee obtained through the above analysis can be bounded by the optimum value of the following linear program, where $y_i := \gamma_i := \frac{i}{n}$ for each $i \in \{0, \dots, n\}$.*

$$\begin{aligned} \min \quad & \alpha \\ \sum_{i=1}^n \delta_i &= 1 \\ \alpha &\geq \sum_{i=1}^n \delta_i \cdot \frac{3}{2\gamma_i} \\ (1 - y_{i-1}) \cdot \alpha &\geq \sum_{j=i+1}^n \delta_j \cdot \exp\left(-\frac{3y_i}{4\gamma_{j-1}}\right) \quad \forall i \in \{1, \dots, n\} \\ \delta_i &\geq 0 \quad \forall i \in \{1, \dots, n\} . \end{aligned}$$

Indeed, for any distribution over γ with support in $(0, 1]$, the value of the LP solution given by $\delta_i = \mathbb{P}[\gamma \in (\gamma_{i-1}, \gamma_i]]$ for each $i \in \{1, \dots, n\}$ and α smallest possible is a lower bound on the approximation guarantee obtained via the above analysis: The first inequality reflects the factor in (2.4) on the x -part of the objective of the [PCTSP LP relaxation](#), whereas for $i \in \{1, \dots, n\}$, the second inequality reflects the factor in (2.4) on the y -part of the objective of the [PCTSP LP relaxation](#) for vertices $v \in V \setminus \{r\}$ with $y_v^* \in [y_{i-1}, y_i]$. Note that any feasible dual solution yields a lower bound on the optimum LP value. We computed a feasible dual solution using Gurobi 10.0.1, yielding a lower bound of 1.773 as mentioned above.

2.2.5 Sampling walks and derandomization

In this subsection, we prove [Lemma 9](#), which is at the core of our randomized algorithm, giving the basis of the returned tour. Furthermore, we show how to replace the randomized steps of our algorithm with deterministic procedures while obtaining the same guarantees. This includes derandomizing [Lemma 9](#), but also derandomizing the random choice of a suitable threshold γ . Together with the analysis presented earlier, these ingredients prove [Theorem 3](#), our main result regarding this algorithm.

Before going into the more technical parts, let us recall the pipage rounding procedure, which we use in a black-box way for both the proof of [Lemma 9](#) and its deterministic analogue. Pipage rounding is a technique that goes back to Ageev and Sviridenko [[AS04](#)] and Srinivasan [[Sri01](#)]. Generalized to a matroid setting and put into a randomized framework by Calinescu, Chekuri, Pál, and Vondrák [[CCPV11](#)], randomized pipage rounding is one of several extensively studied efficient sampling procedures in matroid and matroid base polytopes (also see [[CVZ10](#); [HO14](#)] and references therein). Generally, the idea is to start from a fractional point x , and modify this solution iteratively in a well-chosen way until an integral solution is found. In pipage rounding, each single modification is a maximal step along a direction parallel to some $e_i - e_j$, where e_k is the k^{th} unit vector. Because the directions $e_i - e_j$ are precisely the edge directions

2.2. Overcoming the weaknesses of threshold rounding

of matroid base polytopes, such moves are enough to eventually end up in a vertex. In a randomized setting, probabilities for going either way are chosen such that in every step, marginals of every component are preserved. Concretely, if for a matroid $\mathcal{M}$, $B_{\mathcal{M}}$ denotes the associated matroid base polytope, i.e., the convex hull of all characteristic vectors of bases in $\mathcal{M}$, one can obtain the following.

Theorem 14 ([CVZ10, direct implication of Theorem 1.1]). *Let $x = (x_1, \dots, x_n) \in B_{\mathcal{M}}$ be a fractional solution in the matroid base polytope, and let $X = (X_1, \dots, X_n) \in \{0, 1\}^n$ be an integral solution obtained from x using randomized pipage rounding. Then, the following holds.*

- (i) *For every $i \in [n]$, we have $\mathbb{E}[X_i] = x_i$.*
- (ii) *For every $T \subseteq [n]$, we have*

$$\mathbb{E} \left[\prod_{i \in T} X_i \right] \leq \prod_{i \in T} x_i \quad \text{and} \quad \mathbb{E} \left[\prod_{i \in T} (1 - X_i) \right] \leq \prod_{i \in T} (1 - x_i) .$$

Thus, besides preserving marginals of a point in the spanning tree polytope exactly, randomized pipage rounding also guarantees that the events of edges being sampled as well as those of edges not being sampled are negatively correlated, a property that we crucially exploit to bound the total penalty that our PCTSP solutions incur. We remark that there are other known procedures achieving the same guarantees, in particular randomized swap rounding [CVZ10], or, at a $1 \pm \varepsilon$ error in the marginals, sampling from λ -uniform spanning tree distributions (see, for example, [AGMOS17; KKO21]).

For a deterministic version of pipage rounding, we use a formulation of Harvey and Olver [HO14] that is based on earlier work mentioned above. Here, the decisions on the direction of every single step are guided by a function $g: \mathcal{B}_{\mathcal{M}} \rightarrow \mathbb{R}$ that is required to be *concave under swaps*, i.e., for all $x \in \mathcal{B}_{\mathcal{M}}$ and all unit vectors e_i, e_j , we require that the function $t \mapsto g(x + t(e_i - e_j))$ is concave. This concavity under swaps allows to perform steps along decreasing values of g , and one can obtain the following.

Theorem 15 (Deterministic Pipage Rounding [HO14]). *There is a deterministic, polynomial-time algorithm that, given $x \in B_{\mathcal{M}}$ and a value oracle for a function g that is concave under swaps, outputs an extreme point $\hat{x}$ of $B_{\mathcal{M}}$ with $g(\hat{x}) \leq g(x)$.*

2.2.5.1 Proof of Lemma 9

As already outlined earlier, we prove this lemma by exploiting the decomposition of a solution (x, y) of the PCTSP LP relaxation into a family $\mathcal{T}$ of trees through Lemma 7. Concretely, we will split each tree into two walks with corresponding weights, one of them consisting of backbone edges only, while the other contains two copies of the limb edges on top (see Figure 2.2 on Page 22). To sample walks, we leverage Item (iv) of Lemma 7, concretely the implication that $z - \chi^{e_0}$ is in the spanning tree polytope over the multigraph $G_1 = (V_1, E_1)$, where $E_1 := \{e_0\} \cup \{e_T : T \in \mathcal{T}\}$. In this graph, every edge corresponds to a tree; by duplicating every edge and splitting the edge weight accordingly, we can associate to every edge one of the walks constructed earlier. Also,

the resulting edge weights still form a point in the spanning tree polytope of the blown-up graph, and we sample a spanning tree in this graph with the edge weights as marginals through pipage rounding. Sampled edges can then be replaced by the corresponding walks. The union of the edge sets of all these walks induces the multigraph H claimed by Lemma 9. Using the properties guaranteed by Theorem 14, we can prove Lemma 9.

Proof of Lemma 9. Let $\mathcal{T}$ and weights $\mu \in [0, 1]^T$ be obtained from Lemma 7. We construct a set $\mathcal{W}_0$ of walks as follows. For each $T \in \mathcal{T}$ with anchors $s, t \in V_1$, add one s - t walk $W_1(T)$ to $\mathcal{W}_0$ that consists of $\text{backbone}_{V_1}(T)$ and two copies of $\text{limbs}_{V_1}(T)$, and set $\nu_{W_1(T)} = \frac{3}{4}\mu_T$. As we add two copies of the limb edges, the resulting edge set has even degrees except at s and t , and can thus indeed be cast as an s - t walk. On top of that, add another s - t walk $W_2(T)$ to $\mathcal{W}_0$ that consists of $\text{backbone}_{V_1}(T)$ only, and set $\nu_{W_2(T)} = \frac{1}{4}\mu_T$.

For each walk $W \in \mathcal{W}_0$, let e_W be the direct edge joining its start- and endpoint. Let F be the multiset of all these edges. Now, consider the point $z_0 \in [0, 1]^F$ defined by $z_0(e_W) = \nu_W$ for each $W \in \mathcal{W}_0$. As z_0 and the point $z - \chi^{e_0}$ (where z is defined as in Item (iv) of Lemma 7) differ only by splitting edges into parallel edges of the same total weight, it immediately follows that $z_0 + \chi^{e_0} \in P_{\text{HK}}(G_0)$, where $G_0 = (V_1, F)$. From the polyhedral descriptions of $P_{\text{HK}}(G_0)$ and the spanning tree polytope $P_{\text{ST}}(G_0)$ over G_0 , one can see that this in turn implies that $z_0 \in P_{\text{ST}}(G_0)$. Applying Theorem 14 with z_0 , we thus obtain a spanning tree (V_1, F_1) with $F_1 \subseteq F$ via randomized pipage rounding. Let

$$\mathcal{W} := \{W \in \mathcal{W}_0 : e_W \in F_1\} .$$

Since $\mathcal{W} \subseteq \mathcal{W}_0$, each walk in $\mathcal{W}$ starts and ends in V_1 . Since (V_1, F_1) is spanning, the multigraph H induced by the edge sets of the walks in $\mathcal{W}$ is connected and always spans V_1 . Let $v \in V \setminus V_1$. Then

$$\begin{aligned} \mathbb{P}[v \notin H] &\leq \prod_{W \in \mathcal{W}_0 : v \in V[W]} \mathbb{P}[e_W \notin F] && \text{(by Theorem 14 (ii))} \\ &\leq \prod_{T \in \mathcal{T} : v \in V[T]} \mathbb{P}[e_{W_1(T)} \notin F] \\ &= \prod_{T \in \mathcal{T} : v \in V[T]} \left(1 - \frac{3}{4}\mu_T\right) && \text{(by Theorem 14 (i))} \\ &\leq \exp\left(-\frac{3}{4} \sum_{T \in \mathcal{T} : v \in V[T]} \mu_T\right) && \text{(using } 1 - x \leq \exp(-x)) \\ &= \exp\left(-\frac{3y_v}{4}\right) . && \text{(by Lemma 7)} \end{aligned}$$

Consequently,

$$\mathbb{E}[\pi(V \setminus V[H])] = \sum_{v \in V \setminus V_1} \pi_v \mathbb{P}[v \notin H] \leq \sum_{v \in V \setminus V_1} \pi_v \exp\left(-\frac{3y_v}{4}\right) . \quad (2.5)$$

2.2. Overcoming the weaknesses of threshold rounding

It remains to bound the expected total cost of the sampled walks. Using [Theorem 14 \(i\)](#) again, we get

$$\begin{aligned}
\mathbb{E}[c(E[H])] &= \sum_{W \in \mathcal{W}_0} \nu_W \cdot c(E[W]) \\
&= \sum_{T \in \mathcal{T}} \left(\frac{3}{4} \mu_T \cdot c(E[W_1(T)]) + \frac{1}{4} \mu_T \cdot c(E[W_2(T)]) \right) \\
&= \sum_{T \in \mathcal{T}} \mu_T \left(c(\text{backbone}_{V_1}(T)) + \frac{3}{2} \cdot c(\text{limbs}_{V_1}(T)) \right). \quad (2.6)
\end{aligned}$$

Together, (2.5) and (2.6) imply the bound claimed by [Lemma 9](#). Finally, note that [Theorem 14](#) guarantees that we can sample $\mathcal{W}$, and thus obtain H , efficiently. $\square$

2.2.5.2 A deterministic selection of walks

In this subsection, we show how to deterministically construct the graph H given by [Lemma 9](#) while maintaining the overall guarantees on the performance of our algorithm. To be precise, we show the following deterministic analogue of [Lemma 9](#).

Lemma 16. *Let (x, y) be a feasible solution of the [PCTSP LP relaxation](#) satisfying [Assumption 6](#). Let $\mathcal{T}$ and $\mu \in [0, 1]^{\mathcal{T}}$ be a set of trees and weights with the properties guaranteed by [Lemma 7](#). We can deterministically and in polynomial time obtain a multigraph H on a subset of V with the following properties:*

- (i) *H is connected, spans $V_1 := \{v \in V : y_v = 1\}$, and all vertices in $V \setminus V_1$ have even degree.*
- (ii) *We have*

$$\begin{aligned}
&c(E[H]) + \pi(V \setminus V[H]) \\
&\leq \sum_{T \in \mathcal{T}} \mu_T \left(c(\text{backbone}_{V_1}(T)) + \frac{3}{2} \cdot c(\text{limbs}_{V_1}(T)) \right) + \sum_{v \in V \setminus V_1} \pi_v \exp\left(-\frac{3y_v}{4}\right).
\end{aligned}$$

To prove [Lemma 16](#), we recall the randomized construction from the proof of [Lemma 9](#) (see [Section 2.2.5.1](#)): There, the graph H is based on a spanning tree sampled in an auxiliary graph through randomized pipage rounding. We now show that the deterministic analogue given through [Theorem 15](#) can achieve the same guarantees.

Proof of Lemma 16. Starting from a solution (x, y) of the [PCTSP LP relaxation](#) and repeating the construction from the proof of [Lemma 9](#), we define the multigraph $G_0 = (V_1, F)$, where for each $T \in \mathcal{T}$, there are two parallel edges in F joining the anchors of T : One corresponding to the walk $W_1(T)$ consisting of $\text{backbone}_{V_1}(T)$ and two copies of $\text{limbs}_{V_1}(T)$ with weight $\nu_{W_1(T)} = \frac{3}{4}\mu_T$, and one corresponding to the walk $W_2(T)$ consisting of $\text{backbone}_{V_1}(T)$ with weight $\nu_{W_2(T)} = \frac{1}{4}\mu_T$. Again, we denote by $\mathcal{W}_0$ the set of all walks constructed this way, and for $W \in \mathcal{W}_0$, let e_W be the edge in F corresponding to W . Moreover, we know that $z_0 \in [0, 1]^F$ defined by $z_0(e_W) = \nu_W$ lies in the spanning tree polytope $P_{\text{ST}}(G_0)$.

We want to apply deterministic pipage rounding, i.e., [Theorem 15](#), in the spanning tree polytope $P_{\text{ST}}(G_0)$ starting from z_0 with the function $g: P_{\text{ST}}(G_0) \rightarrow \mathbb{R}$ defined by

$$g(z) := \sum_{W \in \mathcal{W}_0} c(E[W]) z_{e_W} + \sum_{v \in V \setminus V_1} \pi_v \prod_{W \in \mathcal{W}_0: v \in W} (1 - z_{e_W}) .$$

To this end, we prove the following properties.

Claim 17.

- (i) g is concave under swaps.
- (ii) If z is the incidence vector of a spanning tree in G_0 , and H is the multigraph obtained from that spanning tree by replacing every edge e_W by the walk W , then $g(z) = c(E[H]) + \pi(V \setminus V[H])$.
- (iii) We have

$$g(z_0) \leq \sum_{T \in \mathcal{T}} \mu_T \left(c(\text{backbone}_{V_1}(T)) + \frac{3}{2} \cdot c(\text{limbs}_{V_1}(T)) \right) + \sum_{v \in V \setminus V_1} \pi_v \exp\left(-\frac{3y_v}{4}\right) .$$

Clearly, [Lemma 16](#) immediately follows from the claim: [Item \(i\)](#) allows for applying [Theorem 15](#), while [Items \(ii\) and \(iii\)](#) show that the graph H satisfies the bound in [Item \(ii\)](#) of [Lemma 16](#). Moreover, H is clearly connected, spans V_1 , and has even degrees in $V \setminus V_1$ because it was obtained from a spanning tree on V_1 through replacing edges by walks connecting the same endpoints. Consequently, we are left with proving the claim.

Proof of [Item \(i\)](#). The first sum in the definition of $g(z)$ is linear in z , hence it suffices to show that $\bar{g}: P_{\text{ST}}(G_0) \rightarrow \mathbb{R}$ defined by

$$\bar{g}(z) := \sum_{v \in V \setminus V_1} \pi_v \prod_{W \in \mathcal{W}_0: v \in W} (1 - z_{e_W})$$

is concave under swaps. We show that $\bar{g}$ is in fact equal to (a restriction of) the multilinear extension of a supermodular set function $f: 2^F \rightarrow \mathbb{R}$, which readily implies that $\bar{g}$ is concave under swaps [[CCPV11](#)]. For $S \subseteq F$, let $V[S]$ denote the union of all vertices covered by the walks in $\mathcal{W}_0$ that correspond to edges in S , and define the set function $f: 2^F \rightarrow \mathbb{R}$ by

$$f(S) := \pi(V \setminus V[S]) = \sum_{v \in V} \pi_v \cdot (1 - \mathbb{1}_{v \in V[S]}) .$$

Supermodularity follows immediately from the sum representation above, because the indicator function $\mathbb{1}_{v \in V[S]}$ is clearly submodular. By definition, the multilinear extension $\tilde{f}: [0, 1]^F \rightarrow \mathbb{R}$ of f is given by

$$\tilde{f}(z) = \sum_{S \subseteq F} f(S) \prod_{e \in S} z_e \prod_{e \in F \setminus S} (1 - z_e) .$$

2.2. Overcoming the weaknesses of threshold rounding

It remains to show that for $z \in P_{\text{ST}}(G_0)$ we have $\tilde{f}(z) = \bar{g}(z)$. Indeed,

$$\begin{aligned}
\tilde{f}(z) &= \sum_{S \subseteq F} \pi(V \setminus V[S]) \prod_{e \in S} z_e \prod_{e \in F \setminus S} (1 - z_e) \\
&= \sum_{v \in V \setminus V_1} \pi_v \sum_{S \subseteq \{W \in \mathcal{W}_0 : v \notin W\}} \prod_{W \in S} z_{e_W} \prod_{W \in \mathcal{W}_0 \setminus S} (1 - z_{e_W}) \\
&= \sum_{v \in V \setminus V_1} \pi_v \prod_{W \in \mathcal{W}_0 : v \in W} (1 - z_{e_W}) \underbrace{\sum_{S \subseteq \{W \in \mathcal{W}_0 : v \notin W\}} \prod_{W \in S} z_{e_W} \prod_{W \in \{W \in \mathcal{W}_0 : v \notin W\} \setminus S} (1 - z_{e_W})}_{= \prod_{W \in \{W \in \mathcal{W}_0 : v \notin W\}} (z_{e_W} + (1 - z_{e_W})) = 1} \\
&= \bar{g}(z) .
\end{aligned}$$

Proof of Item (ii). If z is integral and corresponds to a spanning tree T of G_0 , we may write

$$g(z) = \sum_{e_W \in T} c(E[W]) + \sum_{v \in V \setminus V_1} \pi_v \prod_{e_W \in E[G_0] : v \in W} \mathbb{1}_{e_W \notin E[T]} .$$

The first term equals $c(E[H])$ by definition of H . In the second term, the sum has nonzero terms only for vertices $v \in V \setminus V_1$ that are not covered by any walk $W \in \mathcal{W}_0$ for which $e_W \in E[T]$, i.e., precisely for the vertices in $V \setminus V_1$ that are not covered by H . As H always covers V_1 by definition, the second term thus equals $\pi(V \setminus V[H])$, as desired.

Proof of Item (iii). By definition of the walks in $\mathcal{W}_0$, we have

$$\begin{aligned}
\sum_{W \in \mathcal{W}_0} c(E[W]) z_0(e_W) &= \sum_{T \in \mathcal{T}} \left(c(E[W_1(T)]) \cdot \frac{3}{4} \mu_T + c(E[W_2(T)]) \cdot \frac{1}{4} \mu_T \right) \\
&= \sum_{T \in \mathcal{T}} \mu_T \cdot \left(c(\text{backbone}_{V_1}(T)) + \frac{3}{2} \cdot c(\text{limbs}_{V_1}(T)) \right) ,
\end{aligned}$$

and

$$\begin{aligned}
\sum_{v \in V \setminus V_1} \pi_v \prod_{W \in \mathcal{W}_0 : v \in W} (1 - z_0(e_W)) &\leq \sum_{v \in V \setminus V_1} \pi_v \prod_{T \in \mathcal{T} : v \in V[T]} \left(1 - \frac{3}{4} \mu_T \right) \\
&\leq \sum_{v \in V \setminus V_1} \pi_v \exp \left(- \frac{3}{4} \sum_{T \in \mathcal{T} : v \in V[T]} \mu_T \right) \\
&= \sum_{v \in V \setminus V_1} \pi_v \exp \left(- \frac{3}{4} y_v \right) ,
\end{aligned}$$

which together gives the claimed relation, finishes the proof of [Claim 17](#), and thus also of [Lemma 16](#). $\square$

2.2.5.3 Deterministic selection of a threshold

In the analysis of [Algorithm 1](#) in [Theorem 12](#), we also exploited a random choice of the threshold parameter γ . We now show that if we start from an optimal solution (x^*, y^*) of the [PCTSP LP relaxation](#), it suffices to try all values in the set $\{y_v^* : v \in V\}$ as thresholds.

Lemma 18. *Let (x^*, y^*) be an optimal solution of the [PCTSP LP relaxation](#), and let $\alpha > 0$. If there exists $\gamma \in (0, 1]$ such that*

$$\frac{3}{2\gamma} \cdot c^\top x^* + \sum_{v \in V : y_v^* < \gamma} \exp\left(-\frac{3}{4\gamma} y_v^*\right) \cdot \pi_v \leq \alpha \cdot (c^\top x^* + \pi^\top(1 - y^*)) \quad , \quad (2.7)$$

then there exists a vertex $v \in V$ such that the above also holds for $\gamma = y_v^*$.

Proof of Lemma 18. Fix $\gamma \in (0, 1]$ such that (2.7) holds. Choose $y := \min\{y_v^* : y_v^* \geq \gamma\}$. We claim that

$$\begin{aligned} \frac{3}{2y} \cdot c^\top x^* + \sum_{v \in V : y_v^* < y} \exp\left(-\frac{3}{4y} y_v^*\right) \cdot \pi_v \\ \leq \frac{3}{2\gamma} \cdot c^\top x^* + \sum_{v \in V : y_v^* < \gamma} \exp\left(-\frac{3}{4\gamma} y_v^*\right) \cdot \pi_v \quad , \quad (2.8) \end{aligned}$$

which obviously implies the lemma. For the sake of deriving a contradiction, assume that (2.8) does not hold, and note that because the sums in (2.8) are over the same subset of vertices by definition of y , this assumption can be rewritten as

$$\frac{3}{2} \left(\frac{1}{\gamma} \cdot c^\top x^* - \frac{1}{y} \cdot c^\top x^* \right) < \sum_{v \in V : y_v^* < y} \left(\exp\left(-\frac{3y_v^*}{4y}\right) - \exp\left(-\frac{3y_v^*}{4\gamma}\right) \right) \cdot \pi_v \quad .$$

As $x \mapsto \exp(-x) + x$ is increasing for $x \geq 0$, we have $\exp(-a) - \exp(-b) \leq b - a$ whenever $b \geq a \geq 0$, hence

$$\begin{aligned} \frac{3}{2} \left(\frac{1}{\gamma} \cdot c^\top x^* - \frac{1}{y} \cdot c^\top x^* \right) &< \sum_{v \in V : y_v^* < y} \frac{3}{4} \left(\frac{y_v^*}{\gamma} - \frac{y_v^*}{y} \right) \pi_v \\ &= \frac{3}{4} \left(\pi^\top(1 - y_{1/y}^*) - \pi^\top(1 - y_{1/\gamma}^*) \right) \quad , \end{aligned}$$

where we use the notation y_λ^* of [Lemma 5](#) for $\lambda = 1/y$ and $\lambda = 1/\gamma$. Through scaling down the nonnegative left-hand side by $\frac{1}{2}$ and rearranging terms, this further implies

$$\frac{1}{\gamma} \cdot c^\top x^* + \pi^\top(1 - y_{1/\gamma}^*) < \frac{1}{y} \cdot c^\top x^* + \pi^\top(1 - y_{1/y}^*) \quad . \quad (2.9)$$

Now consider the function $f : \mathbb{R}_{\geq 1} \rightarrow \mathbb{R}$ defined by $\lambda \mapsto \lambda \cdot c^\top x^* + \pi^\top(1 - y_\lambda^*)$. By definition, f is convex and continuous. Moreover, (2.9) shows that f is not monotone

2.2. Overcoming the weaknesses of threshold rounding

increasing (note that $y > \gamma$). Thus, f must be decreasing at the left endpoint of its domain, i.e., at $\lambda = 1$, implying that there exists $\lambda_0 > 1$ such that $f(\lambda_0) < f(1)$. But $f(\lambda_0)$ is an upper bound on the objective value of the solution $(x_{\lambda_0}, y_{\lambda_0})$ of the **PCTSP LP relaxation** that can be constructed through [Lemma 5](#), while $f(1)$ is the objective value of (x^*, y^*) , hence $f(\lambda_0) < f(1)$ contradicts optimality of (x^*, y^*) . This finishes the proof of [Lemma 18](#). $\square$

2.2.6 Proof of [Theorem 12](#)

We complete this section by proving that the threshold choice suggested in [Section 2.2.4](#) yields the desired approximation guarantee. We repeat [Theorem 12](#) for convenience.

Theorem 12. *Let $b = 0.6945$. Sampling γ from the interval $[b, 1]$ using a distribution with density $f(\gamma) \propto \exp(-b/\gamma)$, and starting [Algorithm 1](#) from a feasible solution (x^*, y^*) of the **PCTSP LP relaxation**, we get in polynomial time a **PCTSP** solution of expected value at most $\alpha \cdot (c^\top x^* + \pi^\top (1 - y^*))$, where $\alpha < 1.774$.*

Proof of [Theorem 12](#). The concrete choice of b is left to the end of this proof; with foresight, we only require $b \in [1 - e^{-3/4}, 1)$ for now. Let $I_b := \int_b^1 \exp(-b/\gamma) d\gamma$. Then $f(\gamma) = \exp(-b/\gamma)/I_b$. By [Theorem 10](#), the expected edge cost of the returned cycle $C = (V_C, E_C)$ is at most

$$\mathbb{E}[c(E_C)] \leq \frac{3}{2} \int_b^1 \frac{f(\gamma)}{\gamma} d\gamma \cdot c^\top x^* = \frac{3}{2I_b} \int_b^1 \frac{\exp(-b/\gamma)}{\gamma} d\gamma \cdot c^\top x^* . \quad (2.10)$$

Moreover, by [Theorem 10](#), the expected penalty cost can be bounded by

$$\begin{aligned} \mathbb{E}[\pi(V \setminus V_C)] &\leq \sum_{v \in V: y_v^* < 1} \pi_v(1 - y_v^*) \cdot \frac{1}{1 - y_v^*} \int_{\max\{y_v^*, b\}}^1 \exp\left(-\frac{3y_v^*}{4\gamma}\right) f(\gamma) d\gamma \\ &= \sum_{v \in V: y_v^* < 1} \pi_v(1 - y_v^*) \cdot \underbrace{\frac{1}{I_b(1 - y_v^*)} \cdot \int_{\max\{y_v^*, b\}}^1 \exp\left(-\frac{3y_v^* + 4b}{4\gamma}\right) d\gamma}_{=: f_b(y_v^*)} . \end{aligned}$$

We investigate for which $y \in [0, 1)$ the function $y \mapsto f_b(y)$ attains its maximum value. If $y \in [0, b]$, convexity of the map $y \mapsto g_\gamma(y) := \frac{1}{1-y} \exp\left(-\frac{3y}{4\gamma}\right)$ implies that $g_\gamma(y) \leq \max\{g_\gamma(0), g_\gamma(b)\}$. Since $b \geq 1 - e^{-3/4}$ and because $\gamma \geq b$, the latter maximum is equal to $g_\gamma(b)$, and we can conclude that $f_b(y)$ attains its maximum for some $y \in [b, 1)$, where

$$f_b(y) = \frac{1}{I_b(1 - y)} \cdot \int_y^1 \exp\left(-\frac{3y + 4b}{4\gamma}\right) d\gamma .$$

Computational experiments suggest that $f_b(y)$ attains its maximum at $y = b$, but it seems hard to prove this analytically. For this reason, we aim for obtaining good bounds on the

maximum value in what follows. Observe that the integrand $h_{b,y}(\gamma) := \exp\left(-\frac{3y+4b}{4\gamma}\right)$ is concave over $\gamma \in [y, 1]$. Hence,

$$\begin{aligned} \int_y^1 h_{b,y}(\gamma) d\gamma &= \int_y^{\frac{y+1}{2}} h_{b,y}(\gamma) d\gamma + \int_{\frac{y+1}{2}}^1 h_{b,y}(\gamma) d\gamma \\ &\leq \frac{1-y}{2} \cdot \left(h_{b,y}\left(\frac{3y+1}{4}\right) + h_{b,y}\left(\frac{y+3}{4}\right) \right). \end{aligned}$$

Altogether, we therefore get that

$$\begin{aligned} \max_{y \in [0,1)} f_b(y) &= \max_{y \in [b,1)} f_b(y) \\ &\leq \max_{y \in [b,1)} \frac{1}{2I_b} \left(h_{b,y}\left(\frac{3y+1}{4}\right) + h_{b,y}\left(\frac{y+3}{4}\right) \right) \\ &= \frac{1}{2I_b} \cdot \max_{y \in [b,1)} \underbrace{\left(\exp\left(-\frac{3y+4b}{3y+1}\right) + \exp\left(-\frac{3y+4b}{y+3}\right) \right)}_{=: \theta_b(y)}. \end{aligned}$$

Together with (2.10), this implies that we get in polynomial time a **PCTSP** solution of expected value at most $\alpha \cdot (c^\top x^* + \pi^\top (1 - y^*))$, where

$$\alpha := \frac{1}{2I_b} \cdot \max \left\{ 3 \cdot \int_b^1 \frac{\exp(-b/\gamma)}{\gamma} d\gamma, \max_{y \in [b,1)} \theta_b(y) \right\}.$$

We numerically evaluate the latter expression and obtain $\alpha < 1.774$ for $b = 0.6945$, as claimed in [Theorem 12](#). More concretely, to bound $\max_{y \in [b,1)} \theta_b(y)$, we note that the derivative of $\theta_b(y)$ on the interval $[b, 1]$ can be bounded by a constant, hence the maximum can be approximated up to a prescribed error by evaluating $\theta_b(y)$ on a sufficiently fine discretization of possible values of y . $\square$

2.3 Simple 2-approximations for PCTSP and PCST through decompositions

In this section we show that for PCTSP and PCST, we can obtain an approximation guarantee similar to the one of the primal-dual approach by Goemans and Williamson [GW95] through very simple algorithms. More precisely, the proposed algorithms return cycles and trees, respectively, of cost at most twice the x -cost of an optimum LP solution plus once the y -cost of an optimum LP solution. The main ingredient giving rise to these algorithms is the following simplified variation of our decomposition lemma used in Section 2.2, which can also be obtained by utilizing an existential result on packing branchings in a directed multigraph by Bang-Jensen, Frank, and Jackson [BFJ95, Theorem 2.6]. This decomposition will also be key to the up-to-date best known approximation algorithm for PCTSP that we discuss in Section 2.4.

Lemma 19. *Let (x, y) be a feasible solution of the PCTSP LP relaxation. We can compute in polynomial time a set of trees $\mathcal{T}$ containing the root r and weights $\mu \in [0, 1]^T$ such that $\sum_{T \in \mathcal{T}} \mu_T = 1$,*

$$\sum_{T \in \mathcal{T}} \mu_T \chi^{E[T]} \leq x, \quad \text{and} \quad \forall v \in V: \sum_{T \in \mathcal{T}: v \in V[T]} \mu_T = y_v.$$

We prove Lemma 19 through a common generalization with Lemma 7 in Section 2.5. Based on Lemma 19, the simple algorithm we propose for PCTSP is the following.

Algorithm 2: A simple 2-approximation for PCTSP

Input: PCTSP instance (G, r, c, π) on $G = (V, E)$.

1. Compute an optimal solution (x^*, y^*) of the PCTSP LP relaxation.
2. Compute a set of trees $\mathcal{T}$ through Lemma 19 applied to (x^*, y^*) .
3. For each tree $T \in \mathcal{T}$, obtain a cycle C_T through duplicating T and shortcutting.
4. Let $S = \arg \min_{T \in \mathcal{T}} (c(E[C_T]) + \pi(V \setminus V[C_T]))$.

return C_S .

Theorem 20. *Algorithm 2 returns, in polynomial time, a cycle C satisfying*

$$c(E[C]) + \pi(V \setminus V[C]) \leq 2 \cdot c^\top x^* + \pi^\top (1 - y^*),$$

where (x^*, y^*) is the optimal solution of the PCTSP LP relaxation computed in Step 1 of the algorithm.

Proof. We do a probabilistic analysis of the algorithm. The trees in $\mathcal{T}$ from Lemma 19 come with weights $\mu \in [0, 1]^T$ such that $\sum_{T \in \mathcal{T}} \mu_T = 1$ and $\sum_{T \in \mathcal{T}} \mu_T \chi^{E[T]} \leq x^*$.

Sampling a tree T from $\mathcal{T}$ with marginals μ gives

$$\mathbb{E}[c(E[C_T])] \leq \sum_{T \in \mathcal{T}} \mu_T \cdot 2c(E[T]) = 2 \cdot c^\top \left(\sum_{T \in \mathcal{T}} \mu_T \chi^{E[T]} \right) \leq 2 \cdot c^\top x^* .$$

Similarly, for the penalty term we get

$$\begin{aligned} \mathbb{E}[\pi(V \setminus V[C_T])] &= \sum_{v \in V} \mathbb{P}[v \notin T] \cdot \pi_v = \sum_{v \in V} \left(1 - \sum_{T \in \mathcal{T}: v \in V[T]} \mu_T \right) \cdot \pi_v \\ &= \sum_{v \in V} (1 - y_v^*) \cdot \pi_v = \pi^\top (1 - y^*) . \end{aligned}$$

Hence, the expected objective value of C_T can be bounded by $2 \cdot c^\top x^* + \pi^\top (1 - y^*)$. Thus, also at least one of the cycles C_T for $T \in \mathcal{T}$ satisfies the desired guarantee. $\square$

For **PCST**, we consider the following very related LP relaxation.

$$\begin{aligned} \min \quad & \sum_{e \in E} c_e x_e + \sum_{v \in V} \pi_v (1 - y_v) \\ & x(\delta(S)) \geq y_v \quad \forall S \subseteq V \setminus \{r\}, v \in S \\ & y_r = 1 \\ & x_e \geq 0 \quad \forall e \in E \\ & y_v \in [0, 1] \quad \forall v \in V . \end{aligned} \quad (\text{PCST LP relaxation})$$

The main step towards mimicking [Algorithm 2](#) is to transform a solution of the **PCST LP relaxation** to one of the **PCTSP LP relaxation** to be able to apply [Lemma 19](#) in a black-box way, and later transform the obtained components back (see [Step 4](#) of [Algorithm 3](#)). The latter is required because edge costs in **PCST** are not necessarily metric.

Algorithm 3: A simple 2-approximation for **PCST**

Input: **PCST** instance (G, r, c, π) on $G = (V, E)$.

1. Compute an optimal solution (x^*, y^*) of the **PCST LP relaxation**.
2. Consider the **PCTSP** instance $(\bar{G}, r, \bar{c}, \pi)$ where $\bar{c}$ denotes the metric closure of the edge lengths c on the complete graph $\bar{G}$ on V . Obtain a solution (x, y^*) of the **PCTSP LP relaxation** from $(2x^*, y^*)$ by splitting off (cf. [Theorem 28](#) on page 48) until all degree constraints are satisfied.
3. Compute a set of trees $\mathcal{T}$ through [Lemma 19](#) applied to (x, y^*) .
4. For each $T \in \mathcal{T}$, let T_0 be obtained from T through replacing edges by shortest paths connecting their endpoints in G .
5. Let $S = \arg \min_{T \in \mathcal{T}} (c(E[T_0]) + \pi(V \setminus V[T_0]))$.

return a minimum spanning tree on $V[S_0]$.

2.3. Simple 2-approximations for PCTSP and PCST through decompositions

Theorem 21. *Algorithm 3 returns, in polynomial time, a tree T satisfying*

$$c(E[T]) + \pi(V \setminus V[T]) \leq 2 \cdot c^\top x^* + \pi^\top(1 - y^*) ,$$

where (x^*, y^*) is the optimal solution of the *PCST LP relaxation* computed in *Step 1* of the algorithm.

Proof. As in [Algorithm 3](#), let $\bar{c}$ denote the metric closure of the (not necessarily metric) edge lengths c . Note that $\bar{c}^\top x \leq 2c^\top x^*$, as splitting off does not increase the total length with respect to metric lengths. Additionally, for the connected multigraphs T_0 computed in [Step 4](#), we have $\bar{c}(E[T]) = c(E[T_0])$ by definition. With this at hand, we can again do a randomized analysis, where we sample a tree T from $\mathcal{T}$ with marginals equal to the weights μ obtained through [Lemma 19](#). This gives an expected objective value

$$\begin{aligned} \mathbb{E}[c(E[T_0]) + \pi(V \setminus V[T_0])] &= \mathbb{E}[\bar{c}(E[T])] + \sum_{v \in V} \mathbb{P}[v \notin T_0] \cdot \pi_v \\ &\leq \sum_{T \in \mathcal{T}} \mu_T \bar{c}(E[T]) + \sum_{v \in V} \left(1 - \sum_{T \in \mathcal{T}: v \in V[T]} \mu_T\right) \cdot \pi_v \\ &\leq \bar{c}^\top x + \sum_{v \in V} (1 - y_v^*) \pi_v \\ &\leq 2c^\top x^* + \pi^\top(1 - y^*) . \end{aligned}$$

Clearly, the same bound holds for a minimum spanning tree on $V[T_0]$. Thus again, we get the desired bound in expectation, hence for at least one $T \in \mathcal{T}$, we obtain the claimed guarantee. $\square$

2.4 Pruning the trees for cheaper parity correction

In this section we prove [Theorem 2](#), our main result on [PCTSP](#). As mentioned earlier, this is joint work with Nathan Klein and Martin Nägele [[BKN24](#)].

Theorem 2. *There is an LP-relative 1.599-approximation algorithm for [PCTSP](#).*

On a high level, the proposed algorithm proceeds as follows. After some preprocessing step which guarantees that for each $v \in V$ either $y_v = 0$ or $y_v \geq \delta$ for some parameter $\delta \in [0, 1]$, some tree is sampled according to the distribution given by [Lemma 19](#). Then, the tree is pruned by removing all subtrees containing only vertices v with $y_v < \gamma$ for some threshold $\gamma \in [\delta, 1]$. The preprocessing as well as the tree pruning are key to an improved bound on the cost of the parity correction while paying a manageable additional amount of penalties. The novel pruning step guarantees that in the resulting tree T each odd cut $F \subseteq E$ either contains at least three tree edges, or $x(F)$ is large. We use this observation to construct a point in the dominant of the $\text{odd}(T)$ -join polytope of small total cost by, roughly spoken, combining the given LP solution with the incidence vector of the sampled tree. Choosing δ and γ with respect to certain probability distributions yields the desired guarantee. We also show how to derandomize the algorithm using a Best-of-Many approach.

[Section 2.4.1](#) illustrates the main ideas yielding the improved approximation guarantee. [Section 2.4.2](#) contains a simple analysis of the proposed algorithm that yields an approximation ratio of $\frac{1+\sqrt{5}}{2} \approx 1.618$, the golden ratio. Exploiting some remaining flexibility in the proof, we finally push the approximation guarantee slightly below 1.6 in [Section 2.4.3](#).

2.4.1 Our algorithm

As our first algorithm discussed in [Section 2.2](#), we follow the basic idea of the Christofides-Serdyukov algorithm for [TSP](#), which is to combine a spanning tree T with a shortest $\text{odd}(T)$ -join, and shortcut an Eulerian tour in the resulting even-degree graph to obtain a cycle. Typically, for an even cardinality set Q (in particular, for $Q = \text{odd}(T)$), the cost of a shortest Q -join is bounded by the cost $c^\top z$ of a point z that is feasible for the dominant of the Q -join polytope, which is given by

$$P_{Q\text{-join}}^\uparrow := \{x \in \mathbb{R}^E : x(\delta(S)) \geq 1 \ \forall S \subseteq V \text{ with } |S \cap Q| \text{ odd}\} ,$$

cf. [Section 1.2.3.4](#).

We use this approach with two additional variations: First, because our setting allows to not include some vertices in the returned tour, we use trees T that may not span all vertices in V . Second, we follow what is known as a Best-of-Many approach, i.e., we construct a polynomial-size set of trees, construct a tour from each of them, and return the best. For the analysis of such an approach, one typically provides a distribution over the involved trees and analyzes the expected cost of the returned tour—which implies the same bound on the best tour.

2.4. Pruning the trees for cheaper parity correction

It is an intriguing question whether parity correction can be done at expected cost $c^\top x/2$. Such a result would immediately lead to a $3/2$ -approximation algorithm for **PCTSP**, matching the guarantee of the Christofides-Serdyukov algorithm for **TSP**. While in the setting of classical **TSP**, we have $x/2 \in P_{Q\text{-join}}^\uparrow$ for any set Q of even cardinality (and can thus bound the cost of parity correction for *any* tree T by $c^\top x/2$), this is no longer true in the prize-collecting setting.

We base our tree construction on the decomposition lemma used in [Section 2.3](#) that we repeat for convenience.

Lemma 19. *Let (x, y) be a feasible solution of the **PCTSP LP relaxation**. We can compute in polynomial time a set of trees $\mathcal{T}$ containing the root r and weights $\mu \in [0, 1]^\mathcal{T}$ such that $\sum_{T \in \mathcal{T}} \mu_T = 1$,*

$$\sum_{T \in \mathcal{T}} \mu_T \chi^{E[T]} \leq x, \quad \text{and} \quad \forall v \in V: \sum_{T \in \mathcal{T}: v \in V[T]} \mu_T = y_v.$$

Given a tree T , we first prune it to obtain a tree T' (in a way we will shortly explain) and then perform parity correction. To analyze the cost of the parity correction, we construct a point $z \in P_{\text{odd}(T)\text{-join}}^\uparrow$ that is of the form

$$z = \alpha \cdot x + \beta \cdot \chi^{E[T']} \tag{2.11}$$

for some coefficients $\alpha, \beta \in \mathbb{R}_{\geq 0}$. (In fact, we will choose different coefficients β for different parts of the tree.) As the existence of cuts $S \subsetneq V$ for which both $x(\delta(S))$ and $|\delta_{T'}(S)|$ are small may require to choose α and β large, we preprocess both the solution (x, y) of the **PCTSP LP relaxation** that we use as well as the trees that we obtain from it through [Lemma 19](#).

In our first preprocessing step, we get rid of cuts S for which $x(\delta(S))$ is very small. To this end, observe that our algorithm may always drop vertices $v \in V \setminus \{r\}$ for which y_v is very small. Concretely, if our tour does not visit a vertex v with $y_v \leq \delta$ for some $\delta \in [0, 1)$, we pay a penalty of π_v , which is at most a factor of $1/(1 - \delta)$ larger than the fractional penalty $\pi_v(1 - y_v)$ occurring in the LP objective. Thus, if we aim for an α -approximation algorithm, we may safely choose $\delta = 1 - 1/\alpha$ and drop all vertices with $y_v \leq \delta$. Crucially for our analysis, we can also perform this dropping on the level of solutions of the **PCTSP LP relaxation** by using the so-called *splitting off* technique [[Lov76](#); [Mad78](#); [Fra92](#)] (see [Section 2.5](#) for details).

Lemma 22 (Splitting off). *Let (x^*, y^*) be a feasible solution of the **PCTSP LP relaxation**. Let $v \in V \setminus \{r\}$. There is a deterministic algorithm that computes in polynomial time a feasible solution (x, y) of the **PCTSP LP relaxation** with $y_v = 0$, as well as*

$$c^\top x \leq c^\top x^* \quad \text{and} \quad \forall u \in V \setminus \{v\}: y_u = y_u^*.$$

Our first preprocessing step then consists of repeatedly applying [Lemma 22](#) to vertices $v \in V \setminus \{r\}$ with $y_v < \delta$ for some parameter $\delta \in [0, 1)$ that we fix later.

Our second preprocessing step affects the trees that we obtain through [Lemma 19](#). While the first preprocessing step guarantees that there are no nontrivial cuts S with

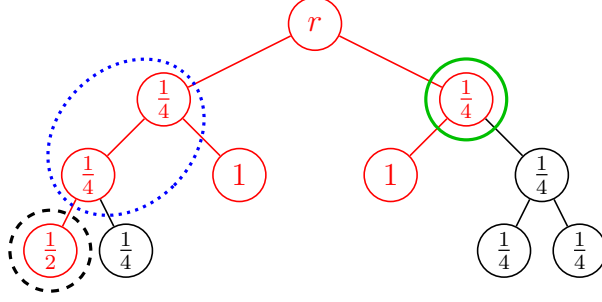


Figure 2.3: The core $T' = \text{core}(T, 1/2)$ of the underlying tree T at threshold $\gamma = 1/2$ is highlighted in red, where y_v is shown for each node $v \in V[T]$. We emphasize the different situations that can occur in terms of cuts: For the dotted blue cut S , $x(\delta(S))$ may be small as it does not contain a vertex with $y_v \geq 1/2$, however there are at least three edges of T' in $\delta(S)$ that make up for this. The dashed black cut is a cut S with only one tree edge in $\delta(S)$, however it contains a vertex with $y_v \geq 1/2$ so $x(\delta(S))$ is large. Finally, for the solid green cut, $x(\delta(S))$ may be small and there are only two tree edges of T' in $\delta(S)$, however for this cut, $|\delta_{T'}(S)|$ is even, so there is no corresponding constraint in the dominant of the $\text{odd}(T')$ -join polytope.

very small $x(\delta(S))$, we also want to eliminate cuts S with moderately small $x(\delta(S))$ for which T has only a single edge in $\delta(S)$, so that the combination z defined in (2.11) gets a significant contribution from at least one of x or $\chi^{E[T]}$ on every nontrivial cut. To this end, we use a pruning step defined as follows (also see Figure 2.3).

Definition 23 (Core). For a fixed solution (x, y) of the *PCTSP LP relaxation*, a tree T containing the root vertex, and a threshold γ , the core of T with respect to γ , denoted by $\text{core}(T, \gamma)$, is the inclusion-wise minimal subtree of T that spans the root and all vertices $v \in V[T]$ with $y_v \geq \gamma$.

Indeed, if $T' = \text{core}(T, \gamma)$, we know that for any nonempty cut $S \subsetneq V[T']$, we either have $x(\delta(S)) \geq 2\gamma$, or $|\delta_T(S)| > 1$. Additionally, the only relevant such cuts S in terms of parity correction on T' are those with $|S \cap \text{odd}(T')|$ odd, which is well known to be equivalent to $|\delta_{T'}(S)|$ being odd because

$$|S \cap \text{odd}(T')| \equiv \sum_{v \in S} \deg_{T'}(v) = 2 \cdot \left| E[T'] \cap \binom{S}{2} \right| + |\delta_{T'}(S)| \equiv |\delta_{T'}(S)| \pmod{2}. \quad (2.12)$$

Thus, for cuts S with $|S \cap \text{odd}(T')|$ odd, $|\delta_{T'}(S)| > 1$ implies $|\delta_{T'}(S)| \geq 3$, thereby further boosting the load on $\delta(S)$ in z for this case (also see Figure 2.3 for examples of the different types of cuts that may appear). Altogether, we are now ready to state our new algorithm for PCTSP.

We remark that Algorithm 4 can be implemented to run in polynomial time. Indeed, by Lemma 19 and Lemma 22, Steps 2 to 4 run in polynomial time. Moreover, we can compute an optimum solution to the PCTSP LP relaxation in polynomial time as the separation problem can be solved by computing minimum r - v cut sizes for each $v \in V \setminus \{r\}$ as mentioned in Section 2.1.

2.4. Pruning the trees for cheaper parity correction

Algorithm 4: Our second new approximation algorithm for **PCTSP**

Input: *PCTSP* instance (G, r, c, π) on $G = (V, E)$, $\delta \in [0, 1)$.

1. Compute an optimal solution (x^*, y^*) of the **PCTSP LP relaxation**.
2. Apply **Lemma 22** to all $v \in V$ with $y_v^* < \delta$, resulting in another feasible solution (x, y) of the **PCTSP LP relaxation**.
3. Compute a set $\mathcal{T}$ of trees through **Lemma 19** applied to (x, y) .
4. Let

$$\mathcal{T}' = \bigcup_{\gamma \in \{y_v : v \in V\}} \{\text{core}(T, \gamma) : T \in \mathcal{T}\} .$$

return *Best tour found by doing parity correction on all trees in $\mathcal{T}'$.*

We show in the next subsection that there is a constant δ for which **Algorithm 4** is a $(1 + \sqrt{5})/2$ -approximation algorithm. To go beyond that and prove **Theorem 2**, we will later allow an instance-specific choice of δ .

2.4.2 A $(1 + \sqrt{5})/2$ -approximation guarantee

In this subsection, we prove the following result that gives the golden ratio as approximation guarantee. This is slightly weaker than what **Theorem 2** claims, but the proof is simple and illustrates our main ideas.

Theorem 24. *Algorithm 4 is an α -approximation algorithm for **PCTSP** with*

$$\alpha := \max \left\{ \frac{5 - 2\delta}{3 - \delta}, \frac{3 - \delta}{2 - \delta}, \frac{1}{1 - \delta} \right\} .$$

In particular, for $\delta = (3 - \sqrt{5})/2 \approx 0.382$, we get $\alpha = (1 + \sqrt{5})/2 \approx 1.618$.

Throughout this subsection, we fix a solution (x, y) of the **PCTSP LP relaxation** that was obtained from an optimal solution (x^*, y^*) through splitting off as in **Step 2** of **Algorithm 4**, and we fix a set $\mathcal{T}$ of trees with weights $(\mu_T)_{T \in \mathcal{T}}$ that is obtained in **Step 3**, i.e., through **Lemma 19** applied to (x, y) . Moreover, we sample a random tree T from the set $\mathcal{T}'$ constructed in **Step 4** of **Algorithm 4** as follows: For a fixed value $\kappa \in [\delta, 1]$, sample a threshold $\gamma \in [\delta, \kappa]$ such that for any $t \in [\delta, \kappa]$, we have

$$\mathbb{P}[\gamma \leq t] = \frac{3 - \delta - \kappa}{3 - \delta - t} . \quad (2.13)$$

Independently, sample a tree $T_0 \in \mathcal{T}$ with marginals given by $(\mu_T)_{T \in \mathcal{T}}$. Then, define

$$T := \text{core}(T_0, \gamma) . \quad (2.14)$$

By definition of the core, it is clear that $T \in \mathcal{T}'$ even if $\gamma \notin \{y_v : v \in V\}$. To prove **Theorem 24**, we bound the expected cost of a tour constructed from T by parity correction. We remark that for proving **Theorem 24**, we only need $\kappa = 1$; we nonetheless proceed in this generality here to be able to reuse some of the following statements in a proof of **Theorem 2**. To start with, we bound the expected tour length.

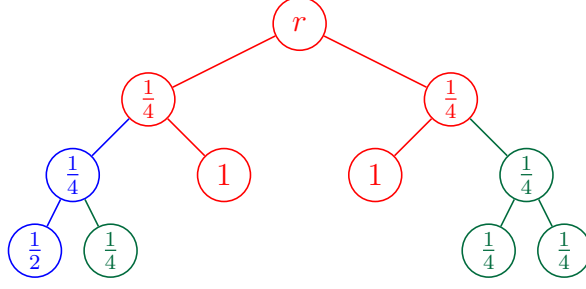


Figure 2.4: Here, the variables η_i are given by the sequence $1, 1/2, 1/4$. Edges in E_1 are drawn in red, those in E_2 in blue, and those in E_3 in green. We also color the nodes for intuition, however the sets E_i consist solely of edges.

Lemma 25. *Let $T = \text{core}(T_0, \gamma)$ be a random tree generated as described in and above (2.14), and let C be the cycle obtained through parity correction on T and shortcutting an Eulerian walk in the resulting graph. Then*

$$\mathbb{E}[c(E[C])] \leq \frac{7 - 2\delta - 2\kappa}{3 - \delta} \cdot c^\top x^* .$$

Proof. Let $\eta_1 > \dots > \eta_k$ such that $\{\eta_1, \dots, \eta_k\} = \{y_v : v \in V\}$. Define

$$E_i := \begin{cases} E[\text{core}(T_0, \eta_1)] & \text{for } i = 1 \\ E[\text{core}(T_0, \eta_i)] \setminus E[\text{core}(T_0, \eta_{i-1})] & \text{for } i \in \{2, \dots, k\} \end{cases} .$$

We refer to Figure 2.4 for an illustration of the sets E_i . In particular, this definition implies that

$$c(E[T]) = c(E[\text{core}(T_0, \gamma)]) = \sum_{i \in [k] : \eta_i \geq \gamma} c(E_i) . \quad (2.15)$$

To bound the cost of parity correction on T , we claim that

$$z := \frac{1}{3 - \delta} \cdot x + \sum_{i \in [k] : \eta_i \geq \gamma} \left(1 - \frac{2\eta_i}{3 - \delta}\right) \cdot \chi^{E_i}$$

lies in the dominant of the $\text{odd}(T)$ -join polytope. This implies that a shortest $\text{odd}(T)$ -join J has length

$$c(E[J]) \leq \frac{1}{3 - \delta} \cdot c^\top x + \sum_{i \in [k] : \eta_i \geq \gamma} \left(1 - \frac{2\eta_i}{3 - \delta}\right) \cdot c(E_i) .$$

Combining this with (2.15) and taking expectations immediately gives the desired

2.4. Pruning the trees for cheaper parity correction

bound:

$$\begin{aligned}
\mathbb{E}[c(E[C])] &\leq \frac{1}{3-\delta} \cdot c^\top x + \sum_{T_0 \in \mathcal{T}} \mu_{T_0} \sum_{i=1}^k \mathbb{P}[\gamma \leq \eta_i] \cdot \left(2 - \frac{2\eta_i}{3-\delta}\right) \cdot c(E_i) \\
&\leq \frac{1}{3-\delta} \cdot c^\top x + \sum_{T_0 \in \mathcal{T}} \mu_{T_0} \sum_{i=1}^k \frac{6-2\delta-2\kappa}{3-\delta} \cdot c(E_i) \quad (\text{using (2.13)}) \\
&= \frac{1}{3-\delta} \cdot c^\top x + \frac{6-2\delta-2\kappa}{3-\delta} \cdot \sum_{T_0 \in \mathcal{T}} \mu_{T_0} c(E[T_0]) \\
&\leq \frac{7-2\delta-2\kappa}{3-\delta} \cdot c^\top x \quad (\text{using Lemma 19})
\end{aligned}$$

By the construction of (x, y) from (x^*, y^*) through splitting off in [Step 2](#) of [Algorithm 4](#), we have $c^\top x \leq c^\top x^*$, hence the above implies the desired. It thus remains to show that $z \in P_{\text{odd}(T)\text{-join}}^\uparrow$, i.e., that $z \geq 0$ and $z(\delta(S)) \geq 1$ for every $S \subseteq V$ with $|S \cap \text{odd}(T)|$ odd. Clearly, we have $z \geq 0$ since $\delta \in [0, 1]$. As remarked in [\(2.12\)](#), $|S \cap \text{odd}(T)|$ is odd if and only if $|\delta_T(S)|$ is. If $|\delta_T(S)| \geq 3$,

$$\begin{aligned}
z(\delta(S)) &= \frac{1}{3-\delta} \cdot x(\delta(S)) + \sum_{i \in [k]: \eta_i \geq \gamma} \left(1 - \frac{2\eta_i}{3-\delta}\right) \cdot |E_i \cap \delta_T(S)| \\
&\geq \frac{2\delta}{3-\delta} + \left(1 - \frac{2}{3-\delta}\right) \cdot 3 = 1,
\end{aligned}$$

where we used that $x(\delta(S)) \geq 2\delta$ because $y_v \geq \delta$ for all $v \in S$, and that $\eta_i \leq 1$ for each $i \in [k]$. Otherwise, we have $|\delta_T(S)| = 1$. Let $i \in [k]$ such that $\delta_T(S) \subseteq E_i$. Then $x(\delta(S)) \geq 2\eta_i$, hence also in this case

$$z(\delta(S)) \geq \frac{2\eta_i}{3-\delta} + \left(1 - \frac{2\eta_i}{3-\delta}\right) = 1. \quad \square$$

Next, we analyze the expected penalty incurred when starting with such a random tree T .

Lemma 26. *Let $T = \text{core}(T_0, \gamma)$ be a random tree generated as described in and above [\(2.14\)](#). Then, for every $v \in V$ with $y_v^* \geq \delta$, we have*

$$\mathbb{P}[v \in V[T]] \geq \begin{cases} y_v^* \cdot \frac{3-\delta-\kappa}{3-\delta-y_v^*} & \text{if } y_v^* \in [\delta, \kappa] \\ y_v^* & \text{if } y_v^* \in (\kappa, 1] \end{cases}.$$

Proof. By construction, the solution (x, y) has the property that for all $v \in V$ with $y_v^* \geq \delta$, $y_v = y_v^*$. Hence, for vertices $v \in V$ with $y_v^* \geq \delta$, we have $\mathbb{P}[v \in V[T_0]] = y_v^*$ by [Lemma 19](#). Hence,

$$\mathbb{P}[v \in V[T]] \geq \mathbb{P}[v \in V[T_0]] \cdot \mathbb{P}[\gamma \leq y_v^*] = y_v^* \cdot \mathbb{P}[\gamma \leq y_v^*].$$

If $y_v^* > \kappa$, then $\mathbb{P}[\gamma \leq y_v^*] = 1$ and $\mathbb{P}[v \in V[T]] = y_v^*$. In the remaining case $y_v^* \in [\delta, \kappa]$, we use [\(2.13\)](#) to obtain the desired. $\square$

Together, [Lemmas 25](#) and [26](#) allow us to conclude [Theorem 24](#).

Proof of Theorem 24. Let T be a random tree generated as described in and above [\(2.14\)](#), and let C be the cycle obtained through parity correction on T and shortcutting an Eulerian walk in the resulting graph.

Let $v \in V$. If $y_v^* < \delta$, then $\mathbb{P}[v \notin V[C]] = 1 \leq \frac{1}{1-\delta}(1 - y_v^*)$. If $y_v^* > \kappa$, then $\mathbb{P}[v \notin V[C]] \leq 1 - y_v^*$ by [Lemma 26](#). If $\delta \leq y_v^* \leq \kappa$ then, again by [Lemma 26](#),

$$\begin{aligned} \mathbb{P}[v \notin V[C]] &\leq 1 - y_v^* \cdot \frac{3 - \delta - \kappa}{3 - \delta - y_v^*} \\ &= \frac{(3 - \delta)(1 - y_v^*) - y_v^*(1 - \kappa)}{3 - \delta - y_v^*} \\ &\leq \frac{3 - \delta}{3 - \delta - \kappa} \cdot (1 - y_v^*) \quad (\text{because } y_v^* \leq \kappa \leq 1) \end{aligned}$$

Hence, together with [Lemma 25](#) we get

$$\begin{aligned} &\mathbb{E}[c(E[C]) + \pi(V \setminus V[C])] \\ &\leq \frac{7 - 2\delta - 2\kappa}{3 - \delta} \cdot c^\top x^* + \max \left\{ \frac{3 - \delta}{3 - \delta - \kappa}, \frac{1}{1 - \delta} \right\} \cdot \pi^\top (1 - y^*) \\ &\leq \max \left\{ \frac{7 - 2\delta - 2\kappa}{3 - \delta}, \frac{3 - \delta}{3 - \delta - \kappa}, \frac{1}{1 - \delta} \right\} \cdot (c^\top x^* + \pi^\top (1 - y^*)) \quad (2.16) \end{aligned}$$

Independently of the realization of the involved random variables, the cycle C is one that is generated in [Algorithm 4](#). The maximum in [\(2.16\)](#) is minimized for $\kappa = 1$ and $\delta = (3 - \sqrt{5})/2$, where it evaluates to $(1 + \sqrt{5})/2$, thus giving the guarantee claimed in [Theorem 24](#). $\square$

2.4.3 Getting below 1.6

To improve upon the golden ratio approximation guarantee that we proved in [Section 2.4.2](#), we exploit some remaining flexibility in the proof: Sampling not only the threshold γ but also δ from a distribution allows for balancing off costs better than before. The choice of distribution here is not best possible (though close to best possible, see [Remark 27](#)), but designed to demonstrate that the true approximability of **PCTSP** is below 1.6 in a way that reduces the use of computers to the evaluation of a “simple” function that does not involve integrals. To still obtain a deterministic algorithm, we also show how actually sampling δ can be avoided by trying polynomially many instance-dependent values.

Proof of Theorem 2. For constants κ_0 and κ to be fixed later, we sample $\delta \in [\kappa_0, \kappa]$ from a distribution with density $f(\delta) = \nu \cdot (3 - \delta)(\kappa - \delta)^{2.2}$, where

$$\begin{aligned} \nu &:= \left(\int_{\kappa_0}^{\kappa} (3 - \delta)(\kappa - \delta)^{2.2} d\delta \right)^{-1} = \left(\int_{\kappa_0}^{\kappa} (3 - \kappa)(\kappa - \delta)^{2.2} + (\kappa - \delta)^{3.2} d\delta \right)^{-1} \\ &= \left(\frac{(3 - \kappa)(\kappa - \kappa_0)^{3.2}}{3.2} + \frac{(\kappa - \kappa_0)^{4.2}}{4.2} \right)^{-1}. \end{aligned}$$

2.4. Pruning the trees for cheaper parity correction

Using this δ , sample a tree T as described in and above (2.14) (using the same κ as here), and let C be the cycle generated from parity correction on T and shortcutting an Eulerian walk in the resulting graph. Using Lemma 25, we get that the expected length of the cycle is

$$\begin{aligned}\mathbb{E}[c(E[C])] &\leq \nu \cdot \int_{\kappa_0}^{\kappa} (7 - 2\delta - 2\kappa)(\kappa - \delta)^{2.2} d\delta \cdot c^\top x^* \\ &= \nu \cdot \underbrace{\left(\frac{(7 - 4\kappa)(\kappa - \kappa_0)^{3.2}}{3.2} + \frac{2(\kappa - \kappa_0)^{4.2}}{4.2} \right)}_{=: g(\kappa, \kappa_0)} \cdot c^\top x^* .\end{aligned}$$

Next, we bound the expected penalty. Let $v \in V$. If $y_v^* < \kappa_0$, then $\mathbb{P}[v \notin V[C]] = 1 \leq \frac{1}{1 - \kappa_0} \cdot (1 - y_v^*)$. If $y_v^* > \kappa$, then $\mathbb{P}[v \notin V[C]] \leq 1 - y_v^*$ by Lemma 26. For $y_v^* \in [\kappa_0, \kappa]$, we again use Lemma 26 and obtain

$$\mathbb{P}[v \notin V[C]] \leq 1 - y_v^* \cdot \nu \cdot \int_{\kappa_0}^{y_v^*} \frac{3 - \delta - \kappa}{3 - \delta - y_v^*} (3 - \delta)(\kappa - \delta)^{2.2} d\delta .$$

Now observe that for $y_v^* \leq \kappa$ the function

$$\begin{aligned}\delta \mapsto \phi(\delta) &:= \frac{(3 - \delta - \kappa)(3 - \delta)}{3 - \delta - y_v^*} = 3 - \delta - \kappa + \frac{(3 - \delta - \kappa)y_v^*}{3 - \delta - y_v^*} \\ &= 3 - \delta - \kappa + y_v^* - \frac{y_v^*(\kappa - y_v^*)}{3 - \delta - y_v^*}\end{aligned}$$

is concave on $[\kappa_0, \kappa]$, hence for each $\delta \in [\kappa_0, \kappa]$, we have

$$\phi(\delta) \geq \phi(\kappa_0) \cdot \frac{\kappa - \delta}{\kappa - \kappa_0} + \phi(\kappa) \cdot \frac{\delta - \kappa_0}{\kappa - \kappa_0} .$$

Plugging in this bound and evaluating the involved integrals gives

$$\begin{aligned}\mathbb{P}[v \notin V[C]] &\leq 1 - \frac{y_v^* \cdot \nu}{\kappa - \kappa_0} \cdot \left(\phi(\kappa_0) \cdot \int_{\kappa_0}^{y_v^*} (\kappa - \delta)^{3.2} d\delta \right. \\ &\quad \left. + \phi(\kappa) \cdot \int_{\kappa_0}^{y_v^*} (\delta - \kappa_0)(\kappa - \delta)^{2.2} d\delta \right) \\ &= 1 - \frac{y_v^* \cdot \nu}{\kappa - \kappa_0} \cdot \left((\phi(\kappa_0) - \phi(\kappa)) \cdot \frac{(\kappa - \kappa_0)^{4.2} - (\kappa - y_v^*)^{4.2}}{4.2} \right. \\ &\quad \left. + \phi(\kappa) \cdot (\kappa - \kappa_0) \cdot \frac{(\kappa - \kappa_0)^{3.2} - (\kappa - y_v^*)^{3.2}}{3.2} \right) \\ &=: h_{y_v^*}(\kappa, \kappa_0) .\end{aligned}$$

Thus

$$\mathbb{E}[\pi(V \setminus V[C])] \leq \underbrace{\max \left\{ \frac{1}{1 - \kappa_0}, \max_{y \in [\kappa_0, \kappa]} \frac{h_y(\kappa, \kappa_0)}{1 - y} \right\}}_{=: h(\kappa, \kappa_0)} \cdot \pi^\top(1 - y^*) , \quad (2.17)$$

and we get a bound on the expected total cost of the form

$$\mathbb{E}[c(E[C]) + \pi(V \setminus V[C])] \leq \max\{g(\kappa, \kappa_0), h(\kappa, \kappa_0)\} \cdot (c^\top x^* + \pi^\top(1 - y^*)) .$$

The latter maximum evaluates to slightly below 1.599 for $\kappa_0 = 0.3724$ and $\kappa = 0.9971$, thus giving the desired guarantee. To calculate the maximum in (2.17), we use that the derivative of $y \mapsto \frac{h_y(\kappa, \kappa_0)}{1 - y}$ on the interval $[\kappa_0, \kappa]$ can be bounded by a constant. Hence, $\max_{y \in [\kappa_0, \kappa]} \frac{h_y(\kappa, \kappa_0)}{1 - y}$ can be approximated up to a minor error by evaluating the function for each y in a sufficiently fine discretization of the interval $[\kappa_0, \kappa]$.

Finally, note that the choice of δ can be derandomized by only trying the instance-specific values in the set $\{y_v^* : v \in V\}$ obtained from the optimal LP solution (x^*, y^*) that is used. Indeed, if $\delta \notin \{y_v^* : v \in V\}$, the bound on the expected cost of the cycle given in Lemma 25 improves (as long as $\kappa \geq 1/2$) by using the minimal δ' in $\{y_v^* : v \in V, y_v^* \geq \delta\}$, whereas the bound on the expected penalty cost does not change. $\square$

Remark 27. *Computational experiments based on discretizing a distribution over pairs (δ, κ) in the same manner as carried out in Remark 13 suggest that an analysis following the one in the above proof cannot achieve an approximation ratio of 1.59. We emphasize that this does not exclude that the actual approximation guarantee of Algorithm 4 is below 1.59.*

2.5 From splitting off to tree decompositions

In this section, we prove [Lemmas 5, 7](#) and [19](#). All these lemmas share a common technique in the background, namely *splitting off*, hence we start by recalling the concept. Splitting off was initially introduced as a fundamental tool in graph theory [[Lov76](#); [Mad78](#); [Fra92](#)] for modifying a graph while maintaining certain connectivity properties. We exploit a weighted version of splitting off. Here, given a complete graph $G = (V, E)$ and edge weights $w: E \rightarrow \mathbb{R}_{\geq 0}$, a splitting operation at a vertex $v \in V$ is the following: For two edges $e = \{v, u\}$ and $f = \{v, w\}$ incident to v and $\delta \in (0, \min\{w(e), w(f)\}]$, reduce the edge weights of e and f by δ , and increase the edge weight of $\{u, w\}$ by δ . In case $e = f = \{v, u\}$, the weight of e is reduced by δ , and we call the splitting operation *degenerate*. A splitting (e, f, δ) at v is called *feasible* if minimum s - t cut sizes in G with respect to the weights w are preserved under the splitting for all $s, t \in V \setminus \{v\}$. A *complete splitting* at $v \in V$ denotes a sequence of feasible splitting operations at v such that all edges incident to v have weight zero after performing the splittings. Frank [[Fra92](#)] showed that such a complete splitting always exists. It is straightforward to see that a complete splitting consisting of less than $|V|^2$ splitting operations can be found through a polynomial number of minimum s - t cut computations, implying the following theorem.

Theorem 28 (Splitting off). *Let $G = (V, E)$ be a complete graph with edge weights $w: E \rightarrow \mathbb{R}_{\geq 0}$. Let $v \in V$ and $\beta \in [0, w(\delta(v))]$. There is a deterministic algorithm that computes in polynomial time a sequence of less than $\text{poly}(|V|)$ many feasible splitting operations at v such that the edges incident to v have total weight β with respect to the resulting weight function. Note that if $\beta = 0$, this results in a complete splitting at v .*

Note that [Lemma 22](#) is a direct consequence of [Theorem 28](#). Indeed, by the triangle inequality, splitting off does not increase the cost of the solution to the [PCTSP LP relaxation](#).

We remark that it is known that already a linear number of splitting operations are sufficient to obtain a complete splitting. For further details, we refer to [[NI97](#); [Nag06](#)] and references therein. Moreover, results of the above kind are typically proven for $\beta = 0$ only. The extension to $\beta > 0$ is immediate, since a sequence of splitting operations at a vertex v can simply be pruned as soon as the edges incident to v have remaining weight β . Having [Theorem 28](#) at hand, we can readily prove [Lemma 5](#).

Proof of Lemma 5. Consider the point $(\lambda x, y_\lambda)$. This point satisfies all cut constraints in the [PCTSP LP relaxation](#), but might violate some of the degree constraints. If not, we are done; hence assume the contrary and fix a vertex $s \in V$ such that the degree constraint at s is violated. We then have $\lambda x(\delta(s)) = 2\lambda y_s > 2y_{\lambda,s} = 2$. Thus, we can apply [Theorem 28](#) with $\beta = 2$ at the vertex s , resulting in a new weight function $\bar{x}$ after performing the splitting. All splitting operations are feasible, hence minimum r - v cut sizes are preserved for all $v \in V \setminus \{r, s\}$, and, in particular, equal to $\bar{x}(\delta(v)) = \lambda x(\delta(v))$ due to the initial degree constraints satisfied by x . Consequently, $(\bar{x}, y_\lambda)$ satisfies all cut constraints in the [PCTSP LP relaxation](#). By the same argument, there cannot be degenerate splitting operations. Hence, degrees of vertices other than s are invariant under the

splitting, i.e., degree constraints satisfied by λx are also satisfied by $\bar{x}$. Additionally, $\bar{x}$ satisfies the degree constraint at s by construction. Iterating this for all vertices $s \in V$ with violated degree constraints leads to a pair (x_λ, y_λ) feasible for the **PCTSP LP relaxation**. By the triangle inequality, applying a splitting off operation does not increase the total cost of the solution, hence $c^\top x_\lambda \leq \lambda c^\top x$, as desired. $\square$

Next, we show **Lemmas 7** and **19**. More precisely, we prove that the following common generalization of the two lemmas is true.

Lemma 29. *Let (x, y) be feasible for the **PCTSP LP relaxation**, and let $U \subseteq V_1$ with $V_1 := \{v \in V : y_v = 1\}$. Assume that there is an edge $e_0 \in \binom{U}{2}$ with $x_{e_0} \geq 1$. We can construct in polynomial time a set $\mathcal{T}$ of trees and weights $\mu \in [0, 1]^{\mathcal{T}}$ with the following properties:*

- (i) *The solution x is a conic combination of the trees in $\mathcal{T}$ with weights μ and the edge e_0 , i.e.,*

$$x = \sum_{T \in \mathcal{T}} \mu_T \chi^{E[T]} + \chi^{e_0} .$$

- (ii) *For every $v \in V \setminus U$,*

$$\sum_{T \in \mathcal{T} : v \in V[T]} \mu_T = y_v .$$

- (iii) *For every $T \in \mathcal{T}$, we have $|V[T] \cap U| = 2$, and we call the vertices in $V[T] \cap U$ the anchors of T .*

- (iv) *For $T \in \mathcal{T}$, let $e_T := V[T] \cap U$ denote the edge joining the anchors of T , and let $H := (U, F)$ be the multigraph with edge set $F := \{e_0\} \cup \{e_T : T \in \mathcal{T}\}$. Then*

$$z := \sum_{T \in \mathcal{T}} \mu_T \chi^{e_T} + \chi^{e_0} \in P_{HK}(H) .$$

We start by showing that indeed, **Lemma 29** is a common generalization of **Lemmas 7** and **19**.

Proof of Lemmas 7 and 19. **Lemma 7** immediately follows from **Lemma 29** by choosing $U = V_1$. In order to utilize **Lemma 29** to prove **Lemma 19**, we note that we can guarantee that **Assumption 6** is satisfied: If not, modify the given instance as described in **Section 2.2.3** and transform the trees computed in the thereby obtained auxiliary graph into trees in the original graph by contracting e_0 and deleting some edge if necessary.

Now, choose U to be the vertex set containing the endpoints of e_0 only. In this case, the graph H in **Item (iv)** consists of parallel edges only, with one of them being e_0 , and every tree in $\mathcal{T}$ corresponding to one of the other edges. Hence $r \in V[T]$ for each $T \in \mathcal{T}$. Moreover, the total edge weight of any feasible solution in the Held-Karp polytope over a two-vertex graph is 2. In z , exactly one unit is taken by the edge e_0 ; hence $\sum_{T \in \mathcal{T}} \mu_T = 1$ as desired. $\square$

It remains to prove **Lemma 29**. On a high level, the proof is organized as follows. We apply **Theorem 28** iteratively to find complete splittings at vertices $v \in V \setminus U$, in

2.5. From splitting off to tree decompositions

order of increasing connectivity y_v , until only vertices in U remain. In the resulting graph, it is trivial to find a decomposition meeting the requirements: We can simply consider each individual edge as a tree, with the corresponding weight being the edge weight. Carefully undoing the splitting off operations in a way similar to Bang-Jensen, Frank, and Jackson [BFJ95], and exploiting that we did them in increasing order of connectivity, we will show how to alter the initial set of trees so to obtain one with the desired properties for the original instance.

Proof of Lemma 29. We prove the statement by induction on $|V \setminus U|$. If $|V \setminus U| = 0$, there is nothing to be done: Creating a tree T from every single edge, with μ_T being equal to the edge weight (or equal to the edge weight minus 1 in case of e_0), clearly satisfies all the properties.

If $|V \setminus U| > 0$, consider a vertex $s \in V \setminus U$ of minimum connectivity y_s . By Theorem 28, we can efficiently compute a sequence of feasible splitting off operations resulting in a complete splitting at s . Note that the weights of minimum r - v -cuts are preserved under the splittings for all $v \in V \setminus \{r, s\}$. By the degree constraints of the PCTSP LP relaxation, this implies that the degrees of vertices different from s are preserved as well. The latter has two implications: First, none of the splitting operations at s can be degenerate, as degenerate splittings would reduce the degree of some vertex other than s . Second, the graph on $V' = V \setminus \{s\}$ satisfies the assumptions of Lemma 29 with respect to the resulting edge weights. Consequently, by the inductive assumption, we can compute in polynomial time a set $\mathcal{T}$ of trees with the desired properties; in particular, for every $v \in V' \setminus U$,

$$\sum_{T \in \mathcal{T}: v \in V[T]} \mu_T = y_v \ .$$

We now undo the splitting operations at s and modify the trees in $\mathcal{T}$ accordingly. We note that while intermediate steps may lead to $\mathcal{T}$ being a multiset of trees, we can merge identical trees at the very end by simply adding their weights. By Item (iv), this does not generate weights larger than 1. Before we start undoing the splitting operations, we initialize auxiliary variables $\text{spare}_v = 0$ for each $v \in V'$.

Let $(e = \{s, u\}, f = \{s, v\}, \delta)$ with $e \neq f$ be one of the splitting operations at s that we want to revert. Let $\mathcal{T}' = \{T_1, \dots, T_k\} \subseteq \mathcal{T}$ be a minimal subset of trees with $\{u, v\} \in E[T]$ for each $T \in \mathcal{T}'$ and $\sum_{T \in \mathcal{T}'} \mu_T \geq \delta$. Let $\varepsilon = \sum_{T \in \mathcal{T}'} \mu_T - \delta$. Note that $\varepsilon < \mu_{T_k}$ by the minimality of $\mathcal{T}'$. If $\varepsilon > 0$, add a copy T'_k of T_k to $\mathcal{T}$, set $\mu_{T'_k} = \varepsilon$, and reduce μ_{T_k} by ε . After this modification, $\sum_{T \in \mathcal{T}'} \mu_T = \delta$. For each $T \in \mathcal{T}'$ do the following:

- (i) If $s \notin V[T]$, remove $\{u, v\}$ from $E[T]$ and add $\{s, u\}$ and $\{s, v\}$ to $E[T]$.
- (ii) If $s \in V[T]$, remove $\{u, v\}$ from $E[T]$ and add either $\{s, u\}$ or $\{s, v\}$ to $E[T]$ such that T remains acyclic. If $\{s, u\}$ is added to $E[T]$, increase spare_v by μ_T . Otherwise, increase spare_u by μ_T .

Note that in case (i), the total weight of trees containing s increases by μ_T . Otherwise, either spare_u or spare_v increases by μ_T . Consequently, through the above operations, $\sum_{T \in \mathcal{T}: s \in V[T]} \mu_T + \sum_{v \in V'} \text{spare}_v$ increases by δ . Since the splitting off operation

(e, f, δ) decreased the degree of s by 2δ , we get, after reverting all splitting off operations at s ,

$$\sum_{T \in \mathcal{T}: s \in V[T]} \mu_T + \sum_{v \in V'} \text{spare}_v = \frac{x(\delta(s))}{2} = y_s .$$

Now, for each $w \in V'$ do the following: If $\text{spare}_w > 0$, find a minimal subset of trees $\mathcal{T}'' = \{T_1, \dots, T_\ell\} \subseteq \mathcal{T}$ with $w \in V[T]$ but $s \notin V[T]$ for each $T \in \mathcal{T}''$, and $\sum_{T \in \mathcal{T}''} \mu_T \geq \text{spare}_w$. Note that such a subset of trees always exists since

$$\sum_{T \in \mathcal{T}: w \in V[T]} \mu_T \geq y_w \geq y_s = \sum_{T \in \mathcal{T}: s \in V[T]} \mu_T + \sum_{v \in V'} \text{spare}_v ,$$

where we used in the second inequality that s has minimum connectivity. Note that the first inequality is satisfied by the inductive assumption: If $w \notin U$, the bound is tight by [Item \(ii\) of Lemma 29](#). If $w \in U$, then $\sum_{T \in \mathcal{T}: w \in V[T]} \mu_T \geq 1 = y_w$ by [Item \(iv\) of Lemma 29](#) and the degree constraints in the Held-Karp polytope (note that e_0 may be incident to w). Let $\varepsilon = \sum_{T \in \mathcal{T}''} \mu_T - \text{spare}_w$. Note that $\varepsilon < \mu_{T_\ell}$ by the minimality of $\mathcal{T}''$. If $\varepsilon > 0$, add a copy T'_ℓ of T_ℓ to $\mathcal{T}$ and set $\mu_{T'_\ell} = \varepsilon$ and reduce μ_{T_ℓ} by ε . For each $T \in \mathcal{T}''$, add $\{s, w\}$ to $E[T]$. Note that this increases the total weight of the trees containing s by spare_w . Hence, after using up all spares, we get

$$\sum_{T \in \mathcal{T}: s \in V[T]} \mu_T = y_s . \quad (2.18)$$

Note that throughout the above operations, the graphs $T \in \mathcal{T}$ are trees, and after reverting all splitting off operations, x is a conic combination of the trees in $\mathcal{T}$ with weights μ and the edge e_0 by construction. Furthermore, the above operations do not change the intersection of the trees in $\mathcal{T}$ with U , and splitting trees does not affect [Item \(iv\)](#). Hence, [Item \(iii\)](#) and [Item \(iv\)](#) still hold after the above operations. Moreover, for every vertex in V' , the total weight of trees covering this vertex is unchanged. Hence, [Item \(ii\)](#) is still satisfied for each $v \in V' \setminus U$, and also for s by (2.18).

Finally, we note that our construction can be executed in polynomial time. Indeed, by [Theorem 28](#), in each step of our inductive procedure, we have to revert less than $\text{poly}(|V|)$ many splitting operations, which increases the total number of trees in $\mathcal{T}$ by an additive $\text{poly}(|V|)$. This implies that the size of $\mathcal{T}$ remains polynomially bounded throughout. $\square$

2.5. From splitting off to tree decompositions

Chapter 3

Improved guarantees for capacitated vehicle routing

In this chapter we devise an improved approximation algorithm for capacitated vehicle routing. We improve and simplify an approach that was initiated in my master’s thesis [Bla20], which led to the first improvement of the approximation ratio of the classical tour partitioning approach. We obtain better guarantees for several major variants of the problem. As main novelty, we leverage an algorithm by Friggstad and Swamy [FS17] for *regret-bounded vehicle routing* to solve a subproblem established in [Bla20]. Moreover, we introduce the concept of weak fractional solutions to this subproblem which allows for a more streamlined analysis. We refine the algorithm and improve and simplify its analysis at several additional places. The results presented in this chapter are joint work with Vera Traub and Jens Vygen. We focus on the improvements compared to [Bla20]. A full version with detailed proofs can be found in [BTV23].

3.1 Introduction

In the *capacitated vehicle routing* problem, we are given a metric space with a depot and customers, each with a positive demand between 0 and 1. The goal is to design tours of minimum total length such that each tour contains the depot, each customer is served by some tour, and the total demand of the customers in one tour does not exceed 1 (after scaling, this is the vehicle capacity). Capacitated vehicle routing generalizes the famous traveling salesperson problem and has obvious applications in logistics. There is a huge body of literature studying heuristics, mixed-integer programming models, and application scenarios, some of which are discussed in the second part of this thesis. Formally, the problem can be stated as follows.

Capacitated Vehicle Routing (CVRP): An instance of capacitated vehicle routing consists of

- a finite set V (of customers),
- a depot s , not belonging to V ,

3.1. Introduction

- a metric c on $\{s\} \cup V$, defining distances (or cost),
- a demand $d(v) \in [0, 1]$ for each customer $v \in V$.

A feasible solution is a set $\mathcal{Q}$ of *tours* such that

- every tour $Q \in \mathcal{Q}$ is a cycle that contains s ,
- every customer belongs to exactly one tour, and
- $\sum_{v \in V[Q] \setminus \{s\}} d(v) \leq 1$ for all $Q \in \mathcal{Q}$.

The task is to find a feasible solution such that the total cost (or distance) $c(\mathcal{Q}) := \sum_{Q \in \mathcal{Q}} c(Q)$ is minimum. We write $c(Q) = \sum_{\{v,w\} \in E[Q]} c(v,w)$.

Throughout, we will denote by $\text{OPT}(\mathcal{I})$ or simply OPT the minimum cost of a feasible solution to a given instance $\mathcal{I}$ of the **CVRP**.

Indeed, this problem generalizes the **TSP** (set $d(v) = 0$ for each $v \in V$) and is hence APX-hard. Note that it is even NP-hard to approximate this problem within a factor better than $3/2$ as it generalizes *bin packing*, which is equivalent to the special case of the **CVRP** where the edge lengths are given by

$$c_e := \begin{cases} 1/2 & \text{if } e \in \delta(s) \\ 0 & \text{else} \end{cases}. \quad (3.1)$$

For a long time, the so-called iterated tour partitioning heuristic persisted to be the best known approximation algorithm: First, a traveling salesperson tour (ignoring the capacity constraint) is computed. Then, the tour is partitioned optimally into segments of total demand at most 1, each of which is then served by a separate tour from the depot. Such an optimal partition can be computed by dynamic programming. The approximation ratio of this algorithm is $\alpha + 2$, where α is the approximation ratio of an algorithm computing the traveling salesperson tour. Essentially the same algorithm has been the best known for the *unit-demand special case* (where all customers have the same demand), and also for the variant called *splittable capacitated vehicle routing*, where a customer's demand can be split and served by more than one tour. For these variants, the achieved approximation ratio is $\alpha + 1$.

These algorithms have been proposed and analyzed in the 1980s by Altinkemer and Gavish [AG87] and Haimovich and Rinnooy Kan [HR85]. Despite many efforts and progress in special cases (cf. Section 3.1.2), they have not been improved for a long time. The first improvement was achieved in [Bla20]: For the unit-demand special case and the splittable variant, an approximation guarantee of $\alpha + 1 - \varepsilon$ is obtained for some ε depending on α only. For unsplittable **CVRP**, the approximation ratio improves by 2ε . For $\alpha = 3/2$, one can choose $\varepsilon > 10^{-12}$. We show how to further refine this algorithm to get a more significant improvement of the approximation guarantee. As main novelty we propose a different strategy for solving a subproblem called *vehicle routing with target groups*. Although the first part of our overall algorithm closely follows [Bla20], we simplify and improve the analysis in several places. We benefit from partitioning the edge set and analyzing both sets separately since our improved upper bound on the cost of the computed solution to vehicle routing with target groups charges different factors on the sets of the partition (cf. Theorem 39 on page 61). Here are our main results:

Theorem 30. *For every $\alpha > 1$ there is an $\varepsilon > 0$ such that the following holds. If there is an α -approximation algorithm for the traveling salesperson problem, then there is an $(\alpha + 2 - 2\varepsilon)$ -approximation algorithm for the **CVRP**. For $\alpha = 3/2$ we have $\varepsilon > \frac{1}{3000}$.*

Theorem 31. *For every $\alpha > 1$ there is an $\varepsilon > 0$ such that the following holds. If there is an α -approximation algorithm for the traveling salesperson problem, then there is an $(\alpha + 1 - \varepsilon)$ -approximation algorithm for the unit-demand case and for splittable **CVRP**. For $\alpha = 3/2$ we have $\varepsilon > \frac{1}{3000}$.*

After our work, Friggstad, Mousavi, Rahgoshay, and Salavatipour [FMRS22] further improved the approximation guarantee for unsplittable **CVRP**: They showed that for any α -approximation algorithm for the **TSP**, and any constant $\delta > 0$, there is an $(\alpha + \ln(2) + 1 + \delta)$ -approximation algorithm for unsplittable **CVRP**. For $\alpha = 3/2$, this yields an approximation guarantee of slightly below 3.2. Their main idea is to distinguish between customers with high demand and customers with small demand. For customers with high demand, they design a configuration LP which can be solved in polynomial time (exploiting that only a constant number of customers with high demand fit into the same tour), and sample tours proportional to their value in an optimum LP solution. On the remaining set of uncovered customers, they call the classical tour partitioning algorithm. In their analysis, they exploit that there are few customers with high demand remaining, which attenuates the bin packing issue.

In their paper, they remark that calling both their algorithm and the one presented in this thesis and returning the better solution gives a slightly better approximation guarantee. Hence, for all three mentioned **CVRP** variants, our algorithm is a building block of the state-of-the-art approximation algorithm.

3.1.1 Outline

As in [Bla20], we analyze instances for which the approximation guarantees of [AG87] and [HR85] are almost tight and exploit their structure to design better solutions. We will call such instances *difficult*. The performance of the classical algorithms can be close to the worst case guarantee only if the following well-known lower bound is almost tight.

Proposition 32. *Let $\mathcal{Q}$ be a feasible solution to a given **CVRP** instance. Then,*

$$c(\mathcal{Q}) \geq \sum_{v \in V} 2d(v) \cdot c(s, v) .$$

Proof. For each $v \in V$ we obtain two s - v -paths by splitting the tour $Q \in \mathcal{Q}$ that contains v . By the triangle inequality, each of these paths has length at least $c(s, v)$, and hence $2c(s, v) \leq c(Q)$. Summation yields

$$\sum_{v \in V} 2d(v)c(s, v) \leq \sum_{Q \in \mathcal{Q}} \sum_{v \in V[Q] \setminus \{s\}} d(v)c(Q) \leq c(\mathcal{Q})$$

as desired. □

We view every tour in a solution to a **CVRP** instance as the union of two paths from the depot to the *peak* of the tour: a point farthest away from the depot. Note that if [Proposition 32](#) is almost tight, then for most tours, the two paths from the depot to the peak of the tour have small detour, i.e., they are approximately shortest paths from the depot to the peak.

We will compute an even number of paths that all start at the depot such that all customers are visited by some path. The total length of these paths will not be much more than the length of an optimum solution to the **CVRP** instance. Then we combine pairs of these paths to tours by adding an edge between their endpoints.

If there exists a set of paths with small total detour (like the one induced by an optimum solution to a difficult instance), then we can find a set of paths that is not much longer in polynomial time. We observe that this problem is closely related to regret-bounded vehicle routing, a problem that has been studied by Friggstad and Swamy [[FS14](#); [FS17](#)]. Here, one asks for a minimum number of paths serving all customers such that the detour of any path is bounded. In [Section 3.4](#), we adapt the LP based approach proposed in [[FS17](#)] to suit our problem. This yields a better performance guarantee than the combinatorial algorithm proposed in [[Bla20](#)]. As a by-product, we will also improve the approximation ratio for regret-bounded vehicle routing from 15 to 10.

Combining pairs of paths to tours can be too expensive. We need to ensure that a relatively cheap matching of the endpoints of the paths exists. Ideally, in each vertex end an even number of paths, then the matching would not cost anything. This is the case if we split each tour in an optimum solution into two paths from the depot to its peak. But of course we do not know these peaks. Therefore, we try to “guess” them similarly as in [[Bla20](#)] (cf. [Section 3.3](#)).

The tours generated this way will generally still not meet the capacity constraint, but we can simply concatenate all these tours (and shortcut) to obtain a traveling salesperson tour. Since this tour will be not much more expensive than an optimum solution to our **CVRP** instance, applying the classical tour partitioning algorithm finishes the job:

Theorem 33 ([[AG87](#); [HR85](#)]). *Given an instance (V, s, c, d) of the **CVRP** and a traveling salesperson tour Q , one can compute a feasible solution of cost at most $c(Q) + \sum_{v \in V} 4d(v)c(s, v)$ in $O(n^2)$ time, where $n = |V|$. For the unit-demand case and for splittable **CVRP** the bound improves to $c(Q) + \sum_{v \in V} 2d(v)c(s, v)$.*

We close this chapter by a short comment on the integrality gap of the natural linear programming relaxation of the **CVRP** in [Section 3.5](#).

3.1.2 Related work

Despite of a huge amount of research on vehicle routing, the best known approximation ratio for capacitated vehicle routing (as well as for the unit-demand and splittable variants) remained unbeaten in more than 30 years until the already mentioned recent improvements. Before, there had been progress on several special cases. First of all, the tour partitioning algorithms by Haimovich and Rinnooy Kan [[HR85](#)] and by Altinkemer

and Gavish [AG87] already yield a slightly better approximation guarantee if the least common denominator k of all demands is bounded (this is often called the *bounded capacity* case). Compared to $\alpha + 2$ and $\alpha + 1$, the approximation ratios reduce by $2\alpha/k$ (for general demands) and α/k (for unit demands). Bompadre, Dror and Orlin [BDO06] gained another $\Omega(1/k^3)$. Zhao and Xiao [ZX22] improved the gain to $\Theta(\sqrt{1/k})$.

There are also several results for geometric instances for the unit-demand case (in which $d(v) = 1/k$ for all $v \in V$). In the Euclidean plane a PTAS is known for constant k (Haimovich and Rinnooy Kan [HR85]), for $k = O(\log(n)/\log \log(n))$, where $n = |V|$ (Asano, Katoh, Tamaki, and Tokuyama [AKTT97]), and for $k \leq 2^{\log^{f(\varepsilon)}(n)}$, where f is some function depending on ε only (Adamszek, Czumaj, and Lingas [ACL10]). The latter uses a result by Das and Mathieu [DM15], who provided a quasi-polynomial time approximation scheme for the Euclidean plane and unbounded k . For higher dimensional Euclidean metrics, Khachay and Dubinin [KD16] found a PTAS for fixed dimension ℓ and $k = O(\log^{1/\ell}(n))$.

Better approximation ratios have also been found for graph metrics arising from graphs with a special structure. For the unit-demand case with constant k , Becker, Klein and Schild [BKS19] devised a PTAS in planar graphs, and Becker, Klein and Saulpic [BKS18] found a PTAS in graphs with bounded highway dimension. Becker [Bec18] designed a $4/3$ -approximation algorithm for splittable CVRP in tree metrics, improving on results by Hamaguchi and Katoh [HK98] and Asano, Kawashima and Katoh [AKK01]. Very recently, Mathieu and Zhou [MZ23a] devised a PTAS for this case. In another work, they provided a $(3/2 + \varepsilon)$ -approximation for any constant $\varepsilon > 0$ for unsplittable CVRP in tree metrics [MZ23b], which is essentially best possible. (Note that even when restricted to tree metrics, the CVRP generalizes bin packing, cf. (3.1).) Their algorithm improves upon a 2-approximation algorithm by Labbé, Laporte, and Mercure [LLM91] and upon a $5/3$ -approximation algorithm for the special case where the tree is a path (i.e., on the line) by Wu and Lu [WL20]. Note that the unit-demand case is polynomially solvable on the line.

One part of the proof of our improved approximation guarantee is leveraging an LP relaxation that was proposed by Friggstad and Swamy [FS17] for regret-bounded vehicle routing. In (additive) regret-bounded vehicle routing, the goal is to find a minimum number of paths starting at the depot and covering all customers such that none of the paths has a detour more than a given bound. Here, the detour (or regret) of a path from s to t is its length minus $c(s, t)$. Friggstad and Swamy [FS14] provided the first constant-factor approximation algorithm for this problem. They improved the approximation ratio from 31 to 15 in [FS17].

Regret-bounded vehicle routing is a special case of the *school bus problem*. In the school bus problem, there is the additional constraint that no path is allowed to contain more customers than a given bound (the vehicle capacity). Bock, Grant, Könemann and Sanità [BGKS11] observed that any μ -approximation algorithm for regret-bounded vehicle routing implies a $(\mu + 1)$ -approximation algorithm for the school bus problem. They provided a 3-approximation algorithm for regret-bounded vehicle routing on trees and thus obtained a 4-approximation algorithm for the school bus problem on trees. The later results by Friggstad and Swamy mentioned above imply a 16-approximation

algorithm for the general school bus problem. As a by-product of our work, we will improve on this in [Section 3.4.4](#).

3.2 Overview of our algorithm

In this section we provide a more detailed outline of our approach and introduce some important definitions.

Our algorithm will compute two solutions to the [CVRP](#) instance and return the better of the two. The first solution results from the classical iterated tour partitioning heuristic [[AG87](#); [HR85](#)]. The second algorithm yields a better approximation ratio on instances where [Proposition 32](#) and thus the analysis of the classical algorithms is almost tight. We will call such instances *difficult*. In order to give a formal definition of difficult instances we fix a constant $0 < \varepsilon < 1$.

Definition 34. *An instance (V, s, c, d) of the [CVRP](#) is called difficult if*

$$2 \cdot \sum_{v \in V} d(v) \cdot c(s, v) > (1 - \varepsilon) \cdot \text{OPT} .$$

If an instance of the [CVRP](#) is not difficult, [Theorem 33](#) says that the iterated tour partitioning heuristic computes a solution of cost at most $(\alpha + 2 \cdot (1 - \varepsilon)) \cdot \text{OPT}$, and at most $(\alpha + 1 - \varepsilon) \cdot \text{OPT}$ for the unit-demand and splittable variants. For difficult instances we will compute a traveling salesperson tour that is shorter than $\alpha \cdot \text{OPT}$; then the better approximation ratios follow from calling [Theorem 33](#).

Informally speaking, in an optimum solution to a difficult instance almost every tour must have the following two properties:

- (i) The tour Q is not much longer than $2 \cdot c(s, \text{peak}(Q))$, where $\text{peak}(Q)$ is a vertex in Q that is farthest away from the depot s .
- (ii) The demand served by the tour is almost 1, and almost all of it is close to the peak.

More precisely, the total length of the tours that do not have the above properties is very small compared to OPT .

Property (ii) implies that most customers can be partitioned into clusters with demand approximately 1. The first step of our algorithm aims at identifying these clusters. Since the clusters around the peaks of different tours can be close to each other and might be difficult to distinguish, we merge such clusters into larger ones. A more detailed description of our clustering algorithm will be given in [Section 3.3](#).

Having identified clusters, we now want to find paths starting at the depot and ending in the clusters, such that all customers are visited by one such path, regardless of whether the customer is part of a cluster or not. To find such paths we compute a solution to an instance of [VEHICLE ROUTING WITH TARGET GROUPS](#), a problem established in [[Bla20](#)]. The *targets* will be some customers inside the clusters, where targets in the same cluster belong to the same *target group*.

Vehicle Routing with Target Groups: An instance of VEHICLE ROUTING WITH TARGET GROUPS consists of

- disjoint finite sets V (of *customers*) and $\bar{T}$ (of *targets*),
- a *depot* s , not belonging to $V \cup \bar{T}$,
- a metric c on $\{s\} \cup V \cup \bar{T}$,
- a partition $\mathcal{T}$ of the target set $\bar{T}$ into *target groups*, and
- $b : \mathcal{T} \rightarrow \mathbb{Z}_{>0}$ specifying how many tours must end in each target group.

A feasible solution is a set $\mathcal{P}$ of *tours* such that

- every tour $P \in \mathcal{P}$ is either
 - an s - t -path for some target $t \in \bar{T}$ such that all internal vertices of P belong to V , or
 - a cycle containing s such that all other vertices belong to V ,
- every element of V belongs to at least one of these tours, and
- for every target group $T \in \mathcal{T}$, exactly $b(T)$ of these tours end in an element of T .

The task is to find a feasible solution whose total cost $c(\mathcal{P}) := \sum_{P \in \mathcal{P}} c(P)$ is minimum, where we again write $c(P) = \sum_{\{v,w\} \in E[P]} c(v,w)$.

Again, we write $\text{OPT}(\mathcal{J})$ for the cost of an optimum solution to instance $\mathcal{J}$.

For a target group T , we will set the number $b(T)$ to roughly twice the demand of the cluster containing T . The number $b(T)$ of paths ending in T will always be even. Therefore, we can turn a solution to our constructed instance of VEHICLE ROUTING WITH TARGET GROUPS into a solution of the CVRP instance as follows. For each target group T we pair up the paths and complete a pair of paths to a tour by adding an edge between their endpoints. Computing a pairing that minimizes the cost of the added edges is a minimum-weight perfect matching problem. In the end we concatenate all these tours and shortcut to obtain a TSP tour, to which we apply Theorem 33.

A key part of our proof is to show that we can compute a cheap solution to our instance of VEHICLE ROUTING WITH TARGET GROUPS. First, we want to show that if our CVRP instance $\mathcal{J}$ is difficult, the instance of VEHICLE ROUTING WITH TARGET GROUPS that we construct has a solution that

- (a) is not much more expensive than the cost $\text{OPT}(\mathcal{J})$ of an optimum solution to our CVRP instance, and
- (b) has small total *detour*, i.e. almost all tours are approximately shortest paths from the depot to the target they end in.

It will be convenient to consider a solution to an instance of VEHICLE ROUTING WITH TARGET GROUPS as a set of walks in a digraph.

Definition 35. For an instance $\mathcal{J} = (V, \bar{T}, s, c, \mathcal{T}, b)$ of VEHICLE ROUTING WITH TARGET GROUPS let $G(\mathcal{J})$ be the digraph with vertex set $V[\mathcal{J}] = \{s\} \cup V \cup \bar{T}$ and edge set $E[\mathcal{J}] = E_1[\mathcal{J}] \cup E_2[\mathcal{J}]$, where

$$\begin{aligned} E_1[\mathcal{J}] &= \{(v, w) : v \in \{s\} \cup V, w \in V, v \neq w\} , \\ E_2[\mathcal{J}] &= \{(v, w) : v \in \{s\} \cup V, w \in \{s\} \cup \bar{T}, v \neq w\} . \end{aligned}$$

3.2. Overview of our algorithm

So in a feasible solution, the final edge of every tour belongs to $E_2[\mathcal{J}]$, and all other edges belong to $E_1[\mathcal{J}]$. Distinguishing between $E_1[\mathcal{J}]$ and $E_2[\mathcal{J}]$ will improve the analysis of our algorithm because our upper bound on the cost of the computed solution to $\mathcal{J}$ charges different factors on $E_1[\mathcal{J}]$ and $E_2[\mathcal{J}]$ (cf. [Theorem 39](#) on page 61).

Definition 36 (walk solution). *Given an instance $\mathcal{J} = (V, \bar{T}, s, c, \mathcal{T}, b)$ of [VEHICLE ROUTING WITH TARGET GROUPS](#), a walk solution is a multi-subset H of $E[\mathcal{J}]$ such that*

- $|\delta_H^+(v)| = |\delta_H^-(v)|$ for all $v \in V$,
- $|\delta_H^-(T)| = b(T)$ for each $T \in \mathcal{T}$, and
- H contains an s - v -path for each $v \in V$.

Here $\delta_H^+(v)$ and $\delta_H^-(v)$ denote the multi-sets of arcs in H leaving and entering v , respectively.

The cost of a walk solution H is $c(H) := \sum_{(v,w) \in H} c(v, w)$, where the sum counts multiplicities. Note that [VEHICLE ROUTING WITH TARGET GROUPS](#) is equivalent to computing a cheapest walk solution since the edge lengths c satisfy the triangle inequality.

Definition 37 (detour). *For an arc $e = (v, w) \in E[\mathcal{J}]$ we define*

$$\text{detour}(e) := \text{detour}(v, w) := c(v, w) + c(s, v) - c(s, w) .$$

For a walk P in $G(\mathcal{J})$ that visits the vertices $v_0, v_1, \dots, v_k$ in this order we write $\text{detour}(P) := \sum_{i=1}^k \text{detour}(v_{i-1}, v_i)$.

For a walk P that starts at the depot s and ends in t we have $\text{detour}(P) = c(P) - c(s, t)$. The detour has been called *excess* by [BCKLMM07](#) and the *regret metric* by [FS14](#). The detour is clearly not symmetric, but it is nonnegative and fulfills the triangle inequality.

Having constructed an instance $\mathcal{J}$ of [VEHICLE ROUTING WITH TARGET GROUPS](#) from a given difficult instance $\mathcal{J}$ of the [CVRP](#), we want to show that $\mathcal{J}$ has a cheap solution (costing not much more than $\text{OPT}(\mathcal{J})$) with small total detour. However, this is not exactly the statement we will show. It will simplify our proofs (compared to [Bla20](#)) to consider what we call *weak fractional solutions* of $\mathcal{J}$ instead of actual solutions. First, we allow taking only fractions of walks. Second, we reduce the amount that must arrive in each target group slightly. The latter is because we cannot assume that the total demand of a tour near its peak is 1 in an optimum solution to $\mathcal{J}$, but only $1 - \tau$ for some constant $\tau \in (0, 1)$ that depends on ε and that we will choose later.

Definition 38 (weak fractional solution). *A weak fractional solution to an instance $\mathcal{J} = (V, \bar{T}, s, c, \mathcal{T}, b)$ of [VEHICLE ROUTING WITH TARGET GROUPS](#) is a vector $x \in \mathbb{R}_{\geq 0}^{E[\mathcal{J}]}$ such that*

$$x = \sum_{P \in \mathcal{P}} \lambda_P \cdot \chi^{E[P]} ,$$

where $\mathcal{P}$ is a set of walks in $G(\mathcal{J})$ and $\lambda_P \in \mathbb{R}_{\geq 0}$ for all $P \in \mathcal{P}$ such that

- every walk $P \in \mathcal{P}$ begins in s , ends in $\{s\} \cup \bar{T}$, and all inner vertices belong to V ,
- for every $v \in V$, we have $\sum_{P \in \mathcal{P}: v \in V[P]} \lambda_P \geq 1$, and
- for every target group $T \in \mathcal{T}$, the total weight of the walks ending in T equals $(1 - \tau) \cdot b(T)$, i.e.,

$$\sum_{P \in \mathcal{P}: P \text{ ends in } T} \lambda_P = (1 - \tau) \cdot b(T) .$$

Here $\chi^{E[P]}$ denotes the incidence vector of $E[P]$. For any vector $x \in \mathbb{R}_{\geq 0}^{E[\mathcal{J}]}$ we write $c(x) := \sum_{(v,w) \in E[\mathcal{J}]} c(v,w)x_{(v,w)}$ and $\text{detour}(x) := \sum_{(v,w) \in E[\mathcal{J}]} \text{detour}(v,w)x_{(v,w)}$.

In Section 3.3, we show that our constructed instance $\mathcal{J}$ of **VEHICLE ROUTING WITH TARGET GROUPS** has a weak fractional solution that has small detour and is not much more expensive than $\text{OPT}(\mathcal{J})$.

While **VEHICLE ROUTING WITH TARGET GROUPS** in general is at least as hard as the traveling salesperson problem, we can compute solutions that are not much longer than the best weak fractional solution with small detour. More precisely, if there exists a weak fractional solution x with small detour, then we can compute a solution to **VEHICLE ROUTING WITH TARGET GROUPS** that is not much more expensive than x . This is formally stated in the following theorem. For sufficiently small detour, we can choose $\theta = 1 - \tau$. Then the factor on the cost of the given weak fractional solution is close to 1 because we will choose the constant $\tau \in (0, 1)$ to be close to 0.

Theorem 39. *Let $\tau \in (0, 1)$. There is a polynomial-time algorithm for **VEHICLE ROUTING WITH TARGET GROUPS** that computes for every instance $\mathcal{J}$ and any given $\theta \in (0, 1 - \tau]$ a feasible solution $\mathcal{P}$ to $\mathcal{J}$ such that*

$$c(\mathcal{P}) < \frac{1}{\theta} \cdot c(x) + \frac{1 - \tau - \theta}{\theta \cdot (1 - \tau)} \cdot c(x|_{E_1[\mathcal{J}]}) + \frac{3}{1 - \theta} \cdot \text{detour}(x|_{E_1[\mathcal{J}]}) ,$$

for every weak fractional solution x of $\mathcal{J}$.

In contrast to [Bla20], we only charge some factor to the detour of edges in $E_1[\mathcal{J}]$, which will improve the final approximation guarantee.

We solve **VEHICLE ROUTING WITH TARGET GROUPS** by leveraging a sophisticated LP relaxation for regret-bounded vehicle routing due to Friggstad and Swamy [FS17] (Section 3.4). We compute a cheap forest and a network flow to obtain tours that visit not necessarily every customer, but every connected component of the forest. In the network flow problem we ensure that the number of tours ending in each target group meets the requirements. Doubling the edges of the forest yields the desired walk solution of **VEHICLE ROUTING WITH TARGET GROUPS**. In contrast to the combinatorial approach proposed in [Bla20], the network flow only uses edges moving away from the depot, i.e. edges (v, w) with $c(s, w) > c(s, v)$, (and some edges entering s). Both the forest and the network flow are obtained from an optimum LP solution. However, setting up an LP relaxation that allows to round fractional solutions to an integral forest

and an (almost) acyclic network flow that visits all connected components of the forest turns out to be tricky. To achieve this, the LP has variables corresponding to a network flow in some auxiliary digraph. In this digraph we have $O(n^2)$ many vertices for each customer $v \in V$, which correspond to different possibilities for the forest component containing v . We combine the rounding approach by [FS17] with a new construction of a fractional solution, which will also enable us to obtain a better approximation ratio for regret-bounded vehicle routing and the school bus problem.

3.3 Reduction to vehicle routing with target groups

In this section we describe an algorithm to construct an instance $\mathcal{J}$ of **VEHICLE ROUTING WITH TARGET GROUPS** from a given **CVRP** instance $\mathcal{J}$. Except for some minor improvements, this algorithm is similar to the one proposed in [Bla20]. In this section we give a high-level description and focus on the simplified and improved analysis. Please refer to [BTV23] for detailed proofs. The resulting instance is the one to which we apply [Theorem 39](#) to obtain our main result. Intuitively, our algorithm tries to “guess” the peaks of the tours of an optimum solution to our **CVRP** instance $\mathcal{J}$; these should become target vertices and in each target t two tours should end, i.e. $b(\{t\}) = 2$.

Given a tour, we call a small area around the peak of this tour its *peak cluster*. The size of the peak cluster depends on ε only (the smaller ε , the smaller the peak cluster). We call a peak cluster *large* if the total demand served by the tour to customers in the peak cluster is larger than $1 - \tau$. With foresight, we choose $\tau \in (0, 1)$ such that τ and ε/τ tend to zero as ε goes to zero.

Our algorithm will not always guess the peaks of the optimum tours correctly. However, for every tour with a large peak cluster, our algorithm will always guess a target vertex that is not far away from its peak by identifying areas with large total demand. More precisely, we traverse the customers in order of non-increasing distance to s , and declare a customer $v \in V$ as target (by adding it to $\bar{T}$) if the cluster C_v around v (of the same size as the peak clusters, cf. [Figure 3.1](#)) contains customers with total demand at least $1 - \tau$. We only count the demand of customers that are not already contained in $\bigcup_{t \in \bar{T}} C_t$, which avoids counting demands twice. If v itself is already contained in $\bigcup_{t \in \bar{T}} C_t$, we do not choose v as target.

For every target vertex $t \in \bar{T}$ chosen this way, we then consider an area B_t around t that is chosen smallest possible to guarantee that each large peak cluster of an optimum solution is fully contained in one of these areas. Note that our algorithm might also guess targets that are not close to any of the peaks of optimum tours, but for difficult instances this happens rarely.

We want to set the numbers b in the **VEHICLE ROUTING WITH TARGET GROUPS** instance large enough so that for every tour with a large peak cluster, two paths are allowed to end in a target close to the peak. Therefore, the number of paths ending in the targets will depend on the total demand in the area B_t around t . In order to avoid requiring a too high number of paths, we want to avoid that the demand $d(v)$ of a customer v is counted twice when v is contained in two of the areas B_t . Therefore, if

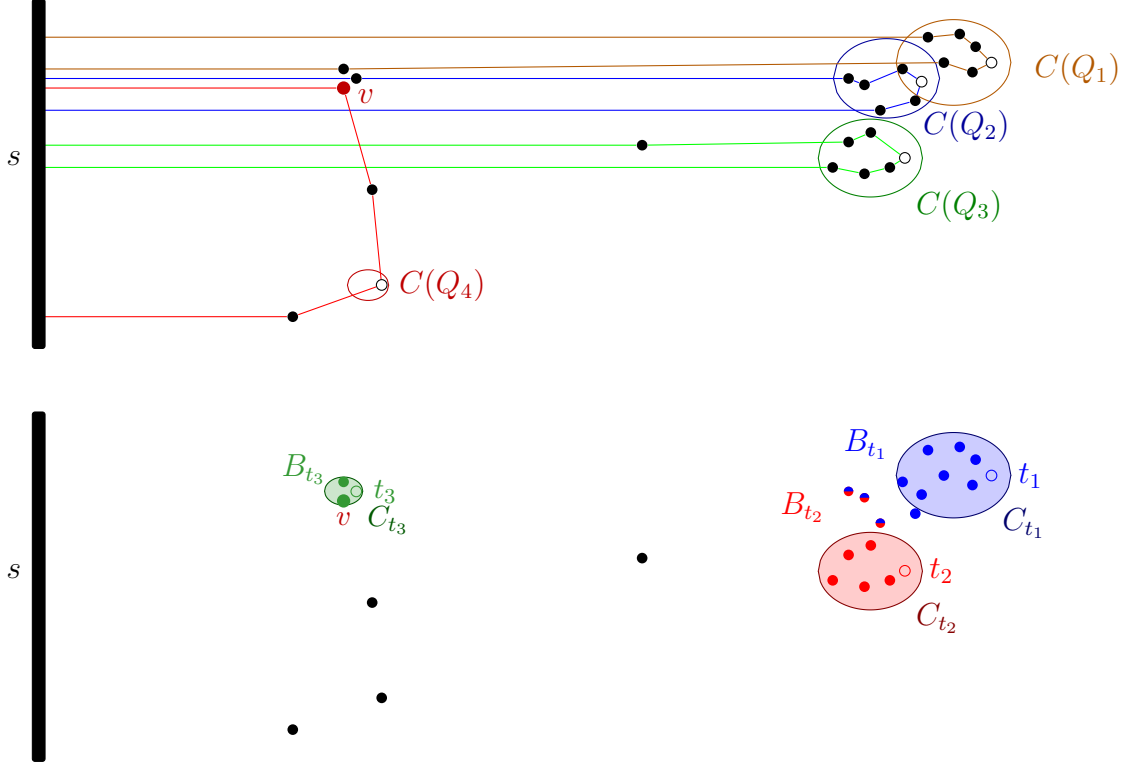


Figure 3.1: The upper figure shows an instance $\mathcal{J} = (V, s, c, d)$ of the **CVRP** with a solution $\mathcal{Q}$. The customer v (shown in red) has demand $\frac{4}{7}$ and all other customers have demand $\frac{1}{7}$. The peak clusters are illustrated for Euclidean distances. The peaks of the tours are shown as empty circles. Let $\tau = \frac{1}{6}$. Then the peak clusters $C(Q_1)$, $C(Q_2)$ and $C(Q_3)$ are large. The lower figure shows the constructed instance $\mathcal{J} = (V, \bar{T}, s, c, \mathcal{T}, b)$ of **VEHICLE ROUTING WITH TARGET GROUPS**. Our algorithm guessed $\bar{T} = \{t_1, t_2, t_3\}$ as targets, shown as empty circles. The customers are colored by their membership to the sets B_{t_1} , B_{t_2} and B_{t_3} . Note that some customers belong to B_{t_1} and B_{t_2} (colored in blue and red). Hence, the targets $\bar{T}$ are partitioned into target groups $\mathcal{T} = \{\{t_1, t_2\}, \{t_3\}\}$. A solution of $\mathcal{J}$ consists of $b(\{t_1, t_2\}) = 6$ tours ending in $\{t_1, t_2\}$ and $b(\{t_3\}) = 2$ tours ending in t_3 . It does not harm that t_3 was selected as a target although it does not belong to any peak cluster since this happens rarely. Note that peak(Q_2) was not identified as a target even though Q_2 has a large peak cluster. However, the peak cluster of Q_2 lies completely in $B_{t_1} \cup B_{t_2}$, which will guarantee that $b(\{t_1, t_2\})$ is chosen large enough so that $\mathcal{Q}$ can be easily transformed into a weak fractional solution of $\mathcal{J}$ that costs not much more than $\mathcal{Q}$.

3.3. Reduction to vehicle routing with target groups

the areas B_t for different targets t overlap, e.g. because the peaks of two tours are close to each other, we merge the areas B_t and the corresponding targets will form a target group. (If there are no overlaps, each target forms its own target group.) See Figure 3.1 for an illustration of the algorithm.

Using that in a difficult **CVRP** instance the total detour of all tours in an optimum solution is small, it can be shown that the constructed instance $\mathcal{J}$ of **VEHICLE ROUTING WITH TARGET GROUPS** admits a cheap weak fractional solution x with

$$\text{detour}(x|_{E_1[\mathcal{J}]}) \leq \varepsilon \cdot \text{OPT}(\mathcal{J}) , \quad (3.2)$$

and

$$c(x) < (1 + \zeta) \cdot \text{OPT}(\mathcal{J}) , \quad (3.3)$$

where ζ depends on ε with $\lim_{\varepsilon \rightarrow 0} \zeta = 0$.

To this end, we start with an optimum solution $\mathcal{Q}$ to our **CVRP** instance $\mathcal{J}$. Then, for every customer v that is contained in a tour $Q \in \mathcal{Q}$, we partition Q into two paths starting at the depot s and ending at v . We extend these paths by an edge connecting v either to a close-by target or to the depot s . The latter can be avoided in most cases, as we will argue shortly. The resulting walks P will then contribute with weight $\lambda_P = d(v)$ to the weak fractional solution that we construct. Finally, for each tour $Q \in \mathcal{Q}$, we add $1 - d(V[Q] \setminus \{s\})$ times its incidence vector (oriented to a directed cycle) to x .

Note that $c(x|_{E_1[\mathcal{J}]}) \leq \text{OPT}(\mathcal{J})$ by construction. Moreover, it can be shown that $c(x|_{E_2[\mathcal{J}]})$ is small compared to $\text{OPT}(\mathcal{J})$: First of all, $c(x|_{E_2[\mathcal{J}] \cap \delta^-(s)})$ is small compared to $\text{OPT}(\mathcal{J})$ by our choice of τ (which guarantees that most of the demand lies inside the target areas B_t) and our choice of $b : \mathcal{T} \rightarrow \mathbb{Z}_{>0}$. Second, one can show that $c(x|_{E_2[\mathcal{J}] \setminus \delta^-(s)})$ is small compared to $\text{OPT}(\mathcal{J})$ using that the size of the peak clusters goes to zero as ε goes to zero, and that the areas B_t are chosen smallest possible such that every large peak cluster is contained in at least one of these areas. This shows (3.3).

For all $v \in V$, the total value of walks ending with an edge (v, t) for some $t \in \{s\} \cup \bar{T}$ is exactly $2d(v)$. Thus

$$\text{detour}(x|_{E_1[\mathcal{J}]}) \leq c(\mathcal{Q}) - \sum_{v \in V} 2d(v)c(s, v) ,$$

implying (3.2) because the considered **CVRP** instance is difficult.

By applying Theorem 39 to the constructed instance $\mathcal{J}$ of **VEHICLE ROUTING WITH TARGET GROUPS** with $\theta = 1 - \tau$, we get a solution $\mathcal{P}$ to $\mathcal{J}$ of total length at most

$$\begin{aligned} c(\mathcal{P}) &< \frac{1}{1 - \tau} \cdot c(x) + \frac{3}{\tau} \cdot \text{detour}(x|_{E_1[\mathcal{J}]}) \\ &\leq \left(\frac{1 + \zeta}{1 - \tau} + \frac{3\varepsilon}{\tau} \right) \cdot \text{OPT}(\mathcal{J}) , \end{aligned}$$

where we used (3.2) and (3.3) in the second inequality. Note that the upper bound tends to $\text{OPT}(\mathcal{J})$ as ε goes to zero.

The areas B_t are chosen small enough such that we can transform $\mathcal{P}$ to a **TSP**-tour on $V \cup \{s\}$ at small extra cost. More precisely, for the transformation we need to perform parity correction for vertices belonging to the same target group. The areas B_t are small enough such that there is a cheap spanning tree on each target group, implying the existence of a cheap matching. Applying [Theorem 33](#) to the resulting **TSP**-tour yields a solution to our **CVRP** instance $\mathcal{J}$ of cost at most

$$(3 + f(\varepsilon)) \cdot \text{OPT}(\mathcal{J})$$

for unsplittable demands, and cost at most

$$(2 + f(\varepsilon)) \cdot \text{OPT}(\mathcal{J})$$

for splittable demands and in the unit-demand case, where $f : \mathbb{R}_{>0} \rightarrow \mathbb{R}_{>0}$ is some function with $\lim_{\varepsilon \rightarrow 0} f(\varepsilon) = 0$. Hence, for ε small enough, our algorithm outperforms the iterated tour partitioning heuristic on difficult instances. (Note that this stays true even if the approximation guarantee for the **TSP** will be further improved.) This implies [Theorem 30](#) and [Theorem 31](#), our main results. Thus, it remains to prove [Theorem 39](#).

3.4 LP-based approach to vehicle routing with target groups

In this section we prove [Theorem 39](#), using an approach of Friggstad and Swamy [\[FS17\]](#). This will lead to a worse running time but a better approximation ratio compared to the combinatorial algorithm in [\[Bla20\]](#). We remark that in our paper we also improved the performance guarantee of this combinatorial algorithm. However, since the guarantees are still inferior in comparison to the LP based approach presented here, we omit its description and refer the reader to [\[BTV23\]](#).

Large parts of this section follow closely [\[FS17\]](#), although our setting is slightly different and we improve on this work at one point (in [Section 3.4.2](#)). In particular, the linear program ([Section 3.4.1](#)) is largely identical to the one in [\[FS17\]](#), and the rounding procedure is taken from that paper ([Section 3.4.3](#)). Nevertheless, we cannot simply use a result of [\[FS17\]](#) in a black-box manner, so we give a self-contained proof here.

Friggstad and Swamy [\[FS17\]](#) devised an LP-based 15-approximation algorithm for regret-bounded vehicle routing. In this problem we have a metric space with a depot and a set of customers as well as a regret bound R and ask for a set $\mathcal{P}$ of paths such that each customer belongs to some path and each path begins at the depot, ends at some customer, and has total detour at most R . The goal is to minimize the number of paths in $\mathcal{P}$. So on one hand, the paths can end anywhere, on the other hand, the detour of each single path is bounded and we minimize the number of paths. In [Section 3.4.4](#) we show how the approximation ratio can be improved from 15 to 10.

In the following, let $\mathcal{J}$ be an instance of **VEHICLE ROUTING WITH TARGET GROUPS** and $\theta \in (0, 1 - \tau]$ as in [Theorem 39](#).

3.4.1 The linear program

Let $G(\mathcal{J})$ be the digraph defined in [Definition 35](#) on page 59.

Definition 40. An s - t -walk P in $G(\mathcal{J})$ with $t \in \bar{T} \cup \{s\}$ is called *monotone* if $c(s, u) < c(s, v)$ for every edge $(u, v) \in E[P]$ with $v \in V$. A solution $\mathcal{P}$ to $\mathcal{J}$ is *monotone* if all $P \in \mathcal{P}$ are monotone.

Suppose $\mathcal{J}$ has a feasible solution $\mathcal{P}$ with small detour. We will observe that then there exists a subset U of customers such that

- the instance $(U, \bar{T}, s, c, \mathcal{T}, b)$ has a monotone solution $\mathcal{P}_U$ that is no longer than $\mathcal{P}$, and
- there is a short forest F connecting every vertex $v \in V \setminus U$ to a vertex in U .

If we knew the solution $\mathcal{P}_U$ for $(U, \bar{T}, s, c, \mathcal{T}, b)$ and the forest F , we could obtain a cheap solution for $\mathcal{J}$ as follows. We take two copies of the edges in F and orient them in different directions. Then we add them to $\mathcal{P}_U$ to obtain a walk solution.

We will consider an LP relaxation for the problem of finding a subset $U \subseteq V$, a monotone solution $\mathcal{P}_U$, and a forest (V, F) connecting every vertex to an element of U .

Let us first explain how we can obtain $\mathcal{P}_U$ and F from a solution $\mathcal{P}$ of $\mathcal{J}$ with small detour. Of course, $\mathcal{P}$ is unknown but our construction leads to a particular choice of $\mathcal{P}_U$ and F for which we can impose certain constraints in the LP.

For every walk $P \in \mathcal{P}$, say from s to t , we color the points in the interval $[0, \max\{c(s, u) : u \in V[P]\}]$ red or blue. A point x is *blue* if there is an edge (v, w) of P such that $c(s, u) < x$ for all $u \in V[P_{[s, v]}] \setminus \{s\}$ and $c(s, u) > x$ for all $u \in V[P_{[w, t]}] \setminus \{t\}$. All other points are *red*, including $c(s, u)$ for all $u \in V[P] \setminus \{s, t\}$. The resulting red intervals form a subset of the set $\mathcal{I}$ of all closed intervals with end-points among $\{c(s, u) : u \in V[P] \setminus \{s, t\}\}$, including the trivial intervals consisting of a single point.

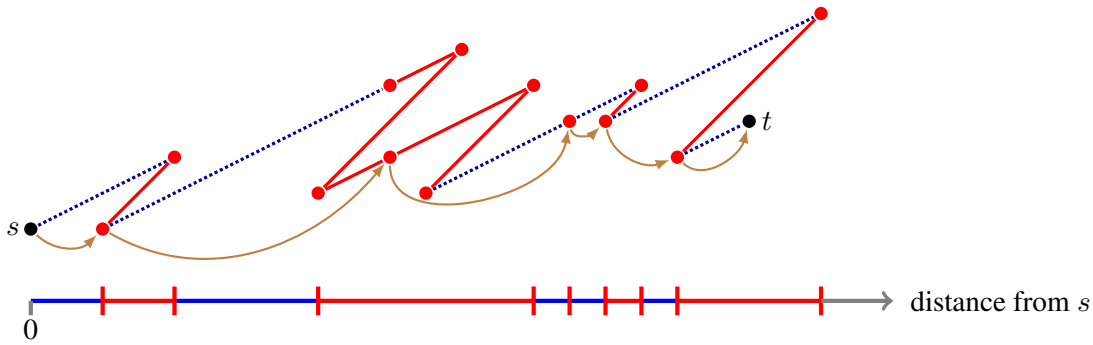


Figure 3.2: An example of a path from s to t , consisting of the dotted blue and solid red edges. The interval $[0, \max\{c(s, u) : u \in V[P]\}]$ is colored blue and red (bottom). The red edges are those of the forest F . The brown directed arcs show the monotone path visiting the sentinels of P .

Then for every vertex $v \in V[P] \setminus \{s, t\}$, the distance $c(s, v)$ is contained in one of the red intervals. We consider the forest F_P that contains those edges (v, w) of P for which $c(s, v)$ and $c(s, w)$ are contained in the same red interval and $v, w \in V[P] \setminus \{s, t\}$,

see Figure 3.2 for an example. We will bound the cost of this forest using the following observation, which is essentially Corollary 3.2 of [BCKLMM07].

Lemma 41 ([BCKLMM07]). *We have $c(F_P) \leq \frac{3}{2} \detour(E[P] \cap E_1[\mathcal{J}])$.*

Proof. Obtain P' from P by deleting the last edge and subdividing edges so that P' alternates between subpaths in blue and subpaths in red intervals. Then the total length of the blue subpaths is at least the total length of the blue intervals, and the total length of the red subpaths is at least three times the total length of the red intervals up to $c(s, v)$, where v is the last vertex of P' . Hence

$$c(s, v) \leq c(P'_{\text{blue}}) + \frac{1}{3}c(P'_{\text{red}}) = c(P') - \frac{2}{3}c(P'_{\text{red}}) \leq c(P') - \frac{2}{3}c(F_P) .$$

Since $\detour(E[P] \cap E_1[\mathcal{J}]) = c(P') - c(s, v)$, this yields the claimed inequality. $\square$

There is a one-to-one correspondence between the connected components of $(V[P] \setminus \{s, t\}, F_P)$ and the red intervals of P . For every red interval I , we choose one arbitrary vertex $v \in V[P] \setminus \{s, t\}$ with $c(s, v) \in I$. We call these vertices the *sentinels* of P . We have exactly one sentinel for each connected component of $(V[P] \setminus \{s, t\}, F_P)$. The sentinels of P are visited by P in an order of increasing distance from s . Hence, shortcutting P such that it visits only the sentinels yields a monotone path of length at most $c(P)$ (cf. Figure 3.2).

The forest F is the union of all F_P for $P \in \mathcal{P}$ and the monotone solution $\mathcal{P}_U$ arises from $\mathcal{P}$ by shortcutting $\mathcal{P}$ such that it only visits the sentinels.

Next we describe a linear programming relaxation for this forest F and monotone solution $\mathcal{P}_U$. For $u \in V$ denote $\mathcal{I}_u := \{I \in \mathcal{I} : c(s, u) \in I\}$. Recall that for every vertex $v \in V$, there is a path $P \in \mathcal{P}$ that contains v , and there is a red interval $I \in \mathcal{I}_v$ of P and a sentinel u such that $I \in \mathcal{I}_u$. We introduce variables

$$a_{v,(I,u)} \geq 0 \quad (u, v \in V, I \in \mathcal{I}_u \cap \mathcal{I}_v)$$

that model the assignment of v to the interval I with sentinel u . We remark that if v is a sentinel of interval I we have $a_{v,(I,v)} = 1$ in the corresponding LP solution. These variables satisfy the following constraints:

$$\begin{aligned} a_{v,(I,u)} &\leq a_{u,(I,u)} & (u, v \in V, I \in \mathcal{I}_u \cap \mathcal{I}_v) \\ \sum_{u \in V, I \in \mathcal{I}_u \cap \mathcal{I}_v} a_{v,(I,u)} &= 1 & (v \in V) . \end{aligned}$$

To model the forest F , we introduce variables $z_e \in \mathbb{R}_{\geq 0}$ for all $e \in \binom{V}{2}$. If an edge $e \in \binom{V}{2}$ is contained in F , we have $z_e = 1$ in the corresponding LP solution (and we have $z_e = 0$ otherwise). If $v \in V[P] \setminus \{s, t\}$ for $P \in \mathcal{P}$, then there is a red interval $I \in \mathcal{I}_v$ of P , and v must be connected to the sentinel u of I in the forest (V, F) . Hence we can introduce the constraints

$$z(\delta(S)) \geq \sum_{u \in V \setminus S, I \in \mathcal{I}_u \cap \mathcal{I}_v} a_{v,(I,u)} \quad (v \in S \subseteq V) .$$

Finally, we introduce flow constraints modelling the monotone solution $\mathcal{P}_U$. Define a digraph B with vertex set $\{(I, u) : u \in V, I \in \mathcal{I}_u\} \cup \{s\} \cup \bar{T}$ and four types of arcs:

3.4. LP-based approach to vehicle routing with target groups

- arcs $(s, (I, u))$ for $u \in V$ and $I \in \mathcal{I}_u$,
- arcs $((I, u), (J, v))$ whenever $c(s, u) < c(s, v)$ and $I \in \mathcal{I}_u$ and $J \in \mathcal{I}_v$ are disjoint,
- arcs $((I, u), t)$ for $t \in \bar{T} \cup \{s\}$, $u \in V$, and $I \in \mathcal{I}_u$, and
- arcs (s, t) for $t \in \bar{T}$.

We remark that the arc set is constructed such that $B - s$ is acyclic. The arcs in B inherit their cost from c , i.e., $c(s, (I, u)) = c(s, u)$, $c((I, u), (J, v)) = c(u, v)$, etc.

We introduce flow variables $f_e \geq 0$ for every arc e in B . For our solution $\mathcal{P}_U$, we define the corresponding flow f in B as follows.

- For an edge (s, u) in $\mathcal{P}_U$, where u is the sentinel of interval I , we set $f_{(s, (I, u))} := 1$.
- For an edge (u, u') in $\mathcal{P}_U$, where u is the sentinel of the interval I and I' is the interval of the sentinel u' we set $f_{((I, u), (I', u'))} := 1$.
- For an edge (u, t) in $\mathcal{P}_U$, where u is the sentinel of the interval I and $t \in \bar{T} \cup \{s\}$, we set $f_{((I, u), t)} := 1$.
- For an edge (s, t) in $\mathcal{P}_U$, where $t \in \bar{T}$, we set $f_{(s, t)} := 1$.

If an edge appears multiple times, the flow values add up. All other flow variables are zero. Then the flow f fulfills the following constraints:

$$\begin{aligned} f(\delta^-(T)) &= b(T) & (T \in \mathcal{T}) \\ f(\delta^-((I, u))) &= f(\delta^+((I, u))) = a_{u, (I, u)} & (u \in V, I \in \mathcal{I}_u) \end{aligned}$$

To prove [Theorem 39](#), we want to construct a feasible LP solution from a weak fractional solution to $\mathcal{J}$. To this end, we will consider the polyhedron Π below, where we replaced the flow constraints $f(\delta^-(T)) = b(T)$ by $f(\delta^-(T)) = (1 - \tau) \cdot b(T)$ and relaxed $\sum_{u \in V, I \in \mathcal{I}_u \cap \mathcal{I}_v} a_{v, (I, u)} = 1$ to $\sum_{u \in V, I \in \mathcal{I}_u \cap \mathcal{I}_v} a_{v, (I, u)} \geq 1$. We define

$$\begin{aligned} \Pi = \left\{ \right. & a, f, z : \\ & f(\delta^-(T)) = (1 - \tau) \cdot b(T) & (T \in \mathcal{T}) \\ & f(\delta^-((I, u))) = f(\delta^+((I, u))) = a_{u, (I, u)} & (u \in V, I \in \mathcal{I}_u) \\ & a_{v, (I, u)} \leq a_{u, (I, u)} & (u, v \in V, I \in \mathcal{I}_u \cap \mathcal{I}_v) \\ & \sum_{u \in V, I \in \mathcal{I}_u \cap \mathcal{I}_v} a_{v, (I, u)} \geq 1 & (v \in V) \\ & z(\delta(S)) \geq \sum_{u \in V \setminus S, I \in \mathcal{I}_u \cap \mathcal{I}_v} a_{v, (I, u)} & (v \in S \subseteq V) \\ & a, f, z \geq 0 \\ & \left. \right\}. \end{aligned} \tag{3.4}$$

Note that the number of constraints can be reduced to a polynomial number by introducing flow variables g_e^v for each $v \in V$ and $e \in \binom{V}{2}$. Require that the flow g^v sends $\sum_{I \in \mathcal{I}_u} a_{v, (I, u)}$ units from v to u and $z \geq g^v$. Then the cut constraints for z can be dropped without changing Π .

Later, in [Section 3.4.3](#), we show how we can construct a cheap solution to $\mathcal{J}$ from any element $(a, f, z) \in \Pi$ of the polyhedron Π . We write $c(z) := \sum_{e=\{v, w\} \in \binom{V}{2}} c(v, w) z_e$ and $c(f) := \sum_{e \in E[B]} c(e) f_e$. We will show:

Lemma 42. *Let $0 < \theta \leq 1 - \tau$. Given $(a, f, z) \in \Pi$ we can compute a solution $\mathcal{P}$ of the instance $\mathcal{J}$ of VEHICLE ROUTING WITH TARGET GROUPS with*

$$c(\mathcal{P}) \leq \frac{4}{1-\theta}c(z) + \frac{1}{\theta} \cdot c(f) + \frac{1-\tau-\theta}{\theta \cdot (1-\tau)} \cdot \sum_{t \in \bar{T}} \sum_{u \in V, I \in \mathcal{I}_u} c(s, u) \cdot f((I, u), t)$$

in polynomial time.

In Section 3.4.2 we show how we can obtain a vector $(a, f, z) \in \Pi$ from a weak fractional solution x of $\mathcal{J}$ and prove the following lemma. Note that we gain a factor 2 in the bound on z , compared to Lemma 41.

Lemma 43. *For any weak fractional solution x of the instance $\mathcal{J}$ of VEHICLE ROUTING WITH TARGET GROUPS there exists a vector $(a, f, z) \in \Pi$ such that*

$$\begin{aligned} c(z) &\leq \frac{3}{4} \text{detour}(x|_{E_1[\mathcal{J}]}) , \\ c(f) &\leq c(x) , \\ \sum_{t \in \bar{T}} \sum_{u \in V, I \in \mathcal{I}_u} c(s, u) \cdot f((I, u), t) &\leq c(x|_{E_1[\mathcal{J}]}) . \end{aligned}$$

From Lemma 42 and Lemma 43 we can now conclude Theorem 39.

Proof of Theorem 39. We compute an optimum solution (a, f, z) to the LP

$$\min \left\{ \frac{4}{1-\theta}c(z) + \frac{1}{\theta} \cdot c(f) + \frac{1-\tau-\theta}{\theta \cdot (1-\tau)} \cdot \sum_{t \in \bar{T}} \sum_{u \in V, I \in \mathcal{I}_u} c(s, u) \cdot f((I, u), t) : (a, f, z) \in \Pi \right\} .$$

By Lemma 43 we have

$$\begin{aligned} &\frac{4}{1-\theta}c(z) + \frac{1}{\theta} \cdot c(f) + \frac{1-\tau-\theta}{\theta \cdot (1-\tau)} \cdot \sum_{t \in \bar{T}} \sum_{u \in V, I \in \mathcal{I}_u} c(s, u) \cdot f((I, u), t) \\ &\leq \frac{3}{1-\theta} \text{detour}(x|_{E_1[\mathcal{J}]}) + \frac{1}{\theta} \cdot c(x) + \frac{1-\tau-\theta}{\theta \cdot (1-\tau)} \cdot c(x|_{E_1[\mathcal{J}]}) . \end{aligned}$$

Hence, applying Lemma 42 completes the proof. $\square$

3.4.2 Existence of a cheap LP solution (Proof of Lemma 43)

In this subsection we prove Lemma 43. Given a weak fractional solution x of $\mathcal{J}$ we will construct a vector $(a, f, z) \in \Pi$. By the definition of weak fractional solutions (Definition 38 on page 60) we can write

$$x = \sum_{P \in \mathcal{P}} \lambda_P \chi^{E[P]} ,$$

where $\mathcal{P}$ is a set of walks in $G(\mathcal{J})$, each starting in s and ending in some $t \in \bar{T} \cup \{s\}$, and $\lambda_P \geq 0$ for all $P \in \mathcal{P}$. For every walk $P \in \mathcal{P}$, we consider the forest (V, F_P) .

3.4. LP-based approach to vehicle routing with target groups

Recall that every connected component of (V, F_P) corresponds to a red interval $I \in \mathcal{I}$ of P . For every connected component Z of (V, F_P) let u_1^Z and u_2^Z be the vertex in Z visited first and visited last by $\mathcal{P}$, respectively.

For each $v \in Z$ we set

$$a_{v,(I,u_1^Z)}^P = a_{v,(I,u_2^Z)}^P = \frac{1}{2}$$

if $u_1^Z \neq u_2^Z$, and $a_{v,(I,u_1^Z)}^P = 1$ if $u_1^Z = u_2^Z$. We set $a_{v,(I,u)}^P = 0$ for all other (I, u) with $u \in V$ and $I \in \mathcal{I}_u \cap \mathcal{I}_v$. We aggregate

$$a_{v,(I,u)} = \sum_{P \in \mathcal{P}: v \in V[P]} \lambda_P \cdot a_{v,(I,u)}^P .$$

Then $a_{v,(I,u)} \leq a_{u,(I,u)}$ for all $u, v \in V$ and $I \in \mathcal{I}_u \cap \mathcal{I}_v$. Moreover, for all $v \in V$,

$$\sum_{u \in V, I \in \mathcal{I}_u \cap \mathcal{I}_v} a_{v,(I,u)} = \sum_{P \in \mathcal{P}: v \in V[P]} \lambda_P \geq 1 .$$

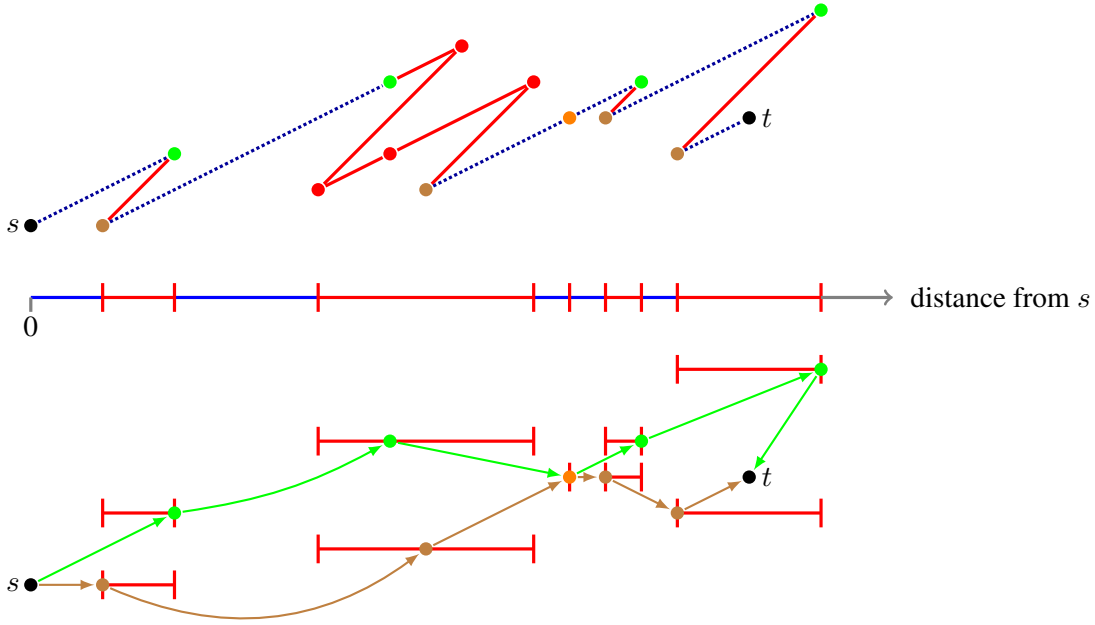


Figure 3.3: A path P (at the top) with vertices u_1^Z in green and orange and vertices u_2^Z in brown and orange. The bottom part of the picture shows the corresponding flow f in the graph B with vertices s , t and (I, u_i^Z) . Both the green and the brown edges have flow value $1/2$.

For each $P \in \mathcal{P}$ we define two monotone walks P_1 and P_2 . The walk P_i results from the s - t -walk P by shortcutting such that it visits only the vertices s , t , and u_i^Z for the connected components Z of (V, F_P) . From these two paths we obtain s - t -walks P'_1 and P'_2 in the graph B by replacing every vertex u_i^Z by (I, u_i^Z) , where I is the red interval corresponding to the component Z . See Figure 3.3. We set

$$f := \sum_{P \in \mathcal{P}} \lambda_P \cdot \frac{1}{2} \cdot \left(\chi^{E[P'_1]} + \chi^{E[P'_2]} \right) .$$

Then we have $f(\delta^-((I, u))) = f(\delta^+((I, u))) = a_{u, (I, u)}$ for all $u \in V$, $I \in \mathcal{I}_u$ and because x is a weak fractional solution of $\mathcal{J}$ we also have $f(\delta^-(T)) = (1 - \tau) \cdot b(T)$ for all $T \in \mathcal{T}$. We have $c(f) \leq c(x)$ and

$$\sum_{t \in \bar{T}} \sum_{u \in V, I \in \mathcal{I}_u} c(s, u) \cdot f((I, u), t) \leq \sum_{P \in \mathcal{P}} \lambda_P \cdot c(E[P] \cap E_1[\mathcal{J}]) \leq c(x|_{E_1[\mathcal{J}]}) .$$

Finally, we define

$$z := \frac{1}{2} \sum_{P \in \mathcal{P}} \lambda_P \cdot \chi^{F_P} .$$

Lemma 44. *We have $z(\delta(S)) \geq \sum_{u \in V \setminus S, I \in \mathcal{I}_u \cap \mathcal{I}_v} a_{v, (I, u)}$ for all $v \in S \subseteq V$.*

Proof. Let $v \in V$ and let $P \in \mathcal{P}$ be a walk containing v . Let $S \subseteq V$ with $v \in S$. We have to show $\frac{1}{2}|F_P \cap \delta(S)| \geq \sum_{u \in V \setminus S, I \in \mathcal{I}_u \cap \mathcal{I}_v} a_{v, (I, u)}^P$. Let Z be the vertex set of the connected component of (V, F_P) that contains v and let I be the red interval corresponding to Z . If Z contains only v , then the inequality holds trivially because its right-hand side is 0 in this case. Otherwise $u_1^Z \neq u_2^Z$ and $a_{v, (I, u_1^Z)}^P = a_{v, (I, u_2^Z)}^P = \frac{1}{2}$ and $a_{v, (I, u)}^P = 0$ for all other (I, u) . The connected component $(Z, F_P[Z])$ of (V, F_P) that contains v is the subpath of P from u_1^Z to u_2^Z . Therefore, the v - u_1^Z -path and the v - u_2^Z -path in (V, F_P) are edge-disjoint. This implies the desired inequality. $\square$

By Lemma 41, we have

$$\begin{aligned} c(z) &= \frac{1}{2} \cdot \sum_{P \in \mathcal{P}} \lambda_P \cdot c(F_P) \\ &\leq \frac{1}{2} \cdot \sum_{P \in \mathcal{P}} \lambda_P \cdot \frac{3}{2} \detour(E[P] \cap E_1[\mathcal{J}]) \\ &= \frac{3}{4} \detour(x|_{E_1[\mathcal{J}]}) . \end{aligned}$$

This completes the proof of Lemma 43.

3.4.3 Rounding LP solutions (Proof of Lemma 42)

Let $(a, f, z) \in \Pi$. First, we explain how we round z to obtain a forest (V, F) . For $\emptyset \neq S \subsetneq V$, let

$$r(S) := \begin{cases} 1 & \text{if } \sum_{u \in S, I \in \mathcal{I}_u \cap \mathcal{I}_v} a_{v, (I, u)} < \theta \text{ for all } v \in S \\ 0 & \text{otherwise.} \end{cases}$$

Lemma 45 ([FS17]). *We can compute a forest (V, F) with $|F \cap \delta(S)| \geq r(S)$ for all $\emptyset \neq S \subsetneq V$ and*

$$c(F) \leq \frac{2}{1 - \theta} \cdot c(z)$$

in polynomial time.

3.4. LP-based approach to vehicle routing with target groups

Proof. Let $\emptyset \neq S \subsetneq V$. Then S contains at least one vertex v and by the LP constraints

$$\begin{aligned} z(\delta(S)) &\geq \sum_{u \in V \setminus S, I \in \mathcal{I}_u \cap \mathcal{I}_v} a_{v,(I,u)} \\ &= \sum_{u \in V, I \in \mathcal{I}_u \cap \mathcal{I}_v} a_{v,(I,u)} - \sum_{u \in S, I \in \mathcal{I}_u \cap \mathcal{I}_v} a_{v,(I,u)} \\ &\geq 1 - \sum_{u \in S, I \in \mathcal{I}_u \cap \mathcal{I}_v} a_{v,(I,u)} . \end{aligned}$$

This implies $z(\delta(S)) \geq 1 - \theta$ for all S with $r(S) = 1$. In other words, $\frac{1}{1-\theta} z(\delta(S)) \geq r(S)$ for all S . Moreover, we observe that r is a downwards-monotone cut-requirement function, i.e. we have $r(X) \geq r(Y)$ for any X, Y with $\emptyset \neq X \subseteq Y$. Hence, an algorithm by Goemans and Williamson [GW94] can be used to compute a forest (V, F) as required. $\square$

Let $\mathcal{Z}$ denote the set of (vertex sets of) connected components of (V, F) . For every $Z \in \mathcal{Z}$ we have $r(Z) = 0$ and hence there is a vertex $w \in Z$ with $\sum_{u \in Z, I \in \mathcal{I}_u \cap \mathcal{I}_w} a_{w,(I,u)} \geq \theta$. We choose such a vertex w for each connected component Z and call it the *witness node* of Z . Then we define

$$\sigma(Z) := \{(I, u) : u \in Z, I \in \mathcal{I}_u \cap \mathcal{I}_w\} .$$

Note that $\sigma(Z) \subset V[B]$.

Lemma 46 ([FS17]). *For every $Z \in \mathcal{Z}$ we have $f(\delta^-(\sigma(Z))) = f(\delta^+(\sigma(Z))) \geq \theta$.*

Proof. Let w be the witness node of Z . Because $w \in I$ for every $(I, u) \in \sigma(Z)$ and because the graph B contains edges $((I, u), (I', u'))$ only if I and I' are disjoint, B contains no edge with both endpoints in $\sigma(Z)$. Hence, for the witness node w of Z the LP constraints imply

$$\begin{aligned} f(\delta^-(\sigma(Z))) &= f(\delta^+(\sigma(Z))) = \sum_{(I,u) \in \sigma(Z)} f(\delta^-((I, u))) \\ &= \sum_{(I,u) \in \sigma(Z)} a_{u,(I,u)} \geq \sum_{u \in Z, I \in \mathcal{I}_u \cap \mathcal{I}_w} a_{w,(I,u)} \geq \theta . \end{aligned}$$

$\square$

Next, we define an order $\prec$ on $\mathcal{Z}$. For $Z, Z' \in \mathcal{Z}$ with witness nodes w and w' respectively, we say that $Z \prec Z'$ if and only if $c(s, w) < c(s, w')$.

We define a directed graph D with vertex set $\{s\} \cup \mathcal{Z} \cup \bar{T}$ and the following types of arcs:

- an arc (s, Z) , for $Z \in \mathcal{Z}$
- an arc (Z, Z') for $Z, Z' \in \mathcal{Z}$ with $Z \prec Z'$,
- an arc (Z, t) for $Z \in \mathcal{Z}$ and $t \in \bar{T} \cup \{s\}$, and
- an arc (s, t) for all $t \in \bar{T}$.

We define $c(s, Z) := \min_{v \in Z} c(s, v)$, $c(Z, Z') := \min_{v \in Z, v' \in Z'} c(v, v')$, and $c(Z, t) := \min_{v \in Z} c(v, t)$.

Lemma 47. *Given the forest (V, F) and a flow g in D with*

- $g(\delta_D^-(Z)) = g(\delta_D^+(Z)) = 1$ for every $Z \in \mathcal{Z}$, and
- $g(\delta_D^-(T)) = b(T)$ for every target group $T \in \mathcal{T}$,

*we can compute a walk solution to the instance $\mathcal{J}$ of **VEHICLE ROUTING WITH TARGET GROUPS** with cost at most $2 \cdot c(F) + c(g)$ in polynomial time.*

Proof. It is well known that there is an optimum flow $\bar{g}$ with the same properties that is integral. Back in $G(\mathcal{J})$, this corresponds to a multi-set $H \subseteq E[\mathcal{J}]$ with $|\delta_H^-(Z)| = |\delta_H^+(Z)| = 1$ for every $Z \in \mathcal{Z}$ and $|\delta_H^-(T)| = b(T)$ for every target group $T \in \mathcal{T}$. Let $Z \in \mathcal{Z}$. If $\delta_H^-(Z) = \{(v, z)\}$ and $\delta_H^+(Z) = \{(z', v')\}$, we let P_Z be the path in (V, F) that connects z and z' , oriented from z to z' . For each $e = \{v, w\} \in F$ such that (v, w) belongs to P_Z for its $Z \in \mathcal{Z}$, we add (v, w) to H . For each $e = \{v, w\} \in F$ such that neither (v, w) nor (w, v) belongs to any P_Z , we add both orientations (v, w) and (w, v) to H . Then H is a walk solution of cost at most $c(\bar{g}) + 2c(F) \leq c(g) + 2c(F)$. $\square$

We will construct a flow g as in Lemma 47 with

$$c(g) \leq \frac{1}{\theta} c(f) + \frac{1 - \tau - \theta}{\theta \cdot (1 - \tau)} \cdot \sum_{t \in \bar{T}} \sum_{u \in V, I \in \mathcal{I}_u} c(s, u) \cdot f((I, u), t) . \quad (3.5)$$

Together with Lemma 45, this will complete the proof of Lemma 42.

First, we shortcut the flow f as follows. Because the graph $B - s$ is acyclic we can decompose f into s - t -walks with $t \in \bar{T} \cup \{s\}$, i.e. we write $f = \sum_{H \in \mathcal{H}} \lambda_H \cdot \chi^H$ where $\mathcal{H}$ is a set of s - t -walks with $t \in \bar{T} \cup \{s\}$ and $\lambda_H > 0$ for all $H \in \mathcal{H}$. Note that $\sum_{H \in \mathcal{H}: V[H] \cap \sigma(Z) \neq \emptyset} \lambda_H = f(\delta^-(\sigma(Z))) \geq \theta$ for all $Z \in \mathcal{Z}$ by Lemma 46.

Then we shortcut each walk $H \in \mathcal{H}$ to a walk H' by skipping vertices that do not belong to $\{s\} \cup \bar{T} \cup \bigcup_{Z \in \mathcal{Z}} \sigma(Z)$. If $\sum_{H \in \mathcal{H}: V[H] \cap \sigma(Z) \neq \emptyset} \lambda_H > \theta$ for some $Z \in \mathcal{Z}$, we will also skip $\sigma(Z)$ in some of the walks so that $\sum_{H \in \mathcal{H}: V[H'] \cap \sigma(Z) \neq \emptyset} \lambda_H = \theta$; this may require to replace a walk by two copies. Shortcutting (skipping vertices) is possible in the graph B , although B is not a complete digraph, because if B contains two edges (b_1, b_2) and (b_2, b_3) for $b_2 \neq s$, it also contains the edge (b_1, b_3) . We define a flow f' in B to be $f' = \sum_{H \in \mathcal{H}} \lambda_H \cdot \chi^{H'}$. Because of the triangle inequality, we have $c(f') \leq c(f)$.

Next we contract every set $\sigma(Z)$ for all $Z \in \mathcal{Z}$. This yields a flow g' in the graph D by identifying the vertex resulting from the contraction of $\sigma(Z)$ with the vertex $Z \in V[D]$. Here we use that for an arc $((I, u), (J, v))$ in B with $u \in Z_u \in \mathcal{Z}$ and $v \in Z_v \in \mathcal{Z}$ we have $c(s, u) < c(s, v)$ and $I \cap J = \emptyset$, implying that $d_1 < d_2$ for every $d_1 \in I$ and $d_2 \in J$. In particular, $c(s, w_u) < c(s, w_v)$ where w_u and w_v are the witness nodes of Z_u and Z_v respectively. We have $c(g') \leq c(f') \leq c(f)$.

Finally, to define g , we start with the flow $\frac{1}{\theta} g'$ in D . The transformation from f to f' ensured that $\frac{1}{\theta} g'(\delta^-(Z)) = \frac{1}{\theta} g'(\delta^+(Z)) = 1$ for every $Z \in \mathcal{Z}$. However, if $\theta < 1 - \tau$, there is too much flow arriving in the target groups. More precisely, $\frac{1}{\theta} g'(\delta^-(T)) = \frac{1-\tau}{\theta} \cdot b(T)$ for every target group $T \in \mathcal{T}$, but we need exactly $b(T)$. For every edge (Z, t)

3.4. LP-based approach to vehicle routing with target groups

with $t \in \bar{T}$ and $Z \in \mathcal{Z}$ we decrease the flow on (Z, t) from $\frac{1}{\theta}g'(Z, t)$ to $\frac{1}{1-\tau}g'(Z, t)$ and increase the flow on (Z, s) by $\frac{1}{\theta}g'(Z, t) - \frac{1}{1-\tau}g'(Z, t) = \frac{1-\tau-\theta}{\theta \cdot (1-\tau)}g'(Z, t)$. For every edge (s, t) with $t \in \bar{T}$ we decrease the flow on (s, t) from $\frac{1}{\theta}g'(Z, t)$ to $\frac{1}{1-\tau}g'(Z, t)$. The resulting flow is the desired flow g .

The cost of g is

$$\begin{aligned} c(g) &\leq \frac{1}{\theta}c(g') + \frac{1-\tau-\theta}{\theta \cdot (1-\tau)} \cdot \sum_{Z \in \mathcal{Z}} \sum_{t \in \bar{T}} c(s, Z) \cdot g'(Z, t) \\ &\leq \frac{1}{\theta}c(f) + \frac{1-\tau-\theta}{\theta \cdot (1-\tau)} \cdot \sum_{u \in V, I \in \mathcal{I}_u} \sum_{t \in \bar{T}} c(s, u) \cdot f'((I, u), t) . \end{aligned}$$

Because the graph B visits pairs (I, u) in an order of increasing $c(s, u)$, we have $c(s, u) < c(s, v)$ whenever we shortcut $((I, u), (J, v))$ and $((J, v), t)$ to an edge $((I, u), t)$ in the construction of f' from f . Hence,

$$\sum_{u \in V, I \in \mathcal{I}_u} \sum_{t \in \bar{T}} c(s, u) \cdot f'((I, u), t) \leq \sum_{u \in V, I \in \mathcal{I}_u} \sum_{t \in \bar{T}} c(s, u) \cdot f((I, u), t) ,$$

implying (3.5). This completes the proof of Lemma 42.

3.4.4 Improving the approximation ratio for regret-bounded vehicle routing

In this subsection we remark that our construction of a fractional LP solution (Section 3.4.2) also leads to an improved approximation ratio for regret-bounded vehicle routing, from 15 [FS17] to 10. (Our LP solution differs from the one constructed in [FS17] by splitting the sentinels, which gains a factor 2 on the bound on $c(z)$.) We will not need this for our main result, but it might be worth noting. Recall that an instance of (additive) regret-bounded vehicle routing consists of a metric space $(\{s\} \cup V, c)$ and a regret bound R ; the task is to compute a minimum number of paths, each beginning at s and having total detour at most R , such that every element of V belongs to one path.

Theorem 48. *There is a 10-approximation algorithm for regret-bounded vehicle routing.*

Proof. For an instance (V, s, c, R) let $\bar{T} = V$, $\mathcal{T} = \{\bar{T}\}$, $\tau = \theta = \frac{1}{2}$, and for $t \in \bar{T}$ let $b(t)$ be twice the number of paths that end in t in an optimum solution. Although we do not know $b(t)$ for any $t \in \bar{T}$, we can guess $b(\bar{T}) = 2 \cdot \text{OPT}(V, s, c, R)$ by trying all possible values $2, 4, \dots, 2|V|$. Then let Π be the polyhedron (3.4) for this $\bar{T}$, $\mathcal{T}$, τ , and $b(\bar{T})$. We set x to be the incidence vector of an optimum solution to (V, s, c, R) and apply the construction in Section 3.4.2, but with the cost of f induced by detour instead of c (like in the work by Friggstad and Swamy [FS17]). We get a vector $(a, f, z) \in \Pi$ such that $\text{detour}(f) \leq \theta \cdot b(\bar{T}) \cdot R$ and $c(z) \leq \frac{3}{4}\theta \cdot b(\bar{T}) \cdot R$. So after adding these constraints the modified polyhedron Π' is still nonempty, and we can find a vector $(a, f, z) \in \Pi'$ in polynomial time. By the construction from the proof of

Lemma 42 (where the last part simplifies to setting $g = \frac{1}{\theta}g'$ because $\theta = 1 - \tau$), we get a solution $\mathcal{P}$ with $b(\bar{T})$ paths and

$$\begin{aligned} \text{detour}(\mathcal{P}) &\leq \frac{4}{1-\theta} \cdot c(z) + \frac{1}{\theta} \cdot \text{detour}(f) \\ &\leq \left(\frac{3}{1-\theta} + \frac{1}{\theta} \right) \cdot R \cdot \text{OPT}(V, s, c, R) \\ &= 8 \cdot R \cdot \text{OPT}(V, s, c, R) . \end{aligned}$$

After splitting each path whenever it would exceed the detour bound R (cf. Lemma 2.2 in [FS14]), we get a set of paths that all have detour at most R , and the number of these paths is at most $b(\bar{T}) + 8 \cdot \text{OPT}(V, s, c, R) = 10 \cdot \text{OPT}(V, s, c, R)$. $\square$

By the simple reduction from [BGKS11], this immediately implies an 11-approximation algorithm for the school bus problem.

3.5 Comments on integrality ratios

The classical tour partitioning algorithm also yields an upper bound on the integrality ratio of the following well-known linear programming relaxation for the **CVRP**:

$$\begin{aligned} \min \quad & \sum_{v,w \in V \cup \{s\}} x_{\{v,w\}} \cdot c(v, w) \\ & x(\delta(A)) \geq 2 \cdot d(A) \quad \forall A \subseteq V \\ & x(\delta(A)) \geq 2 \quad \forall A \subseteq V, A \neq \emptyset \\ & x(\delta(v)) = 2 \quad \forall v \in V \\ & x \geq 0 . \end{aligned} \tag{3.6}$$

Let LP denote the value of this linear program (for a given instance), and let β denote the integrality ratio of the Held-Karp relaxation of the traveling salesperson problem. It is easy to see that $\sum_{v \in V} 2d(v)c(s, v) \leq \text{LP}$. By Theorem 33, this implies that the integrality ratio of (3.6) is at most $\beta + 2$. We know $\frac{4}{3} \leq \beta < \frac{3}{2} - 10^{-34}$ [GKL23].

A natural question is whether our techniques lead to an improved bound on the integrality ratio. However, this seems to be difficult even for the strengthened linear programming relaxation where we replace $x(\delta(A)) \geq 2 \cdot d(A)$ by $x(\delta(A)) \geq 2 \cdot \lceil d(A) \rceil$. Let LP' denote the value of this stronger linear program. A crucial fact in our proof was that difficult instances are “clustered”. One might hope that even all instances with $\sum_{v \in V} 2d(v)c(s, v) > (1 - \varepsilon)\text{LP}'$ are “clustered”, but this is not always the case as the example in Figure 3.4 shows.

We remark that solving the stronger linear program is NP-hard [Dia17], but it can be approximated arbitrarily well because an approximate separation oracle can be obtained by enumerating all cuts for which $x(\delta(A))$ is bounded by a constant (these can be enumerated in polynomial time by [NNI97]).

Proving stronger upper bounds on the integrality ratios remains an open question.

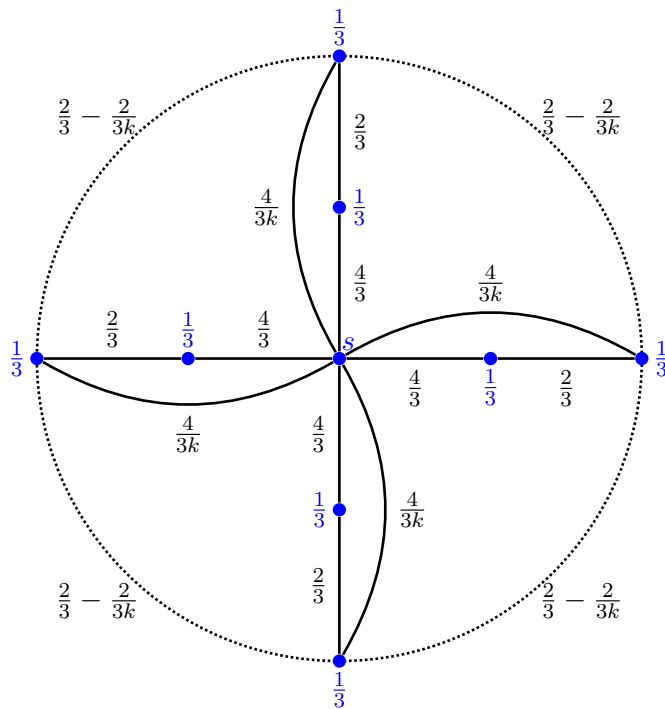


Figure 3.4: An instance of unit-demand CVRP with $|V| = 2k$ (here $k = 4$). The capacity is 3, so $d(v) = \frac{1}{3}$ for all $v \in V$. Every dotted edge has length 0, every solid straight edge has length 1, other distances are given by the metric closure of this graph. (In particular, the solid bent edges have length 2.) A feasible LP solution, even to the stronger linear program, is given by the numbers next to the edges; thus $\text{LP}' \leq 2k + \frac{8}{3}$. Note that $\sum_{v \in V} 2d(v)c(s, v) = 2k$. For large enough k , we have $\sum_{v \in V} 2d(v)c(s, v) > (1 - \varepsilon)\text{LP}'$. Nevertheless, only half of the customers are clustered.

Part II

Solving real-world vehicle routing problems: The BonnTour algorithm

Chapter 4

Introduction

In the second part of this thesis, we describe algorithms that are designed for solving real-world vehicle routing problems. These algorithms incorporate various additional constraints that typically arise in practice, such as working time restrictions and heterogeneous vehicle fleets. We implemented these algorithms and integrated them into a software package called BonnTour. BonnTour is developed in cooperation with our industry partner [Greenplan](#) and is being used successfully in various practical scenarios.¹

Most of the presented content is based on joint work with Stephan Held, Dirk Müller, Niklas Schlomberg, Vera Traub, Thorben Tröbst, and Jens Vygen [BHMSTTV22]. Luise Puhmann and our student assistants also make significant contributions to the BonnTour code.

As stated in [Chapter 1](#), our algorithm takes into account time-dependent travel time fluctuations. Although it is well known that the driving times in many cities vary a lot during the day, the majority of the vehicle routing algorithms—both academic and industrial—still work with fixed travel times. While this might often lead to acceptable results when there are not too many time windows, it inevitably leads to poor or infeasible results when many pickups or deliveries must take place in certain time windows.

We discuss theoretical foundations and practical algorithms for vehicle routing with time-dependent travel times. More precisely, we provide a comprehensive overview of basic operations on piecewise linear arrival time functions and how to evaluate insertion and deletion operations in tours efficiently. Then, we develop a local-search based heuristic to solve a wide class of vehicle routing problems. Our model incorporates various constraints arising in practice, so that we can represent the real-world scenarios of [Greenplan's](#) customers accurately in our model. Finally, we report experimental results. We compare our algorithm with state-of-the-art vehicle routing algorithms by running BonnTour on classical benchmarks. Since most of these do not have time-dependent travel times, we generate and publish new benchmark instances that are based on real-world data. This data also demonstrates the importance of considering time-dependent travel times in instances with tight time windows. Finally, we demonstrate selected features of our algorithm, and discuss real-world scenarios in which the BonnTour al-

¹As mentioned earlier, a selection of partners relying on Greenplan and the BonnTour algorithm can be found on <https://greenplan.de>.

gorithm is successfully used in a bit more detail.

4.1 Problem description

We work with a very general model, where time-dependent travel times are modeled by piecewise linear *arrival time functions*. An arrival time function a describes the arrival time $a(t)$ depending on the departure time t ; then $a(t) - t$ is the *travel time*. Our model fulfills the monotonicity property known as the *first-in-first-out principle*, which says that if we start traveling at a later time, we will not arrive earlier. This is the most commonly used model for time-dependent travel times; see, e.g., [IGP03]. For a precise description of this model, see [Section 5.1](#).

An *instance* of (a simple version of) the vehicle routing problem with pickup and delivery and time windows consists of a list of items, each associated with a pickup address, a pickup time window, a pickup duration, a delivery address, a delivery time window, and a delivery duration. Moreover, an instance contains a road network (see [Section 5.1](#)) and an address where tours start and end.

The task is to compute a set of tours such that every item appears in a tour. A *tour* consists of a tour sequence and a feasible schedule.

A *tour sequence* is a sequence $s_1, \dots, s_k$ of actions, beginning with $s_1 = \text{START}$, ending with $s_k = \text{STOP}$, where each action s_i ($1 < i < k$) is either the pickup of an item or the delivery of an item, such that every item that appears in an action of the tour is first picked up and then delivered. Each item is part of exactly two actions, the pickup at its pickup address and the delivery at its delivery address.

A *schedule* of a tour assigns a start time to every action in the tour such that starting action s_i at time t_i , performing it and then traveling to the location of s_{i+1} takes at most until t_{i+1} for $1 \leq i \leq k - 1$. A schedule is *feasible* if each action is performed in its time window.

Our model implicitly allows that vehicles can wait at a pickup or delivery address until the time window opens. In practice there are further constraints, for instance vehicle capacities, working time limits, and many more. Moreover, the total cost can depend on which vehicles we use, the duration of the tours, and more. We will comment on how to incorporate all this when describing our overall algorithm in [Chapter 6](#). The simple model is sufficient to describe our main techniques.

4.2 Outline and contributions

We have developed fast algorithms that guarantee to obey time window constraints with respect to travel times that depend on the time of the day, while considering all major constraints that are relevant in practice. While our overall approach is heuristic, it is based on solving certain well-defined subproblems optimally.

In [Chapter 5](#), we discuss key operations of local search algorithms for vehicle routing with time-dependent travel times. Some of the described algorithms are already known; we explain them to provide a comprehensive overview of the challenges when

dealing with time-dependent travel times. First, in [Section 5.1](#), we describe how we model time-dependent travel times using arrival time functions and discuss several basic operations on these functions that we need in our algorithm. In [Section 5.2](#), we show how to efficiently evaluate insertions (and removals) of actions (or sequences of actions) in tours.

In [Section 6.1](#), we explain how we combine these components to obtain our overall local search algorithm. We start by preprocessing the road network and computing a contraction hierarchy [[GS10](#); [BGSV13](#)] to allow for efficient fastest-path queries. Next, we preprocess the items: For each item, we compute an estimate for the cost this item causes and identify items that seem to fit well into the same tour. This information will serve as guidance at several places in our algorithm. Next, we construct tours using a variant of the average-regret heuristic [[FP93](#)]. During and after constructing tours, we apply various local search operations, some of which seem to be new. We leverage the “ruin-and-recreate” framework by Accorsi and Vigo [[AV21](#)] for further post-optimization and adapt it to suit our general model.

As we describe in [Section 6.2](#), our algorithm can handle numerous constraints arising in real-world scenarios, including heterogeneous fleets, hard and soft time windows, various cost models, alternative pickup and delivery locations, and many others. Thanks to our general model, our algorithm is being leveraged successfully in various real-world scenarios as mentioned above.²

We present experimental results in [Chapter 7](#). Although we have designed our algorithm to handle time-dependent travel times and many practical constraints, it yields only slightly worse results on standard benchmarks with fixed travel times than algorithms that were designed to perform best on such instances (cf. [Section 7.1](#)). For some benchmark instances with time-dependent travel times, we provide new best known solutions.

Most of the publicly available benchmarks have fixed travel times, and those with time-dependent travel times are rather unrealistic and small. Hence, since time-dependent travel times are necessary to evaluate our algorithm (and other algorithms in the future), we generate and publish new benchmarks based on real-world data in major cities. We describe how we generated these in [Section 7.2](#). Our results on these new benchmarks (cf. [Section 7.3](#)) demonstrate once again the importance of considering time-dependent travel times in vehicle routing algorithms. We obtain more reliable tours while avoiding higher costs that would result from working with constant pessimistic travel times. In [Section 7.4](#), we conduct experiments that demonstrate some features of our algorithm that are discussed in [Section 6.2](#). We run BonnTour on (modifications of) our new benchmark set, as well as on real-world instances from our cooperation with [Greenplan](#).

²A restricted version of the BonnTour API can also be found on <https://doc.greenplan.de/#/?id=main>.

4.3 Related work

All kinds of vehicle routing problems include the traveling salesperson problem and are therefore APX-hard. Approximation algorithms exist only for some restricted models (e.g., [BBCM04; CKP12; NR11] and those discussed in the first part of this thesis), and they are—as of today—not suitable for practical purposes. There is a large body of work on mixed-integer programming models. Recent works (e.g., [PSUV19]) can solve instances with several hundred customers optimally but need too much computation time for practical purposes. Hence, heuristics are mostly used in practice. In particular, many variants of local search techniques have been proposed, see, e.g., [VCGP13; AS19; AV21; QSU21; Vid22], often embedded in some “ruin-and-recreate” framework that iteratively optimizes pertubated solutions in order to escape local optima. A general overview of various approaches was given by Toth and Vigo [TV14]. Besides, the 12th implementation challenge of the Center for Discrete Mathematics and Theoretical Computer Science (DIMACS) was dedicated to the study of heuristics for basic vehicle routing problems.³

Most works considered fixed travel times between any pair of points. Of course, travel times in road networks vary substantially, in particular in cities and in rush hours. To model this, time-dependent travel times were already considered in 1992 by Malandraki and Daskin [MD92], with piecewise constant travel times. By now the most common model works with piecewise constant speeds, which leads to piecewise linear travel time functions [IGP03].

While some papers assume a given travel time function (or equivalently arrival time function) for any pair of addresses [DMCRG08; Fig12; DRVD13; PZL21a; PZL21b], others start with a road network where travel time functions are given for all edges of the graph [Man14; HZVG17; Tic17; TAFQV19; GGLP21]. So do we.

A survey on vehicle routing with time-dependent travel times was given by Gendreau, Ghiani, and Guerriero [GGG15].

Batz, Geisberger, Sanders, and Vetter [BGSV13] showed how to preprocess a road network with time-dependent travel times in order to allow relatively fast shortest path queries. A less memory-consuming variant was proposed by Strasser, Wagner, and Zeitz [SWZ21]. While a contraction hierarchy needs to be recomputed only when the road network changes, Geisberger and Sanders [GS10] proposed an additional instance-specific preprocessing to speed up fastest-path queries.

Computing the arrival time function from one address to the other based on the data from the road network requires certain operations on arrival time functions, in particular composition and pointwise minimum. The same operations are also needed when composing a tour or making local changes. While the number of breakpoints grows linearly when composing arrival time functions, this is not true for the piecewise minimum [Wie86; HS86]. The resulting large number of breakpoints suggests simplifying arrival time functions. Imai and Iri showed how to approximate a continuous piecewise linear function optimally with a guaranteed error bound in linear time [II86].

³See <http://dimacs.rutgers.edu/programs/challenge/vrp/> for the results of the implementation challenge.

Any local-search based algorithm needs operations to insert items into or remove items from a tour. Even more frequently the algorithm will evaluate the effect of such operations without actually performing them. Even the efficient evaluation whether the insertion of an item would lead to violations of time windows is nontrivial and requires the maintenance of data structures for fast updates. Some papers, including [GGLP21], precomputed the latest feasible departure time at any address. If we have a fixed start time or also precompute the earliest possible arrival time at any address, we can check feasibility with a constant number of function evaluations when inserting a sequence of constant length. However, this does not allow to bound the tour duration (unless the start time is fixed) or to compute the cost of the tour resulting from the insertion accurately. [GGLP21] used a heuristic.

The feasibility of an insert operation with respect to all time constraints can easily be checked with $O(1)$ operations on arrival time functions if we have precomputed the composed arrival time function for each *tour segment* (consecutive sequence of actions in a tour) [FGG04; HYI08]. Based on this preprocessing, [HYI08] showed how to find the cheapest way to insert an action into a tour in linear time and how to implement certain local search operations efficiently. However, a tour with n actions has $\Theta(n^2)$ tour segments, and the update after an actual insertion in the middle of the tour would require the recomputation of half of these tour segments. Visser and Spliet [VS20] proposed to store only $\Theta(n \log n)$ segments such that the information for every segment can still be computed from these with a constant number of compositions. This approach is described in more detail and generalized in Section 5.2.1.

The evaluation of the cost of a tour (after insertion or deletion of an item) also requires a scheduling step, in particular determining the optimal start time. Hashimoto, Yagiura and Ibaraki [HYI08] showed how to do this in quadratic time in the number of breakpoints for a very general model, even relaxing the first-in-first-out property of arrival time functions.

Several methods have been proposed for optimizing a single tour, including both the tour sequence and the schedule, under time window constraints and with time-dependent travel times. Branch-and-bound algorithms have been proposed in [AGGGP19; AGG20], and branch-and-price algorithms have been developed in [MMM17; LM18; HJS21]. Vu, Hewitt, Boland, and Savelsbergh [VHBS18] gave a mixed-integer linear programming formulation on a time-expanded network, which they then use within a dynamic discretization framework. Malandraki and Dial [MD96] described a dynamic programming algorithm for the time-dependent TSP without time windows (working with piecewise constant travel times). A dynamic programming algorithm for the general setting was developed by Lera-Romero, Miranda-Bront, and Soullignac [LMS20b], building on earlier work by Tilk and Irnich [TI17], who considered the TSP with time windows and constant travel times.

For the time-dependent vehicle routing problem with time windows, branch-and-price and branch-price-and-cut algorithms have been proposed in [DRVD13; LBS18; LMS20a].

Many benchmarks for different kinds of vehicle routing problems have been published. However, most benchmarks consider constant travel times (e.g., [UPPPVS17;

[AGS19; Sol87; GH99; LL03]). The dataset for capacitated vehicle routing by Uchoa et al. [UPPPVS17] contains instances with up to 1000 customers. Even more recently, Arnold, Gendreau and Sörensen [AGS19] provided very large-scale instances with up to 30,000 customers based on real-world data in Belgium. Benchmark instances for vehicle routing with time windows and 100 customers were published by Solomon [Sol87] and with up to 1000 customers by Gehring and Homberger [GH99]. These are artificial instances based on Euclidean distances and scattered time windows. Based on these benchmarks, Li and Lim [LL03] provided instances for vehicle routing with time windows and general pickup and delivery.

Due to the lack of benchmarks with time-dependent travel times, the Solomon benchmarks have been modified by adding speed profiles to the arcs [DRVD13], [Fig12], [DMCRG08], [IGP03]. This resulted in instances with time-dependent point-to-point driving times. The Euclidean distances are scaled using speed distributions. In [Fig12] and [IGP03] all arcs use the same speed distribution, while [DRVD13] and [DMCRG08] assign to each arc one out of a few different speed distributions. Using average constant speeds for the planning, [DMCRG08] and [IGP03] report time window violations and a significant cost increase when evaluating such solutions in the time-dependent model. Similar results were observed on a real-world instance in [DMCRG08].

Gmira et al. [GGLP21] and Ticha et al. [TAFQV21] added time-dependent travel times to the so-called NEWLET benchmarks [TAFQ17] for the vehicle routing problem with time windows and road network information. The road graphs are randomly generated sparse planar graphs with up to 200 vertices and randomly assigned speed profiles. The instances have up to 50 deliveries and are small enough to be tractable by a branch-cut-and-price framework. Gaps but neither (optimum) solutions, solution values, nor lower bounds have been reported.

Chapter 5

Incorporating time-dependent travel times

As mentioned in [Section 4.2](#), the purpose of this section is to provide a comprehensive overview of techniques for handling time-dependent travel times in key ingredients of local-search based algorithms for vehicle routing. We start with a formal description of the model we use to describe time-dependent travel times.

5.1 Basic operations on arrival time functions

We are given a road network with travel time information that we transform into a directed graph R with an arrival time function for every arc of R . For a vehicle traveling through the arc (v, w) , the arrival time function a describes the arrival time $a(t)$ at w depending on the departure time t at v .

Definition 49. An arrival time function (ATF) is a piecewise linear function $a : (-\infty, t_{\max}] \rightarrow \mathbb{R}$ for some $t_{\max} \in \mathbb{R}$ such that

- (i) a is continuous,
- (ii) a is non-decreasing,
- (iii) $a(t) \geq t$ for all $t \leq t_{\max}$, and
- (iv) a is initially constant, i.e., $a(t) = a(t_{\min})$ for some $t_{\min} \leq t_{\max}$ and all $t \leq t_{\min}$.

We will also define $a(t) := \infty$ for $t > t_{\max}$. We write $\text{dom}(a) := (-\infty, t_{\max}]$.

This is the standard model to describe time-dependent travel times [\[IGP03\]](#). Each condition in [Definition 49](#) can be seen to hold for reasonable models of arrival times in road networks. For condition (iv) we can simply set $t_{\min}$ to the beginning of the considered time period. Condition (iii) says that travel times are always nonnegative, and condition (ii) (also called the *first-in-first-out principle*) simply follows from the fact that we could always wait in place before traversing an arc. Condition (i) might be the most contentious as there are some discontinuous events associated with road networks (e.g. the lowering of a bridge, the arrival of a train, an accident, etc.). However, in practice, arrival times are obtained via statistical sampling, and thus they rather model the *expected* arrival times in our road network. These should always be continuous.

5.1. Basic operations on arrival time functions

On the other hand, the assumption that arrival time functions are piecewise linear is made so that we can represent ATFs via a sorted sequence of breakpoints, i.e., pairs $(t_i, a(t_i))$ for $i = 1, \dots, b$, where $b \geq 1$ and $t_1 = t_{\min}$, $t_b = t_{\max}$, and $t_i < t_{i+1}$ for $i = 1, \dots, b-1$ (then we say that a has b breakpoints and b segments). We call $(t_1, a(t_1)), \dots, (t_{b-1}, a(t_{b-1}))$ the *inner* breakpoints of a . We assume that the slope changes at every inner breakpoint (otherwise it is redundant).

Arrival time functions can also be used to model time windows and the time needed for a pickup or a delivery. For example, if a delivery takes 0.1 time units and must start between 4 and 5, the corresponding arrival time function is $a : (-\infty, 5] \rightarrow \mathbb{R}$ with $a(t) = \max\{4, t\} + 0.1$ for $t \leq 5$. Thus we assume that we have ATFs not just for every arc in our road network but also for every action.

Representing time-dependent travel times by ATFs facilitates simple and efficient implementations of several key operations. Most of these operations are rather straightforward and well-known; we will discuss them for completeness. Similar algorithms are employed in previous works on time-dependent routing problems such as [VS20].

Any vehicle routing algorithm must know how long it takes a vehicle to get from a point p to another point q . For any fixed path P from p to q in R , we can obtain this information by composing the arrival time functions of the arcs of P . Similarly, if we perform an action s (e.g., pickup or delivery) at the location q , we can obtain the ATF that models traveling along the path P and performing s by composing the ATF of P with the ATF of s . Composing two ATFs is easy:

Proposition 50. *Given arrival time functions a_1 and a_2 with b_1 and b_2 breakpoints respectively, we can compute the composition $a_2 \circ a_1$ in $O(b_1 + b_2)$ time. Moreover, $a_2 \circ a_1$ is itself an arrival time function with at most $b_1 + b_2 - 1$ breakpoints.*

Proof. The function $a_2 \circ a_1$ can be computed using a simple sweep through $\text{dom}(a_1)$. Note that in general this only gives us an $O(b_1 \cdot b_2)$ time algorithm. However, since a_1 is non-decreasing, when we sweep t through $\text{dom}(a_1)$, $a_1(t)$ sweeps through $\text{dom}(a_2)$ only once. Hence, only $O(b_1 + b_2)$ time is needed to perform the sweep. Moreover, this also shows that $a_2 \circ a_1$ has at most $(b_1 - 1) + (b_2 - 1)$ inner breakpoints since each inner breakpoint of the composition must originate from a distinct inner breakpoint of a_1 or a_2 . $\square$

For composing a sequence of ATFs we have the following.

Corollary 51. *Given arrival time functions $a_1, \dots, a_k$ such that a_i has b_i breakpoints ($i = 1, \dots, k$), we can compute $a_k \circ \dots \circ a_1$ in $O((\log k) \sum_{i=1}^k b_i)$ time. Moreover, the resulting ATF has at most $1 + \sum_{i=1}^k (b_i - 1)$ breakpoints.*

Proof. By induction on k . Using the associativity of functional composition, we can apply the induction hypothesis to compute $a_k \circ \dots \circ a_{\lceil \frac{k}{2} \rceil + 1}$ and $a_{\lceil \frac{k}{2} \rceil} \circ \dots \circ a_1$, and then apply Proposition 50. Every breakpoint of an original ATF a_i takes part in $O(\log k)$ many compositions. $\square$

For a fixed path P , we denote the composition of the ATFs of its arcs by a_P . Of course, between two given points p and q , different paths can be optimal at different

times of the day. See [Figure 7.1](#) on page 110 for an example. Thus, the optimal arrival time between p and q is modeled by the function

$$a_{p,q}(t) := \min\{a_P(t) : P \text{ is a } p\text{-}q\text{-path}\}.$$

The pointwise minimum of multiple ATFs can also be computed efficiently as we will see shortly.

However, a road network can contain exponentially many paths between two points p and q . Foschini, Hershberger and Suri [[FHS14](#)] showed that even if the arc ATFs are all affine, the function $a_{p,q}$ may have $n^{\Omega(\log n)}$ breakpoints, where n is the number of vertices of the digraph R .

Thus, an exact computation of the relevant ATFs may be practically infeasible. We preprocess the road network using contraction hierarchies (cf. [Section 6.1.1](#)) and approximate ATFs where necessary. Any such approach relies crucially on the ability to compute the pointwise minimum of arrival time functions very efficiently.

Proposition 52. *Given arrival time functions a_1 and a_2 with b_1 and b_2 breakpoints respectively, we can compute the pointwise minimum $\min\{a_1, a_2\}$ in $O(b_1 + b_2)$ time. If $\text{dom}(a_1) = \text{dom}(a_2)$, then $\min\{a_1, a_2\}$ is itself an arrival time function with at most $2(b_1 + b_2) - 3$ breakpoints.*

Proof. We can use a standard sweepline algorithm to compute the points where a_1 and a_2 intersect. Since we assume that we are given a sorted list of the breakpoints of those functions, this can easily be implemented in linear time. It is also clear that $\min\{a_1, a_2\}$ is again initially constant, non-decreasing, and satisfies $\min\{a_1(t), a_2(t)\} \geq t$. The function is continuous if $\text{dom}(a_1) = \text{dom}(a_2)$.

Inner breakpoints of $\min\{a_1, a_2\}$ can only appear at inner breakpoints of a_1 and a_2 or where a_1 and a_2 intersect. But in each interval in which both ATFs are affine they intersect at most once. Because there are at most $(b_1 - 1) + (b_2 - 1)$ inclusionwise maximal intervals in which a_1 and a_2 are affine and intersect, this concludes the proof. $\square$

Unlike the composition of two functions, taking the minimum of two ATFs may in general increase the total number of breakpoints. See [Figure 5.1](#) for an example.

However, it is known that the lower envelope (i.e., pointwise minimum) of n line segments in the plane can have at most $O(n\alpha(n))$ breakpoints, where $\alpha(n)$ is the inverse Ackermann function [[HS86](#)]. This upper bound is sharp [[Wie86](#)]. This implies that the number of breakpoints in the pointwise minimum of many arrival time functions is almost linear in the size of the input. A simple recursive approach as in [Corollary 51](#) will give a running time of $O(m\alpha(m)\log n)$ where n is the number of functions of which we are taking the minimum and m is the total number of breakpoints [[AS00](#)]. The running time can be reduced to $O(m\log n)$ as shown in [[Sch20](#); [BHMSTTV22](#)].

The operations of composing or forming the pointwise minimum of ATFs can lead to ATFs with many breakpoints. In order to speed up our algorithms, it is essential to reduce the number of breakpoints, which we will do by approximating ATFs. In order to guarantee the feasibility of our solutions, the approximated ATF should be an upper

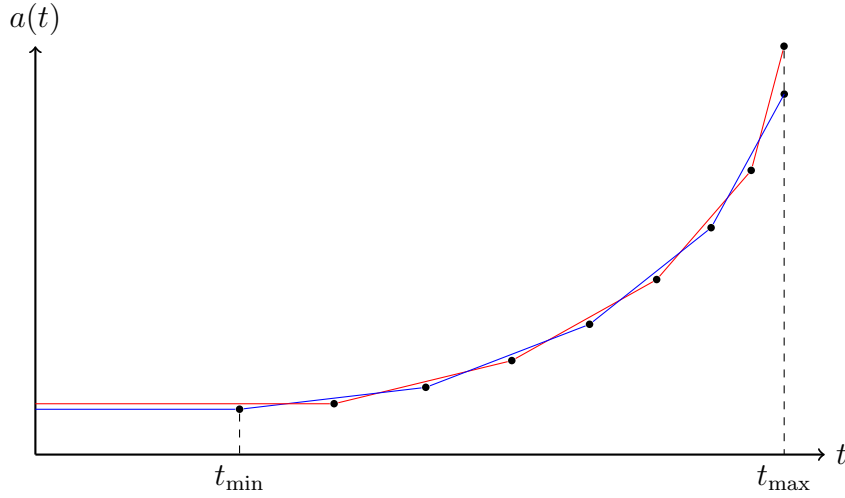


Figure 5.1: Taking the minimum of two ATFs may increase the number of inner breakpoints by a factor of 2.

bound on the original ATF. Moreover, since we have to approximate ATFs millions of times, the algorithm must be very efficient.

For these reasons we adapt the classic polygonal line simplification algorithm due to Imai and Iri [II86]. Given $\varepsilon > 0$ and a continuous, piecewise linear function f defined on a compact interval, the algorithm computes in linear time another such function g with $f \leq g \leq f + \varepsilon$ such that g has a minimum number of breakpoints. The algorithm traverses the corridor between f and $f + \varepsilon$ and computes for $i = 1, 2, \dots$ the area in which the i th breakpoint can be, as illustrated in Figure 5.2. However, the Imai–Iri algorithm is not designed to preserve monotonicity in general. Moreover, the domain of ATFs begins at $-\infty$. One can see that monotonicity can be easily restored in a linear time post-opt routine (as done for example in [BGSV13]). However, as shown in [BHMSTTV22], with an appropriate choice of the last breakpoint, the algorithm will preserve monotonicity for free. Either way, one gets a linear time algorithm for approximating an ATF with respect to a prescribed approximation error ε .

Theorem 53 ([BGSV13; BHMSTTV22]). *Given an ATF $a : [-\infty, t_{\max}] \rightarrow \mathbb{R}$ with b breakpoints and $\varepsilon > 0$, we can compute an ATF $a' : [-\infty, t_{\max}] \rightarrow \mathbb{R}$ with $a(t) \leq a'(t) \leq a(t) + \varepsilon$ for all $t \leq t_{\max}$ such that a' has the fewest possible number of breakpoints in $O(b)$ time.*

When applying this theorem to approximate ATFs, we choose ε to be a fixed percentage (e.g., 0.5 %) of the minimum travel time $\min\{a(t) - t : t \leq t_{\max}\}$.

While Theorem 53 guarantees the minimum number of breakpoints for a fixed error ε , it will often output solutions with fairly poor average error for this minimum number of breakpoints (Figure 5.2 shows an example). Therefore, we employ a simple post-processing heuristic that moves individual breakpoints or line segments optimally, keeping the remaining ATF fixed and maintaining the properties guaranteed by Theorem 53.

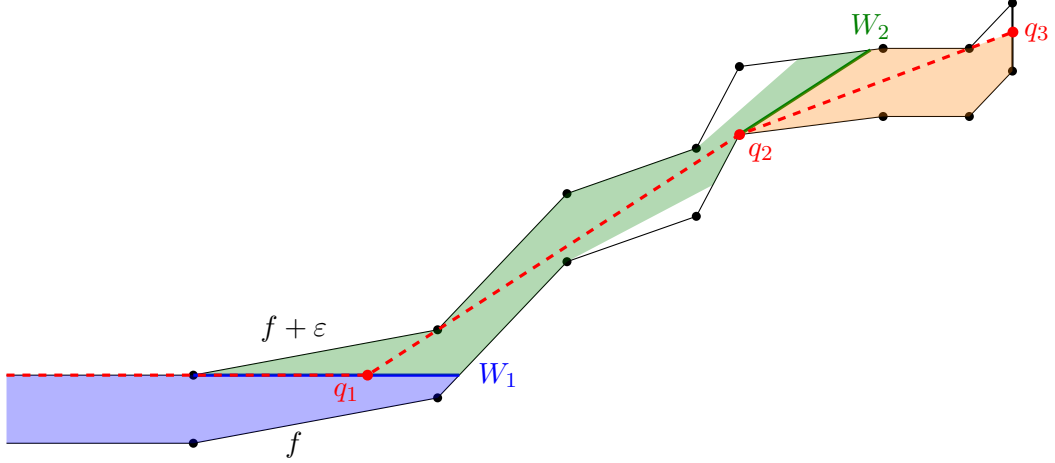


Figure 5.2: The Imai–Iri algorithm progresses through the corridor between f and $f + \varepsilon$ by constructing visibility polygons (shaded blue, green, orange) and their windows on the right boundary. The blue area is *weakly visible* from infinitely far to the left: each point in the blue area can be connected via a straight line segment lying inside the corridor to infinitely far to the left. The green area is weakly visible from the blue area, and in fact from the window W_1 . The orange area is weakly visible from W_2 . A possible output g is the ATF shown red and dashed, with breakpoints q_1, q_2, q_3 .

5.2 Evaluating and updating tours

A task that our overall algorithm performs very often is to insert an item (with its pickup and its delivery) into or remove an item from an existing tour. Even more often we want to evaluate the effect of such a change, i.e., whether it is possible at all to insert the item and if so at which cost, or how much cheaper a tour becomes when we remove an item. Only a small subset of the evaluated changes will actually be performed. Therefore we maintain data structures that allow fast queries and update them whenever we insert or remove an item. These data structures also enable us to efficiently evaluate exchanges of sequences of consecutive items between existing tours. The data structures we use are based on those proposed by Visser and Spliet [VS20].

Whenever we evaluate an insertion or removal of an item (or a sequence of items), we first compute the ATF of the resulting tour and then perform a scheduling in order to determine the cost of that tour. In Section 5.2.1, we describe our data structure for maintaining ATFs of tours and sections of tours. In Section 5.2.2, we describe our scheduling algorithm.

5.2.1 Maintaining arrival time functions of tours

Recall that a tour sequence is a sequence of actions. For each action i we have an arrival time function a_i . If we arrive at the location of action i at time t , the time $a_i(t)$ is the earliest possible arrival time at the location of the following action after having performed action i and traveled to the next location. This can include a waiting time if action i cannot be performed immediately.

5.2. Evaluating and updating tours

Our data structure is built on ATFs $a_1, \dots, a_n$ and allows us to efficiently compute the compositions $a_{i,j} := a_j \circ \dots \circ a_{i+1}$ for all $0 \leq i < j \leq n$. While for some simple vehicle routing problems, the ATFs $a_{0,i}$ and $a_{i,n}$ for $i = 1, \dots, n-1$ are sufficient to evaluate insertions and removals of single items, our data structure works for general pickup and delivery problems and also allows for efficiently evaluating larger changes such as swapping longer segments between tours.

Note that if a_i has b_i breakpoints for each i , the functions $a_{i,j}$ have at most $b_{i,j} := \sum_{h=i+1}^j b_h$ breakpoints (cf. Proposition 50). Therefore, we want to bound the query time of some $a_{i,j}$ roughly by $O(b_{i,j})$, but also initialization and update time for the data structure is relevant.

A first efficient data structure was given by Visser and Spliet [VS20], storing the actions in a binary search tree (see Figure 5.3). Let $b := \sum_{i=1}^n b_i$.

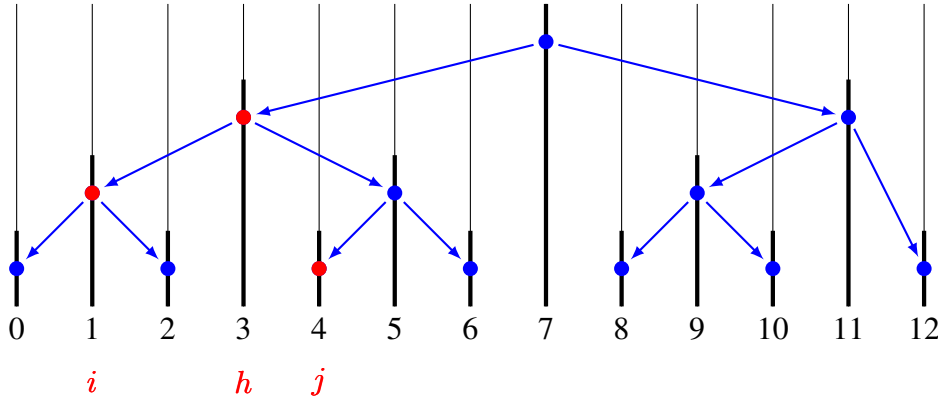


Figure 5.3: A balanced binary search tree. To compute $a_{i,j}$ we find the lowest common ancestor h of i and j and compose $a_{i,h}$ and $a_{h,j}$. These ATFs are precomputed because h is an ancestor of i and j .

Theorem 54 (Visser, Spliet [VS20]). *With $O(n \log n)$ compose operations and $O(nb)$ total initialization time, we can construct a data structure that allows the computation of any $a_{i,j}$ for $0 \leq i < j \leq n$ in $O(b_{i,j} + \log n)$ time with at most one compose operation.*

Proof. We build a balanced binary search tree on $\{0, \dots, n\}$ and precompute all $a_{i,j}$ such that i is an ancestor or a descendant of j in that tree. We say that a vertex is at level k if it has distance k from the root in that tree. The tree has levels $k = 0, \dots, \lfloor \log_2(n+1) \rfloor$.

For each level k we consider all vertices at level k , and for each such vertex h and each descendant i of h we compute $a_{h,i}$ (if $h < i$) or $a_{i,h}$ (if $i < h$). For a fixed vertex h , let p and q be the smallest and largest descendant of h , respectively. Then we compute the functions $a_{i,h}$ for $p \leq i < h$ in the order of decreasing i and we compute the functions $a_{h,i}$ for $h < i \leq q$ in the order of increasing i . This requires a single composition for the computation of each of these functions and hence we need in total $q - p$ compositions for each fixed vertex h . Because the intervals $[p, q]$ are disjoint for the different vertices h on the same level k , we need a total of at most n compose operations for each level.

For computing a single ATF $a_{h,i}$ (for $h < i$; the case $i < h$ is symmetric), the running time is $O(b_{h,i})$ because $a_{h,i}$ is computed by composing $a_{h,i-1}$ (with at most $b_{h,i-1}$ breakpoints) and a_i (with at most b_i breakpoints), leading to a running time of $O(b_{h,i-1} + b_i) = O(b_{h,i})$, which is $O(b_{p,q})$. For each vertex h , we compute at most $\frac{n}{2^{k-1}}$ such functions. Therefore, because the intervals $[p, q]$ are disjoint for the different vertices h at level k , the total running time at level k is $O(\frac{nb}{2^k})$.

Given some $0 \leq i < j \leq n$, we can find the lowest common ancestor h of i and j in $O(\log n)$ time and compute $a_{i,j}$ by composing $a_{i,h}$ and $a_{h,j}$. $\square$

When one ATF a_i changes, we can update this data structure with $O(n)$ compose operations: we only need to update the functions $a_{h,j}$ where $h < i \leq j$ and h is an ancestor or descendant of j . Then the search tree property implies that h or j is an ancestor of i . The ancestor of i at level k has at most $\frac{n}{2^{k-1}}$ descendants. This implies that we need to update at most $4n$ functions $a_{h,j}$.

When an action i is removed from the tour, we can replace a_i by the identity function and apply the above update. When a new action is inserted, we can insert a node into the search tree without rebalancing, and only after $O(\log n)$ insert or removal steps we recompute the data structure from scratch. This does not change any of the above-mentioned running time bounds, but saves time in practice.

In practice it is also useful to store the ATFs $a_{0,i}$ and $a_{i,n}$ for all $i = 1, \dots, n-1$ because these initial and final sequences are queried often. Again, this does not increase any of the above running time bounds.

If we allow more than one compose operation for a query, the initialization time (and the update time) can be improved further by building the data structure on small segments separately as shown in [BHMSTTV22].

Theorem 55 ([BHMSTTV22]). *Let $k \in \mathbb{N}$. With $O(n \log n)$ compose operations and $O(kn^{\frac{1}{k}}b)$ total initialization time, we can construct a data structure that allows the computation of any $a_{i,j}$ for $0 \leq i < j \leq n$ in $O(b_{i,j} \log(k+1) + \log n)$ time with at most $2k-1$ compose operations, and at most $k-1$ compose operations if $i = 0$ or $j = n$.*

Proof. We prove the statement by induction on k . The case $k = 1$ follows from Theorem 54 and by precomputing the ATFs $a_{0,i}$ and $a_{i,n}$ as described above.

For larger k , we choose $p := \lceil n^{\frac{1}{k}} \rceil$ and $m := \lceil \frac{n}{p} \rceil$. First, we build the data structure from Theorem 54 on each of the sets $A_i := \{a_{(i-1)p+1}, \dots, a_{ip}\}$ for $i = 1, \dots, m-1$ and $A_m := \{a_{(m-1)p+1}, \dots, a_n\}$. We also store the ATFs of the initial and final sequences within each A_i .

Second, we apply the induction hypothesis to construct a top-level data structure on ATFs a'_i , which are the compositions of the ATFs in A_i . More precisely, $a'_i := a_{(i-1)p,ip}$ for $i = 1, \dots, m-1$ and $a'_m := a_{(m-1)p,n}$.

The data structures for the A_i can be initialized with $O(n \log p) = O(\frac{1}{k}n \log n)$ compose operations in total and total time $O(pb) = O(n^{\frac{1}{k}}b)$. By the induction hypothesis, the data structure for $a'_1, \dots, a'_m$ can be initialized with $O(m \log m) = O(\frac{k-1}{k}n \log n)$ compose operations in total time $O((k-1)m^{\frac{1}{k-1}}b) = O((k-1)n^{\frac{1}{k}}b)$. Since the con-

5.2. Evaluating and updating tours

stant hidden in the O -notation is always the same, this implies the claimed bounds for the initialization.

For the query of some $a_{i,j}$ where a_{i+1} and a_j are not in the same set A_h , we set $s := \lceil \frac{i}{p} \rceil$ and $t := \lfloor \frac{j}{p} \rfloor$. Note that $a_{i,j}$ is the composition of the ATFs $a_{i,sp}$ and $a_{sp,tp} = a'_{s,t}$ and $a_{tp,j}$. By the induction hypothesis we can write $a'_{s,t}$ as the composition of $2(k-1)$ precomputed ATFs. The ATFs $a_{i,sp}$ and $a_{tp,j}$ are precomputed in the data structures on the sets A_s and A_{t+1} . Hence we can write $a_{i,j}$ as the composition of $2k$ precomputed ATFs. By induction, we can find these in $O(\log m + \log p) = O(\log n)$ time. Since they have a total of $b_{i,j}$ breakpoints, the running time follows from [Corollary 51](#).

For querying $a_{0,j}$, the induction hypothesis yields that $a_{0,tp}$ is the composition of $k-1$ precomputed ATFs, and $a_{tp,j}$ is precomputed in the data structure on the set A_{t+1} . So we need to compose k ATFs, and the same holds for querying $a_{i,n}$ by symmetry. $\square$

This multi-level data structure also allows for lazy update, which can save time in practice. Here is an example how our data structure allows fast evaluations:

Corollary 56. *Let $k \in \mathbb{N}$. Given the data structure of [Theorem 55](#), it takes at most $4k + 3$ compose operations and $O(b \log(k+1))$ total time to compute the total arrival time function of a tour after inserting the pickup action and the delivery action of an item at given positions, where b is the total number of breakpoints of the ATFs of all actions in the resulting tour.*

Proof. Let $a_1, \dots, a_n$ be the ATFs of the n actions of the current tour sequence. If we insert the pickup action after action i and the delivery action after action j , we need to compose $a_{j,n} \circ a_d \circ a'_j \circ a_{i,j-1} \circ a_p \circ a'_i \circ a_{0,i-1}$, where a'_i is the arrival time function for performing action i and then traveling to the pickup position, a'_j is the arrival time function for performing action j and then traveling to the delivery position, a_p is the arrival time function for performing the pickup action and then traveling to position of action $i+1$, and a_d is the arrival time function for performing the delivery action and then traveling to the position of action $j+1$. All of these four ATFs are precomputed. By [Theorem 55](#), computing $a_{0,i-1}$ and $a_{j,n}$ requires $k-1$ compose operations each, and computing $a_{i,j-1}$ requires $2k-1$ compose operations. Hence the total number of compose operations is $4k+3$. The running time bound follows from [Theorem 55](#) and [Corollary 51](#). $\square$

5.2.2 Tour scheduling

In the previous subsection we have discussed how we can efficiently maintain the total arrival time function $a: (-\infty, t_{\max}] \rightarrow \mathbb{R}$ for every tour. In order to evaluate the quality of a tour, we will frequently need to find the optimal starting time t_0 at which the tour should start. The problem of computing t_0 has been called the *optimal starting time problem* in [[HYI08](#)].

The total cost of a tour depends on various properties of the tour, many of which depend on the start time. Examples are the tour duration and the total driving distance. The driving distance depends on the start time of the tour because different paths are

optimal at different times; see [Figure 7.1](#) on page 110. If the cost is proportional to the driving distance, the cost is a piecewise constant lower semi-continuous function of the start time of the tour. (At breakpoints where two paths result in the same arrival time, we take the cheaper path; this guarantees the lower semi-continuity.)

In general we associate a piecewise constant, lower semi-continuous cost function c_f with every ATF f that we store. Whenever we compose two ATFs f and g , we compute the cost function associated with the composition via $c_{g \circ f} = c_g \circ f + c_f$. The number of discontinuities of $c_{g \circ f}$ is at most the sum of the numbers of discontinuities of c_f and c_g .

These piecewise constant cost functions can also be used to model further cost components, e.g., tolls or robustness costs for performing a pickup or delivery close to the end of its time window. For instance, we might impose an artificial penalty for arriving within the last 10 minutes of a time window. Such soft time windows will make the schedule more robust as we will see in [Section 7.3](#).

The most important component of the cost function is the tour duration $a(t_0) - t_0$, where a is the ATF of the tour. To model overtime costs, we allow for a piecewise linear, continuous, and non-decreasing function $c_{\text{ot}} : [0, \infty) \rightarrow \mathbb{R}$ and describe the resulting cost by $c_{\text{ot}}(a(t_0) - t_0)$. If the cost is proportional to the tour duration, c_{ot} is a linear function. In practice, c_{ot} has very few breakpoints.

Finally, the working time cost may depend on the time of the day. For example drivers might be paid more if they are scheduled to work outside of normal working hours. We describe the cost per hour by a piecewise constant function $c_{\text{wt}} : [t_{\min}, t_{\max}] \rightarrow \mathbb{R}$ and pay

$$\int_{t_0}^{a(t_0)} c_{\text{wt}}(t) dt$$

if we start at time t_0 .

Combining the different cost components, our goal is to compute a $t_0 \in [t_{\min}, t_{\max}]$ minimizing the total cost

$$c_{\text{tot}}(t_0) := c_a(t_0) + c_{\text{ot}}(a(t_0) - t_0) + \int_{t_0}^{a(t_0)} c_{\text{wt}}(t) dt.$$

Note that most of the cost components in this function can also be modeled in the cost model in [\[HYI08\]](#). The algorithm proposed in [\[HYI08\]](#) also optimizes waiting time if this reduces the cost, while in our setting we allow waiting only when we arrive at a stop before the time window begins. This seems a reasonable assumption in practice and allows us to solve the optimal starting time problem much faster:

Theorem 57 ([\[BHMSTTV22\]](#)). *Let a , c_a , c_{ot} and c_{wt} have b_a , b_{c_a} , $b_{c_{\text{ot}}}$ and $b_{c_{\text{wt}}}$ breakpoints respectively. Then we can compute the least $t_0 \in [t_{\min}, t_{\max}]$ minimizing c_{tot} in $O(b_{c_{\text{ot}}} b_a + b_{c_a} + b_{c_{\text{wt}}})$ time.*

Proof. Note that $c_{\text{tot}} : [t_{\min}, t_{\max}] \rightarrow \mathbb{R}$ is itself a piecewise linear, lower semi-continuous function. So in order to find its minimum, it suffices to construct all of its breakpoints. To do this we perform a linear scan in the interval $[t_{\min}, t_{\max}]$.

5.2. Evaluating and updating tours

Starting with $t = t_{\min}$, we may compute $c_{\text{tot}}(t) = c_{\text{tot}}(t_{\min})$ easily in $O(b_a + b_{c_a} + b_{c_{\text{ot}}} + b_{c_{\text{wt}}})$ time. Now as t increases, c_{tot} can have a breakpoint at t only if one of the following events occurs:

- (i) c_a has a discontinuity at t ,
- (ii) a has a breakpoint at t ,
- (iii) c_{ot} has a breakpoint at $a(t) - t$,
- (iv) c_{wt} has a discontinuity at t , or
- (v) c_{wt} has a discontinuity at $a(t)$.

Given t and the relevant segments of a , c_a , c_{ot} and c_{wt} , we can find the next of these events in constant time. Note that event (i) can occur b_{c_a} times, event (ii) can occur b_a times, event (iii) can occur $b_{c_{\text{ot}}} b_a$ times, and events (iv) and (v) can each occur $b_{c_{\text{wt}}}$ times since a is non-decreasing. So we can iterate through all breakpoints of c_{tot} in $O(b_{c_{\text{ot}}} b_a + b_{c_a} + b_{c_{\text{wt}}})$ time to find the first minimizing breakpoint. $\square$

In practice, both c_{ot} and c_{wt} tend to have very few breakpoints. So the running time tends to be effectively $O(b_a + b_{c_a})$. By approximating both a and c_a during the algorithm, it is thus possible to query the optimal starting time of a tour and hence its total cost very efficiently.

Chapter 6

A local-search based heuristic for real-world vehicle routing problems

In this chapter we explain our overall algorithm for vehicle routing with time-dependent travel times. Of course it will call the operations described in the previous chapter many times.

When describing the algorithm in [Section 6.1](#), we consider a still restricted model. This model contains the most important aspects and suffices to explain the main algorithmic ideas. The full model is described in [Section 6.2](#). In case that the adaption of our algorithm to the more general model is not straightforward, we describe how to adapt the algorithm in [Section 6.2](#) as well.

6.1 Constructing and improving solutions

As mentioned, we consider a still restricted version of our model. We assume that all arrival time functions between points of interest are given in the input. [Section 6.1.1](#) comments on how to compute these arrival time functions in a fast way. We end up with the following problem description.

The BonnTour model (simplified) : We are given a list of items, a list of vehicles, and a complete directed graph $G = (V, E)$. For each vehicle (type), we are given ATFs on the edges in E satisfying the triangle inequality. Additionally, for each item, we are given

- a pickup location and a delivery location, corresponding to vertices in V ,
- a pickup duration and a delivery duration,
- a pickup time window and a delivery time window,
- a demand in several dimensions (e.g., weight and size), and
- a penalty for non-delivery. (If the penalty is paid, the item does not need to be picked up as well.)

For each vehicle, we are given

- a start location and an end location, corresponding to vertices in V ,

6.1. Constructing and improving solutions

- a capacity in several dimensions,
- a setup cost, and
- distance and working time costs as described in [Section 5.2.2](#).

A feasible tour consists of a vehicle, a tour sequence and a feasible schedule (cf. [Section 4.1](#)) such that

- the tour starts and ends at the start and end location of the vehicle, respectively,
- the vehicle capacity is never exceeded.

The goal is to find a set of feasible tours such that no vehicle is used twice and the total cost, consisting of distance and working time costs, vehicle setup costs, and penalties for non-delivery, is minimized.

On a very high level, we follow a two-stage approach. First, after preprocessing the street map and the items (cf. [Sections 6.1.1](#) and [6.1.2](#)), we construct an initial solution by iteratively inserting items. (We optimize intermediate solutions to avoid opening too many tours.) The tour construction is described in detail in [Section 6.1.3](#). Then, we run extensive local search on the resulting tour set as described in [Section 6.1.4](#). We follow the “ruin-and-recreate” framework that has been proposed for capacitated vehicle routing by Accorsi and Vigo [[AV21](#)]: Iteratively, shake the solution by removing and reinserting items that are part of a random walk. Then, optimize the resulting tour set via local search. The decision whether to jump back to the old solution is based on a simulated annealing acceptance strategy. We adapt this algorithm at several places to suit our much more general model.

6.1.1 Preprocessing the street map and computing fastest paths

Before running our main algorithm, we have a preprocessing step that prepares the road network information. This step is independent of the addresses and thus needs to be run again only when the road network or the vehicle fleet changes. Input to preprocessing is a road network with all relevant information on restrictions and travel times. This is modeled as a graph with undirected and directed edges. For each edge and each vehicle type the input contains an ATF, which is infinite if the vehicle is not allowed on the road corresponding to that edge. We end up with a separate graph for each vehicle type. Turn restrictions (forbidden sequences of vertices) are modeled by making copies of certain vertices, depending on the predecessor(s) on a path. With each ATF we also store a lower and an upper bound on the travel time in order to speed up certain operations later. We also approximate ATFs at several stages.

Next, we compute a *contraction hierarchy*, similarly to [[BGSV13](#)]. In this step, a total order of all vertices of a graph is computed. If $v_1, \dots, v_n$ is this order, we process $k = 1, \dots, n$ and add a shortcut edge from v_i to v_j (with composed ATFs) whenever $i > k$ and $j > k$ and there are edges (including previously added shortcut edges) from v_i to v_k and from v_k to v_j such that these (might) form a fastest path from v_i to v_j at some time of the day (here we use lower and upper bounds). This guarantees that there is always a fastest path whose vertex indices are first increasing and then

decreasing (an *up-down path*). Occasionally, a slower path could be cheaper than the fastest path, but this effect is negligible in practice because drivers typically choose a fastest path anyway. The contraction hierarchy typically roughly doubles the number of edges but allows for faster fastest-path queries later. The total preprocessing time is less than 10 minutes on typical instances up to 8 million road segments and up to 2 000 start and end addresses (including those used to generate our benchmarks described in [Section 7.2](#)) on an AMD EPYC 7742 server. This divides into up to 6 minutes for computing the contraction hierarchy and up to 4 minutes for computing the matrix of arrival time functions for each pair of addresses.

When processing an actual instance with pickup and delivery addresses given by geographic coordinates, we first map each address to the nearest well-accessible road segment. We add a copy of the corresponding edge, subdivide it, and put the new vertex at the beginning of our order. Keeping the original edge ensures that we still have a fastest up-down path for every pair of vertices. For each such vertex v we then compute a forward cone and a backward cone (as suggested by Geisberger and Sanders [GS10]). The forward cone contains all vertices that are reachable from v via a path with increasing indices ($i < j$ for each edge (v_i, v_j) on the path) that is fastest at some time. The backward cone contains all vertices from which v is reachable via a path with decreasing indices ($i > j$ for each edge (v_i, v_j) on the path) that is fastest at some time. Moreover we store the minimum of the composed ATFs of such paths at the elements of the cones and approximate the resulting ATF as described in [Section 5.1](#). Most cones contain at most few hundred vertices and can be computed in a few milliseconds.

When asking for a fastest path from v to w (a frequently needed operation), we intersect the forward cone of v with the backward cone of w and sort the intersection according to the sum of the two minimum travel times (a lower bound). To compute the total ATF from v to w , we scan the intersection in that order, compose the two ATFs and take the minimum with the previous total ATF. We can stop when the lower bound exceeds the current maximum travel time (which happens after fewer than ten steps normally). The total computation time for this step is less than a tenth of a millisecond on average. For each segment of the ATF we also store the time-independent part of the cost of the corresponding path (e.g., cost proportional to the driving distance). The actual path is not stored, instead it is simply recomputed in the very end when all tour sequences are fixed; then we also recompute optimal parking positions.

6.1.2 Preprocessing items

Before starting with our tour construction, we compute item specific data that we use repeatedly.

First, we identify infeasible items, i.e., items that cannot be shipped by any vehicle even if the vehicle is reserved for this item only. For each infeasible item, we check whether it could be shipped when relaxing certain constraints. We output this information which might be valuable to customers or to detect input errors. We ignore infeasible items henceforth.

Next, we identify subsets of items for which we are almost sure that they should be

handled by the same vehicle (i.e., same pickup and same delivery address and compatible time windows), and treat them as a single item to speed up tour construction.

For each remaining item a , we identify items that seem to fit well into the same tour as a (although not almost identical to a in the above sense) and declare them as *friends* of a . These friends will play an important role when defining local search neighborhoods as discussed later. To identify friends, we estimate the cost increase when shipping both items at once compared to the cost when shipping only one of the items. Each item will end up with a constant number of friends. Very close friends are grouped to a *bundle*. They are inserted at once during tour construction to reduce computation time. When optimizing our solution with local search, we dissolve the bundles as otherwise our search space would be too restrictive. Hence, bundled items may end up in different tours of our final solution.

In another preprocessing step, we compute an approximation of the *happy nucleolus*, i.e., a cost allocation that distributes as much cost as possible to the items under certain fairness conditions [BETV24]. We call the cost allocated to some item its *share*. The share of an item can be interpreted as an estimate of the cost this item causes. We will use this estimate repeatedly as discussed later. Note that the share of an item cannot be higher than its penalty for non-delivery.

6.1.3 Constructing tours

Many heuristics have been proposed to construct initial solutions for vehicle routing problems. A comparison of classical tour construction heuristics can be found in [Sol87]. We choose (and adapt) a heuristic first proposed by Foisy and Potvin [FP93], which generalizes to our model in a quite natural way. We construct our initial solution by opening some tours delivering items that are well spread and important (i.e., we should handle them, but it is not easy to handle them), and insert the remaining items similar to [FP93]. This way, we always maintain a feasible solution taking vehicle availabilities into account. We optimize intermediate solutions to avoid opening too many tours.

We remark that compared to other local-search based vehicle routing algorithms, we put much more effort in constructing the initial solution. Due to our general model including time-dependent travel times, the evaluation of a single tour change is quite time consuming, even when using the advanced data structure described in Section 5.2.1. Hence, we cannot afford as extensive post-optimization as vehicle routing algorithms working with constant travel times. Moreover, on complex instances with many constraints, reassigning vehicles at some later point in the algorithm without violating feasibility is typically difficult. Hence, we put special focus on creating initial solutions of high quality. In the following, we describe our tour construction in more detail.

In a first step, we want to identify a subset of items that are important (i.e., we should handle them, but it is not easy to handle them) and not at all friends of each other. These will serve as *seeds*: we open a separate tour for each seed item as a starting point of our tour construction. Note that in case of a restricted vehicle fleet there might be many more shipping requests than we can deal with. In such a case, we greedily construct a

solution preferring items with a large penalty for non-delivery but small share. (Items with a small share utilize few resources.) When constructing this greedy solution, we assume that there are twice as many vehicles available to take into account that greedy tours are usually much less efficient than tours of an optimum solution. All items not covered by this greedy solution will be ignored during seed selection. Next, we compute another greedy solution on the restricted set of items, but this time, we do not double the vehicle fleet and start with items that are important but difficult to handle (i.e., high penalty for non-delivery and high share), somewhat similar to [FP93]. Then, we choose at most one item out of each tour as seed such that the chosen seeds are well spread. (When choosing the next seed, we prefer items that are not at all friends of already chosen seeds.) In contrast, Foisy and Potvin [FP93] choose exactly one seed per tour. However, we want to avoid opening more seed tours than needed in the end. (The number of tours in the greedy solution is typically larger than the number of tours in an optimum solution.) Each time we open a tour for one of our seeds, we have to decide for a vehicle. For a set of promising vehicles (small cost and small utilization in terms of capacity, working time, etc.) we greedily fill the tour and take the vehicle for which the resulting tour is most efficient (small cost and high sum of covered item shares).

Then we insert the remaining item bundles one by one. We always guarantee to maintain a feasible solution. In each step we pick a bundle whose *average regret* is maximum, i.e., the difference of the expected cost of inserting it into a random tour and the cost of inserting it into the best tour. Again, this is similar to the approach proposed by [FP93]. For measuring the insertion cost, we use the algorithm described in Section 5.2.1, followed by the scheduling as described in Section 5.2.2. If inserting a bundle into a tour is infeasible (e.g., due to violated capacity constraints or time windows), the corresponding cost is replaced by the cost of shipping it by a new separate tour (or a large constant if no further vehicle is available). When we detect that more tours are needed than we have, we compute additional seed tours.

Moreover, at certain stages, in particular after creating new seed tours, and more extensively at the very end, we perform local search operations (see the next subsection) to optimize the existing tours. After having computed different candidate solutions with parallel threads, making different random choices at various places, we continue post-optimizing the best of these solutions in the end with all threads.

6.1.4 Post-optimization

We implemented various local search operations, most of which are well-known from the vehicle routing literature. The basic operations used by all of them are those described in Section 5.2, but we apply them often to contiguous sequences of items in a tour rather than single items. Formally, by a *sequence* we mean a set of items in a tour for which all their pickup events are contiguous in this tour and all their delivery events are contiguous in this tour. It is straightforward to extend the insertion and removal algorithms of Section 5.2.1 to sequences.

To evaluate the cost resulting from an insertion or removal, we need to run the scheduling algorithm from Section 5.2.2. For inserting a segment into a tour we usually

try several positions for the pickup sequence as well as for the delivery sequence. If an insertion at a certain position is infeasible due to capacity constraints, incompatibility, or sorting rules, this is detected quickly. The scheduling algorithm can determine infeasibility due to time windows or exceeding the working time limit. Often our lower bounds can be used to detect infeasibility much faster even without composing ATFs and scheduling.

At any stage we maintain a feasible solution. When we try inserting or removing a sequence of items, we do not update the main data structures until we know that we actually want to perform this change; see [Section 5.2.1](#). Only a small percentage of the evaluated changes will actually prove beneficial and be performed.

After significant changes to a tour, the tour sequence is re-optimized. We first run the Lin-Kernighan heuristic [[LK73](#)] adapted to the asymmetric path [TSP](#), but only on segments in which reorderings that reduce the total travel time are likely to maintain feasibility (with respect to sorting, capacity constraints, and the current schedule). We assume constant travel times during optimizing a segment and evaluate the tour with respect to time-dependent travel times afterwards. We discard the tour changes if the actual cost turned out to be worse. This routine performs especially well on long tours with few constraints. Next we run a (slower but accurate) dynamic programming algorithm similar to [[LMS20b](#)], but we randomly preselect a subset of (typically long) edges of the tour and force the dynamic program to keep the sequences in between those edges contiguous to reduce the running time. This routine performs particularly well on short tours. Last, we run the Lin-Kernighan-Helsgaun-3 heuristic, which was developed for a wide range of vehicle routing problems [[Hel17](#)]. To speed up the computation, we only evaluate k -opt steps that are improving with respect to optimistic constant bounds on the time-dependent travel times. This routine performs very well on long tours with many constraints.

For optimizing a set of tours, we have implemented various local search operations.

Segment swapping operates on two tours, removes a sequence from each, and inserts these sequences (possibly reversed) into the other tour. Most vehicle routing algorithms in the literature restrict the swaps to explore to sequences of constant size, e.g., [[VCGP13](#); [AV21](#); [Vid22](#)]. However, we allow segments of arbitrary size to take into account that it is not unusual for real-world instances to be locally clustered. Note that there are $\Theta(n_1^2 \cdot n_2^2)$ many choices for the two segments, where n_1 and n_2 denote the number of items handled by the first and second tour, respectively. Hence, for long tours, we cannot afford evaluating all swaps. We reduce the number of evaluations as follows. Consider two (possibly reversed) segments, one from each tour. Let c_1 be an estimate of the cost difference between traversing the connections to the startpoints of the segments before and after the swap. Let c_2 be an estimate of the cost difference between traversing the connections starting in the endpoints of the segments before and after the swap. We only evaluate the swap if it is among the k best with respect to c_1 or among the k best with respect to c_2 for constant k . This way, the number of swaps that we evaluate reduces to $O(n_1 \cdot n_2)$. To further reduce the number of evaluations, we only consider swaps for which the number of connections between friends does not decrease.

Matching removes short (possibly empty) sequences from all tours and possibly

takes few items that are currently not served by any tour, and finds an optimal bipartite matching, assigning these sequences to the so-reduced tours (and possibly to “outside any tour”). To improve running time, we do not build the complete bipartite graph, but only evaluate insertions of removed segments into tours containing friends of items in this segment. We run a constant number of iterations. The sequences are chosen randomly, where sequences that seem to fit better in other tours are chosen with higher probability. To identify such sequences, we compute potential insertion costs (ignoring feasibility constraints) when inserting the sequence directly before or directly after a friend in some other tour. For each tour, we do not remove any item with constant probability. Distributions for different tours are independent. Note that the probability distributions only need to be recomputed in case we find a new best solution. If there are items that are currently not served by any tour, we select a constant number of these in every matching iteration to be considered for insertion. The probability of selecting a certain item is initially proportional to its penalty for non-delivery and decreases exponentially with the number of times this item was chosen to be considered for insertion.

Minimum mean cycle considers a complete digraph whose vertices are the tours, and in which the weight of the edge from tour A to tour B is an estimate of the (possibly negative) cost of removing a selected sequence from tour A and inserting it into tour B, ignoring feasibility constraints. We find a cycle with minimum mean weight in this digraph. If its total weight is negative, we check whether the corresponding set of operations is actually feasible and beneficial. If not, we increase the weight of an edge whose weight was too optimistic and iterate.

Vehicle optimization reassigns vehicles to tours by computing an optimal bipartite matching. We do not call this routine until we have completed the initial tour construction, as we may have proactively opted for larger and more expensive vehicles so that we can add more items to existing tours.

Chain opt tries to insert an item that is not yet shipped. For each promising tour we compute a cheapest replace, i.e., a sequence such that removing this sequence and inserting the considered item minimizes the total cost (where we charge the sum of item shares for the removed sequence as this estimates the expected insertion cost into another tour). We iterate the procedure with the removed sequence, building a chain until (hopefully) we managed to insert all items. Such *ejection-chains* have been introduced by Glover [Glo96] for the TSP and were extended to vehicle routing problems, e.g., in [Reg01; SHU05; SSRR17; AV21]. Note that we branch the chain by trying to insert each sequence into a small number of promising tours (that contain many friends of items in the sequence to insert).

6.1.5 Random walk

Random walks were introduced by [ACR03] to iterate local search for the TSP. We follow the approach proposed by Accorsi and Vigo [AV21] for capacitated vehicle routing: removing a random sequence from a tour, re-inserting the items as described in Section 6.1.3 or in other orders as suggested by [AV21], and post-optimizing the involved tours (and a small number of additional tours containing friends of involved items, as

these tours may be part of improving local search operations) using the above-described operations. Then, we decide whether to proceed with the resulting solution by a simulated annealing acceptance strategy. After a certain number of iterations we return the best among all intermediate solutions.

Note that it is unlikely that we dissolve a whole tour in one random walk and manage to re-insert all items in the same iteration. However, in real-world instances, we often have high setup costs per vehicle. Hence, we adapt the algorithm as follows. At any point in time, we maintain two solutions that we optimize in parallel. On the first solution we run the framework described above. On the second solution, we dissolve a whole tour and do not allow to open a new tour when re-inserting the removed items. In case that we do not manage to re-insert all items we still proceed with this solution. In the following iterations, we try to re-insert all items that we removed. In order to allow for small progress from one iteration to the next, we use our precomputed item shares (recall that the share of an item can be interpreted as an estimate of the cost this item causes) as penalties for non-delivery so that the artificial total cost improves if we insert an item that seems difficult to handle and instead remove an item that seems rather easy to handle. Every time an item is not part of the solution at the end of an iteration, we increase its share as this item seems to be harder to handle than we initially thought, similar to [CLQL18; SB18]. As soon as we managed to re-insert all items and found a new best solution, we update the first branch as well and start all over. Note that we do not actively try to dissolve a tour in the first branch so that we still optimize other parts of the objective (e.g., working time costs or fuel costs). When dissolving a tour, we prefer inefficient tours (high tour cost compared to covered item shares) for which the covered items seem to be rather easy to reinsert into other tours (small total item shares). In both branches, after each iteration, we jump back to the best solution found so far with small probability. Figure 7.2 on page 114 illustrates the improvements found by this framework on one of our new benchmark instances.

On complex instances with a heterogeneous vehicle fleet, it is more difficult to make a good decision about which tour to dissolve. (We might decide to dissolve an inefficient tour with large capacity, but it is only possible to dissolve a tour with small capacity.) In such scenarios one could try to jump back to the best solution found so far with higher probability. Moreover, it might be the case that dissolving a whole tour and reassigning its items is impossible, but instead, one can assign a less expensive vehicle with smaller capacity to some tour by reassigning some of its items. Currently, we do not actively look for such improvements. This is left as potential future work.

For the final solution returned by BonnTour we guarantee that every item that is not part of the solution cannot be inserted via cheapest insert by a smaller cost than its penalty for non-delivery. Moreover, each tour of the final solution has been optimized after the latest insertion or deletion (using our TSP algorithms). Furthermore, we check that for no set of items occurring subsequently in the same tour we benefit from removing these items and paying their penalty instead.

6.2 Detailed model description

Now we describe the model our algorithm covers in full detail. As before, part of an instance is a list of vehicles and a list of items to be shipped.

Each vehicle has characteristics that define the road restrictions and speed profiles such as maximum speed and vehicle dimensions. Moreover, each vehicle has a start address and an end address. It is also possible that start and/or end addresses can be chosen freely. Each vehicle has capacities in several dimensions (e.g., weight and size), possibly with a soft limit that can be exceeded at an extra cost until the hard capacity limit is reached. Each vehicle can have a setup cost, a cost per driving distance, and a cost per used time. Moreover, one can define costs for caused emissions. A vehicle (with a driver) is available for a specified duration within a certain time period, with a specified flexibility regarding the shift start. One can define overtime options at certain extra costs per used time depending on the shift length and the time of day as described in [Section 5.2.2](#). Additionally, we might have given a maximum allowed driving time (due to driving time regulations) and a maximum distance that an (electric) vehicle can go. There might be further restrictions due to break regulations. In [Section 6.2.1](#), we discuss how we handle such regulations. Finally, sorting of loaded items might be forbidden. In this case we can only deliver the item that we picked up last.

In contrast to the simplified models considered earlier, each item might have several alternative pickup and delivery addresses. For example, identical copies of certain items might be available at several locations (depots or shops), and we can pick up the item at any of these. We discuss how to extend our algorithm to this more general model in [Section 6.2.2](#). Additionally, each item has a pickup time window and a delivery time window. We can specify extra costs if we pickup or deliver an item at the end of its time window [[Sch20](#)]. This way we can create an incentive to include some time buffer, making the overall solution more robust against unforeseeable delays (see [Section 7.3](#) for experimental results). It is planned to extend the model to allow several alternative time windows depending on the pickup or delivery option (e.g., it might be feasible to deliver an item either in the morning or in the evening). An item can be classified as *direct transport*. In this case we are not allowed to pickup or deliver any other item as long as this item is loaded. This way we can model transferring a trailer that blocks the access to the actual loading space. Finally, an item comes with a penalty for non-delivery that we have to pay in case we decide not to ship it. This feature is mainly used in the case when the number of vehicles is not sufficient to handle all items; then some items can be prioritized.

There is a time required for pickup and for delivery, an extra time for visiting an address (that we only need to pay once for subsequent pickups and deliveries at the same address) and an extra time for parking: thus we can save some time by parking only once for several pickups or deliveries nearby. All these times can depend on the vehicle type. A maximum parking radius can be specified to avoid too long walking distances, as well as the maximum weight a driver can carry.

The set of vehicles that can ship an item can be restricted, or alternatively, we can charge vehicle-dependent extra costs for each delivered item. We can set a vehicle-

and address-dependent surcharge for visiting an address. This way, we can create an incentive to deliver all items going to the same address at once, which is usually in the customer's interest.

Tours passed in the input can be used as starting point for optimization. This way, we can integrate adhoc shipping requests during the execution of the computed tour plan. One often wants to integrate such adhocs at small extra cost but keep the number of tour changes small. Item-vehicle assignment costs can be used to model such preferences. If preferred, one can even opt to forbid any tour changes except insertions of adhocs.

6.2.1 Breaks

Due to regulations, drivers have to take breaks after a certain amount of time. In a multiday schedule, there might be such regulations between two subsequent breaks (e.g., night break and lunch break) as well as between two night breaks. We deal with such regulations in the following way. For a more detailed description see [\[Ros22\]](#).

As input we allow a list of breaks of which we have to schedule an initial subsequence. For each break we specify a minimum duration and a time window in which the break has to take place. Each break has a specified level. (Currently, the implementation allows two levels.) One can add time limits (and limits on the pure driving time and the covered distance) between breaks if all breaks in between have strictly lower level (e.g., in a multiday schedule, we assign level 0 to lunch breaks and level 1 to night breaks). This restriction simplifies the implementation significantly and covers almost all common regulations. In scheduling, we first heuristically choose which breaks to schedule and determine for each such break a position in the tour by specifying before/after which pickup or delivery action the break should take place. In this first step, we use pessimistic constant travel time estimates. (Recall that scheduling is typically called millions of times. Hence, we need to find a reasonable solution in short computation time.) Then, we compute the optimum start time under the additional constraint that all break regulations must be met by a straightforward adaption of the algorithm described in [Section 5.2.2](#).

6.2.2 Multiple pickup and delivery locations

Recently, we extended the model so that one can pass several alternative pickup and delivery locations per item. E.g., this can be used to model multi-depot scenarios where some goods are available at several depots (see [Section 7.4.1](#) for experimental results). In another scenario we have the choice to deliver some item directly to the customer, or bring the item to some packing station, which might come at certain extra cost as customer satisfaction suffers.

In order to keep the increase in running time manageable when allowing several pickup and delivery options per item, we keep the address choice of already inserted items fixed until a tour changes. Hence, when evaluating the insertion of a sequence of items into a tour, we only have to optimize the address choice for the sequence to insert. For each feasible pair of start- and end-address for the sequence to insert, we precompute

good address choices for the intermediate stops (by a simple dynamic program using constant lower bounds on the driving times). Then, for each insertion position, we decide for the best pair of start- and end-address (again, with respect to constant lower bounds) and evaluate the insertion at this position only for this address choice. Note that we might miss the cheapest insertion, but by keeping the address choice for items that are already part of the tour fixed we can use the data structure described in [Section 5.2.1](#) for fast ATF queries. In segment swapping, we even keep the address choice for the removed sequences fixed for the same reason. When actually performing a tour change, we re-optimize the address choice of all items in the tour, again, by a simple dynamic program working with constant lower bounds on the travel times. As tour changes are performed much less frequently than evaluated, one could try to increase the effort when re-optimizing addresses after a tour change. This is left as future work.

Currently, when optimizing a single tour, we first call our [TSP](#) routines as described in [Section 6.1.4](#) and optimize the address choice afterwards as mentioned above. This could potentially be improved by optimizing the stop order and the address choice simultaneously.

6.2.3 Compact tours

Although minimizing total cost is the main objective, solutions in which nearby items are handled by the same tour are often preferred. Such *compact* solutions simplify the driver-tour assignment by taking driver-specific area knowledge into account. We propose to charge small relative extra costs for *spread* tours, where we measure the spread of a tour by one minus the average fraction of friends¹ per item that are covered by the same tour (with close friends having a higher weighting). These extra costs are only charged during optimization so that there is an incentive to keep tours compact. Usually, we decrease these artificial penalties towards the end of the algorithm to find a good compromise between solution cost and compactness of the computed solution. This way, we get significantly more compact tours at comparable total costs (see [Section 7.4.2](#) for experimental results).

¹Recall that friends are items that seem to fit well into the same tour, cf. [Section 6.1.2](#), although here, we ignore time windows when computing the friends of an item as we aim for solutions in which nearby items are handled by the same tour.

Chapter 7

Experimental results

We implemented the described algorithms in C++ and integrated them into a software package called BonnTour. To demonstrate the efficiency of BonnTour we carried out extensive experiments. First, in [Section 7.1](#), we used existing capacitated vehicle routing benchmarks without and with delivery time windows. Second, as large-scale and realistic public benchmarks with time-dependent travel times were missing, we created a new set of instances, which we make publicly available at <https://gitlab.com/muelleratorunibonn/vrptdt-benchmark>. We describe how we generated these instances in [Section 7.2](#) and report experimental results in [Section 7.3](#). These results show the importance of incorporating time-dependent travel times during optimization. In [Section 7.4](#), we demonstrate some of the BonnTour features discussed in [Section 6.2](#) by conducting experiments on (modifications of) our new benchmark set as well as on instances that have been generated by [Greenplan](#) and model real-world vehicle routing problems.

All our experiments were carried out on a 2-way AMD EPYC 7742 Linux server with 128 physical cores in total. We ran multiple jobs simultaneously (each using up to 8 threads), but we never used more than 128 threads in total.

7.1 Results on standard vehicle routing benchmarks

We tested BonnTour on the benchmarks for the capacitated vehicle routing problem (CVRP) by Uchoa et al. [[UPPPVS17](#)], on the benchmarks for the capacitated vehicle routing problem with time windows (CVRPTW) by Gehring and Homberger [[GH99](#)], and on the pickup-and-delivery problem with time windows (PDPTW) by Li and Lim [[LL03](#)]. These are popular benchmark sets for state-of-the-art vehicle routing algorithms. While there are 100 Uchoa et al. instances with sizes ascending from 100 to 1000 customers, there are 300 Gehring and Homberger instances, with 60 instances in each of the groups with instance size 200, 400, 600, 800, and 1000. All these instances have a single depot for pickup. There are 354 Li and Lim instances ranging from 100 to 1000 tasks (each task is either a pickup or a delivery).¹ Each of the instances by Li

¹In their paper, Li and Lim only describe how they generated the instances with 100 tasks. All instances can be found on <https://www.sintef.no/projectweb/top/pdptw/>.

7.1. Results on standard vehicle routing benchmarks

Instance class # instances	BKS		BonnTour default mode				BonnTour high-effort mode			
	tours	distance	tours (% gap)	distance (% gap)	<i>t</i>		tours (% gap)	distance (% gap)	<i>t</i>	
U (100–246) 32	759	1009991	777	1019246 (0.9)	157		773	1015816 (0.6)	1354	
U (250–490) 36	1567	2040726	1594	2070983 (1.5)	246		1587	2059175 (0.9)	2051	
U (501–1000) 32	2683	3259953	2755	3316501 (1.7)	482		2744	3296898 (1.1)	3292	
GH (200) 60	694	168034	694 (0.0)	173135	123		694 (0.0)	170630	1004	
GH (400) 60	1380	387654	1386 (0.4)	404162	196		1382 (0.1)	396773	1733	
GH (600) 60	2065	784653	2074 (0.4)	834728	331		2068 (0.1)	811157	2714	
GH (800) 60	2732	1329290	2758 (1.0)	1392141	394		2750 (0.7)	1360467	3402	
GH (1000) 60	3416	2019978	3435 (0.6)	2167893	494		3423 (0.2)	2102217	3902	
LL (100) 56	402	58060	402 (0.0)	58081	69		402 (0.0)	58066	546	
LL (200) 60	600	183720	608 (1.3)	183328	120		602 (0.3)	184939	987	
LL (400) 60	1129	437440	1160 (2.7)	451067	217		1143 (1.2)	446268	1802	
LL (600) 60	1619	889271	1673 (3.3)	909281	290		1640 (1.3)	921799	2654	
LL (800) 60	2102	1483301	2181 (3.8)	1522891	370		2138 (1.7)	1524887	3567	
LL (1000) 58	2562	2184255	2682 (4.7)	2244576	454		2607 (1.8)	2259760	4517	
D (25) 56		663546	200	664192 (0.1)	16			664172 (0.1)	134	
D (50) 56		1273212	275	1276041 (0.2)	43			1274952 (0.1)	370	
D (100) 56		2416331	444	2423347 (0.3)	99			2420352 (0.2)	828	

Table 7.1: Relative gaps of the two modes of BonnTour compared to the best known solutions on popular benchmark instances for CVRP (U: Uchoa et al. [UPPPVS17], with a total of 100 instances ranging from 100 to 1000 customers), CVRPTW (GH: Gehring and Homberger [GH99], with a total of 300 instances from 200 to 1000 customers), PDPTW (LL: Li and Lim [LL03], with a total of 354 instances from 100 to 1000 tasks), and TDVRPTW (D: Dabia et al. [DRVD13], with a total of 168 instances from 25 to 100 customers). Results for the main objective are shown in bold. (For the TDVRPTW, the objective is to minimize the total working time, here shown as distance.) The average computation time *t* per instance is given in seconds. For the TDVRPTW instances, the number of tours are not reported in [DRVD13; LBS18; LMS20a; PZL21a].

and Lim is either based on one of the Gehring and Homberger instances, or on one of the 56 Solomon instances for capacitated vehicle routing with time windows with 100 customers each [Sol87]. For the Uchoa et al. instances the objective is to minimize the total distance, whereas for the Gehring and Homberger instances and for the Li and Lim instances the main objective is to minimize the number of tours (used vehicles), using the total distance only as second criterion.

These instances have fixed travel times for any pair of addresses. Because BonnTour has the implementation overhead of maintaining time-dependent travel times and handling many additional constraints, we cannot expect to outperform the best implementations for these special instance types. Here, we show that we are close to the best known solutions, which are reported at <https://www.sintef.no/projectweb/top/vrptw/>² for the Gehring and Homberger instances, at <http://vrp.atd-lab.inf.puc-rio.br/index.php/en/> for the Uchoa et al. instances, and at <https://www.sintef.no/projectweb/top/pdptw/> for the Li and Lim instances. We used two modes of BonnTour: the default mode that is also mostly used in practice and a high-effort mode with the only difference that more random walk iterations are performed. Table 7.1 shows results for different instance classes and sizes. The two columns under “BKS” report the total number and total length of all tours of the best known solutions in that given class. Then, we report our results both for the “BonnTour default mode” and the “BonnTour high-effort mode”. The results

²We checked all web addresses mentioned in this section on March 13, 2024.

and the gaps of the primary objective to the best known solution are highlighted in bold. The average computation time t per instance is given in seconds.

The results show that our algorithm produces good solutions in reasonable computation time on these problems, despite the overhead resulting from the more general implementation allowing time-dependent travel times. The gap is consistently below 2% in the high effort mode, and for instances with all pickups at a depot this is even true in the much faster default mode.

Table 7.1 also contains results on benchmarks for time-dependent capacitated vehicle routing with time windows by Dabia, Røpke, Van Woensel and De Kok [DRVD13], which are generated from the Solomon instances [Sol87] by assigning three different types of speed profiles to the arcs. There are 168 instances ranging from 25 to 100 customers. Solution values are reported in [DRVD13; LBS18; LMS20a; PZL21a]. For nine instances with 100 customers each, we found a better solution than any of these papers³; the “BKS” value in Table 7.1 includes these new values. The objective is to minimize the total working time, shown as distance in Table 7.1.

7.2 New benchmarks with time-dependent travel times

To demonstrate the effectiveness of our algorithms and to foster future research on vehicle routing with time-dependent travel times, we created a set of realistic instances with time-dependent travel times. These are based on maps from OpenStreetMap (OSM, publicly available at <https://www.openstreetmap.org>) and speed data that was published by Uber. Our instances are available at <https://gitlab.com/muelleratorunibonnde/vrptdt-benchmark>.

We chose 10 cities from four sub-continents, for which we found a sufficient coverage of streets with Uber speed data. We used data for the set D of the ten Mondays from January 6 to March 9, 2020 (hence excluding effects of the Covid-19 pandemic on travel times). For each road segment r and each hour $h \in \{15, \dots, 21\}$, let $D(r, h) \subseteq D$ be the set of these Mondays for which the data contains a speed profile for r and h . Then we defined speed values for r and h by computing

- the average of the Uber speed data values provided for r and (d, h) for all days $d \in D(r, h)$, if $D(r, h) \neq \emptyset$;
- $p(h) \cdot f(r)$ otherwise, where $f(r)$ denotes the *free-flow speed* on r as defined below, and $p(h)$ the average fraction of free-flow speed during $[h, h + 1)$, over all road segments fully covered by speed data within our time horizon.

Where possible we defined the free-flow speed $f(r)$ using the 85 % quantile speed values for r provided by Uber in their Q1/2020 data set, or alternatively using the OSM maxspeed value, or, as a last resort, based on the OSM highway attribute.

For each city we then chose a depot address, in most cases close to a major airport, and then defined three address sets with 2000, 1000, and 500 addresses. We picked these addresses randomly from OSM nodes tagged as shops and not more than 500 m

³better solutions (instance/value): r112/17727.65, rc104/19256.52, c204/95608.05, r205/19314.75, r210/18520.02, r211/15389.66, rc205/23638.27, rc206/20607.01, rc208/15886.30

7.2. New benchmarks with time-dependent travel times

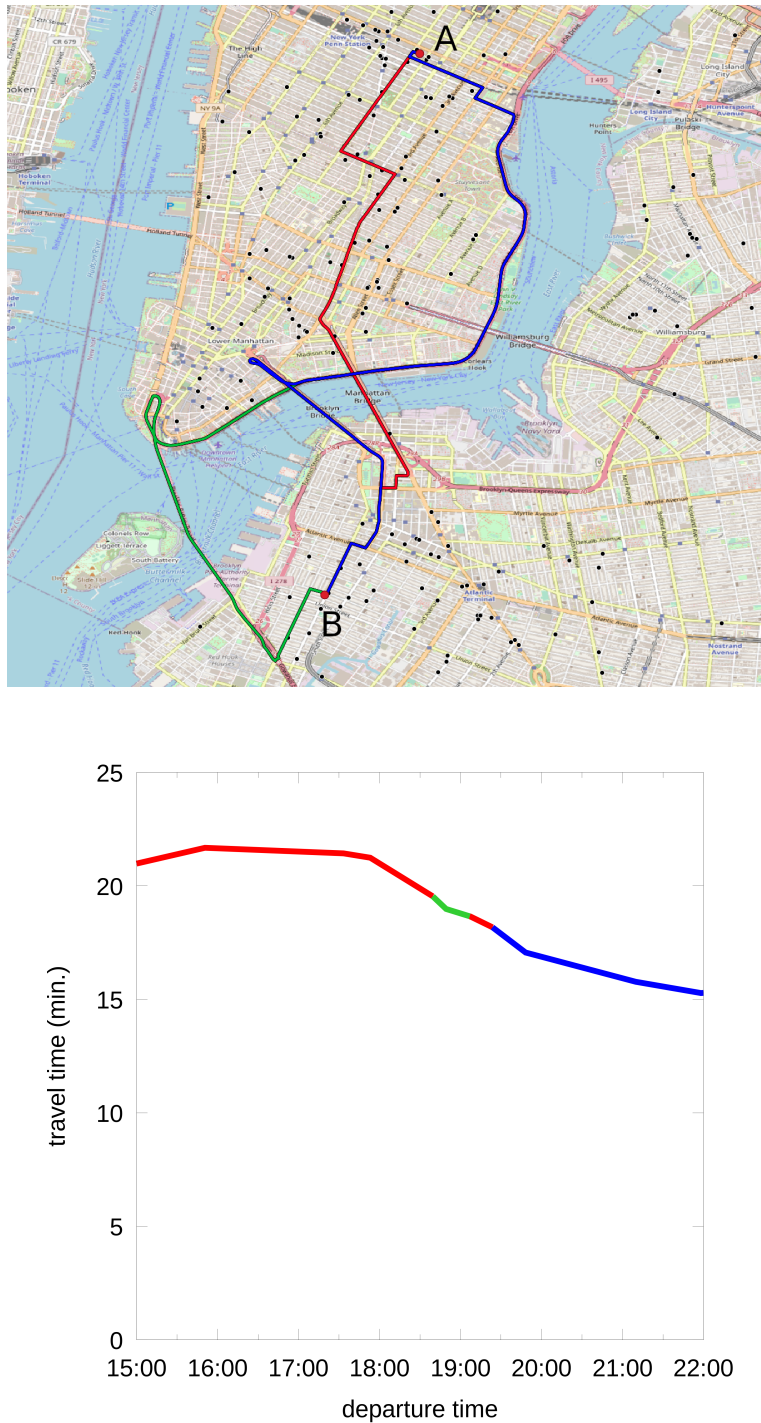


Figure 7.1: The map at the top shows three paths in the 500-customer New York instance that are fastest at some point between 3 pm and 10 pm from some customer in Manhattan (A) to a customer in Brooklyn (B). Other customers are shown as black dots. At the bottom, the corresponding travel time function from A to B is depicted. The segments are colored in the color of the path that is the fastest at this departure time.

away from a road segment covered by Uber speed data. We computed (approximately) fastest time-dependent travel times for each pair of start and end address as described in [Section 6.1.1](#). The resulting matrix of arrival time functions is part of our published data. We used the road network preprocessing described in [Section 6.1.1](#) to generate this data; now the road network is no longer needed to process our instances.

[Figure 7.1](#) shows part of the addresses in the 500-customer New York instance. Two addresses A and B are highlighted, and the paths from A to B that are the fastest path at some time between 15:00 and 22:00 are shown. [Figure 7.1](#) also shows the travel time function from A to B (to obtain the arrival time function a from the travel time function f , set $a(x) := x + f(x)$).

In all our instances there is only one type of vehicles. Each vehicle has a fixed cost of \$200 and unlimited capacity. The number of vehicles available is also unlimited. The objective is to minimize total fixed cost plus the working time cost, which we set to \$20 per hour. The traveled distance is not part of our objective function. Nevertheless, our published benchmark data also contains the length of each fastest path because it might be useful for future studies.

For each city and each address set we define two instances: one with all pickups at the depot, and one general pickup-and-delivery instance. We first describe the instances with all pickups at the depot. For each address a , we defined one item to be picked up at the depot at any time and delivered at a . Time windows for deliveries were chosen independently at random as follows. With 50 % probability we chose a one-hour time window beginning at a full or half hour chosen uniformly at random from $\{15:30, 16:00, \dots, 19:30, 20:00\}$; the remaining items have a large delivery time window from 15:30 to 21:00. We stipulate that each delivery takes three minutes and must start within the delivery time window, but is allowed to extend past its end. All tours must start at the depot no earlier than 15:00 and end at the depot at any time. Pickup at the depot takes zero time, i.e., we assume pre-loaded vehicles.

Our general pickup-and-delivery instances are based on the same addresses and arrival time functions. Here some of the addresses with a large time window are called *shops*; the other addresses (except the depot) are called *customers*. Now 80 % of the items are to be picked up at a shop and delivered to a customer, while 20 % of the items are to be picked up at a customer and delivered to a shop. Each visit at a shop or customer address takes three minutes, independent of the number of items picked up or delivered during that visit, and has to start within the time window of the address. Details are described on the website containing the instances.

7.3 Results on new benchmarks

In [Table 7.2](#) and [Table 7.3](#) we show the result of our algorithm for the instances with 500, 1000, and 2000 customers in the 10 cities. For each instance we show the number of tours and the total cost of our solution (computed by BonnTour default mode), as well as the total computation time of our algorithm. We also show the corresponding data for the high-effort mode.

7.3. Results on new benchmarks

Instance city # customers	BonnTour default mode			BonnTour high-effort mode		
	# tours	cost	t	# tours	cost	t
berlin_500	9	2859	525	9	2827	3417
cincinnati_500	9	2892	457	9	2907	2837
kyiv_500	10	3170	576	10	3146	3592
london_500	15	4847	424	15	4821	3569
madrid_500	10	3202	507	10	3169	3562
nairobi_500	10	3126	534	10	3113	3097
new_york_500	11	3432	504	11	3418	3434
san_francisco_500	12	3900	435	12	3854	2706
sao_paulo_500	13	4108	608	12	3910	3690
seattle_500	10	3194	434	10	3179	3385
total	109	34726	5004	108	34342	33289
berlin_1000	16	5027	1479	16	4981	6358
cincinnati_1000	16	5114	1150	16	5059	6239
kyiv_1000	18	5643	1281	17	5391	6770
london_1000	25	8126	959	24	7871	6085
madrid_1000	17	5437	1213	17	5421	6370
nairobi_1000	17	5361	1368	17	5311	8297
new_york_1000	18	5700	1328	17	5455	6216
san_francisco_1000	20	6500	993	20	6456	6100
sao_paulo_1000	21	6802	1314	21	6753	8085
seattle_1000	17	5397	1098	17	5389	6665
total	185	59102	12183	182	58082	67186
berlin_2000	29	9077	3584	28	8850	13077
cincinnati_2000	28	8955	2979	27	8723	12306
kyiv_2000	31	9842	3379	30	9568	13559
london_2000	42	13714	2501	40	13274	11882
madrid_2000	30	9558	3009	29	9309	14270
nairobi_2000	29	9110	3253	28	8888	16124
new_york_2000	31	9854	3220	30	9561	13851
san_francisco_2000	36	11533	2428	34	11022	12297
sao_paulo_2000	36	11493	3376	35	11259	16310
seattle_2000	29	9283	2981	29	9180	13968
total	321	102412	30711	310	99629	137643

Table 7.2: Results of the two modes of BonnTour on our new benchmark instances with all pick-ups at a single depot. In each row, the cost is rounded up to full dollars. The computation time t is given in seconds.

Instance city # customers	BonnTour default mode			BonnTour high-effort mode		
	# tours	cost	t	# tours	cost	t
berlin_500_pdp	15	4644	507	14	4404	3209
cincinnati_500_pdp	15	4704	433	14	4495	2918
kyiv_500_pdp	17	5327	470	16	4987	3044
london_500_pdp	29	9471	498	28	9261	3196
madrid_500_pdp	17	5311	636	16	5005	3857
nairobi_500_pdp	17	5394	540	16	5046	3418
new_york_500_pdp	18	5702	556	17	5380	3647
san_francisco_500_pdp	21	6670	500	19	6177	3182
sao_paulo_500_pdp	21	6650	658	20	6361	3981
seattle_500_pdp	17	5418	492	16	5126	2920
total	187	59288	5290	176	56237	33374
berlin_1000_pdp	27	8420	1264	26	8117	5758
cincinnati_1000_pdp	27	8670	971	25	8048	4776
kyiv_1000_pdp	34	10462	1049	30	9517	5590
london_1000_pdp	54	17526	1146	51	16766	6082
madrid_1000_pdp	32	10080	1333	30	9546	6194
nairobi_1000_pdp	33	10367	1222	32	10067	6354
new_york_1000_pdp	33	10484	1260	32	10086	6051
san_francisco_1000_pdp	38	12219	1157	35	11437	6226
sao_paulo_1000_pdp	41	13171	1349	38	12263	7467
seattle_1000_pdp	30	9654	1008	29	9231	5987
total	349	111049	11759	328	105074	60483
berlin_2000_pdp	54	17016	2887	51	16048	11711
cincinnati_2000_pdp	54	17225	2569	50	15958	10441
kyiv_2000_pdp	61	19362	3071	58	18398	10163
london_2000_pdp	105	33530	2649	94	30870	13577
madrid_2000_pdp	58	18548	2924	55	17506	11789
nairobi_2000_pdp	63	19957	3033	60	19051	14007
new_york_2000_pdp	66	20880	3018	61	19410	14044
san_francisco_2000_pdp	75	23985	2600	68	22048	12046
sao_paulo_2000_pdp	80	25401	3323	73	23651	14851
seattle_2000_pdp	56	17813	2655	52	16642	10681
total	672	213714	28730	622	199577	123310

Table 7.3: Results of the two modes of BonnTour on our new pickup-and-delivery benchmark instances. In each row, the cost is rounded up to full dollars. The computation time t is given in seconds.

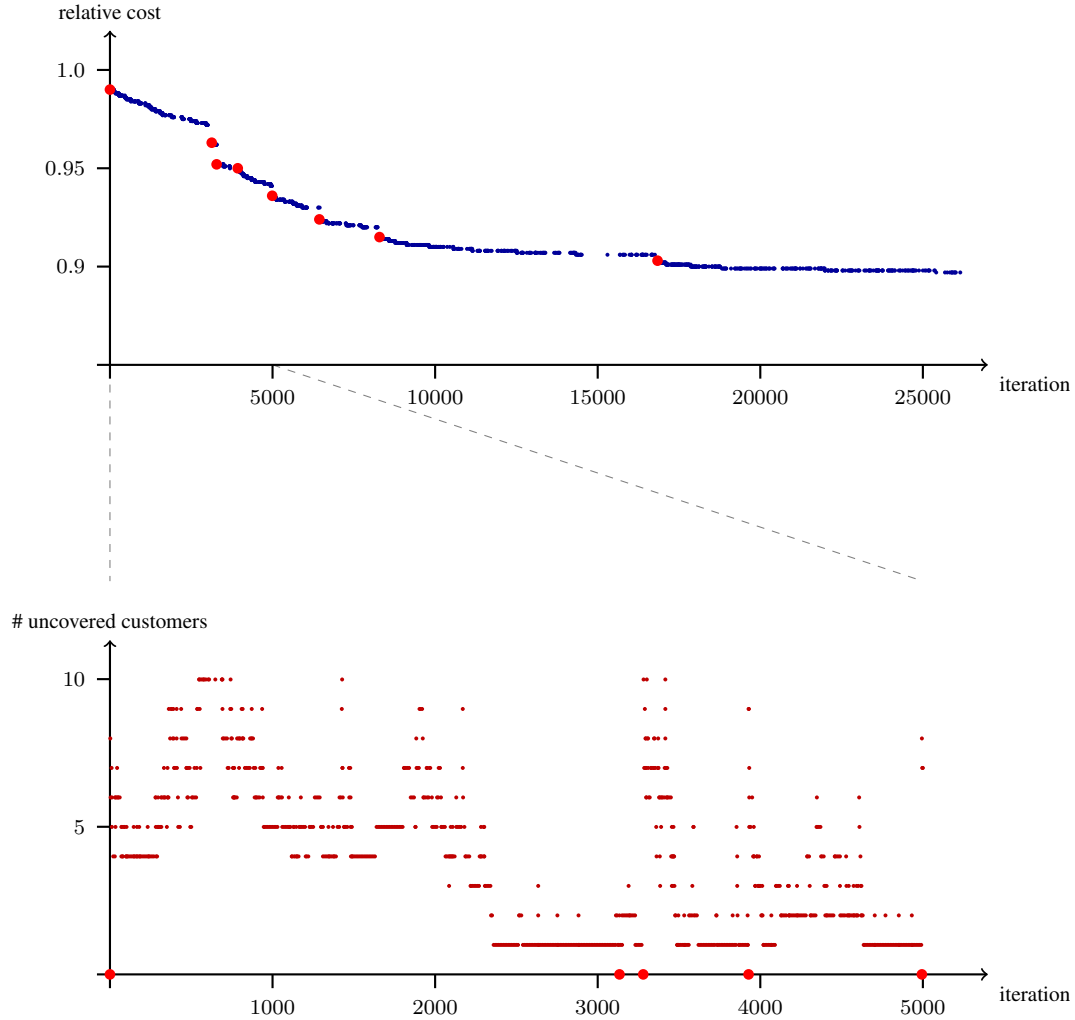


Figure 7.2: Improvements on the San Francisco benchmark instance with 2000 customers and general pickups and deliveries that are obtained by the approach described in [Section 6.1.5](#). The shown data is obtained with high-effort mode. The upper figure contains for each update of the best solution found so far one data point that shows the number of random walk iterations that have been performed and the cost of this solution relative to the cost of the solution we started with. (Recall that the initial solution is constructed as described in [Section 6.1.3](#).) We maintain two branches. In one branch, we proceed as proposed in [\[AV21\]](#) but using our local search routines described in [Section 6.1.4](#). Improvements achieved by this branch are colored blue. In the other branch, we dissolve one tour and try to reassign its customers to other tours. Improvements achieved by this branch are colored red. The bottom figure shows for the first 5000 random walk iterations the number of customers in the latter branch that are not covered by any tour. Data points that correspond to feasible solutions (all customers are covered by tours) are highlighted. Numbers greater than 10 are not depicted.

Results on the 30 new instances with all pickups at a single depot:

travel times	# tours	cost (% gap)	# late deliveries	maximum delay (sec.)
time-dependent	600	192052	0	0
worst-case	653	206877 (7.7)	0	0
50 % worst, 50 % average	629	199496 (3.9)	124	197
average	600	191936 (-0.1)	1656	763

Results on the 30 new general pickup and delivery instances:

travel times	# tours	cost (% gap)	# late pickups/deliveries	maximum delay (sec.)
time-dependent	1126	360887	0	0
worst-case	1257	397555 (10.2)	0	0
50 % worst, 50 % average	1184	375862 (4.1)	406	337
average	1111	356550 (-1.2)	3371	1094

Table 7.4: Results when planning with time-dependent and constant travel times. Each row shows the total result summed over 30 new benchmark instances with 35000 tasks in total. For the single-depot instances (top), 17368 deliveries have a one-hour time window. For the pickup-and-delivery instances (bottom), 4638 pickups and 17051 deliveries have a one-hour time window. All results are evaluated with time-dependent travel times.

Recall that the high-effort mode differs from the default mode only by the number of iterations of our “ruin-and-recreate” framework described in [Section 6.1.5](#), which is based on an approach proposed by Accorsi and Vigo [[AV21](#)]. [Figure 7.2](#) illustrates improvements on one of our new benchmark instances obtained by this framework. Recall that in contrast to [[AV21](#)], we maintain two branches. In one of these branches, we actively try to reduce the number of tours, which leads to significant improvements as shown in [Figure 7.2](#). For the considered instance, 8 tours have been saved in comparison to the initial solution we started with. The total cost has been reduced by more than 10 %.

As said in the beginning, most other vehicle routing algorithms—both academic and industrial—still work with fixed travel times. To assess the benefit of handling time-dependent travel times, we conducted several experiments where we tried to compute good solutions when working only with fixed travel times. To this end, we applied our algorithm in exactly the same way (in high-effort mode), but using constant travel times. As travel times we took the worst case (in the domain of our ATFs), the average travel time, and the arithmetic mean of these two. In the end we evaluated the resulting tours with respect to the time-dependent travel times.

[Table 7.4](#) shows that working with worst-case travel times increases the cost by roughly 8–10 %. Working with average travel times leads to similar total cost but many unacceptable time window violations. The arithmetic mean of worst-case and average travel times might be an acceptable compromise in practice, but, as expected, the results are still much worse than using time-dependent travel times.

Another option to work with fixed travel times and still meet most of the time windows, is to work with average travel times but reduce the length of the time windows upfront. As [Table 7.5](#) shows, a hard reduction of every time window by 10 minutes ends up with fewer time window violations, but already leads to more than 4 % higher cost.

7.3. Results on new benchmarks

Results on the 30 new instances with all pickups at a single depot:

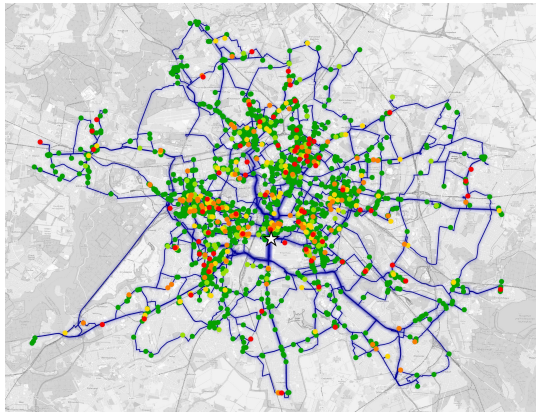
travel times	time windows	# tours	cost (% gap)	# deliveries with slack within:			
				[10, 15)	[5, 10)	[0,5)	late
time-dependent	original	600	192052	1691	1721	2798	0
time-dependent	soft reduction	606	195038 (1.6)	1091	792	263	0
time-dependent	aggressive soft reduction	617	197856 (3.0)	691	404	80	0
average	original	600	191936 (-0.1)	1654	1702	1860	1656
average	reduced by 10 min	635	200744 (4.5)	2015	1472	388	35*
average	soft reduction	607	195292 (1.7)	1872	1120	592	214
average	aggressive soft reduction	615	197606 (2.9)	1763	873	373	157

Results on the 30 new general pickup and delivery instances:

travel times	time windows	# tours	cost (% gap)	# pickups/deliveries with slack within:			
				[10, 15)	[5, 10)	[0,5)	late
time-dependent	original	1126	360887	2326	2561	4629	0
time-dependent	soft reduction	1131	364194 (0.9)	2035	1818	1189	0
time-dependent	aggressive soft reduction	1155	371201 (2.9)	1673	1449	758	0
average	original	1111	356550 (-1.2)	2381	2538	2824	3371
average	reduced by 10 min	1187	377126 (4.5)	2997	2573	1235	604*
average	soft reduction	1131	363727 (0.8)	2881	2134	1514	1154
average	aggressive soft reduction	1147	369559 (2.4)	3012	2172	1378	878

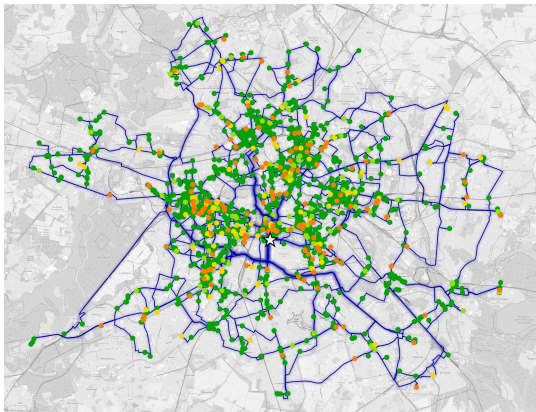
Table 7.5: Results when planning with the original hard time windows, soft time window reductions, or a hard reduction of every time window by 10 minutes. Each row shows the total result summed over 30 new benchmark instances with 35000 tasks in total. For the single-depot instances (top), 17368 deliveries have a one-hour time window. For the pickup-and-delivery instances (bottom), 4638 pickups and 17051 deliveries have a one-hour time window. All results are evaluated with time-dependent travel times and the original hard time windows. The costs do not include the artificial penalty for soft time window reductions. The last four columns contain the number of deliveries with a certain slack until the deadline, e.g. [10, 15) contains the number of pickups/deliveries that happen at least 10 minutes but less than 15 minutes before the deadline. *When planning with a hard reduction of every time window by 10 minutes, one customer in the single-depot São Paulo instances with 1000 and 2000 customers and one customer in the pickup-and-delivery London instance with 500 customers cannot be visited within its reduced delivery time window when starting at the depot at 3 pm. Hence, this customer is not visited at all in the computed solution (and counted as late in the table), which is another drawback of this approach.

As described in [Section 5.2.2](#) we can handle soft time window reductions. If we impose an artificial penalty of \$1 when arriving less than 15 but at least 10 minutes before the deadline, \$2 within the next 5 minutes, and \$4 in the final 5 minutes and still strictly forbid arrivals after the deadline, the overall algorithm will avoid arrivals late in a time window wherever this is possible at moderate cost. [Table 7.5](#) shows the results: most arrivals at the end of the time window can indeed be avoided at a moderate cost increase. If we double the artificial penalties (aggressive soft reduction in [Table 7.5](#)), the effect is stronger as expected. Soft time window reductions also help in combination with average travel times, but do not prevent late arrivals completely. The combination of time-dependent travel times and soft time window reductions leads to schedules that are cheap and significantly more robust; this is also illustrated by [Figure 7.3](#).

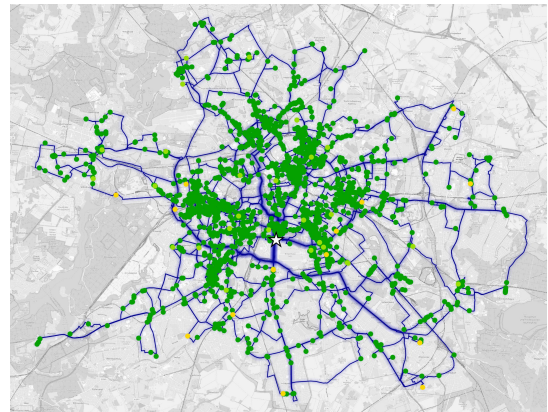


- ☆ Depot
- 15 min $\leq$ slack
- 10 min $\leq$ slack < 15 min
- 5 min $\leq$ slack < 10 min
- 0 min $\leq$ slack < 5 min
- slack < 0 min

a) Constant average travel times: 28 tours, cost = 8859



b) Time-dependent travel times: 28 tours, cost = 8850



c) Time-dependent travel times and aggressive soft time window reduction: 29 tours, cost = 9149

Figure 7.3: Illustrating some results of Table 7.5: three solutions obtained with high-effort mode on our Berlin benchmark instance with 2000 customers and all pickups at a single depot. Working with average travel times misses many time windows (red). Planning with time-dependent travel times and soft time window reductions largely avoids deliveries very late in the time window (yellow, orange).

7.4 Demonstrating selected BonnTour features

We conclude this chapter by showcasing the practical use of two BonnTour features that have been discussed in [Section 6.2](#). The experimental results shown in this section are generated with default mode, which is the one typically used in practice.

7.4.1 Multiple pickup and delivery locations

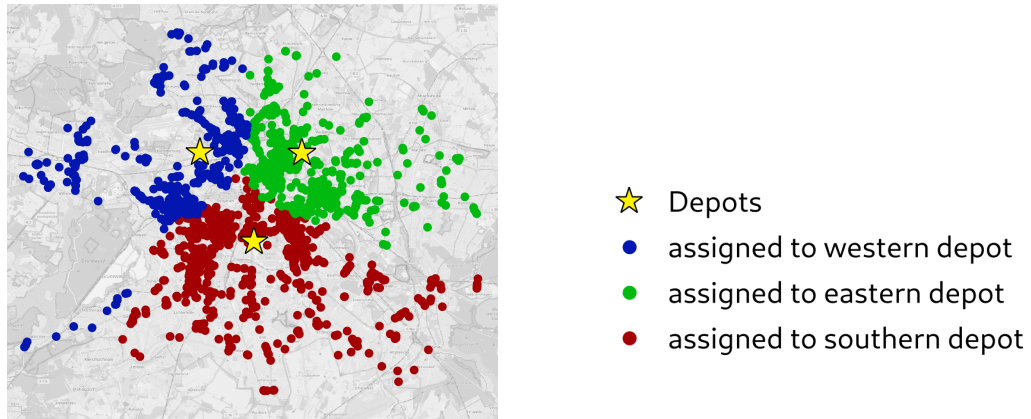
As discussed in [Section 6.2.2](#), it is possible to pass for each item alternative pickup and delivery locations to BonnTour. We demonstrate how to leverage this feature to model multi-depot scenarios in which each good is available at several depots. For this, we modify the instances of our new benchmark set with all pickups at a single depot. For each city, we choose two additional addresses that will serve as depot so that in total there are three depots in each city. We assume that for each item, identical copies are available at any of these depots, resulting in three feasible pickup locations. All tours must start at one of the three depots (no earlier than 15:00 as in the original benchmark), and end at any time at the same depot where the tour started.

We compare our approach discussed in [Section 6.2.2](#) to the naïve approach of assigning each item to the depot for which the average travel time from the depot to the delivery location and back to the depot is smallest. Recall that in our approach, we do not fix the depot assignment upfront but instead re-optimize the address choice whenever a tour changes.

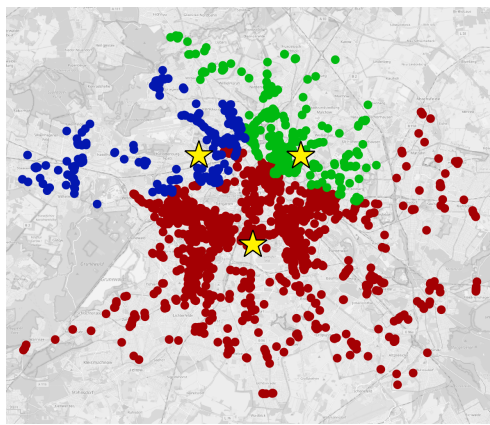
We conducted experiments once with the original time windows, and once with large uniform delivery time windows from 15:30 to 21:00. [Figure 7.4](#) shows the depot assignment for the Berlin instance with 2000 customers. As shown in [Table 7.6](#), assigning each item to the closest depot in advance results in 2–4 % higher cost compared to our flexible approach. In the case of heterogeneous delivery time windows, it is particularly advantageous not to specify the depot assignment in advance, which is also illustrated by [Figure 7.4](#).

time windows	depot assignment	# tours	cost (% gap)		t
original	flexible	614	193325		2369
	assignment to closest depot	637	201128	(4.0)	1555
large & uniform	flexible	487	153206		4828
	assignment to closest depot	497	156518	(2.2)	3852

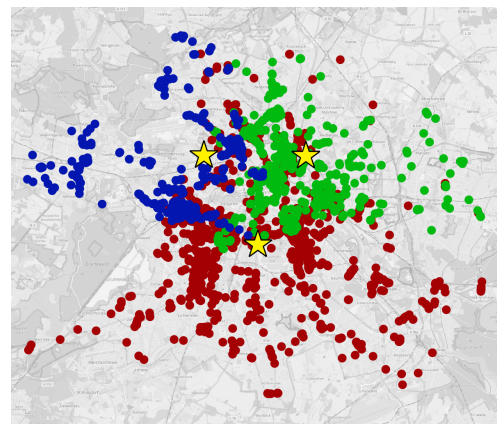
Table 7.6: Results when assigning delivery requests in advance to the closest depot, and results with flexible assignment of the delivery requests to depots as described in [Section 6.2.2](#). Each row shows the total result summed over 30 instances. The average computation time t per instance is given in seconds.



a) Assignment to closest depot. Large & uniform time windows: 25 tours, cost = 7742; original time windows: 30 tours, cost = 9403



b) Flexible depot assignment and large & uniform time windows: 24 tours, cost = 7521



c) Flexible depot assignment and original time windows: 29 tours, cost = 9045

Figure 7.4: Illustrating some results of Table 7.6: three solutions obtained with default mode on the modified Berlin benchmark instance with 2000 customers. Following the naïve approach of assigning each delivery request to the closest depot comes at a significant loss in quality, especially with non-uniform time windows, which is also reflected in Table 7.6.

7.4.2 Compact tours

As described in [Section 6.2.3](#), we can charge artificial penalties for spread tours. This way, we get more compact solutions that are often preferred in practice because driver-specific area knowledge can be exploited when assigning drivers to tours. We conducted experiments on our new benchmark set, once with the original time windows, and once with large uniform time windows from 15:30 to 21:00.

Additionally, we ran BonnTour on selected instances that model real-world vehicle routing problems. As mentioned earlier, BonnTour is used successfully in various practical scenarios thanks to the cooperation with our industry partner [Greenplan](#). Each of the chosen instance classes has specific characteristics, which showcases the variety of use cases in which the BonnTour algorithm is leveraged:

- **C-010-3:** This instance set contains 7 instances with general pickups and deliveries. A heterogeneous fleet is given with several vans and trucks (with limited availability) that differ in terms of road usage, capacity, and driving speeds. These instances contain items that are flagged as “direct transport” to model transferring a trailer that blocks the access to the actual loading space (cf. [Section 6.2](#)). Some items are incompatible to certain vehicles.
- **C-018-2:** This instance set consists of 8 instances. All items must be picked up at a single depot. However, some items are only available at noon or in the evening, whereas other items can be picked up in the morning. Hence, in each reasonably good solution, the drivers must perform several trips. The delivery time windows at customers are tight as well.
- **C-020:** This instance set contains 6 large-scale instances with up to 13,000 items at more than 7,000 different addresses. Pickup and delivery time windows are large and uniform, and the operation area is relatively small, resulting in tours that handle more than 250 items on average.
- **C-028-2:** This instance set consists of 5 instances that model the operation of service technicians. There are many more requests than can be fulfilled by the technicians. Penalties for rejected requests are used to model priorities.
- **C-030-3:** This instance set contains 25 instances with drivers that are available at different times of the day. For some drivers we need to schedule more than one break due to the given break regulations. Many items have restrictive time windows.

As shown in [Table 7.7](#) and [Figure 7.5](#), we get more compact tours at comparable costs when charging moderate artificial penalties for spread tours. As one can see in [Table 7.7](#), the more complex the instances, the higher the average spread of the tours. This is not surprising: On complex instances with tight time windows, we benefit from our flexible solution approach that is not tied to strict assignments of drivers to specific areas. Charging high artificial costs for spread tours on such instances already leads to a noticeable cost increase.

Results on the 30 new instances with all pickups at a single depot:

time windows	charged spread cost	# tours	cost (% gap)	avg. spread per tour
original	none	622	197710	0.51
	moderate	616	196544 (-0.6)	0.32
	high	621	198586 (0.4)	0.23
large & uniform	none	491	156334	0.2
	moderate	490	156239 (-0.1)	0.12
	high	489	156412 (0.0)	0.08

Results on the 30 new general pickup and delivery instances:

time windows	charged spread cost	# tours	cost (% gap)	avg. spread per tour
original	none	1203	382817	0.77
	moderate	1206	383266 (0.1)	0.74
	high	1216	385958 (0.8)	0.65
large & uniform	none	878	278921	0.63
	moderate	884	280271 (0.5)	0.57
	high	894	283859 (1.8)	0.43

Results on real-world instances:

instance class	charged spread cost	# tours	cost (% gap)	avg. spread per tour
C-010-3	none	297	31963710	0.51
	moderate	297	31881402 (-0.3)	0.44
	high	297	32627404 (2.1)	0.38
C-018-2	none	430	81185	0.81
	moderate	440	81373 (0.2)	0.77
	high	446	82379 (1.5)	0.72
C-020	none	256	97469	0.19
	moderate	256	97617 (0.2)	0.08
	high	257	98124 (0.7)	0.05
C-028-2	none	292	118383	0.47
	moderate	293	118566 (0.2)	0.45
	high	292	118973 (0.5)	0.4
C-030-3	none	442	565188	0.61
	moderate	442	564887 (-0.1)	0.53
	high	444	566533 (0.2)	0.41

Table 7.7: Results when planning with and without artificial extra costs for spread tours. The reported costs do not include the artificial penalties for spread tours. The last column shows the average spread of the computed tours. Recall that the spread of a tour is a number between 0 and 1 and is computed as described in [Section 6.2.3](#). We remark that the costs in the bottom table are reported in the national currency of the corresponding instance class and are hence not comparable with each other.

7.4. Demonstrating selected BonnTour features



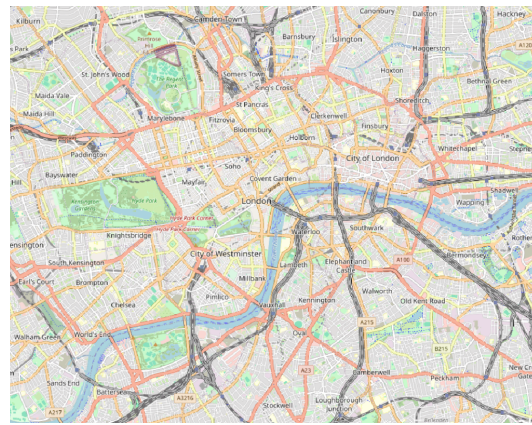
a) No incentive for compact tours: 31 tours, cost = 10153, average spread per tour = 0.21



b) Moderate incentive for compact tours: 31 tours, cost = 10187, average spread per tour = 0.15



c) High incentive for compact tours: 31 tours, cost = 10177, average spread per tour = 0.09



d) Map of the city center of London for which tours are shown in the images **a)**–**c)**. The map is taken from OpenStreetMap.

Figure 7.5: Illustrating some results of Table 7.7: three excerpts of solutions obtained with default mode on our London benchmark instance with 2000 customers, all pickups at a single depot, and large uniform delivery time windows from 15:30 to 21:00. The images **a)**–**c)** show the tours delivering items in the city center that have been computed by BonnTour with different penalties for spread tours. Driving actions from/to the depot (which is located to the west of the depicted excerpts) are not depicted. The delivery location of an item s is colored by the fraction x_s of friends that are covered by the same tour (with close friends having a higher weighting). Delivery locations of items s with $x_s \leq 0.5$ are colored black, and delivery locations of items with $x_s = 1.0$ are colored white, using some interpolation for values in between. Charging artificial costs for spread tours yields more compact tours at comparable costs. The spread of the depicted tours ranges from 0.01 (red tour on the left side of image **c)**) to 0.47 (red tour in the center of image **a)**).

Summary

In this thesis, we studied **TSP** generalizations and vehicle routing problems from both theoretical and practical perspectives.

We improved the approximation guarantee for two natural generalizations of the **TSP** with obvious applications in logistics. For the prize-collecting **TSP**, we presented two new approximation algorithms, both beating the previous best known approximation guarantee of roughly 1.914 by Goemans [Goe09]. We pushed the approximation ratio to 1.599, which significantly reduces the gap between the approximability of the **TSP** and its prize-collecting variant. Our bound is with respect to the optimum value of the natural linear programming relaxation of this problem, hence giving the same upper bound on its integrality gap.

The second **TSP** generalization for which we improved the approximation guarantee is the capacitated vehicle routing problem. Our algorithm also gives improved guarantees for the unit-demand case and for the variant in which customer demands can be split and served by several tours. For all three variants, our improvement is in the order of 10^{-3} in comparison to the guarantees obtained by Haimovich and Rinnooy Kan [HR85] and Altinkemer and Gavish [AG87] in their classical analysis of the tour partitioning heuristic. For general unsplittable demands, the approximation guarantee has recently been further improved Friggstad, Mousavi, Rahgoshay, and Salavatipour [FMRS22]. In their work, they remark that running their algorithm and ours, and returning the better solution slightly improves the approximation ratio they obtained. In our work, as a by-product, we also improved the approximation guarantee for regret-bounded vehicle routing from 15 [FS17] to 10, which directly gives an 11-approximation algorithm for the school bus problem using the simple reduction in [BGKS11].

In the second part of this thesis, we discussed a heuristic that is designed to compute high-quality solutions to real-world vehicle routing problems in reasonable computation time. Our algorithm, called BonnTour, is developed in cooperation with our industry partner **Greenplan**.

It is well known that travel times in cities vary a lot over the day, mainly due to congestion. Hence, we consider a road network with time-dependent travel times, which leads to much more reliable route plans. Thanks to our very general model that includes, e.g., general pickup and delivery locations, hard and soft time windows, and heterogeneous vehicle fleets, BonnTour is successfully used in various practical scenarios.⁴

We conducted extensive experiments that demonstrate the quality of our algorithm.

⁴A selection of partners relying on the BonnTour algorithm can be found on the Greenplan website: <https://greenplan.de>.

7.4. Demonstrating selected BonnTour features

First, we showed that on public vehicle routing benchmarks with constant travel times, we only get slightly worse results than algorithms that are designed to perform best on these instances, despite the huge handicap that BonnTour is designed for a much more general model which comes at a significant running time overhead. On some benchmark instances with time-dependent travel times, we report new best known solutions.

Due to the lack of realistic and large-scale benchmark instances for vehicle routing with time-dependent travel times, we designed new benchmark instances with up to 2000 customers that are based on real-world data.⁵ We compare the tour plans obtained by BonnTour with those obtained when using constant travel time estimates. When evaluating the latter with the actual time-dependent travel times, one either gets a lot of time window violations (when using average travel times) or solutions of significantly higher cost (when using pessimistic estimates). Using the arithmetic mean of average and worst-case travel times leads to an unsatisfying trade-off. Finally, we demonstrated some features of our algorithm, both on our new benchmark set as well as on real-world instances from [Greenplan](#).

⁵Our instances are publicly available at <https://gitlab.com/muelleratorunibonnde/vrptdt-benchmark>.

Bibliography

- [ABHK11] A. Archer, M. Bateni, M. Hajiaghayi, and H. Karloff. Improved approximation algorithms for prize-collecting Steiner tree and TSP. In: *SIAM Journal on Computing* 40.2 (2011), pp. 309–332 (cit. on pp. [3](#), [13](#), [14](#)).
- [ABLM07] G. Ausiello, V. Bonifaci, S. Leonardi, and A. Marchetti-Spaccamela. Prize-collecting traveling salesman and related problems. In: *Handbook of Approximation Algorithms and Metaheuristics*. Chapman and Hall, 2007. Chap. 40, pp. 1–13 (cit. on p. [17](#)).
- [ACL10] A. Adamaszek, A. Czumaj, and A. Lingas. PTAS for k -tour cover problem on the plane for moderately large values of k . In: *International Journal of Foundations of Computer Science* 21 (2010), pp. 893–904 (cit. on p. [57](#)).
- [ACR03] D. Applegate, W. Cook, and A. Rohe. Chained Lin-Kernighan for large traveling salesman problems. In: *INFORMS Journal on Computing* 15.1 (2003), pp. 82–92 (cit. on p. [101](#)).
- [AG87] K. Altinkemer and B. Gavish. Heuristics for unequal weight delivery problems with a fixed error guarantee. In: *Operations Research Letters* 6 (1987), pp. 149–158 (cit. on pp. [4](#), [54–58](#), [123](#)).
- [AGG20] T. Adamo, G. Ghiani, and E. Guerriero. An enhanced lower bound for the time-dependent travelling salesman problem. In: *Computers & Operations Research* 113 (2020), Article 104795 (cit. on p. [83](#)).
- [AGGGP19] A. Arigliano, G. Ghiani, A. Grieco, E. Guerriero, and I. Plana. Time-dependent asymmetric traveling salesman problem with time windows: Properties and an exact algorithm. In: *Discrete Applied Mathematics* 261 (2019), pp. 28–39 (cit. on p. [83](#)).
- [AGHJM24a] A. Ahmadi, I. Gholami, M. Hajiaghayi, P. Jabbarzade, and M. Mahdavi. 2-approximation for prize-collecting Steiner forest. In: *Proceedings of the 35th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2024, pp. 669–693 (cit. on p. [17](#)).
- [AGHJM24b] A. Ahmadi, I. Gholami, M. Hajiaghayi, P. Jabbarzade, and M. Mahdavi. Prize-collecting Steiner tree: A 1.79 approximation. In: *Proceedings of the 56rd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*. 2024, to appear (cit. on p. [16](#)).

- [AGMOS17] A. Asadpour, M. X. Goemans, A. Madry, S. Oveis Gharan, and A. Saberi. An $O(\log n / \log \log n)$ -approximation algorithm for the asymmetric traveling salesman problem. In: *Operations Research* 65.4 (2017), pp. 1043–1061 (cit. on p. 28).
- [AGS19] F. Arnold, M. Gendreau, and K. Sörensen. Efficiently solving very large-scale routing problems. In: *Computers & Operations Research* 107 (2019), pp. 32–42 (cit. on p. 84).
- [AKK01] T. Asano, N. Katoh, and K. Kawashima. A new approximation algorithm for the capacitated vehicle routing problem on a tree. In: *Journal of Combinatorial Optimization* 5 (2001), pp. 213–231 (cit. on p. 57).
- [AKTT97] T. Asano, N. Katoh, H. Tamaki, and T. Tokuyama. Covering points in the plane by k -tours: Towards a polynomial time approximation scheme for general k . In: *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC)*. 1997, pp. 275–283 (cit. on p. 57).
- [ALM00] G. Ausiello, S. Leonardi, and A. Marchetti-Spaccamela. On salesmen, repairmen, spiders, and other traveling agents. In: *Proceedings of the 4th Italian Conference on Algorithms and Complexity*. 2000, pp. 1–16 (cit. on p. 17).
- [AS00] P. K. Agarwal and M. Sharir. Davenport–Schinzel sequences and their geometric applications. In: *Handbook of Computational Geometry*. Elsevier, 2000, pp. 1–47 (cit. on p. 87).
- [AS04] A. A. Ageev and M. I. Sviridenko. Pipe rounding: A new method of constructing algorithms with proven performance guarantee. In: *Journal of Combinatorial Optimization* 8.3 (2004), pp. 307–328 (cit. on pp. 18, 27).
- [AS19] F. Arnold and K. Sörensen. Knowledge-guided local search for the vehicle routing problem. In: *Computers & Operations Research* 105 (2019), pp. 32–46 (cit. on p. 82).
- [AV21] L. Accorsi and D. Vigo. A fast and scalable heuristic for the solution of large-scale capacitated vehicle routing problems. In: *Transportation Science* 55.4 (2021), pp. 832–856 (cit. on pp. 81, 82, 96, 100, 101, 114, 115).
- [Bal89] E. Balas. The prize collecting traveling salesman problem. In: *Networks* 19.6 (1989), pp. 621–636 (cit. on pp. 14, 17).
- [BBCM04] N. Bansal, A. Blum, S. Chawla, and A. Meyerson. Approximation algorithms for deadline-TSP and vehicle routing with time-windows. In: *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*. 2004, pp. 166–174 (cit. on p. 82).

-
- [BCEHKM11] M. Bateni, C. Chekuri, A. Ene, M. Hajiaghayi, N. Korula, and D. Marx. Prize-collecting Steiner problems on planar graphs. In: *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2011, pp. 1028–1049 (cit. on p. 17).
 - [BCKLMM07] A. Blum, S. Chawla, D. R. Karger, T. Lane, A. Meyerson, and M. Minkoff. Approximation algorithms for orienteering and discounted-reward TSP. In: *SIAM Journal on Computing* 37 (2007), pp. 653–670 (cit. on pp. 17, 60, 67).
 - [BDO06] A. Bompadre, M. Dror, and J. B. Orlin. Improved bounds for vehicle routing solutions. In: *Discrete Optimization* 3 (2006), pp. 299–316 (cit. on p. 57).
 - [Bec18] A. Becker. A tight $4/3$ approximation for capacitated vehicle routing in trees. In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*. 2018, 3:1–3:15 (cit. on p. 57).
 - [BETV24] J. Blauth, A. Ellerbrock, V. Traub, and J. Vygen. Cost allocation for set covering: The happy nucleolus. 2024. arXiv: [2401.04040](https://arxiv.org/abs/2401.04040) (cit. on p. 98).
 - [BFJ95] J. Bang-Jensen, A. Frank, and B. Jackson. Preserving and increasing local edge-connectivity in mixed graphs. In: *SIAM Journal on Discrete Mathematics* 8.2 (1995), pp. 155–178 (cit. on pp. 15, 16, 19, 36, 50).
 - [BGKS11] A. Bock, E. Grant, J. Könnemann, and L. Sanità. The school bus problem on trees. In: *Algorithmica* 67 (2011), pp. 10–19 (cit. on pp. 57, 75, 123).
 - [BGSV13] G. V. Batz, R. Geisberger, P. Sanders, and C. Vetter. Minimum time-dependent travel times with contraction hierarchies. In: *ACM Journal of Experimental Algorithmics* 18 (2013), Article 1.4 (cit. on pp. 81, 82, 88, 96).
 - [BGSW93] D. Bienstock, M. X. Goemans, D. Simchi-Levi, and D. Williamson. A note on the prize collecting traveling salesman problem. In: *Mathematical Programming* 59 (1993), pp. 413–420 (cit. on pp. 2, 3, 14, 18–20).
 - [BHMSTTV22] J. Blauth, S. Held, D. Müller, N. Schlömlberg, V. Traub, T. Tröbst, and J. Vygen. Vehicle routing with time-dependent travel times: Theory, practice, and benchmarks. 2022. arXiv: [2205.00889](https://arxiv.org/abs/2205.00889) (cit. on pp. 5, 79, 87, 88, 91, 93).
 - [BKN24] J. Blauth, N. Klein, and M. Nägele. A better-than-1.6-approximation for prize-collecting TSP. In: *Proceedings of the 25th Conference on Integer Programming and Combinatorial Optimization (IPCO)*. 2024, to appear (cit. on pp. 3, 15, 39).

- [BKS18] A. Becker, P. N. Klein, and D. Saulpic. Polynomial-time approximation schemes for k -center, k -median, and capacitated vehicle routing in bounded highway dimension. In: *Proceedings of the 26th Annual European Symposium on Algorithms (ESA)*. 2018, 8:1–8:15 (cit. on p. 57).
- [BKS19] A. Becker, P. N. Klein, and A. Schild. A PTAS for bounded-capacity vehicle routing in planar graphs. In: *Proceedings of the 16th International Symposium on Algorithms and Data Structures (WADS)*. 2019, pp. 99–111 (cit. on p. 57).
- [Bla20] J. Blauth. An improved approximation algorithm for the capacitated vehicle routing problem. Master’s thesis, University of Bonn. 2020 (cit. on pp. 4, 53–56, 58, 60–62, 65).
- [BN23] J. Blauth and M. Nägele. An improved approximation guarantee for prize-collecting TSP. In: *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC)*. 2023, pp. 1848–1861 (cit. on pp. 2, 3, 15, 18).
- [BTV23] J. Blauth, V. Traub, and J. Vygen. Improving the approximation ratio for capacitated vehicle routing. In: *Mathematical Programming* 197 (2023), pp. 451–497 (cit. on pp. 4, 53, 62, 65).
- [CCPV11] G. Calinescu, C. Chekuri, M. Pál, and J. Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. In: *SIAM Journal on Computing* 40.6 (2011), pp. 1740–1766 (cit. on pp. 18, 27, 31).
- [CDV08] D. Chakrabarty, N. R. Devanur, and V. V. Vazirani. New geometry-inspired relaxations and algorithms for the metric Steiner tree problem. In: *Proceedings of the 13th Conference on Integer Programming and Combinatorial Optimization (IPCO)*. 2008, pp. 344–358 (cit. on p. 16).
- [Chr22] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. In: *Operations Research Forum* 3 (2022) (cit. on pp. 9, 16).
- [Chr76] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie Mellon University. 1976 (cit. on pp. 9, 16, 40).
- [CJJ20] T.-H. H. Chan, H. Jiang, and S. H.-C. Jiang. A unified PTAS for prize collecting TSP and Steiner tree problem in doubling metrics. In: *ACM Transactions on Algorithms* 16.2 (2020), pp. 1–23 (cit. on p. 17).
- [CKP12] C. Chekuri, N. Korula, and M. Pál. Improved algorithms for orienteering and related problems. In: *ACM Transactions on Algorithms* 8.3 (2012), pp. 1–27 (cit. on pp. 17, 82).

-
- [CLQL18] T. Curtois, D. Landa-Silva, Y. Qu, and W. Laesanklang. Large neighbourhood search with adaptive guided ejection search for the pickup and delivery problem with time windows. In: *EURO Journal on Transportation and Logistics* 7 (2018), pp. 151–192 (cit. on p. 102).
- [CVZ10] C. Chekuri, J. Vondrak, and R. Zenklusen. Dependent randomized rounding via exchange properties of combinatorial structures. In: *Proceedings of the 51st IEEE Annual Symposium on Foundations of Computer Science (FOCS)*. 2010, pp. 575–584 (cit. on pp. 27, 28).
- [DFPS22] S. Dezfuli, Z. Friggstad, I. Post, and C. Swamy. Combinatorial algorithms for rooted prize-collecting walks and applications to orienteering and minimum-latency problems. In: *Proceedings of the 23rd Conference on Integer Programming and Combinatorial Optimization (IPCO)*. 2022, pp. 195–209 (cit. on p. 17).
- [Dia17] I. Diarrassouba. On the complexity of the separation problem for rounded capacity inequalities. In: *Discrete Optimization* 25 (2017), pp. 86–104 (cit. on p. 75).
- [DM15] A. Das and C. Mathieu. A quasi-polynomial time approximation scheme for Euclidean capacitated vehicle routing. In: *Algorithmica* 73 (2015), pp. 115–142 (cit. on p. 57).
- [DMCRG08] A. V. Donati, R. Montemanni, N. Casagrande, A. E. Rizzoli, and L. M. Gambardella. Time dependent vehicle routing problem with a multi ant colony system. In: *European Journal of Operational Research* 185.3 (2008), pp. 1174–1191 (cit. on pp. 82, 84).
- [DRVD13] S. Dabia, S. Røpke, T. Van Woensel, and T. De Kok. Branch and price for the time-dependent vehicle routing problem with time windows. In: *Transportation Science* 47.3 (2013), pp. 380–396 (cit. on pp. 82–84, 108, 109).
- [Edm70] J. Edmonds. Submodular functions, matroids and certain polyhedra. In: *Proceedings of the Calgary International Conference on Combinatorial Structures and Their Applications 1969*. 1970, pp. 69–87 (cit. on p. 8).
- [Edm73] J. Edmonds. Edge-disjoint branchings. In: *Combinatorial algorithms*. Academic Press, 1973, pp. 91–96 (cit. on p. 19).
- [EJ73] J. Edmonds and E. Johnson. Matching, Euler tours and the chinese postman. In: *Mathematical Programming* 5 (1973), pp. 88–124 (cit. on p. 8).
- [EK72] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. In: *Journal of the ACM* 19.2 (1972), pp. 248–264 (cit. on p. 8).

- [FGG04] B. Fleischmann, M. Gietz, and S. Gnutzmann. Time-varying travel times in vehicle routing. In: *Transportation Science* 38.2 (2004), pp. 160–173 (cit. on p. 83).
- [FHS14] L. Foschini, J. Hershberger, and S. Suri. On the complexity of time-dependent shortest paths. In: *Algorithmica* 68.4 (2014), pp. 1075–1097 (cit. on p. 87).
- [Fig12] M. A. Figliozzi. The time dependent vehicle routing problem with time windows: Benchmark problems, an efficient solution algorithm, and solution characteristics. In: *Transportation Research Part E: Logistics and Transportation Review* 48.3 (2012), pp. 616–636 (cit. on pp. 82, 84).
- [FMRS22] Z. Friggstad, R. Mousavi, M. Rahgoshay, and M. R. Salavatipour. Improved approximations for capacitated vehicle routing with unsplittable client demands. In: *Proceedings of the 23rd Conference on Integer Programming and Combinatorial Optimization (IPCO)*. 2022, pp. 251–261 (cit. on pp. 4, 55, 123).
- [FP93] C. Foisy and J.-Y. Potvin. Implementing an insertion heuristic for vehicle routing on parallel hardware. In: *Computers & Operations Research* 20.7 (1993), pp. 737–745 (cit. on pp. 81, 98, 99).
- [Fra92] A. Frank. On a theorem of Mader. In: *Discrete Mathematics* 101.1 (1992), pp. 49–57 (cit. on pp. 40, 48).
- [FS14] Z. Friggstad and C. Swamy. Approximation algorithms for regret-bounded vehicle routing and applications to distance-constrained vehicle routing. In: *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC)*. 2014, pp. 744–753 (cit. on pp. 56, 57, 60, 75).
- [FS17] Z. Friggstad and C. Swamy. Compact, provably-good LPs for orienteering and regret-bounded vehicle routing. In: *Proceedings of the 19th Conference on Integer Programming and Combinatorial Optimization (IPCO)*. 2017, pp. 199–211 (cit. on pp. 4, 53, 56, 57, 61, 62, 65, 71, 72, 74, 123).
- [Gar05] N. Garg. Saving an epsilon: A 2-approximation for the k -MST problem in graphs. In: *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*. 2005, pp. 396–402 (cit. on p. 17).
- [GB93] M. X. Goemans and D. J. Bertsimas. Survivable networks, linear programming relaxations and the parsimonious property. In: *Mathematical Programming* 60 (1993), pp. 145–166 (cit. on p. 20).
- [GGG15] M. Gendreau, G. Ghiani, and E. Guerriero. Time-dependent routing problems: A review. In: *Computers & Operations Research* 64 (2015), pp. 189–197 (cit. on p. 82).

- [GGLP21] M. Gmira, M. Gendreau, A. Lodi, and J.-Y. Potvin. Tabu search for the time-dependent vehicle routing problem with time windows on a road network. In: *European Journal of Operational Research* 288.1 (2021), pp. 129–140 (cit. on pp. 82–84).
- [GH99] H. Gehring and J. Homberger. A parallel hybrid evolutionary meta-heuristic for the vehicle routing problem with time windows. In: *Proceedings of EUROGEN 99*. University of Jyväskylä, 1999, pp. 57–64 (cit. on pp. 84, 107, 108).
- [GKL23] L. Gurvits, N. Klein, and J. Leake. From trees to polynomials and back again: New capacity bounds with applications to TSP. 2023. arXiv: [2311.09072](#) (cit. on pp. 3, 9, 16, 23, 75).
- [GKT51] D. Gale, H. Kuhn, and A. Tucker. Linear programming and the theory of games. In: *Activity Analysis of Production and Allocation* (1951), pp. 317–329 (cit. on p. 7).
- [Glo96] F. Glover. Ejection chains, reference structures and alternating path methods for traveling salesman problems. In: *Discrete Applied Mathematics* 65.1 (1996), pp. 223–253 (cit. on p. 101).
- [GLS81] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. In: *Combinatorica* 1.2 (1981), pp. 169–197 (cit. on p. 7).
- [Goe09] M. X. Goemans. Combining approximation algorithms for the prize-collecting TSP. 2009. arXiv: [0910.0553](#) (cit. on pp. 3, 14, 15, 17–19, 25, 26, 123).
- [GS10] R. Geisberger and P. Sanders. Engineering time-dependent many-to-many shortest paths computation. In: *10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2010, pp. 74–87 (cit. on pp. 81, 82, 97).
- [GW17] K. Genova and D. P. Williamson. An experimental evaluation of the best-of-many Christofides’ algorithm for the traveling salesman problem. In: *Algorithmica* 78 (2017), pp. 1109–1130 (cit. on p. 19).
- [GW94] M. X. Goemans and D. P. Williamson. Approximating minimum-cost graph problems with spanning tree edges. In: *Operations Research Letters* 16 (1994), pp. 183–189 (cit. on p. 72).
- [GW95] M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. In: *SIAM Journal on Computing* 24.2 (1995), pp. 296–317 (cit. on pp. 3, 14, 16, 18, 26, 36).
- [Hel17] K. Helsgaun. An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems. Technical Report, Roskilde University. 2017 (cit. on p. 100).

- [HJ06] M. T. Hajiaghayi and K. Jain. The prize-collecting generalized Steiner tree problem via a new approach of primal-dual schema. In: *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm (SODA)*. 2006, pp. 631–640 (cit. on p. 17).
- [HJS21] C. Hansknecht, I. Joormann, and S. Stiller. Dynamic shortest paths methods for the time-dependent TSP. In: *Algorithms* 14.1 (2021), Article 21 (cit. on p. 83).
- [HK98] S. Hamaguchi and N. Katoh. A capacitated vehicle routing problem on a tree. In: *Proceedings of the 9th International Symposium on Algorithms and Computation (ISAAC)*. 1998, pp. 399–407 (cit. on p. 57).
- [HO14] N. J. A. Harvey and N. Olver. Pipage rounding, pessimistic estimators and matrix concentration. In: *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2014, pp. 926–945 (cit. on pp. 27, 28).
- [HR85] M. Haimovich and A. H. G. Rinnooy Kan. Bounds and heuristics for capacitated routing problems. In: *Mathematics of Operations Research* 10 (1985), pp. 527–542 (cit. on pp. 4, 54–58, 123).
- [HS86] S. Hart and M. Sharir. Nonlinearity of Davenport–Schinzel sequences and of generalized path compression schemes. In: *Combinatorica* 6.2 (1986), pp. 151–177 (cit. on pp. 82, 87).
- [HYI08] H. Hashimoto, M. Yagiura, and T. Ibaraki. An iterated local search algorithm for the time-dependent vehicle routing problem with time windows. In: *Discrete Optimization* 5.2 (2008), pp. 434–456 (cit. on pp. 83, 92, 93).
- [HZVG17] Y. Huang, L. Zhao, T. Van Woensel, and J.-P. Gross. Time-dependent vehicle routing problem with path flexibility. In: *Transportation Research Part B: Methodological* 95 (2017), pp. 169–195 (cit. on p. 82).
- [IGP03] S. Ichoua, M. Gendreau, and J.-Y. Potvin. Vehicle dispatching with time-dependent travel times. In: *European Journal of Operational Research* 144.2 (2003), pp. 379–396 (cit. on pp. 5, 80, 82, 84, 85).
- [II86] H. Imai and M. Iri. An optimal algorithm for approximating a piecewise linear function. In: *Information Processing Letters* 9.3 (1986), pp. 159–162 (cit. on pp. 82, 88).
- [Kar72] R. M. Karp. Reducibility among combinatorial problems. In: *Complexity of Computer Computations*. The IBM Research Symposia Series. Boston, 1972, pp. 85–103 (cit. on p. 9).
- [KD16] M. Khachay and R. Dubinin. PTAS for the Euclidean capacitated vehicle routing problem in $\mathbb{R}^d$. In: *Proceedings of the 9th International Conference on Discrete Optimization and Operations Research (DOOR)*. 2016, pp. 193–205 (cit. on p. 57).

-
- [KKO21] A. R. Karlin, N. Klein, and S. Oveis Gharan. A (slightly) improved approximation algorithm for metric TSP. In: *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*. 2021, pp. 32–45 (cit. on pp. [3](#), [9](#), [16](#), [21](#), [23](#), [28](#)).
 - [KKO22] A. Karlin, N. Klein, and S. Oveis Gharan. A (slightly) improved bound on the integrality gap of the subtour LP for TSP. In: *Proceedings of the 63rd IEEE Annual Symposium on Foundations of Computer Science (FOCS)*. 2022, pp. 832–843 (cit. on p. [3](#)).
 - [KKO23] A. R. Karlin, N. Klein, and S. Oveis Gharan. A deterministic better-than- $3/2$ approximation algorithm for metric TSP. In: *Proceedings of the 22nd Conference on Integer Programming and Combinatorial Optimization (IPCO)*. 2023, pp. 261–274 (cit. on pp. [3](#), [9](#), [16](#), [23](#)).
 - [KLS15] M. Karpinski, M. Lampis, and R. Schmied. New inapproximability bounds for TSP. In: *Journal of Computer and System Sciences* 81.8 (2015), pp. 1665–1677 (cit. on p. [9](#)).
 - [KOPRSV17] J. Könemann, N. Olver, K. Pashkovich, R. Ravi, C. Swamy, and J. Vygen. On the integrality gap of the prize-collecting Steiner forest LP. In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*. 2017, 17:1–17:13 (cit. on p. [17](#)).
 - [Kru56] J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. In: *Proceedings of the American Mathematical Society* 7.1 (1956), pp. 48–50 (cit. on p. [8](#)).
 - [LBS18] G. Lera-Romero, J. J. M. Bront, and F. J. Soullignac. An enhanced branch and price algorithm for the time-dependent vehicle routing problem with time window. *Optimization Online*. 2018 (cit. on pp. [83](#), [108](#), [109](#)).
 - [LK73] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. In: *Operations Research* 21.2 (1973), pp. 498–516 (cit. on p. [100](#)).
 - [LL03] H. Li and A. Lim. A metaheuristic for the pickup and delivery problem with time windows. In: *International Journal on Artificial Intelligence Tools* 12.02 (2003), pp. 173–186 (cit. on pp. [84](#), [107](#), [108](#)).
 - [LLM91] M. Labbé, G. Laporte, and H. Mercure. Capacitated vehicle routing on trees. In: *Operations Research* 39 (1991), pp. 616–622 (cit. on p. [57](#)).
 - [LM18] G. Lera-Romero and J. J. Miranda-Bront. Integer programming formulations for the time-dependent elementary shortest path problem with resource constraints. In: *Electronic Notes in Discrete Mathematics* 69 (2018), pp. 53–60 (cit. on p. [83](#)).

- [LMS20a] G. Lera-Romero, J. J. Miranda Bront, and F. J. Soullignac. Linear edge costs and labeling algorithms: The case of the time-dependent vehicle routing problem with time windows. In: *Networks* 76.1 (2020), pp. 24–53 (cit. on pp. [83](#), [108](#), [109](#)).
- [LMS20b] G. Lera-Romero, J. J. Miranda-Bront, and F. J. Soullignac. Dynamic programming for the time-dependent traveling salesman problem with time windows. Optimization Online. 2020 (cit. on pp. [83](#), [100](#)).
- [Lov76] L. Lovász. On some connectivity properties of Eulerian graphs. In: *Acta Mathematica Academiae Scientiarum Hungarica* 28.1 (1976), pp. 129–138 (cit. on pp. [40](#), [48](#)).
- [Mad78] W. Mader. A reduction method for edge-connectivity in graphs. In: *Annals of Discrete Mathematics* 3 (1978), pp. 145–164 (cit. on pp. [40](#), [48](#)).
- [Man14] S. Mancini. Time dependent travel speed vehicle routing and scheduling on a real road network: The case of Torino. In: *Transportation Research Procedia* 3 (2014), pp. 433–441 (cit. on p. [82](#)).
- [MD92] C. Malandraki and M. S. Daskin. Time dependent vehicle routing problems: Formulations, properties and heuristic algorithms. In: *Transportation Science* 26 (1992), pp. 161–260 (cit. on p. [82](#)).
- [MD96] C. Malandraki and R. B. Dial. A restricted dynamic programming heuristic algorithm for the time dependent traveling salesman problem. In: *European Journal of Operational Research* 90.1 (1996), pp. 45–55 (cit. on p. [83](#)).
- [MMM17] A. Montero, I. Méndez-Díaz, and J. J. Miranda-Bront. An integer programming approach for the time-dependent traveling salesman problem with time windows. In: *Computers & Operations Research* 88 (2017), pp. 280–289 (cit. on p. [83](#)).
- [MZ23a] C. Mathieu and H. Zhou. A PTAS for capacitated vehicle routing on trees. In: *ACM Transactions on Algorithms* 19.2 (2023) (cit. on p. [57](#)).
- [MZ23b] C. Mathieu and H. Zhou. A tight $(1.5 + \varepsilon)$ -approximation for unsplittable capacitated vehicle routing on trees. In: *50th International Colloquium on Automata, Languages, and Programming (ICALP)*. 2023, 91:1–91:16 (cit. on p. [57](#)).
- [Nag06] H. Nagamochi. A fast edge-splitting algorithm in edge-weighted graphs. In: *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* E89-A.5 (2006), pp. 1263–1268 (cit. on p. [48](#)).
- [Neu63] J. von Neumann. Discussion of a maximum problem. Working paper. In: *John von Neumann, Collected Works* 6 (1963), pp. 27–28 (cit. on p. [7](#)).

-
- [Ngu13] V. H. Nguyen. A primal-dual approximation algorithm for the asymmetric prize-collecting TSP. In: *Journal of Combinatorial Optimization* 25 (2013), pp. 265–278 (cit. on p. 17).
- [NI97] H. Nagamochi and T. Ibaraki. Deterministic $\tilde{O}(nm)$ time edge-splitting in undirected graphs. In: *Journal of Combinatorial Optimization* 1.6 (1997), pp. 5–46 (cit. on p. 48).
- [NNI97] H. Nagamochi, K. Nishimura, and T. Ibaraki. Computing all small cuts in an undirected network. In: *SIAM Journal on Discrete Mathematics* 10 (1997), pp. 469–481 (cit. on p. 75).
- [NR11] V. Nagarajan and R. Ravi. The directed orienteering problem. In: *Algorithmica* 60.4 (2011), pp. 1017–1030 (cit. on p. 82).
- [PFFSW20] A. Paul, D. Freund, A. Ferber, D. B. Shmoys, and D. P. Williamson. Budgeted prize-collecting traveling salesman and minimum spanning tree problems. In: *Mathematics of Operations Research* 45.2 (2020), pp. 576–590 (cit. on p. 17).
- [PFFSW22] A. Paul, D. Freund, A. Ferber, D. B. Shmoys, and D. P. Williamson. Erratum to ‘budgeted prize-collecting traveling salesman and minimum spanning tree problems’. In: *Mathematics of Operations Research* (2022) (cit. on p. 17).
- [Pri57] R. C. Prim. Shortest connection networks and some generalizations. In: *The Bell System Technical Journal* 36.6 (1957), pp. 1389–1401 (cit. on p. 8).
- [PSUV19] A. A. Pessoa, R. Sadykov, E. Uchoa, and F. Vanderbeck. A generic exact solver for vehicle routing and related problems. In: *Proceedings of the 20th Conference on Integer Programming and Combinatorial Optimization (IPCO)*. 2019, pp. 354–369 (cit. on p. 82).
- [PZL21a] B. Pan, Z. Zhang, and A. Lim. A hybrid algorithm for time-dependent vehicle routing problem with time windows. In: *Computers & Operations Research* 128 (2021), Article 105193 (cit. on pp. 82, 108, 109).
- [PZL21b] B. Pan, Z. Zhang, and A. Lim. Multi-trip time-dependent vehicle routing problem with time windows. In: *European Journal of Operational Research* 291.1 (2021), pp. 218–231 (cit. on p. 82).
- [QSU21] E. Queiroga, R. Sadykov, and E. Uchoa. A POPMUSIC matheuristic for the capacitated vehicle routing problem. In: *Computers & Operations Research* 136 (2021), Article 105475 (cit. on p. 82).
- [Reg01] C. Rego. Node-ejection chains for the vehicle routing problem: Sequential and parallel algorithms. In: *Parallel Computing* 27.3 (2001), pp. 201–222 (cit. on p. 101).
- [Ros22] F. Rosebrock. Scheduling breaks in vehicle routing with time-dependent travel times. Master’s thesis, University of Bonn. 2022 (cit. on p. 104).

- [RV99] S. Rajagopalan and V. V. Vazirani. On the bidirected cut relaxation for the metric Steiner tree problem. In: *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 1999, pp. 742–751 (cit. on p. 16).
- [SB18] C. S. Sartori and L. S. Buriol. A matheuristic approach to the pickup and delivery problem with time windows. In: *Computational Logistics*. 2018, pp. 253–267 (cit. on p. 102).
- [Sch20] N. Schlomberg. Scheduling in vehicle routing. Master’s thesis, University of Bonn. 2020 (cit. on pp. 87, 103).
- [Ser87] A. I. Serdyukov. O nekotorykh ekstremal’nykh obkhodakh v grafakh. In: *Upravlyaemye sistemy* 17 (1987), pp. 76–79 (cit. on pp. 9, 16, 40).
- [SHU05] H. Sontrop, P. van der Horn, and M. Uetz. Fast ejection chain algorithms for vehicle routing with time windows. In: *Hybrid Metaheuristics*. 2005, pp. 78–89 (cit. on p. 101).
- [Sol87] M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. In: *Operations Research* 35.2 (1987), pp. 254–265 (cit. on pp. 84, 98, 108, 109).
- [Sri01] A. Srinivasan. Distributions on level-sets with applications to approximation algorithms. In: *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 2001, pp. 588–597 (cit. on pp. 18, 27).
- [SSRR17] M. Soto, M. Sevaux, A. Rossi, and A. Reinholz. Multiple neighborhood search, tabu search and ejection chains for the multi-depot open vehicle routing problem. In: *Computers & Industrial Engineering* 107 (2017), pp. 211–222 (cit. on p. 101).
- [SWZ21] B. Strasser, D. Wagner, and T. Zeitz. Space-efficient, fast and exact routing in time-dependent road networks. In: *Algorithms* 14.3 (2021), p. 90 (cit. on p. 82).
- [TAFQ17] H. B. Ticha, N. Absi, D. Feillet, and A. Quilliot. Empirical analysis for the VRPTW with a multigraph representation for the road network. In: *Computers & Operations Research* 88 (2017), pp. 103–116 (cit. on p. 84).
- [TAFQV19] H. B. Ticha, N. Absi, D. Feillet, A. Quilliot, and T. Van Woensel. A branch-and-price algorithm for the vehicle routing problem with time windows on a road network. In: *Networks* 73.4 (2019), pp. 401–417 (cit. on p. 82).
- [TAFQV21] H. B. Ticha, N. Absi, D. Feillet, A. Quilliot, and T. Van Woensel. The time-dependent vehicle routing problem with time windows and road-network information. In: *Operations Research Forum* 2 (2021), Article 4 (cit. on p. 84).

-
- [TI17] C. Tilk and S. Irnich. Dynamic programming for the minimum tour duration problem. In: *Transportation Science* 51.2 (2017), pp. 549–565 (cit. on p. 83).
- [Tic17] H. B. Ticha. Vehicle routing problems with road-network information. PhD thesis, Université Clermont Auvergne. 2017 (cit. on p. 82).
- [Trö19] T. Tröbst. Capacitated vehicle routing and cycle covering problems. Master’s thesis, University of Bonn. 2019 (cit. on p. 4).
- [TV14] P. Toth and D. Vigo, eds. *Vehicle routing: Problems, methods, applications*. 2nd ed. MOS-SIAM, 2014 (cit. on p. 82).
- [TV24] V. Traub and J. Vygen. *Approximation Algorithms for Traveling Salesman Problems*. Cambridge University Press. 2024, to appear (cit. on p. 10).
- [TVZ22] V. Traub, J. Vygen, and R. Zenklusen. Reducing path TSP to TSP. In: *SIAM Journal on Computing* 51.3 (2022), STOC20-24–STOC20-53 (cit. on p. 16).
- [UPPPVS17] E. Uchoa, D. Pecin, A. Pessoa, M. Poggi, T. Vidal, and A. Subramanian. New benchmark instances for the capacitated vehicle routing problem. In: *European Journal of Operational Research* 257.3 (2017), pp. 845–858 (cit. on pp. 83, 84, 107, 108).
- [VCGP13] T. Vidal, T. G. Crainic, M. Gendreau, and C. Prins. A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. In: *Computers & Operations Research* 40 (2013), pp. 475–489 (cit. on pp. 82, 100).
- [VHBS18] D. M. Vu, M. Hewitt, N. Boland, and M. Savelsbergh. Solving time dependent traveling salesman problems with time windows. *Optimization Online*. 2018 (cit. on p. 83).
- [Vid22] T. Vidal. Hybrid genetic search for the CVRP: Open-source implementation and SWAP* neighborhood. In: *Computers & Operations Research* 140 (2022), Article 105643 (cit. on pp. 82, 100).
- [VS20] T. R. Visser and R. Spliet. Efficient move evaluations for time-dependent vehicle routing problems. In: *Transportation Science* 54.4 (2020), pp. 1091–1112 (cit. on pp. 83, 86, 89, 90).
- [vS20] R. van Bevern and V. A. Slugina. A historical note on the 3/2-approximation algorithm for the metric traveling salesman problem. In: *Historia Mathematica* 53 (2020), pp. 118–127 (cit. on pp. 9, 16).
- [Wie86] A. Wiernik. Planar realizations of nonlinear Davenport–Schinzel sequences by segments. In: *Proceedings of the 27th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*. 1986, pp. 97–106 (cit. on pp. 82, 87).

Bibliography

- [Wil90] D. P. Williamson. Analysis of the Held-Karp heuristic for the traveling salesman problem. Master's thesis, Massachusetts Institute of Technology. 1990 (cit. on p. 3).
- [WL20] Y. Wu and X. Lu. Capacitated vehicle routing problem on line with unsplittable demands. In: *Journal of Combinatorial Optimization* 44 (2020), pp. 1953–1963 (cit. on p. 57).
- [Wol80] L. A. Wolsey. Heuristic analysis, linear programming and branch and bound. In: *Combinatorial Optimization II*. 1980, pp. 121–134 (cit. on pp. 10, 19).
- [ZX22] J. Zhao and M. Xiao. Improved approximation algorithms for capacitated vehicle routing with fixed capacity. 2022. arXiv: [2210.16534](https://arxiv.org/abs/2210.16534) (cit. on p. 57).