Exact and Approximation Algorithms for Many-Visits Travelling Salesperson Problems

Vom Promotionsausschuss der
Technischen Universität Hamburg
zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften (Dr. rer. nat.)

genehmigte Dissertation

von
Roland Vincze

aus
Nové Zámky, Tschechoslowakei

2022

Betreuer:
Prof. Dr. Matthias Mnich (Technische Universität Hamburg)

Gutachter:
Prof. László Végh (London School of Economics, United Kingdom)
Prof. Mohit Singh (Georgia Institute of Technology, United States of America)

Datum der mündlichen Prüfung:
1. Dezember 2021

DOI: https://doi.org/10.15480/882.4226

ORCID: Roland Vincze
https://orcid.org/0000-0002-5997-6564

Contents

Preface

This thesis contains the outcomes of my research activity as a PhD candidate at Maastricht University from September 2016 until August 2019 and at Hamburg University of Technology from September 2019 until January 2021. My supervisors at Maastricht University were André Berger and Matthias Mnich, and my supervisor at Hamburg University of Technology was Matthias Mnich. My research was supported by grant MN 59/4-1 of Deutsche Forschungsgemeinschaft (DFG), as well as the DAAD with funds of the Bundesministerium für Bildung und Forschung (BMBF).

At the time of submitting this thesis, some of the results have already been published. The content of Chapter 1: Exact Algorithms for the Many-Visits TSP is joint work with André Berger, László Kozma and Matthias Mnich. They first appeared at SODA 2019 [19], and an extended version with the final time complexities appeared in the journal ACM Transactions on Algorithms [20]. The content of Chapter 2: Degree-Bounded Problems and Approximations is joint work with Kristóf Bérczi, André Berger and Matthias Mnich, and they first appeared on arXiv [14]. Finally, the content of Chapter 3: Constant-Factor Approximations for the Many-Visits Path TSP and Chapter 4: Approximation Algorithms for Multiple-Agent MVTSP is joint work with Kristóf Bérczi and Matthias Mnich, and they first appeared (separately) on arXiv [15, 16].

All results of this thesis are outcomes of collaborations, and all the participants named above contributed essentially an equal share to the relevant results.

Acknowledgements

First of all, I would like to thank my supervisors André Berger and Matthias Mnich for their extraordinary help and support throughout my PhD. I am also thankful to my coauthors Kozma Laci and Bérczi Kristóf for their expertise and hard work, and to Végh Laci and Mohit Singh for reading my thesis and for their insightful comments.

I would like to thank my former and current colleagues in Maastricht, Hamburg and Augsburg, my friends in Budapest and all around the globe for making my time as a PhD student much more colourful.

Finally, I am grateful to my parents for making this journey possible in the first place, and my girlfriend for her support during the final, bumpiest part of the journey.

INTRODUCTION

The travelling salesperson problem (TSP) is one of the cornerstones of combinatorial optimisation, with origins going back (at least) to the 19^{th} century work of Hamilton. For surveys on the rich history, variants, and current status of TSP we refer to the dedicated books [5, 28, 52, 82]. In the standard TSP, given n cities and their pairwise distances, we seek a tour of minimum total distance that visits each city at least once. If the distances obey the triangle inequality, then an optimal tour necessarily visits each city *exactly once* (apart from returning to the starting city in the end). In the general case of the TSP with arbitrary distances, the optimal tour may visit a city multiple times. Instances with non-metric distances arise from various applications that are modelled by the TSP, e.g. from scheduling problems.

In this work we study a more general problem where each city has to be visited *exactly* a given number of times. More precisely, we are given a set V of n vertices, with nonnegative pairwise distances (or costs) $c(uv)$, for all edges uv, where $u, v \in V$. Also given are integers $r(v) \geqslant 1$ for $v \in V$, which we refer to as *visit requests* or simply *requests*. A valid tour of length r is a sequence $(v_1, \ldots, v_r) \in V^r$, where $r = \sum_{v \in V} r(v)$, such that each $v \in V$ appears in the sequence exactly $r(v)$ times. The cost of the tour is $\sum_{i=1}^{r-1} c(v_i v_{i+1}) + c(v_r v_1)$. Our goal is to find a valid tour with minimum cost.

The problem is known as the *many-visits TSP* (MVTSP), and to the best of our knowledge, it was first considered in 1966 by Rothkopf [97]. As an alternative name, *high-multiplicity TSP* also appears in the literature. The MVTSP is NP-complete, as it contains the classical TSP as a special case, when $r(v) = 1$ for all $v \in V$. Similarly to the TSP, one can consider the *path version* of the problem: in the Many-Visits Path TSP, special end-vertices s and t are given, and the aim is to find a s-t-walk[1] of minimum cost, that visits each vertex v exactly $r(v)$ times.

As a natural TSP-generalisation, MVTSP is a fundamental problem of independent interest. In addition, MVTSP has proved to be useful for modelling other problems. The *aircraft sequencing problem* or *aircraft landing problem* is one of the most referred applications in the literature [12, 21, 84, 94], where the goal is to find a schedule of departing and/or landing aeroplanes that minimises an objective function and satisfies certain constraints. The aircraft are categorised into a small number of classes, and for each pair of classes a nonnegative lower bound is given, denoting the minimum amount of time needed to pass between the take off/landing of two planes from the given classes.

[1] In graph theory, a *walk* is an alternating sequence of vertices and edges, that begins and ends with a vertex; moreover, both vertices and edges can appear multiple times in the sequence.

The problem can be embedded in the MANY-VISITS PATH TSP model by considering the classes to be cities and the separation times to be the cost of the edge between them, while the number of aeroplanes in a class corresponds to the number of visits of a city.

Suppose there are r jobs of n different types to be executed on a single, universal machine; in particular, there are $r(j)$ jobs of type j. Processing a job of type j bears a cost $p(j)$, switching from a type i to type j job costs $s(ij)$, and the goal is to find the sequence of jobs with minimal total cost. Modelling this problem as a MVTSP instance is straightforward, by letting $c(ij)$ denote the sum of $p(j)$ and $s(ij)$. Note that $c(ii)$ is not necessarily smaller than $c(ij)$ for $i \neq j$, meaning in some applications it is beneficial to switch between job types. Emmons and Mathur [33] also describe an application of the MVTSP to the no-wait flow shop problem. There is only a handful of constant-factor approximation algorithms for scheduling problems with setup times [3]; see, for example, the results of Jansen et al. [64] and Deppert and Jansen [30] that consider sequence-independent batch setup times, or van der Veen et al. [109] that considers sequence-dependent setup times with a special structure.

A different kind of application comes from geometric approximation. A standard technique to approximate geometric optimisation problems is to reduce the size of the input by grouping certain input points together. Each group is then replaced by a single representative, and the reduced instance is solved exactly. For example, we may snap input points to nearby grid points, if doing so does not significantly affect the objective cost. Recently, this technique was used by Kozma and Mömke, to give an efficient polynomial-time approximation scheme (EPTAS) for the MAXIMUM SCATTER TSP in doubling metrics [79], addressing an open question of Arkin et al. [6]. In this case, the reduced problem is exactly the MVTSP. Yet another application of the MVTSP is in settling the parameterised complexity of finding a Hamiltonian cycle in a graph class with restricted neighbourhood structure [81].

Outline

In Chapter 1 of this thesis, we provide a family of exact algorithms, that solve the Many-Visits Travelling Salesman Problem optimally. These algorithms work under the most general setting, in the sense that the edge costs can be asymmetric, and they do not need to satisfy the triangle inequality. Our main result is a polynomial-space algorithm with a single-exponential-time complexity that improves on the previously best algorithm by Cosmadakis and Papadimitrou [29], making it the first improvement in over 35 years. This algorithm is asymptotically best possible, assuming the Exponential Time Hypothesis. It also implies an improved PTAS for the Maximum Scatter TSP [79], reducing the space dependence of the error parameter to polynomial. The algorithms in this chapter use enumeration of directed spanning trees, and the polynomial-space one relies on a recursive, "divide and conquer" approach inspired by the polynomial-space TSP algorithm of Gurevich and Shelah [51].

Chapter 2 is a short detour into matroid theory. We consider the BOUNDED DEGREE G-POLYMATROID ELEMENT WITH MULTIPLICITIES problem: given a g-polymatroid and a hypergraph on a common base set, one seeks a minimal cost element from the g-polymatroid, such that the intersection of this element with the hyperedges of the hypergraph satisfies certain bounds. We generalise the result of Király et al. [73] on the MINIMUM BOUNDED DEGREE MATROID BASIS PROBLEM, and provide a polynomial-time approximation algorithm to the BOUNDED DEGREE G-POLYMATROID ELEMENT WITH MULTIPLICITIES problem that outputs an element of at most optimal cost, that violates the hyperedge constraints by a certain amount. Using this algorithm as a black box, one can obtain a solution to the MINIMUM BOUNDED DEGREE k-COMPONENT MULTIGRAPH problem with cost at most the optimum, where each degree lower bound is violated by at most 1.

Obtaining an approximate solution to the MINIMUM BOUNDED DEGREE k-COMPONENT MULTIGRAPH problem serves as a key component in the main algorithmic result of Chapter 3. In this chapter we consider the Many-Visits Path TSP, with the assumption that the edge costs are symmetric and they satisfy the triangle inequality. We provide a polynomial-time $3/2$-approximation for this problem, where the constant factor matches that of the best polynomial-time approximation algorithm for the single-visit Path TSP, by Zenklusen [121]. The algorithm uses the techniques of Zenklusen, adapted to the many-visits setting, as well as our approximation algorithm from Chapter 2.

Finally, in Chapter 4, we further generalise the MVTSP by introducing multiple agents, that may or may not have to start and end their journey in their respective depots. The Multiple-Agent MVTSP (MA-MVTSP) and Multiple-Depot MVTSP (MD-MVTSP) variants ask for k MVTSP tours that together visit each vertex exactly $r(v)$ times, such that the overall cost of the tours is minimised. We present a full description of the different problem variants, as well as polynomial-time constant-factor approximations for all variants, assuming symmetric and metric edge costs. Finally, with the help of the techniques presented in Chapter 2 and used in Chapter 3, we provide polynomial-time 2-approximations to two of the problem variants.

Basic notation

Throughout this thesis, a graph $G = (V, E)$ denotes a complete graph on n vertices. There is a nonnegative cost $c(uv)$ defined on the edges $uv \in E$. In the most general settings these costs are asymmetric, but in certain parts of this work they are defined to be symmetric (i.e. $c(uv) = c(vu)$ for all $u, v \in V$), or metric (i.e. satisfying the triangle inequality, $c(uw) \leqslant c(uv) + c(vw)$ for all $u, v, w \in V$). Note that the edge set E includes *self-loops*, i.e. edges vv for each vertex $v \in V$, which might occur a nonzero cost.

We use the terms 'city' and 'vertex' interchangeably. In case of symmetric edge costs (Chapters 3 and 4), by 'requirement' we strictly mean the *degree requirement* of a vertex, this quantity is twice as much as the *request* (denoted by the $r(v)$ values) of a vertex. In

case of asymmetric edge costs (Chapter 1), we will distinguish between *outdegree* and *indegree* requirements, and will refer to them collectively as *degree requirements*.

A *multigraph* X is a graph on the vertex set V with a multiset $E(X)$ as edge set, that is, $E(X)$ might contain several copies of the same edge. For a subset $F \subseteq E$ of edges, the *set of vertices covered by* F is denoted by $V(F)$. The *number of connected components* of the graph $(V(F), F)$ is denoted by $\mathrm{comp}(F)$. We will simply use the term *components* of a graph G referring to the connected components of G. For a subset $W \subseteq V$ of vertices, the *set of edges spanned by* W is denoted by $E(W)$. For $W \subseteq V$, the (multi)graph induced by W is $G[W]$.

Given a multiset F of edges, the multiset of edges leaving the vertex set $C \subseteq V(F)$ is denoted by $\delta_F(C)$. Similarly, the multiset of regular edges (i.e. excluding self-loops) in F incident to a vertex $v \in V$ is denoted by $\delta_F(v)$. The multiset of all edges (i.e. including self-loops) in F incident to a vertex $v \in V$ is denoted by $\dot{\delta}_F(v)$. Then the *degree* of v in F is denoted by $\deg_F(v) := |\dot{\delta}_F(v)|$, where every copy of the self-loop at v in F is counted twice. We will omit the subscript when F contains all the edges of G, that is, $F = E(G)$. The *multiplicity* of an edge uv in a directed multigraph X means the number of copies of an edge uv in X, and is denoted $\mathrm{mul}_X(uv)$. If X is directed, the *outdegree* of a vertex $v \in V$ is $\deg^{\mathrm{O}}_X(v) = \sum_{w \in V} \mathrm{mul}_X(vw)$, the *indegree* of a vertex $v \in V$ is $\deg^{\mathrm{I}}_X(v) = \sum_{u \in V} \mathrm{mul}_X(uv)$. Given edge costs $c : V \times V \to \mathbb{N} \cup \{\infty\}$, the *cost* of X is simply the sum of its edge costs, i.e. $\mathsf{cost}(X) = \sum_{u,v \in V} \mathrm{mul}_X(uv) \cdot c(uv)$.

Let us denote the set of edges between two disjoint vertex sets A and B by $\delta(A, B)$. Given two graphs or multigraphs H_1, H_2 on the same vertex set, $H_1 + H_2$ denotes the multigraph obtained by taking the union of the edge sets of H_1 and H_2. Observe that as an effect, in case of directed graphs, outdegrees and indegrees are also added pointwise. Formally, $\mathrm{mul}_{H_1+H_2}(uv) = \mathrm{mul}_{H_1}(uv) + \mathrm{mul}_{H_2}(uv)$, $\deg^{\mathrm{O}}_{H_1+H_2}(v) = \deg^{\mathrm{O}}_{H_1}(v) + \deg^{\mathrm{O}}_{H_2}(v)$, and $\deg^{\mathrm{I}}_{H_1+H_2}(v) = \deg^{\mathrm{I}}_{H_1}(v) + \deg^{\mathrm{I}}_{H_2}(v)$, for all $u, v \in V$. The relation $\mathsf{cost}(H_1 + H_2) = \mathsf{cost}(H_1) + \mathsf{cost}(H_2)$ clearly holds.

For a vector $x \in \mathbb{R}^E$, we denote the sum of the x-values on the edges incident to v by $x(\dot{\delta}(v))$. Note that the x-value of the self-loop at v is counted twice in $x(\dot{\delta}(v))$. Given a vector $x \in \mathbb{R}^S$ and a set $Z \subseteq S$, we use $x(Z) = \sum_{s \in Z} x(s)$. The *lower integer part* of x is denoted by $\lfloor x \rfloor$, so $\lfloor x \rfloor(s) = \lfloor x(s) \rfloor$ for every $s \in S$. This notation extends to sets, so by $\lfloor x \rfloor(Z)$ we mean $\sum_{s \in Z} \lfloor x(s) \rfloor$. The *support of* x *on a set* $Z \subseteq S$ is denoted by $\mathrm{supp}_Z(x)$, that is, $\mathrm{supp}_Z(x) = \{s \in Z \mid x(s) \neq 0\}$. When talking about the support on the whole base set, that is $Z = S$, we simply write $\mathrm{supp}(x)$ instead of $\mathrm{supp}_S(x)$. The *difference of set* B *from set* A is denoted by $A - B = \{s \in A \mid s \notin B\}$. We denote a single-element set $\{s\}$ by s, and with a slight abuse of notation, we write $A - s$ to indicate $A - \{s\}$. Let us denote the *symmetric difference* of two sets A and B by $A \triangle B := (A - B) \cup (B - A)$ and the *characteristic vector* of a set A by χ_A.

When discussing time and space complexity, we will use the standard "big O" notation. For example by $f(n) = O(g(n))$ we mark that f is bounded from above by g asymptotically, i.e. there exists a positive real M, such that $f(n) \leqslant Mg(n)$ holds for suf-

ficiently large values of n. We also write $f(n) = \Omega(g(n))$ if $g(n) = O(f(n))$ is true. By $f(n) = o(g(n))$ we mean f is dominated by g asymptotically, i.e. there exists a positive real M, such that $f(n) < Mg(n)$ holds for sufficiently large values of n. Finally, we write $f(n) = \omega(g(n))$ if $g(n) = o(f(n))$ is true. The term *superexponential* refers to a quantity of the form $2^{\omega(n)}$. Some algorithms in this thesis have a time and/or space complexity (super-)exponential in n, and we are often interested in the exponential components, and not so much in the polynomial ones. In these instances we use the $O^*(\cdot)$ notation, which suppresses the factors bounded by a polynomial of n.

For a problem instance I, the *encoding length* of I is denoted by $|I|$. The classical computational model uses a binary representation for the numerical parameters of an instance I, which means a parameter p is stored using $\log p$ bits. An algorithm is a *polynomial-time* algorithm, if the number of operations it performs on an input I is $O(\mathrm{poly}(|I|))$, where $\mathrm{poly}(\cdot)$ is a polynomial function. A *polynomial-space* algorithm uses $O(\mathrm{poly}(|I|))$ bits of memory/storage during its execution.

Sometimes it is more natural to consider that the elementary arithmetic operations (addition, subtraction, multiplication, division) can be performed in $O(1)$ time, regardless of the size of the numbers participating in the operation. Suppose that a problem instance I consists of N rational numbers. We say that an algorithm is *strongly polynomial*, if the number of arithmetic operations is bounded by $\mathrm{poly}(N)$ and the algorithm runs using polynomial space. In other words, the runtime of a strongly polynomial algorithm depends only on the *number* of numerical parameters and not the *size* of them.

Consider a MVTSP tour $\mathsf{TOUR} = (v_1, \ldots, v_r) \in V^r$. We refer to the unique directed multigraph X consisting of the edges $(v_1, v_2), \ldots, (v_{r-1}, v_r), (v_r, v_1)$ as the *edge set of* TOUR. We state a simple but crucial observation.

Lemma 1. *Let X be a directed multigraph over V with outdegrees $\deg^{\mathrm{O}}_X(\cdot)$ and indegrees $\deg^{\mathrm{I}}_X(\cdot)$. Then X is the edge set of a MVTSP tour that visits each vertex $v \in V$ exactly $r(v)$ times if and only if both of the following conditions hold:*

(i) the underlying graph of X is connected, and

(ii) for all $v \in V$, we have $\deg^{\mathrm{O}}_X(v) = \deg^{\mathrm{I}}_X(v) = r(v)$.

Proof. The fact that the connectedness of X and $\deg^{\mathrm{O}}_X(v) = \deg^{\mathrm{I}}_X(v)$ is equivalent with the existence of a tour that uses each edge of X exactly once is the well-known "Euler's theorem". (See Bang-Jensen and Gutin [9, Thm 1.6.3] for a short proof.) Clearly, visiting each vertex v exactly $r(v)$ times is equivalent with the condition that the tour contains $r(v)$ edges of the form $\bullet v$ and $r(v)$ edges of the form $v\bullet$. □

Note that the description $\mathsf{TOUR} = (v_1, \ldots, v_r)$ takes $\Omega(r)$ space, when stored as a sequence, which is *exponential* in the input size $O(n^2 \log r)$. For this reason, when keeping track of a MVTSP tour TOUR, we use a *compact representation* of its multigraph X. One can store X using $O(n^2 \log r)$ space, by storing each edge uv of X along with its multiplicity $x(uv)$.

Moreover, given such a representation of X, a tour TOUR' with edge set X can easily be recovered, such that $\mathsf{cost}(\mathsf{TOUR}') = \mathsf{cost}(X) = \mathsf{cost}(\mathsf{TOUR})$. This amounts to finding an Eulerian tour of X, which can be done in time linear in the length r of the tour. To avoid a linear dependence on r, we can apply the algorithm **ConvertToSequence** by Grigoriev and van de Klundert [50] that calculates a classical cycle-decomposition of a circulation, and thus constructs a compact representation of TOUR' in time $O(n^4 \log r)$. For the sake of completeness, we include it as Algorithm 1.

Algorithm 1 ConvertToSequence by Grigoriev and van de Klundert [50]

Input: A multigraph X, given by the multiplicity $x(uv)$ for every edge uv.
Output: A closed walk in X, represented by a collection $\mathcal{C}$ of pairs (C, μ_C), where C is a simple cycle in X and μ_C its multiplicity.

1: $\mathcal{C} := \emptyset$, $q = 1$.
2: **while** $x(uv) \not\equiv 0$ **do**
3: Find a simple closed walk $C_q = (v_1v_2, v_2v_3, \ldots, v_tv_1)$, where $v_i \in V$ for $i = 1, \ldots, t$, such that $x(v_iv_{i+1}) > 0$ for $i = 1, \ldots, t-1$ and $x(v_tv_1) > 0$.
4: Let $\mu_q = \min\{x(uv) \mid uv \in C_q\}$ and $\mathcal{C} := \mathcal{C} \cup \{(C_q, \mu_q)\}$.
Update $x(uv) := x(uv) - \mu_q$ for $uv \in C_q$, and $q = q + 1$.
5: **return** $\mathcal{C}$

The number of iterations in Algorithm 1 can be bounded by $O(n^2)$ as at least one variable $x(uv)$ becomes 0 in each iteration. The polynomial size of the output $\mathcal{C}$ also follows from this fact. Furthermore, the graph operations in Lines 3 and 4 can be done in $O(n^2 \log r)$ time, hence the time complexity of Algorithm 1 is $O(n^4 \log r)$.

One can obtain an explicit MVTSP tour TOUR' from $\mathcal{C}$ the following way: traverse an arbitrary cycle C exactly μ_C times, and whenever a vertex u is reached for the first time, traverse $\mu_{C'}$ copies of every cycle C' containing u. As the edge sets of TOUR and TOUR' are equal, TOUR' also visits each $v \in V$ exactly $r(v)$ times and $\mathsf{cost}(\mathsf{TOUR}') = \mathsf{cost}(X) = \mathsf{cost}(\mathsf{TOUR})$. Thus, in solving MVTSP we only focus on finding a minimum cost directed multigraph whose underlying undirected graph is connected, and whose degrees match the visit requests in the problem instance.

CHAPTER 1

EXACT ALGORITHMS FOR THE MANY-VISITS TSP

Introduction

To date, the fastest known exact algorithms for TSP (both in the metric and non-metric cases) are due to Bellman [13] and Held and Karp [56], running in time $O(2^n n^2)$ for n-city instances; both algorithms also require space $\Omega(2^n)$. When it comes to *polynomial-space* algorithms, the state of the art is a recursive, divide and conquer approach by Gurevich and Shelah [51], that solves the TSP in $O^*(4^{n+o(n)})$ time.

As for the MVTSP, Psaraftis [94] gave a dynamic programming algorithm with run time $O(n^2 \cdot \prod_{v \in V} (r(v) + 1))$. Observe that this quantity may be as high as $(r/n + 1)^n$, which is prohibitive even for moderately large values of r. In 1984, Cosmadakis and Papadimitriou [29] observed that the MVTSP can be decomposed into a connectivity subproblem and an assignment subproblem. Taking advantage of this decomposition, they designed a family of algorithms, the best of which has run time $O^*(n^{2n} 2^n + \log r)$. This result can be seen as an early example of *fixed-parameter tractability*, where the rapid growth in complexity is restricted to a certain parameter (the number of vertices), rather than the total size of the input.

The analysis of the algorithm of Cosmadakis and Papadimitriou is highly non-trivial, combining graph-theoretic insights and involved estimates of various combinatorial quantities. The complexity analysis of the algorithm is not known to be tight, but a lower bound of $2^{\Omega(n \log n)}$ is known to hold for its run time [29, p. 104]. Similarly, in the space requirement of the algorithm, a term of the form $2^{\Omega(n \log n)}$ appears hard to avoid. While it extends the tractability of TSP to a new range of parameters, the usefulness of the Cosmadakis-Papadimitriou algorithm is limited by its superexponential dependence on n in the run time, with the issue of superexponential *space* perhaps even more worrisome.

There have been further studies of the MVTSP. Van der Veen and Zhang [110] discuss a problem equivalent to the MVTSP, called *K-group TSP*, and describe an algorithm with polylogarithmic dependence on the number r of visits, similarly to Cosmadakis and Papadimitriou. The value n however is assumed constant, and its effect on the run time is not explicitly computed (the dependence can be seen to be superexponential). Finally, Grigoriev and van de Klundert [50] give an ILP formulation for the MVTSP with

$O(n^2)$ variables. Applying Kannan's improvement [67] of Lenstra's algorithm [83] for solving fixed-dimensional ILPs to this formulation yields an algorithm with run time $n^{O(n^2)} \log r$. Further ILP formulations for the MVTSP are due to Sarin et al. [98] and Aguayo et al. [2], both of which again require superexponential time to be solved by standard algorithms.

Results

The main result of this chapter improves both the time and space complexity of the best known algorithm for the MVTSP, the first improvement in over 35 years. Specifically, we show that a *logarithmic* dependence on the number r of visits, a *single-exponential* dependence on the number n of cities, and a *polynomial* space complexity are simultaneously achievable. Moreover, while we build upon ideas from the previous best approach, our algorithm is arguably easier to describe, easier to implement, and easier to analyse than its predecessor. To introduce the techniques step by step, we describe *three* algorithms for solving the MVTSP. We refer to the three algorithms, after their core subroutines, as ENUM-MV, DP-MV, and DC-MV. The acronyms stand for enumeration, dynamic programming, and divide and conquer, respectively. We also describe an improved variant of the third algorithm, called DC-MV2. All our algorithms are deterministic. Their complexities are summarised in Theorem 1.1, proved in §1.1.

Theorem 1.1.

(i) ENUM-MV *solves MVTSP using space* $O(n^2)$*, in time* $O^*(n^n + \log r)$.

(ii) DP-MV *solves MVTSP using space* $O(5^n)$*, in time* $O^*(5^n + \log r)$.

(iii) DC-MV2 *solves MVTSP using space* $O(n^2)$*, in time* $O^*(16^{n+o(n)} + \log r)$.

The Exponential Time Hypothesis (ETH) [60, 61] implies that TSP cannot be solved in $2^{o(n)}$, i.e. sub-exponential, time. Under this hypothesis, the run time of our algorithm DC-MV2 is asymptotically optimal for the MVTSP, up to the base of the exponential. Further, note that the space requirement of DC-MV2 is also (essentially) optimal, as a compact solution encodes for each of the $\Omega(n^2)$ edges the number t of times that this edge is traversed by an optimal tour. Note that throughout the chapter we assume that each request can be stored in a constant number of machine words; if this is not the case, e.g. if r is exponential in n, a factor $O(\log r)$ should be applied to the given space bounds.

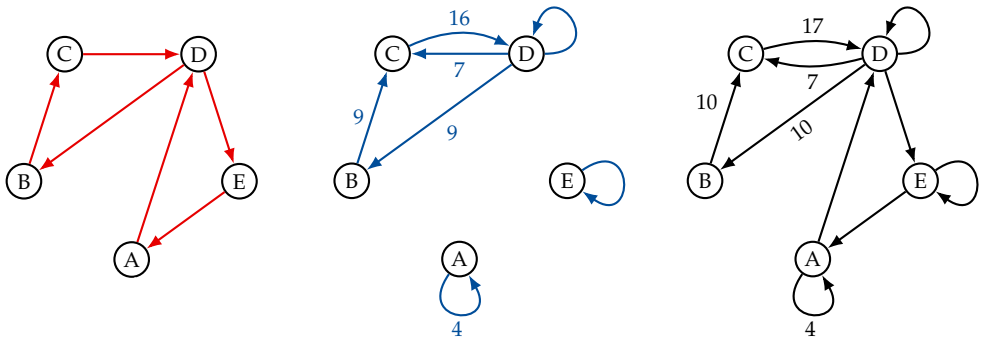
It is interesting to contrast our results for the MVTSP with recent results for the *r-simple k-path* problem, where a path of length k is sought that visits each vertex *at most* r times. For that problem, the fastest known algorithms—due to Abasi et al. [1] and Gabizon et al. [46]—have run time *exponential* in $k/r \cdot \log r$, and such exponential dependence is necessary assuming ETH.

The results of this chapter are joint work with André Berger, László Kozma and Matthias Mnich. They first appeared at SODA 2019 [19], and an extended version

with the final time complexities appeared in the ACM Journal Transactions on Algorithms [20]. After the publication of our results, Kowalik et al. [78] made further fine-grained time complexity improvements. The authors provided a tight analysis of DP-MV, proving a $O^*(4^n)$ time and space complexity, and designed a new polynomial-space algorithm running in time $O(7.88^n)$ by "using a more powerful minimum cost flow network, which allows for computing the cheapest outbranchings (referred to them as rooted spanning trees throughout this chapter) in smaller subgraphs".

Overview of techniques. The Cosmadakis-Papadimitriou algorithm is based on the following high-level insight, common to most work on the TSP problem, whether exact or approximate. The task of finding a valid tour may be split into two separate tasks: (1) finding a structure that connects all vertices, and (2) augmenting the structure found in (1) in order to ensure that each vertex is visited the required number of times. Indeed, such an approach is also used in the well-known $3/2$-approximation algorithm of Christofides and Serdyukov for metric TSP [27, 103]. There, the structure that guarantees connectivity is a minimum spanning tree, and the Eulerian property is enforced by the addition of a perfect matching that connects odd-degree vertices, ensuring that all vertices have even degree, and can thus be entered and exited, as required.

In the case of MVTSP, Cosmadakis and Papadimitriou ensure connectivity (part (1)) by finding a *minimal connected Eulerian digraph* on the set V of input vertices. Indeed, such a digraph must be part of every solution, since a tour must balance every vertex (equal outdegree and indegree), and all vertices must be mutually reachable. Minimality is meant here in the sense that no proper subgraph is Eulerian, and is required only to reduce the search space. Assuming that a connected Eulerian subdigraph of the solution is found, it needs to be extended to the edge set of a valid tour (part (2)). If this is done with the cheapest possible set of edges, then the optimum must have been found. This second step amounts to solving a transportation problem, which takes polynomial time.



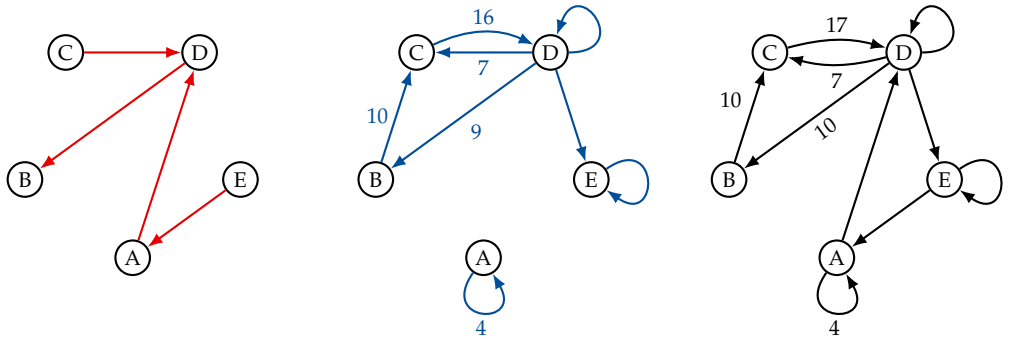
a) Eulerian digraph b) Transportation problem c) MVTSP solution

FIGURE 1.1: Decomposition of a MVTSP solution by the Cosmadakis-Papadimitriou algorithm. Visit requests are $(5, 10, 17, 19, 2)$. The numbers specify the multiplicity of a certain edge in the multigraph.

The first step, however, requires us to implicitly consider all possible minimal Eulerian digraphs. As it is NP-complete to test the non-minimality of an Eulerian digraph [93], the authors relax minimality and suggest the use of heuristics for pruning out non-minimal instances in practice. On the other hand, they obtain a saving in run time by observing that among all digraphs with the same *degree sequence* only one with smallest cost needs to be considered. Otherwise, in the final tour, the connected Eulerian subdigraph could be swapped with a cheaper one, while maintaining the validity of the tour.

Cosmadakis and Papadimitriou thus iterate over feasible degree sequences of connected Eulerian digraphs; for each such degree sequence they construct the cheapest realisation (which may not be minimal) by dynamic programming. Finally they construct, for each obtained Eulerian digraph, the cheapest extension to a valid tour, by solving a transportation problem. The cheapest tour found over all iterations is returned as the solution. Iterating and optimising over these structures is no easy task, and Cosmadakis and Papadimitriou invoke a number of graph-theoretic and combinatorial insights. For estimating the total cost of their procedure a sophisticated global counting argument is developed.

The key insight of our approach is that the machinery involving Eulerian digraphs is *not necessary* for solving MVTSP. To ensure connectivity (i.e. task (1) above), a *directed spanning tree* is sufficient. This may seem surprising, as a directed tree fails to satisfy the main property of connected Eulerian digraphs, *strong connectivity*. Observe however, that a collection of directed edges with the same outdegrees and indegrees as a valid MVTSP tour is itself a valid tour, as long as the digraph determined by the collection of edges is weakly connected. Requiring the solution to contain a tree is sufficient to avoid the case of disjoint cycles. The fact that the tree can be assumed to be *rooted*, i.e. all of its edges are directed away from some vertex, follows from the strong connectedness of the tour.[1]



a) Directed spanning tree b) Transportation problem c) MVTSP solution

FIGURE 1.2: Decomposition of a MVTSP solution in our approach, $r = (5, 10, 17, 19, 2)$.

[1]We thank Andreas Björklund for the latter observation which led to an improved run time and a simpler correctness argument.

Directed spanning trees are easier to enumerate and optimise over than Eulerian digraphs; this fact alone explains the reduced complexity of our approach. However, to obtain our main result, further ideas are needed. In particular, we find the cheapest directed spanning tree that is feasible for a given degree sequence, first by dynamic programming, then by a recursive partitioning of trees, based on centroid-decompositions.

1.1 Improved algorithms for the Many-Visits TSP

In this section we describe and analyse our three algorithms. The first, ENUM-MV is based on exact enumeration of trees (§1.1.1), the second, DP-MV uses a dynamic programming approach to find an optimal tree (§1.1.4), and the third, DC-MV is based on divide and conquer (§1.1.5 and §1.1.6). Before presenting the algorithms, we introduce some notation and structural observations that are subsequently used.

In this chapter, a *directed spanning tree* of V is a tree with vertex set V whose edges are directed; a *rooted spanning tree* is a directed spanning tree where each edge is directed *away* from some vertex $v_0 \in V$, in other words, the tree contains a directed path from v_0 to every other vertex in V. Rooted spanning trees are alternatively called *branchings*, *arborescences*, or *out-trees*. We refer to the vertex v_0 as the *root* of the tree. We observe that every valid tour contains a rooted spanning tree.

Lemma 1.2. *Let X be the edge set of a tour on V with arbitrary nonzero requests, and let $v_0 \in V$ be an arbitrary vertex. Then there is a rooted spanning tree T of X with root v_0, and a directed multigraph Z, such that $X = T + Z$.*

Proof. We choose T to be the single-source *shortest path tree* in X with source v_0. More precisely, let $d(v)$ denote the *distance* (i.e. number of edges) from v_0 to v in X. Observe that in a valid tour all vertices are mutually reachable, so $d(v)$ is finite for all $v \in V$. Let us now build T, by adding, for each vertex $v \in V \setminus \{v_0\}$, an edge wv, where $w \in V$ is an arbitrary vertex such that $d(w) = d(v) - 1$. Such a w must exist, as the predecessor of v on a shortest path from v_0 has this property. The fact that $d(\cdot)$ strictly increases along edges ensures that T is cycle-free, i.e. a tree. □

We can thus split the MVTSP problem into finding a rooted spanning tree T with an arbitrary root v_0 and an extension Z, such that $T + Z$ is a valid tour. We claim that in the decomposition $X = T + Z$ of an optimal tour X, both T and Z are optimal with respect to their degree sequences.

Lemma 1.3. *Let X be the edge set of an optimal tour for the MVTSP, let T be a rooted spanning tree, and let Z be a directed multigraph such that $X = T + Z$. Then, T has the smallest cost among all rooted spanning trees with degrees $\deg^{\mathrm{O}}_T(\cdot)$ and $\deg^{\mathrm{I}}_T(\cdot)$, and Z has the smallest cost among all directed multigraphs with degrees $\deg^{\mathrm{O}}_Z(\cdot)$ and $\deg^{\mathrm{I}}_Z(\cdot)$.*

Proof. Suppose there is a rooted spanning tree T' such that $\mathsf{cost}(T') < \mathsf{cost}(T)$, and $\deg^{\mathrm{O}}_{T'}(v) = \deg^{\mathrm{O}}_T(v)$, and $\deg^{\mathrm{I}}_{T'}(v) = \deg^{\mathrm{I}}_T(v)$ for all $v \in V$. But then $T' + Z$ is connected,

has the same degree sequence as X, while $\mathsf{cost}(T' + Z) < \mathsf{cost}(X)$, contradicting the optimality of X.

Similarly, suppose there is a directed multigraph Z' such that $\mathsf{cost}(Z') < \mathsf{cost}(Z)$, and $\deg^{\mathrm{O}}_{Z'}(v) = \deg^{\mathrm{O}}_{Z}(v)$, and $\deg^{\mathrm{I}}_{Z'}(v) = \deg^{\mathrm{I}}_{Z}(v)$ for all $v \in V$. But then $T + Z'$ is connected, has the same degree sequence as X, while $\mathsf{cost}(T + Z') < \mathsf{cost}(X)$, contradicting the optimality of X. □

1.1.1 ENUM-MV: polynomial space and superexponential time

Given the vertex set V, requests $r(v) \in \mathbb{N}$ and cost function c, we wish to find a tour of minimum cost that visits each $v \in V$ exactly $r(v)$ times. From Lemma 1.2, our first algorithm presents itself. It simply iterates over all rooted spanning trees T with vertex set V, and extends each of them optimally to a valid tour X. Among all valid tours constructed, one with smallest cost is returned (Algorithm 1.1).

This simple algorithm already improves on the previous best run time (although it is still superexponential), and reduces the space requirement from superexponential to polynomial.

Algorithm 1.1 ENUM-MV for solving MVTSP using enumeration

Input: Vertex set V, cost function c, requests $r(v)$.
Output: A tour of minimum cost that visits each $v \in V$ exactly $r(v)$ times.
1: **for** each rooted spanning tree T with root $v_0 \in V$ **do**
2: Find a minimum cost directed multigraph Z such that for all $v \in V$:
$\deg^{\mathrm{O}}_{Z}(v) = r(v) - \deg^{\mathrm{O}}_{T}(v)$,
$\deg^{\mathrm{I}}_{Z}(v) = r(v) - \deg^{\mathrm{I}}_{T}(v)$.
3: Let $X \leftarrow T + Z$.
return X with smallest cost

The correctness of the algorithm is immediate: from Lemma 1.2 it follows that every multigraph X considered is valid (connected, and with degrees matching the visit requests), and by Lemma 1.3, the optimal tour $X^\star$ must be considered during the execution.

The iteration of Line 1 requires us to enumerate all labelled trees with vertex set V. There are n^{n-2} such trees [23], and standard techniques can be used to enumerate them with a constant overhead per item (see e.g. Kapoor and Ramesh [68]). For each considered tree we orient the edges in a unique way, away from v_0.

Let T be the current tree. In Line 2 we find a minimum cost directed multigraph Z, with given outdegree and indegree sequence, such as to extend T into a valid tour. If, for some vertex v, it holds that $\deg^{\mathrm{I}}_{T}(v) > r(v)$ or $\deg^{\mathrm{O}}_{T}(v) > r(v)$, we proceed to the next spanning tree, since the current tree cannot be extended to a valid tour. Observe that this may happen only if $r(v) < n - 1$, for some $v \in V$. Otherwise, we find the optimal Z by solving a transportation problem in polynomial time. Throughout the execution we remember the best tour found so far, which we output in the end. We describe next the

transportation subroutine, which is common to all our algorithms, and is essentially the same as in the Cosmadakis-Papadimitriou algorithm.

The transportation problem. The subproblem we need to solve is finding a minimum cost directed multigraph Z over vertex set V, with given outdegree and indegree requirements. We can map this problem to an instance of the Hitchcock transportation problem [58], where the goal is to transport a given amount of goods from N warehouses to M outlets with given pairwise shipping costs. Note that this is a special case of the more general min-cost max-flow problem [32].

More precisely, let us define a digraph with vertices $\{\sigma, \tau\} \cup \{\sigma_i, \tau_i \mid i \in V\}$. Edges are $\{(\sigma, \sigma_i), (\tau_i, \tau) \mid i \in V\}$, and $\{(\sigma_i, \tau_j) \mid i, j \in V\}$. We set cost 0 to edges (σ, σ_i) and (τ_i, τ), and cost $c(ij)$ (i.e. the costs given in the MVTSP instance) to (σ_i, τ_j). We set capacity ∞ to edges (σ_i, τ_j), capacities $r(i) - \deg^{\mathrm{O}}_T(i)$ to (σ, σ_i), and $r(i) - \deg^{\mathrm{I}}_T(i)$ to (τ_i, τ). We can think of these capacities as the supplies at the warehouses and demands at the outlets, respectively.

The construction is identical to the one used by Cosmadakis and Papadimitriou, apart from the fact that in our case the capacity of (σ, σ_i) may be different from the capacity of (τ_i, τ). Observe that the sum of capacities of (σ, σ_i)-edges still equals the sum of capacities of (τ_i, τ)-edges over all $i \in V$; thus, a maximal $\sigma - \tau$ flow saturates all these edges. The amount of flow transmitted on the edge (σ_i, τ_j) gives the multiplicity of edge (i, j) in the sought after multigraph, for all $i, j \in V$. A minimum cost maximum flow clearly maps to a minimum cost edge set with the given degree constraints.

In the Cosmadakis-Papadimitriou algorithm, the transportation subproblems are solved via the scaling method of Edmonds and Karp [32]. This algorithm proceeds by solving $O(\log r)$ approximate versions of the problem, where the costs are the same as in the original problem, but the capacities are scaled (i.e. divided and rounded down) by a factor 2^p for $p = \lceil \log r \rceil, \ldots, 0$. Each approximate problem is solved in $O(n^3)$ time, by performing flow augmentations on the optimal flow found in the previous approximation, multiplied by two. The overall run time for solving the described transportation problem is therefore $O(n^3 \log r)$.

Cosmadakis and Papadimitriou describe an improvement which also applies for our case. Namely, they show that the total run time for solving several instances with the same costs can be reduced, if the capacities on corresponding edges in two different instances may differ by at most n. The strategy is to solve all but the last $\log n$ approximate problems only once, as these are (essentially) the same for all instances. For different instances we only need to solve the last $\log n$ approximate problems, i.e. at the finest levels of approximation. This gives a run time of $O(n^3 \log r)$ for solving the "master problem", and $O(n^3 \log n)$ for solving each individual instance. We refer to Cosmadakis and Papadimitriou [29] as well as to Edmonds and Karp [32] for details.

In our case, the different instances of the transportation problem are for finding the directed multigraphs Z for different trees T. Each of these instances agree in the underlying graph and cost function, and may differ only in the capacities. As the maximum degree of each tree T is at most $n - 1$, the differences are bounded, as required.

As an alternative to the Edmonds-Karp algorithm, we may solve the arising transportation problems with a *strongly polynomial* algorithm, e.g. the one by Orlin [92] or its extension due to Kleinschmidt and Schannath [74]. Note that these algorithms were not yet available when Cosmadakis and Papadimitriou obtained their result. The run time for the transportation subproblem then becomes $O(n^3 \log n)$, i.e. independent of r. Such an improvement is likely of theoretical interest only; furthermore, it assumes that operations on the requests $r(v)$ take constant time. If this assumption is unrealistic, e.g. if r is exponential in n, we may fall back to the Edmonds-Karp algorithm, with the term $O(n^3 \log r)$ added to the run time.

Analysis of Algorithm 1.1. We iterate over all $O(n^{n-2})$ rooted spanning trees and solve a transportation problem for each, with run time $O(n^3 \log n)$. The total run time $O(n^{n+1} \log n)$ follows. The space requirement of the algorithm is dominated by that of solving a (single) transportation problem, and of storing the edge set of a single tour (apart from minor bookkeeping).

1.1.2 Degree sequences of directed trees

Observe that the solution of the transportation problem depends only on the degree sequence of the current tree T, and not the actual edges of T. Therefore, different trees with the same degree sequence can be extended in the same way. Notice that several trees may have the same degree sequence; in an extreme case, all $(n-2)!$ simple Hamiltonian paths with the same endpoints have the same degree sequence. This means that ENUM-MV considers the same transportation problem multiple times.

Characterising degree sequences of directed spanning trees

In order to omit the redundant calculations, the straightforward idea is to group trees by their degree sequences, and calculate the underlying transportation problem solution only once for each group. We start exploring this idea by characterising the feasible degree sequences of trees.

Although our final algorithm will use rooted spanning trees, as a warm-up we first consider directed spanning trees. Observe that since every rooted spanning tree is a directed spanning tree as well, Lemmas 1.2 and 1.3 hold for (non-rooted) directed spanning trees as well. Therefore we start with directed spanning trees for the sake of simplicity, then show that the same results hold when switching to rooted spanning trees, with an improved time complexity.

The feasibility of degree sequences for various graph classes is a well-studied subject, see e.g. [18, 34, 44, 53, 55, 72, 75]. The simple condition we state is similar to the condition for *undirected* trees, given by Berge [17, page 117].

Lemma 1.4. *Let $V = \{x_1, \dots, x_n\}$ be a set of vertices, where $n \geqslant 2$. There is a directed spanning tree of V whose outdegrees and indegrees are respectively $\deg^{\mathrm{O}}(\cdot)$ and $\deg^{\mathrm{I}}(\cdot)$, if and only if*

(i) $\deg^{\mathrm{O}}(x_i) + \deg^{\mathrm{I}}(x_i) > 0$ *for* $i = 1, \ldots, n$, *and*

(ii) $\sum_i \deg^{\mathrm{O}}(x_i) = \sum_i \deg^{\mathrm{I}}(x_i) = n - 1$.

Moreover, the number of such sequences is

$$\mathtt{DS}(n) = \sum_{z=1}^{n-1} \binom{n}{z} \binom{n-2}{n-z-1} \binom{2n-z-2}{n-1} < n \cdot 2^{3.471n} = O(11.09^n) \ . \tag{1.1}$$

Proof. In the forward direction, a tree on n vertices has $n - 1$ edges, proving *(ii)*. As the tree is connected, each vertex has at least one incident edge, outgoing or incoming, proving *(i)*.

In the backward direction, we argue by induction on n. Let us denote the total degree of a vertex x_i as $\deg(x_i) = \deg^{\mathrm{O}}(x_i) + \deg^{\mathrm{I}}(x_i)$. In the case $n = 2$, there are two vertices, and from *(i)* and *(ii)* it follows that one of them has indegree 1, outdegree 0, and the other has indegree 0, outdegree 1. Thus, we connect them with an appropriate edge, satisfying the degree requirements.

Consider now the case of n vertices. From *(i)* and *(ii)* it follows that there is a vertex x_i such that $\deg(x_i) = 1$. Suppose first that $\deg^{\mathrm{I}}(x_i) = 1$. Then for some $j \in \{1, \ldots, n\} \setminus \{i\}$, we have that $\deg^{\mathrm{O}}(x_j) \geqslant 1$ and $\deg(x_j) \geqslant 2$. We decrease $\deg^{\mathrm{O}}(x_j)$ by one. It can be seen that conditions *(i)* and *(ii)* hold for $V \setminus \{x_i\}$. By induction, we can build a tree on $V \setminus \{x_i\}$, and attach x_i to this tree as a leaf, with the edge (x_j, x_i). The case when $\deg^{\mathrm{O}}(x_i) = 1$ is symmetric.

Now we turn to the number of such sequences. Let $\mathtt{DS}(n)$ denote the number of different pairs of sequences $(d'_1, \ldots, d'_n), (d''_1, \ldots, d''_n)$, that are feasible for an directed spanning tree, i.e. for vertex set $V = \{x_1, \ldots, x_n\}$, for some directed spanning tree T, we have $\deg^{\mathrm{O}}_T(x_i) = d'_i$, and $\deg^{\mathrm{I}}_T(x_i) = d''_i$, for all $i \in \{1, \ldots, n\}$. We show the exact expression and upper bound for $\mathtt{DS}(n)$.

We enumerate outdegree sequences (or out-sequences) by their number of zero-entries z, and for each out-sequence we count the number of possible indegree sequences (or in-sequences).

According to condition *(i)*, for an out-sequence with z zeros ($1 \leqslant z \leqslant n - 1$), the corresponding in-sequences can have at most $n - z$ zeros. For each of the $\binom{n}{z}$ ways to distribute z zeros among n entries, there are $\binom{n-2}{n-z-1}$ ways of distributing $n - 1$ degrees among the $n - z$ non-zero entries. (We can think of this as choosing $n - z - 1$ internal separators to split the items, out of the $n - 2$ possibilities of a separator.)

As no vertex can have zero outdegree and indegree, we assign one indegree to each node with zero outdegrees, leaving us with the task of distributing $n - z$ indegrees between n nodes arbitrarily. This can be done in $\binom{2n-z-2}{n-z-1} = \binom{2n-z-2}{n-1}$ ways, choosing the position of $n - z - 1$ separators in a sequence of $(n - 1) + (n - z - 1)$ objects (items and separators combined) in total, yielding the result. [2]

[2] This counting approach is also known as *stars and bars* [37, II.5].

Finally, we use Stirling's approximation for $\binom{n}{k}$ with the binary entropy function (see e.g. [85, page 2]) and find that the summed quantity is maximized when $z/n \approx 0.382$. □

Unfortunately, subsequent trees might have different degree sequences, when enumerated using standard techniques [68], and storing the current best (minimal cost) tree or transportation problem solution for each degree sequence would lead to an exponential space complexity. Hence we change the order of the trees considered. We iterate through each possible degree sequence, and for each such sequence choose a minimum cost tree and calculate the solution to the relevant transportation problem.

Computing a minimum-cost spanning tree for a degree sequence is an NP-complete problem, as it contains the PATH TSP as a special case, when $\deg^{\mathrm{I}}(v) = 1$ for all vertices but the starting one s and $\deg^{\mathrm{O}}(v) = 1$ for all vertices but the endvertex t. For this reason we find an optimal tree using exhaustive approaches: first by simply iterating through all directed spanning trees grouped by their degree sequences in §1.1.3, then calculating a minimum cost directed spanning tree for each degree sequence using dynamic programming in §1.1.4 and finally by a recursive approach in §1.1.5.

Generating degree sequences of directed trees

Algorithm 1.2 iterates over all feasible degree sequences of directed trees. It begins by first generating all feasible outdegree sequences outseq, which means distributing $n-1$ outdegrees among n vertices, allowing for vertices with zero degree. This is equivalent to generating all $\binom{2n-2}{n-1}$ ways of choosing $n-1$ positions of separators between $n-1$ items, and it is done in Line 4 of the algorithm. Then it generates all matching indegree sequences inseq in Line 7, while making sure to fulfil the conditions of Lemma 1.4, in particular that $\mathsf{inseq}[i] \geqslant 1$ holds for all $i \in \{1, \dots, n\}$, such that $\mathsf{outseq}[i] = 0$. This is done by fixing one indegree at such vertices $v \in Z$ in Line 9, then distributing the remaining $n-1-|Z|$ degree among $n-1$ vertices; similarly to the outsequence, this means all positions $n-1$ separators between $n-1-|Z|$ elements. Finally, Algorithm 1.2 outputs all pairs $(\mathsf{outseq}, \mathsf{inseq})$.

Algorithm 1.2 Generating all degree sequences of directed trees.

1: **Input:** Positive integer n.
2: **Output:** A generator of all directed tree degree sequences on n vertices.
3: **procedure** DIRECTEDTREEDS(n)
4: **for** every array1 in COMBINATIONS($2n-2, n-1$) **do**
5: outseq ← COMBINATIONTOSEQUENCE(array1)
6: $Z \leftarrow$ indices i with $\mathsf{outseq}[i] = 0$
7: **for** every array2 in COMBINATIONS($2n - |Z| - 2, n-1$) **do**
8: inseq ← COMBINATIONTOSEQUENCE(array2)
9: $\mathsf{inseq}[i] \leftarrow \mathsf{inseq}[i] + 1$ for each $i \in Z$
10: **yield** $(\mathsf{outseq}, \mathsf{inseq})$

Note that Algorithm 1.2 does not return the list of degree sequences (outseq, inseq) at once, it is instead a *generator* that serves as the head of the **for** loop in the exact algorithms presented later in the chapter, outputting a different degree sequence in every iteration. We used the keyword **yield** instead of **return** as a Python-style indication that the algorithm does not terminate after "returning" the first result.

Algorithm 1.2 uses two simple procedures. First, COMBINATIONS outputs all possible *positions* of the separators, when distributing m elements into k parts, then COMBINATIONSTOSEQUENCE converts it into a sequence containing the number of elements in each part. For the sake of completeness, we included these procedures in Algorithms 1.3 and 1.4 below.

Algorithm 1.3 Generating all possible k-subsets of $\{1, \ldots, m\}$.

```
1:  Input: Positive integers m and k.
2:  Output: A generator of all possible k-subsets of {1,...,m}.
3:  procedure COMBINATIONS(m, k)
4:      comb ← [1,...,k]
5:      yield comb
6:      while true do
7:          i ← last index such that comb[i] ≠ m − k + i, if no such index exists, break
8:          comb[i] ← comb[i] + 1
9:          for every index j ← i+1,...,k do
10:             comb[j] ← comb[j − 1] + 1
11:         yield comb
```

The procedure COMBINATIONS implements algorithm NEXKSB by Nijenhuis and Wilf [90, page 26]. It takes two integers as input, m and k, and generates all (ordered) subsets of size k, of the base set $\{1, \ldots, m\}$. It starts with the set $[1, 2, \ldots, k]$ and generates all k-subsets in lexicographical order, up to $[m-k+1, m-k+2, \ldots, m-k+k] = [m-k+1, m-k+2, \ldots, m]$. In every iteration, it increments the rightmost entry comb$[i]$ not equal to $m - k + i$, and makes comb$[j]$ equal to comb$[j - 1] + 1$ for all j indices between $[i + 1, k]$. The algorithm stops when there are no such indices i, which happens when reaching $[m - k + 1, \ldots, m]$.

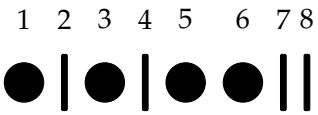


FIGURE 1.3: Sequence $[2, 4, 7, 8]$ representing the degree sequence $[1, 1, 2, 0, 0]$

An example is shown in Figure 1.3. The corresponding integer sequence would be $[1, 1, 2, 0, 0]$, however COMBINATIONS returned the sequence $[2, 4, 7, 8]$, that is, a sequence of the positions (separating bars) of the integer sequence above. In order to obtain the actual degree sequence, we use a short script COMBINATIONTOSEQUENCE in Algorithm 1.4 that converts the sequence with the positions of the bars to the degree sequence.

Algorithm 1.4 Converting pos into $[\mathsf{pos}[1], \mathsf{pos}[2] - \mathsf{pos}[1] - 1, \ldots, m + k - \mathsf{pos}[k]]$.

Input: List of positions $\mathsf{pos} = \mathsf{pos}[1], \ldots, \mathsf{pos}[k]$, integer m.
Output: A sequence of $k+1$ integers that sum up to m.
procedure COMBINATIONTOSEQUENCE(pos, m)
    $\mathsf{seq}[1] \leftarrow \mathsf{pos}[1] - 1$
    for every index $i \leftarrow 2, \ldots, k$ **do**
        $\mathsf{seq}[i] \leftarrow \mathsf{pos}[i] - \mathsf{pos}[i-1] - 1$
    $\mathsf{seq}[k+1] \leftarrow m + k - \mathsf{pos}[k]$
return seq

Generating trees with a given degree sequence

Now that we can iterate through degree sequences, we turn into generating all directed trees with a given degree sequence. In the proof of Lemma 1.5, we generate *one* directed tree from its degree sequence. In this subsection we show how to generate *all* trees for a given degree sequence (Algorithm 1.5).

Algorithm 1.5 Generating all directed trees for a given degree sequence

Input: Degree sequence $(\deg^{\mathrm{O}}, \deg^{\mathrm{I}})$ and forest $\mathbf{x}^{\mathrm{stub}}$.
Output: A generator of all trees with degree sequence $(\deg^{\mathrm{O}}, \deg^{\mathrm{I}})$, attached to $\mathbf{x}^{\mathrm{stub}}$.
procedure BUILDDIRECTEDTREE($\deg^{\mathrm{O}}$, $\deg^{\mathrm{I}}$, $\mathbf{x}^{\mathrm{stub}}$)
    if $\sum \deg(\cdot) = 2$ **then**
        Let i and j be the indices where $\deg^{\mathrm{O}}(x_i) = 1$ and $\deg^{\mathrm{I}}(x_j) = 1$.
        yield $\mathbf{x}^{\mathrm{stub}} \cup \{x_i x_j\}$
    else $(\sum \deg(\cdot) > 2)$
        Let i be the first index where $\deg(x_i) = 1$.
        if $\deg^{\mathrm{O}}(x_i) = 1$ **then**
            for every index $j \neq i$, such that $\deg(x_j) \geqslant 2$ and $\deg^{\mathrm{I}}(x_j) \geqslant 1$ **do**
                $\left(\deg^{\mathrm{O}'}, \deg^{\mathrm{I}'}\right) \leftarrow \left(\deg^{\mathrm{O}}, \deg^{\mathrm{I}}\right)$
                $\deg^{\mathrm{O}'}(x_i) \leftarrow \deg^{\mathrm{O}'}(x_i) - 1$
                $\deg^{\mathrm{I}'}(x_j) \leftarrow \deg^{\mathrm{I}'}(x_j) - 1$
                yield BUILDDIRECTEDTREE($\deg^{\mathrm{O}'}$, $\deg^{\mathrm{I}'}$, $\mathbf{x}^{\mathrm{stub}} \cup \{x_i x_j\}$)
        else $(\deg^{\mathrm{I}}(x_i) = 1)$
            for every index $j \neq i$, such that $\deg(x_j) \geqslant 2$ and $\deg^{\mathrm{O}}(x_j) \geqslant 1$ **do**
                $\left(\deg^{\mathrm{O}'}, \deg^{\mathrm{I}'}\right) \leftarrow \left(\deg^{\mathrm{O}}, \deg^{\mathrm{I}}\right)$
                $\deg^{\mathrm{O}'}(x_j) \leftarrow \deg^{\mathrm{O}'}(x_j) - 1$
                $\deg^{\mathrm{I}'}(x_i) \leftarrow \deg^{\mathrm{I}'}(x_i) - 1$
                yield BUILDDIRECTEDTREE($\deg^{\mathrm{O}'}$, $\deg^{\mathrm{I}'}$, $\mathbf{x}^{\mathrm{stub}} \cup \{x_j x_i\}$)

The initial call to the BUILDDIRECTEDTREE procedure is with a feasible input degree sequence $(\deg^{\mathrm{O}}, \deg^{\mathrm{I}})$ and an empty "stub" ($\mathbf{x}^{\mathrm{stub}} = \{\}$) as arguments. The algorithm finds the first vertex that is either a leaf of the final tree or whose subtree is already complete (that is, $\deg(x_i) = 1$), then it finds all possibilities for attaching x_i to the rest of the tree: this means finding indices $j \neq i$ such that $\deg^{\mathrm{I}}(x_j) \geqslant 1$ if $\deg^{\mathrm{O}}(x_i) = 1$ holds, or such that $\deg^{\mathrm{O}}(x_j) \geqslant 1$ if $\deg^{\mathrm{I}}(x_i) = 1$ holds. The procedure is then called recursively

with modified arguments: edge x_ix_j (or x_jx_i) is added to the stub and the degrees of x_i and x_j are decremented.

At each recursive level there are as many new calls as possible candidates for the next edge, with both degree demands decreased by one, and with the stub gaining one additional edge. During the intermediate calls the stub may be disconnected, i.e. it represents a forest. At the $(n-1)$-th level exactly two degree-one vertices remain, say, x_i with outdegree 1 and x_j with indegree 1. Adding the edge x_ix_j finishes the construction.

Observe that there are no "dead ends" during this process, i.e. every call of the procedure BUILDDIRECTEDTREE eventually results in a valid directed tree in the last level of the recursion, and there are no discarded graphs during the process. The argument mirrors the one in the proof of Lemma 1.5. In every call to BUILDDIRECTEDTREE, the degree sequence $(\deg^{\mathrm{O}}, \deg^{\mathrm{I}})$ encodes a valid directed tree on the subset of vertices with nonzero degree. Let h denote the number of vertices with nonzero degree. We argue by induction on h that the construction succeeds. For $h = 2$ (Lines 2–4) this is clear. For $h \geqslant 3$ (Lines 5–18), we attach x_i to a vertex x_j of a valid tree, constructed by the recursive call with $h-1$ vertices with nonzero degree.

For every directed tree with the required degree-sequence, repeatedly peeling off the lowest-index leaf gives a valid edge-insertion order that is found by the algorithm. The order in which vertices are considered is essentially the *Prüfer sequence* [88] of the tree. Due to this nature of the procedure, the resulting objects will be indeed spanning trees. Moreover, the trees constructed by the procedure have different Prüfer sequences, hence no tree is constructed more than once.

Similarly to Algorithm 1.2, Algorithm 1.5 also does not return all directed trees with a certain degree sequence at once, instead serves as the head of a **for** loop that outputs a different tree in every iteration.

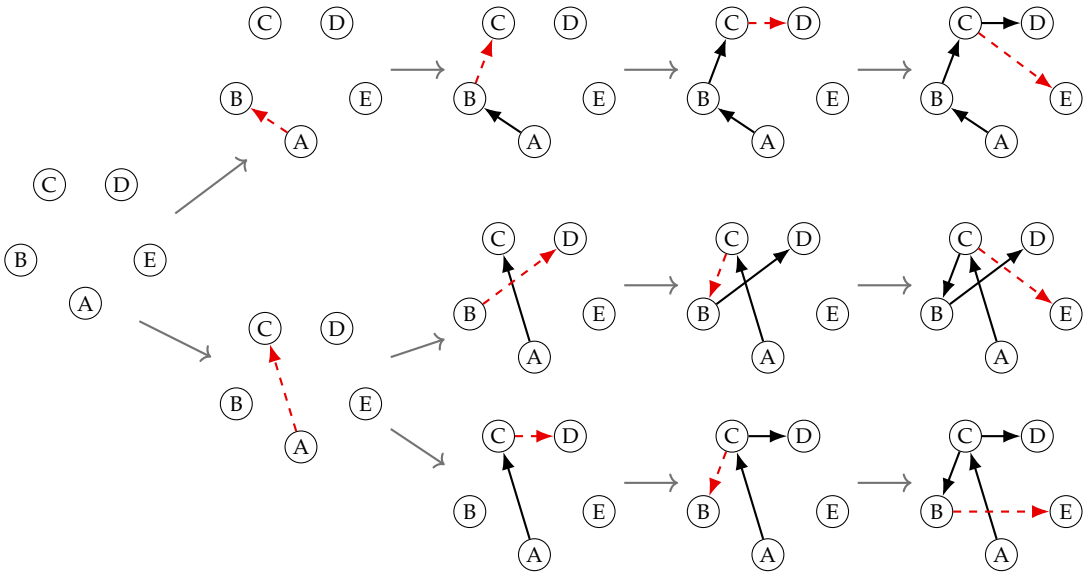


FIGURE 1.4: Example: constructing all directed trees realising the degree sequence $(1,1,2,0,0),(0,1,1,1,1)$.

Example Suppose we want to build all directed trees realising the degree sequence $(1,1,2,0,0),(0,1,1,1,1)$; observe that such trees are rooted at vertex A. Let us label the vertices by A, B, C, D and E. The lowest indexed leaf is A with outdegree 1, hence we need to consider all non-leaves with a positive indegree: B and C. We first consider B, and after adding the edge AB we update the degree sequence to $(0,1,2,0,0),(0,0,1,1,1)$. The procedure continues the same way, considering only vertices B–E and vertex B as the leaf with the lowest index. The whole construction is shown in the first row of Figure 1.4.

The algorithm then considers the edge AC instead of AB, resulting in an updated degree sequence $(0,1,2,0,0),(0,1,0,1,1)$. The next leaf is D, with two possible vertices to connect, B or C. The two paths of construction are shown in the second and third rows of Figure 1.4. Note that the algorithm did not start by connecting the root vertex A because it was the root, but because it was the leaf with the lowest-index at the beginning of the construction.

Degree sequences of rooted directed trees

Now we turn to the analysis of rooted trees, and show that they yield an improved exact algorithm.

Lemma 1.5. *Let $V = \{x_1, \ldots, x_n\}$ be a set of vertices, where $n \geqslant 2$. There is a rooted spanning tree of V with root x_1, whose outdegrees and indegrees are respectively $\deg^{\mathrm{O}}(\cdot)$ and $\deg^{\mathrm{I}}(\cdot)$, if and only if*

(i) $\deg^{\mathrm{I}}(x_1) = 0$,

(ii) $\deg^{\mathrm{I}}(x_i) = 1$ *for* $i = 2, \ldots, n$,

(iii) $\deg^{\mathrm{O}}(x_1) > 0$, *and*

(iv) $\sum_i \deg^{\mathrm{O}}(x_i) = n - 1$.

Moreover, the number of such sequences is

$$\texttt{DSr}(n) = \binom{2n-3}{n-1} \leqslant 4^n \ . \tag{1.2}$$

Proof. In the forward direction, in a directed spanning tree all non-root vertices have exactly one parent, proving *(i)* and *(ii)*. The root must have at least one child *(iii)*, and the total number of edges is $n-1$, proving *(iv)*.

In the backward direction, we argue by induction on n. In the case $n = 2$, we have $\deg^{\mathrm{I}}(x_1) = \deg^{\mathrm{O}}(x_2) = 0$, and $\deg^{\mathrm{I}}(x_2) = \deg^{\mathrm{O}}(x_1) = 1$, hence an edge x_1x_2 satisfies the degree requirements.

Consider now the case of $n > 2$ vertices. From *(ii)–(iv)* it follows that for some $k \in \{2, \ldots, n\}$, we have $\deg^{\mathrm{I}}(x_k) = 1$ and $\deg^{\mathrm{O}}(x_k) = 0$, i.e. x_k is a leaf.

Let x_j be a vertex for some $j \in \{1, \ldots, n\} \setminus \{k\}$ such that $\deg^{\mathrm{O}}(x_j) \geqslant 1$, and $\deg^{\mathrm{I}}(x_j) + \deg^{\mathrm{O}}(x_j) \geqslant 2$; by *(ii)–(iv)* there must be such a vertex. We decrease $\deg^{\mathrm{O}}(x_j)$ by one.

Conditions *(i)–(iv)* clearly hold for $V \setminus \{x_k\}$. By induction, we can build a tree on $V \setminus \{x_k\}$, and attach x_k to this tree as a leaf, with the edge (x_j, x_k).

Let $\texttt{DSr}(n)$ denote the number of different pairs of sequences $(d'_1, \ldots, d'_n), (d''_1, \ldots, d''_n)$ that are feasible for a directed spanning tree, i.e. for vertex set $V = \{x_1, \ldots, x_n\}$, for some directed spanning tree T with root x_1, we have $\deg^{\mathrm{O}}_T(x_i) = d'_i$, and $\deg^{\mathrm{I}}_T(x_i) = d''_i$, for all $i \in \{1, \ldots, n\}$. By the properties *(i)–(iv)*, $\texttt{DSr}(n)$ equals the number of ways to distribute $n-1$ outdegrees to n vertices, such that a designated vertex has nonzero outdegree. This task is the same as distributing $n-2$ balls arbitrarily into n bins, of which there are $\binom{2n-3}{n-1} = O(2^{2n}) = O(4^n)$ ways. □

According to Lemma 1.5, the number of degree sequences of rooted trees is smaller than that of all directed trees, albeit still exponential. However, as we show later, using rooted trees instead of directed trees can still reduce the number of iterations. Moreover, degree sequences of rooted trees can also be generated via a simpler method, due to their properties, see Algorithm 1.6.

Algorithm 1.6 Generating all degree sequences of rooted trees.

```
1: Input: Positive integer n.
2: Output: A generator of all rooted tree degree sequences on n vertices.
3: procedure ROOTEDTREEDS(n)
4:     inseq[1] ← 0
5:     inseq[i] ← 1 for i = {2, ..., n}
6:     for every array in COMBINATIONS(2n − 3, n − 1) do
7:         outseq ← COMBINATIONTOSEQUENCE(array)
8:         outseq[1] ← outseq[1] + 1
9:     yield (outseq, inseq)
```

ROOTEDTREEDS constructs degree sequences that satisfy the conditions *(i)–(iv)* of Lemma 1.5. It first fixes the indegrees (0 for the root, 1 for the other vertices), then proceeds to assign the outdegrees. This is done by fixing one outdegree for the root (Line 8), then distributing the rest of $n-2$ outdegrees among $n-1$ vertices (Line 6).[3]

Since the indegree sequence of a rooted tree is fixed for a size n, the algorithm generating rooted trees also becomes simplified, compared to Algorithm 1.5, the directed tree counterpart. The simpler algorithm BUILDROOTEDTREE for rooter trees is presented in Algorithm 1.7.

The initial call to the BUILDROOTEDTREE procedure is with a feasible input degree sequence $(\deg^{\mathrm{O}}, \deg^{\mathrm{I}})$ and an empty "stub" ($\mathbf{x}^{\text{stub}} = \{\}$) as arguments. The algorithm finds the first *unattached* vertex that is either a leaf of the final tree or whose subtree is already complete (that is, $\deg^{\mathrm{O}}(x_i) = 0$), then it finds all possibilities for attaching x_i to the rest of the tree. Observe that the fact that x_i has no more capacity for outgoing edges prevents cycles. The procedure is then called recursively with modified arguments:

[3]Note that Algorithm 1.2 and Algorithm 1.6 would be implemented by performing the operations in Line 9 and Line 8, respectively, outside of the (second) **for**-loop; the operations were presented in a different order as it allows for a simpler presentation.

Algorithm 1.7 Generating all rooted directed trees for a given degree sequence

Input: Degree sequence $(\deg^{\mathrm{O}}, \deg^{\mathrm{I}})$ and forest $\mathbf{x}^{\text{stub}}$.
Output: A generator of all rooted trees with degree sequence $(\deg^{\mathrm{O}}, \deg^{\mathrm{I}})$, attached to $\mathbf{x}^{\text{stub}}$.

1: **procedure** BUILDROOTEDTREE($\deg^{\mathrm{O}}$, $\deg^{\mathrm{I}}$, $\mathbf{x}^{\text{stub}}$)
2: **if** $\sum \deg^{\mathrm{I}}(\cdot) = 1$ **then**
3: Let i be the index where $\deg^{\mathrm{I}}(x_i) = 1$.
4: **yield** $\mathbf{x}^{\text{stub}} \cup \{(x_1, x_i)\}$
5: **else**
6: Let i be the first index where $\deg^{\mathrm{O}}(x_i) = 0$ and $\deg^{\mathrm{I}}(x_i) = 1$.
7: **for** every index $j \neq i$, such that $\deg^{\mathrm{O}}(x_j) + \deg^{\mathrm{I}}(x_j) \geqslant 2$ **do** ▷ attach x_i to parent x_j
8: $\left(\deg^{\mathrm{O}'}, \deg^{\mathrm{I}'}\right) \leftarrow \left(\deg^{\mathrm{O}}, \deg^{\mathrm{I}}\right)$
9: $\deg^{\mathrm{O}'}(x_j) \leftarrow \deg^{\mathrm{O}'}(x_j) - 1$
10: $\deg^{\mathrm{I}'}(x_i) \leftarrow 0$
11: BUILDROOTEDTREE($\deg^{\mathrm{O}'}$, $\deg^{\mathrm{I}'}$, $\mathbf{x}^{\text{stub}} \cup \{(x_j, x_i)\}$)

edge (x_j, x_i) is added to the stub and the out-degree of x_j and the in-degree of x_i are decremented.

At each recursive level there are as many new calls as possible candidates for the next edge, with both degree demands decreased by one, and with the stub gaining one additional edge. During the intermediate calls the stub may be disconnected, i.e. it represents a forest. At the $(n-1)$-th level exactly two degree-one vertices remain, say, x_i with in-degree 1 and the root x_1 with out-degree 1. Adding the edge (x_1, x_i) finishes the construction.

1.1.3 Improved enumeration algorithm

The improved algorithm is presented as Algorithm 1.8. It iterates over all trees grouped by their degree sequences, and solves the transportation problem only once for each degree sequence (there are $\texttt{DSr}(n)$ of them). Procedure ROOTEDTREEDS from Algorithm 1.6 serves as the **for** loop in Line 1, and BUILDROOTEDTREE from Algorithm 1.7 serves as the **for** loop in Line 3.

Algorithm 1.8 ENUM-MV for solving MVTSP using enumeration (improved)

Input: Vertex set V, cost function c, requests $r(v)$.
Output: A tour of minimum cost that visits each $v \in V$ exactly $r(v)$ times.

1: **for** each feasible degree sequence $\deg^{\mathrm{O}}(\cdot), \deg^{\mathrm{I}}(\cdot)$ of a directed spanning tree with root v_0 **do**
2: Find a minimum cost directed multigraph Z such that for all $v \in V$:
$\deg^{\mathrm{O}}_Z(v) = r(v) - \deg^{\mathrm{O}}(v)$,
$\deg^{\mathrm{I}}_Z(v) = r(v) - \deg^{\mathrm{I}}(v)$.
3: **for** each directed spanning tree T with $\deg^{\mathrm{O}}_T = \deg^{\mathrm{O}}$ and $\deg^{\mathrm{I}}_T = \deg^{\mathrm{I}}$ **do**
4: Let $X \leftarrow T + Z$.
return X with smallest cost

The correctness is immediate, as all directed spanning trees T are still considered as before.

Since we need $O(n)$ time to calculate the cost of a spanning tree, we obtain the run time $O(n^{n-2} \cdot n + \texttt{DSr}(n) \cdot n^4 \log n) = O(n^{n-1} + 4^n n^4 \log n)$. The space requirement is asymptotically unchanged.

1.1.4 DP-MV: exponential space and single-exponential time

Next, we improve the run time to single-exponential, by making use of Lemma 1.3. Specifically, we observe that for every feasible degree sequence only the tree with the minimum cost needs to be considered. The enumeration in Algorithm 1.5 can easily be modified to return, instead of all trees, just one with smallest cost, this, however, would not improve the asymptotic run time. Instead, in this section we describe a dynamic programming approach resembling the algorithms by Bellman [13], and Held and Karp [56] for *directly* computing the best directed tree for a given degree sequence.

The outline of the algorithm is shown in Algorithm 1.9, and it is identical for DP-MV, DC-MV and DC-MV2, described in §1.1.4–1.1.6. The algorithms DP-MV, DC-MV and DC-MV2 differ in the way they find the minimum cost directed tree, i.e. Line 2 of the generic Algorithm 1.9.

Algorithm 1.9 Solving MVTSP by optimising over trees (common for DP-MV, DC-MV and DC-MV2).

Input: Vertex set V, cost function c, requests $r(v)$.
Output: A tour of minimum cost that visits each $v \in V$ exactly $r(v)$ times.
1: **for** each feasible degree sequence $\deg^{\mathrm{O}}(\cdot), \deg^{\mathrm{I}}(\cdot)$ of a directed spanning tree **do**
2: Find a minimum-cost directed tree T with $\deg^{\mathrm{I}}_T = \deg^{\mathrm{I}}$ and $\deg^{\mathrm{O}}_T = \deg^{\mathrm{O}}$.
3: Find a minimum-cost directed multigraph Z such that for all $v \in V$:
$\deg^{\mathrm{I}}_Z(v) = r(v) - \deg^{\mathrm{I}}_T(v)$,
$\deg^{\mathrm{O}}_Z(v) = r(v) - \deg^{\mathrm{O}}_T(v)$.
4: Let $X \leftarrow T + Z$.
return X with smallest cost

The dynamic programming approach (DP-MV) resembles Algorithm 1.5 for iterating over all directed trees with a given degree sequence. Specifically, let $(\deg^{\mathrm{O}}, \deg^{\mathrm{I}})$ be the degree sequence for which we wish to find a minimum-cost tree. We build a dynamic programming table $\mathcal{T}$ that holds an optimal tree and its cost for *every* feasible degree sequence on every possible proper subsets of V. The solution can thus be read from $\mathcal{T}[\deg^{\mathrm{O}}, \deg^{\mathrm{I}}]$. Observe that specifying a degree sequence allows us to restrict the problem to arbitrary subsets of V, by simply setting the degrees of non-participating vertices to zero.

To compute $\mathcal{T}[\deg^{\mathrm{O}}, \deg^{\mathrm{I}}]$, we find the leaf with the smallest index x_i and all non-leaves x_j that may be connected to x_i by an edge in the optimal tree (similarly to Algorithm 1.5). For each choice of x_j we compute the optimal tree by adding the connecting edge $x_j x_i$ to the optimal tree over $V \setminus \{x_i\}$, with the degree sequence suitably updated.

The correctness of the dynamic programming algorithm follows from an observation similar to Lemma 1.3; every subtree of the optimal tree must be optimal for its degree

sequence, as otherwise it could be swapped for a cheaper subtree. The details are shown in Algorithm 1.10.

Algorithm 1.10 DP-MV for generating the optimal directed tree for a given degree sequence.

Input: Degree sequence $(\deg^{\mathrm{O}}, \deg^{\mathrm{I}})$, and edge costs c.
Output: (T, cost), where T is optimum tree with root x_1, degrees $\deg^{\mathrm{O}}, \deg^{\mathrm{I}}$ and cost denotes its cost.

procedure $\mathcal{T}\left[\deg^{\mathrm{O}}, \deg^{\mathrm{I}}\right]$
  if $\sum \deg^{\mathrm{I}}(\cdot) = 1$ **then**
    Let i be the index where $\deg^{\mathrm{I}}(x_i) = 1$.
    return $(\{x_1x_i\}, c(x_1x_i))$
  else $(\sum \deg^{\mathrm{I}}(\cdot) > 1)$
    Let i be the first index where $\deg^{\mathrm{O}}(x_i) = 0$ and $\deg^{\mathrm{I}}(x_i) = 1$.
    for every index $j \neq i$, such that $\deg^{\mathrm{O}}(x_j) + \deg^{\mathrm{I}}(x_j) \geqslant 2$ **do** ▷ attach x_i to parent x_j
      $\left(\deg^{\mathrm{O}'}, \deg^{\mathrm{I}'}\right) \leftarrow \left(\deg^{\mathrm{O}}, \deg^{\mathrm{I}}\right)$
      $\deg^{\mathrm{O}'}(x_j) \leftarrow \deg^{\mathrm{O}'}(x_j) - 1$
      $\deg^{\mathrm{I}'}(x_i) \leftarrow 0$
      $(T_j, \mathsf{cost}_j) \leftarrow \mathcal{T}[\deg^{\mathrm{O}'}, \deg^{\mathrm{I}'}]$
      $T_j \leftarrow T_j \cup \{x_jx_i\}$
      $\mathsf{cost}_j \leftarrow \mathsf{cost}_j + c(x_jx_i)$
    return (T_j, cost_j) with minimum cost_j

Analysis of Algorithm 1.10. Observe that the number of possible entries $\mathcal{T}[\cdot,\cdot]$, i.e. the number of feasible degree sequences for trees on subsets of V, is at most $\sum_{k=1}^{n} \binom{n}{k} \mathtt{DSr}(k)$. Using our previous estimate $\mathtt{DSr}(n) = O(4^n)$, this sum evaluates, by the binomial theorem, to $O(5^n)$. The overall run time of Algorithm 1.9, including the transportation subproblem, with Algorithm 1.10 as a subroutine is $O^*(5^n)$, since the entry for a given degree sequence is computed at most once over all iterations, and the values are stored through the entire execution. In practice, storing an entire tree in each $\mathcal{T}[\cdot,\cdot]$ is wasteful; for the optimum tree to be constructible from the table, it is sufficient to store the node to which the lowest-index leaf is connected. The claimed time and space complexities follow.

1.1.5 DC-MV: polynomial space and single-exponential time

In this section we show how to reduce the space complexity to polynomial, while maintaining a single-exponential run time.

The outer loop (Algorithm 1.9) remains the same, but we replace the subroutine for finding an optimal directed spanning tree (Algorithm 1.10) with an approach based on divide and conquer (Algorithm 1.11). The approach is inspired by the algorithm of Gurevich and Shelah [51] for finding an optimal TSP tour.

The algorithm relies on the following observation about tree-separators. Let (V_1, V_2) be a partition of V, i.e. $V_1 \cup V_2 = V$, and $V_1 \cap V_2 = \emptyset$, and let $|V| = n$. We say that

(V_1, V_2) is *balanced* if $\max\{|V_1|, |V_2|\} \leqslant \lceil 2n/3 \rceil$. The following result is folklore (see e.g. Nishizeki and Chiba [91, § 9.1]), we give a short proof for completeness.

Lemma 1.6. *Every tree T with vertex set V admits a balanced partition (V_1, V_2) of V such that all edges of T between V_1 and V_2 are incident to a vertex $v \in V_1$.*

Proof. A very old result of Jordan [66] states that every tree has a *centroid*, which is a vertex whose removal splits the tree into subtrees not larger than half of the original tree. To find such a centroid, move from an arbitrary vertex, one edge at a time, towards the largest subtree.

Let $T_1, \ldots, T_m$ be the vertex sets of the trees, in decreasing order of size, resulting from deleting the centroid of T. Let $e_1, \ldots, e_m$ denote the edges that connect the respective trees to the centroid in T. If $|T_1| \geqslant \lfloor n/3 \rfloor$, then we have the balanced partition $(V \setminus T_1, T_1)$, with a single crossing edge.

Otherwise, let m' be the smallest index such that $\lfloor n/3 \rfloor \leqslant |T_1| + \cdots + |T_{m'}| \leqslant \lceil 2n/3 \rceil$. Observe that as $|T_i| < \lfloor n/3 \rfloor$ for all i, such an index m' must exist. Now the balanced partition is $(V \setminus (T_1 \cup \cdots \cup T_{m'}), T_1 \cup \cdots \cup T_{m'})$. The partition fulfils the stated condition as all crossing edges are incident to the centroid, which is on the left side. □

Algorithm 1.11 DC-MV for generating the optimal directed tree for a given degree sequence.

Input: Vertex set V, degree sequence $\left(\deg^{\mathrm{O}}, \deg^{\mathrm{I}}\right)$, edge costs c, and $n = |V|$.

Output: (T, cost), where T is optimum tree for $\deg^{\mathrm{O}}, \deg^{\mathrm{I}}$ and cost denotes its cost.

procedure $\mathcal{T}\left[V, \deg^{\mathrm{O}}, \deg^{\mathrm{I}}\right]$
  if $|V| \leqslant 5$ **then**
    Directly find optimum tree T with cost cost.
    return (T, cost)
  else
    for each partition (V_1, V_2) of V, such that $|V_1|, |V_2| \leqslant \lceil 2n/3 \rceil$ **do**
      for each $v \in V_1$ **do**
        $\left(\mathsf{excess}_v^{\mathrm{out}}, \mathsf{excess}_v^{\mathrm{in}}\right) \leftarrow \left(\sum_{u \in V_1} \deg^{\mathrm{O}}(u) - |V_1| + 1,\ \sum_{u \in V_1} \deg^{\mathrm{I}}(u) - |V_1| + 1\right)$
        $\left(\deg^{\mathrm{O}'}, \deg^{\mathrm{I}'}\right) \leftarrow \left(\deg^{\mathrm{O}}, \deg^{\mathrm{I}}\right)$
        $\deg^{\mathrm{O}'}(v) \leftarrow \deg^{\mathrm{O}'}(v) - \mathsf{excess}_v^{\mathrm{out}}, \quad \deg^{\mathrm{I}'}(v) \leftarrow \deg^{\mathrm{I}'}(v) - \mathsf{excess}_v^{\mathrm{in}}$
        $V_2' \leftarrow V_2 \cup \{w\}$
        $c(wu) \leftarrow c(vu),\ \ c(uw) \leftarrow c(uv),\ \ \text{for all } u \in V_2$
        $\left(\deg^{\mathrm{O}'}(w), \deg^{\mathrm{I}'}(w)\right) \leftarrow \left(\mathsf{excess}_v^{\mathrm{out}}, \mathsf{excess}_v^{\mathrm{in}}\right)$
        if $\deg^{\mathrm{O}'}(\cdot), \deg^{\mathrm{I}'}(\cdot) \geqslant 0, \deg^{\mathrm{O}'}(v) + \deg^{\mathrm{I}'}(v) \geqslant 1, \deg^{\mathrm{O}'}(w) + \deg^{\mathrm{I}'}(w) \geqslant 1$ **then**
          $\mathsf{cost}_1 \leftarrow \mathcal{T}\left[V_1,\ \deg^{\mathrm{O}'}|_{V_1},\ \deg^{\mathrm{I}'}|_{V_1}\right]$
          $\mathsf{cost}_2 \leftarrow \mathcal{T}\left[V_2',\ \deg^{\mathrm{O}'}|_{V_2'},\ \deg^{\mathrm{I}'}|_{V_2'}\right]$
          $\mathsf{cost} \leftarrow \mathsf{cost}_1 + \mathsf{cost}_2$
          $T \leftarrow$ merge T_1 and T_2 by identifying v and w
    return (T, cost) with minimum cost

At a high-level, DC-MV works as follows. It *guesses* the partition (V_1, V_2) of the vertex set V according to a balanced separator of the (unknown) optimal rooted tree T, satisfying the conditions of Lemma 1.6. It also *guesses* the distinguished vertex $v \in V_1$ to which all edges that cross the partition are incident. (Throughout the description of the algorithm, guessing should be understood as iterating through all possible choices.)

The balanced separator splits T into a tree with vertex set V_1 and a forest with vertex set V_2. There are two cases to consider, depending on whether the root $r = x_1$ of T falls in V_1 or V_2. In the first case, V_1 induces a directed subtree of T rooted at r, in the second case, V_1 induces a directed subtree of T rooted at v.

Observe that the outdegrees and indegrees of vertices in V_1 are feasible for a directed tree, except for vertex v which has additional degree due to the edges crossing the partition. We refer to this outdegree and indegree as the *excess* of v. The excess of v can be computed using Lemma 1.5. This excess determines the number and orientation of edges across the cut, even if the endpoints, other than v, of the edges are unknown.

By the same argument as in Lemma 1.3, the subtree of T induced by V_1 is optimal for its corresponding degree sequence after subtracting the excess of v, therefore we can find it by a recursive call to the procedure.

On the other side of the partition we have a *collection* of trees. To obtain an instance of our original problem, we add a virtual vertex w to V_2 to play the role of v. For all vertices u we set the distance between u and the virtual vertex w to be the same as the distance between u and v. We set the outdegree and indegree of w to the *excess* of v, to allow it to connect to all vertices of V_2 that v connects to in T. Now we can find the optimal tree on this side too, by a recursive call to the procedure.

On both sides of the partition, the roots of the directed trees are uniquely determined by the remaining degrees. Observe that if the original root coincides with the centroid v, then v and w will be the roots of the trees in the recursive calls. After obtaining the optimal trees on the two sides (assuming the guesses were correct), we reconstruct T by gluing the two trees together, identifying the vertices v and w. As this operation adds all degrees, we obtain a valid tree for the original degree sequence, furthermore, the tree must be optimal. We illustrate the two cases of this process in Figure 1.5 with pseudocode given in Algorithm 1.11.

The threshold value 5 in Line 2 of Algorithm 1.11 is to ensure progress; the subproblem size $\lceil 2n/3 \rceil + 1$ (the second term accounts for the virtual vertex w) needs to be less than n, which holds for $n > 5$. For $n \leqslant 5$, we proceed by a brute-force approach, constructing all feasible trees and choosing one with minimal cost. In Line 8, the excess outdegree and indegree of v are calculated. Both types of degrees need to sum to $|V_1| - 1$, from which the expression follows. The condition in Line 14 ensures that we only solve feasible problems. The notation $\deg|_X$ indicates a restriction of a degree sequence to a subset X of the vertices.

Analysis of Algorithm 1.11. For a set V of size n we consider at most 2^n partitions (Line 6) and at most $\lceil 2n/3 \rceil$ choices of v (Line 7), and we recur on subsets of size at most $\lceil 2n/3 \rceil + 1$. All remaining operations take $O(n)$ time. The run time $t(n)$ of DC-MV thus

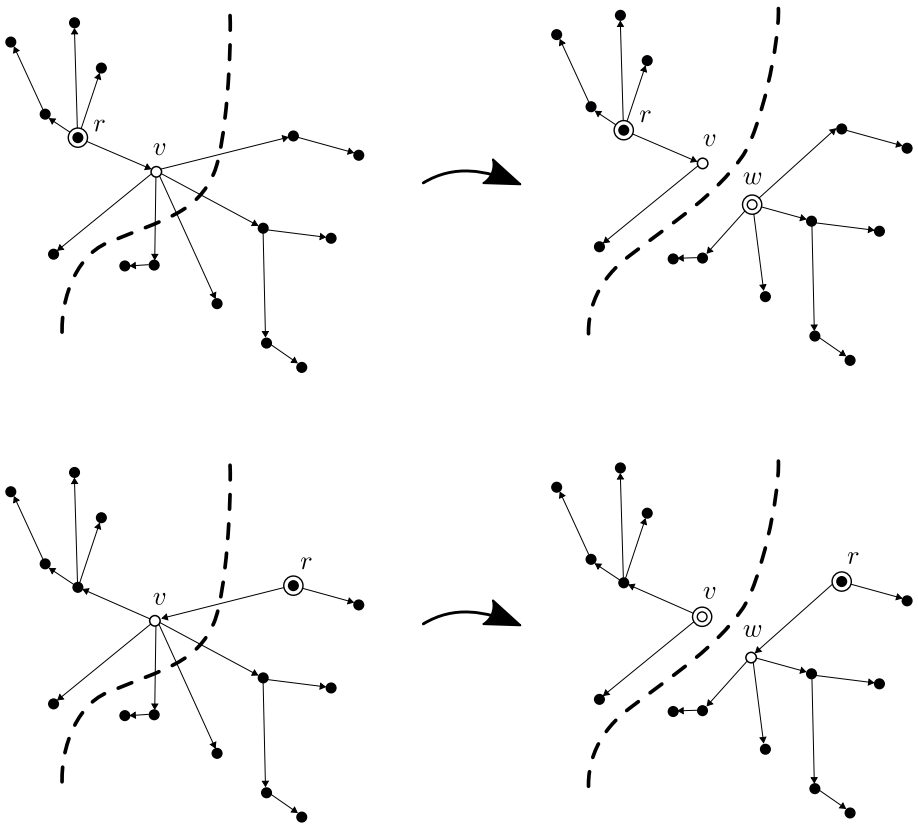


FIGURE 1.5: Illustration of DC-MV. (left) optimal tree and balanced centroid-partitioning, the centroid shown as a hollow circle; (right) optimal trees on the two sides of the partitioning, centroid v and its virtual pair w shown as hollow circles; (above) root is in the left side of the partition, shown as double circle; (below) root is in the right side of the partition, shown as double circle.

satisfies:

$$\begin{aligned} t(n) &\leqslant 2^n \cdot n \cdot 2t(\lceil 2n/3 \rceil + 1) \\ &= n^{O(\log n)} \cdot 2^{n\left(\sum_k (2/3)^k\right)+O(\log n)} \\ &= n^{O(\log n)} \cdot O(2^{3n}) \ . \end{aligned}$$

Including the transportation problem, the overall run time of Algorithm 1.9 using the procedure DC-MV is thus $\texttt{DSr}(n) \cdot (8^n n^{O(\log n)} + n^3 \log n) + O(n^3 \log r) = O^*(32^{n+o(n)} + \log r)$. For the space complexity, observe that (as opposed to DP-MV) the algorithm does not precompute and store $\mathcal{T}[\cdot,\cdot,\cdot]$ and, apart from minor bookkeeping, only a single tour is stored at each time, spread over $O(\log n)$ recursive levels. The claimed bounds follow.

1.1.6 DC-MV2: polynomial space and improved single-exponential time

Finally, we modify the divide and conquer algorithm, improving its run time for a given degree sequence, from $8^{n+o(n)}$ to $4^{n+o(n)}$, while maintaining polynomial space.

The result is based on a strengthening of the structural observation about tree-separators (Lemma 1.6). Specifically, we observe that the vertex set of a tree can be par-

titioned in a "perfectly balanced" way, if we allow the edges across the partition to be incident to a *logarithmic* number of vertices on one side (instead of a single vertex as in Lemma 1.6).

Again, let (V_1, V_2) be a partition of $V(T)$, and let $|V(T)| = n$. We say that (V_1, V_2) is *perfectly balanced* if $\max\{|V_1|, |V_2|\} \leqslant \lceil n/2 \rceil$.

Lemma 1.7. *Every tree T admits a perfectly balanced partition (V_1, V_2) such that all edges of T between V_1 and V_2 are incident to a vertex in $\{v_1, \ldots, v_k\} \subseteq V_1$, where $k \leqslant \lfloor \log_2 n \rfloor$.*

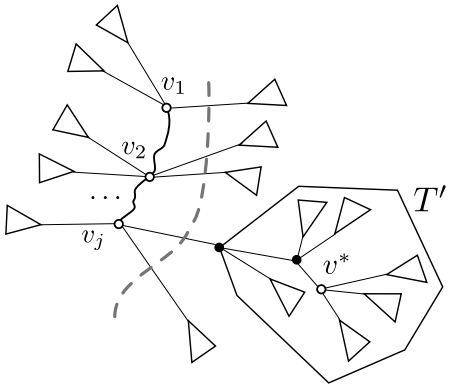


FIGURE 1.6: Illustration of the proof of Lemma 1.7.

Proof. We proceed by finding the special vertices v_j one-by-one for $j = 1, \ldots, \lfloor \log_2 n \rfloor$ and updating the partition (V_1, V_2) in each iteration, maintaining the property that all edges between V_1 and V_2 are incident to a vertex in $\{v_1, \ldots, v_j\}$. We additionally require the following conditions in every iteration: (1) $v_1, \ldots, v_j$ are, in this order, on a simple path entirely within V_1, and (2) $|V_1| \leqslant |V_2|$. Furthermore, if $|V_1| < |V_2|$, then we require that there is a subtree T' of T attached to v_j such that (3) T' is entirely contained in V_2, (4) $|T'| \leqslant n/2^j$, and (5) $|V_1| + |T'| > n/2$. In words, moving T' to the left side of the partition would make the left side larger than the right side. The claim of the Lemma follows from conditions (1)–(5), when $j = \lfloor \log_2 n \rfloor$.

Let us prove that the conditions hold by induction on j. For the base case, $j = 1$, we reuse our construction from Lemma 1.6. Let v_1 be the centroid of T, and iteratively move the subtrees of T attached to v_1 to the initially empty V_2 from the initially full V_1, until $|V_2| \geqslant |V_1|$. Denote the last subtree that is moved as T'. Conditions (1)–(5) are easily verified.

For the inductive step, assume that $v_1, \ldots, v_j$ have been found, satisfying conditions (1)–(5), and let T' be the subtree attached to v_j that fulfils conditions (3)–(5). Let v^* be the centroid of T', and let $q_1, \ldots, q_m$ be the internal vertices (in this order) on the path from v_j to v^*. Let $q_{m+1} = v^*$. Iteratively, for $\ell = 1, \ldots, m+1$, move the nodes q_ℓ from V_2 to V_1, and after moving each q_ℓ, move one-by-one the subtrees of T' attached to q_ℓ, other than the one containing the centroid v^*. By the definition of the centroid, and by (4), all subtrees that we move have size at most $n/2^{j+1}$. We stop when $|V_1| \geqslant |V_2|$. Observe that this must happen eventually, due to condition (5).

If the last move is a vertex q_ℓ, then $|V_2| \leqslant |V_1| \leqslant |V_2| + 1$, so a perfectly balanced partition has been found; we label q_ℓ as v_{j+1} and stop the entire process. In this case, verifying conditions (1)–(5) is no longer needed. If the last move is a subtree, then we move it back to V_2, label it T', and label the last moved vertex q_ℓ as v_{j+1}. By construction, all edges across the partition are incident to one of the vertices $v_1, \ldots, v_{j+1}$. This is because intermediate vertices $q_1, \ldots, q_{\ell-1}$ have been moved across the partition together with all their pendent trees, they cannot thus be incident to any edges across the partition. (Recall that $v_1, \ldots, v_{j+1}$ form a simple path, but are not necessarily consecutive vertices on this path.) By construction, conditions (1)–(5) are satisfied with v_{j+1}, the new T', and the current V_1, V_2. See Figure 1.6 for illustration. □

Our improved algorithm DC-MV2 follows the same outline as DC-MV, described in §1.1.5. We highlight the main differences.

The algorithm *guesses* the partition (V_1, V_2) of the vertex set V, according to a perfectly balanced separator of the (unknown) optimal rooted tree T, satisfying the conditions of Lemma 1.7. It also guesses the distinguished vertices $\{v_1, \ldots, v_k\} \subseteq V_1$ to which all edges that cross the partition are incident. Again, the separator splits T into a tree with vertex set V_1 and a forest with vertex set V_2. Again, we consider separately the cases when the root $r = x_1$ of T falls in V_1 or V_2. In the first case, V_1 induces a directed subtree of T rooted at r, in the second case, V_1 induces a directed subtree of T rooted at v_i, for some $1 \leqslant i \leqslant k$. (See Figure 1.7.)

Whereas in DC-MV all excess degrees belong to a single vertex, they are now distributed among $v_1, \ldots, v_k$. Therefore, we compute the *total excess* of vertices $v_1, \ldots, v_k$ and we *guess* their distribution. Distributing the excess *indegree* is easy. Assuming that our guesses were correct, at most one of the vertices $v_1, \ldots, v_k$ has an incoming edge across the partition. We just need to guess therefore, which vertex v_i (if any) has excess indegree 1, while all others have excess indegree 0 (see Figure 1.7). For the excess *outdegrees*, we need to distribute a total value of at most n among $k = O(\log n)$ vertices, amounting to $n^{O(\log n)}$ possible choices.

We update the degrees of $v_1, \ldots, v_k$ in V_1, removing their excesses. Assuming our guesses to be correct, the subtree of T induced by V_1 is optimal for its degree sequence, by Lemma 1.2.

On the side of V_2 we have a collection of trees. To obtain an instance of our problem (i.e. a single tree), we add virtual vertices $w_1, \ldots, w_k$ to V_2, to play the role of $v_1, \ldots, v_k$, and a virtual root w_0 forcibly connected to the virtual vertices.

To achieve the correct connections, we first set the outdegrees and indegrees of $w_1, \ldots, w_k$ to match the excesses of $v_1, \ldots, v_k$. This allows them to connect to all vertices of V_2 that $v_1, \ldots, v_k$ connect to in T. Observe that the location of the root of T is determined by the indegrees of the virtual vertices. If all w_i have indegree 0, then the root is on the left side of the partition (i.e. in V_1). If w_ℓ, for some $\ell \in \{1, \ldots, k\}$ has indegree 1, then the root is on the right side of the partition (i.e. in V_2), in a subtree attached to v_ℓ.

Algorithm 1.12 DC-MV2 for generating the optimal directed tree for a given degree sequence.

Input: Vertex set V, degree sequence $\left(\deg^{\mathrm{O}}, \deg^{\mathrm{I}}\right)$, edge costs c, and $n = |V|$.

Output: (T, cost), where T is optimum tree for $\deg^{\mathrm{O}}, \deg^{\mathrm{I}}$ and cost denotes its cost.

1: **procedure** $\mathcal{T}\left[V, \deg^{\mathrm{O}}, \deg^{\mathrm{I}}\right]$
2: **if** $|V| \leqslant 9$ **then**
3: Directly find optimum tree T with cost cost.
4: **return** (T, cost)
5: **else**
6: **for** each partition (V_1, V_2) of V, such that $|V_1|, |V_2| \leqslant \lceil n/2 \rceil$ **do**
7: **for** each $\{v_1, \ldots, v_k\} \subseteq V_1$, where $k \leqslant \lfloor \log_2 n \rfloor$ **do**
8: $\left(\mathsf{excess}^{\mathsf{out}}, \mathsf{excess}^{\mathsf{in}}\right) \leftarrow \left(\sum_{u \in V_1} \deg^{\mathrm{O}}(u) - |V_1| + 1,\ \sum_{u \in V_1} \deg^{\mathrm{I}}(u) - |V_1| + 1\right)$
9: **for** each *excess-partition* $\left\{\mathsf{excess}^{\mathsf{out}}_{v_i}\right\}_{1 \leqslant i \leqslant k}, \left\{\mathsf{excess}^{\mathsf{in}}_{v_i}\right\}_{1 \leqslant i \leqslant k}$ **do**
10: $\left(\deg^{\mathrm{O}'}, \deg^{\mathrm{I}'}\right) \leftarrow \left(\deg^{\mathrm{O}}, \deg^{\mathrm{I}}\right)$
11: Update degrees and edge costs appropriately. ▷ Algorithm 1.13
12: **if** $\deg^{\mathrm{O}'}, \deg^{\mathrm{I}'}$ are *valid* for directed trees **then**
13: $\mathsf{cost}_1 \leftarrow \mathcal{T}\left[V_1,\ \deg^{\mathrm{O}'}|_{V_1},\ \deg^{\mathrm{I}'}|_{V_1}\right]$
14: $\mathsf{cost}_2 \leftarrow \mathcal{T}\left[V_2',\ \deg^{\mathrm{O}'}|_{V_2'},\ \deg^{\mathrm{I}'}|_{V_2'}\right]$
15: $\mathsf{cost} \leftarrow \mathsf{cost}_1 + \mathsf{cost}_2$
16: $T \leftarrow$ merge T_1 and T_2 by identifying v_i and w_i and removing w_0
17: **return** (T, cost) with minimum cost

In the first case, we make sure that all edges (w_0, w_i) are directed away from w_0. In the second case, edge (w_ℓ, w_0) is directed towards w_0, and all edges (w_0, w_i) for $i = 1, \ldots, k$, and $i \neq \ell$ are directed away from w_0 (see Figure 1.7). To enforce these connections, we appropriately adjust the indegrees and outdegrees of $w_0, \ldots, w_k$. We also set distances involving virtual vertices $w_0, \ldots, w_k$, such as to forbid unwanted connections. (See Algorithm 1.13.)

After obtaining the optimal trees on the two sides (assuming the guesses were correct), we reconstruct T by gluing the two trees together, removing w_0 with all its incident edges, and identifying v_i with w_i for all $i = 1, \ldots, k$. The optimality of the obtained tree follows by observing that a tree with smaller cost would also induce an improved solution on at least one of the subproblems, contradicting their optimality. Again, we remark that "guessing" means iterating through all possible choices. We refer to the detailed pseudocode described in Algorithm 1.12 and the illustration in Figure 1.7.

Some further remarks about Algorithm 1.12 are in order. The threshold 9 in Line 2 is to ensure progress; we need the recursive subproblem size $\lceil n/2 \rceil + \lfloor \log_2 n \rfloor + 1$ (the additive logarithmic term is due to the virtual vertices) to be less than n, which holds for $n > 9$. In practice, even some values $n \leqslant 9$ may be too large to be solved by brute force; in such cases we can switch back to Algorithm 1.11 (DC-MV) or Algorithm 1.10 (DP-MV), without affecting the asymptotic guarantees.

Algorithm 1.13 Updating degrees and edge costs in DC-MV2: Line 11 of Algorithm 1.12.

1: $\deg^{O'}(v_i) \leftarrow \deg^{O'}(v_i) - \text{excess}^{\text{out}}_{v_i}$, $\deg^{I'}(v_i) \leftarrow \deg^{I'}(v_i) - \text{excess}^{\text{in}}_{v_i}$, for $1 \leqslant i \leqslant k$
2: $V_2' \leftarrow V_2 \cup \{w_0, w_1, \ldots, w_k\}$ ▷ virtual vertices
3: $\deg^{O'}(w_i) \leftarrow \text{excess}^{\text{out}}_{v_i}$, $\deg^{I'}(w_i) \leftarrow \text{excess}^{\text{in}}_{v_i} + 1$, for all $1 \leqslant i \leqslant k$
4: $c(w_0 w_i) \leftarrow 0$, $c(w_i w_0) \leftarrow 0$, for all $1 \leqslant i \leqslant k$
5: $c(w_0 u) \leftarrow \infty$, $c(u\, w_0) \leftarrow \infty$, for all $u \in V_2 \setminus \{w_1, \ldots, w_k\}$
6: $c(w_i w_j) \leftarrow \infty$, for all $1 \leqslant i, j \leqslant k$
7: $c(w_i u) \leftarrow c(v_i u)$, and $c(u\, w_i) \leftarrow c(u\, v_i)$ for all $1 \leqslant i \leqslant k$, and $u \in V_2 \setminus \{w_0, \ldots, w_k\}$
8: $\deg^{O'}(w_0) \leftarrow k$, $\deg^{I'}(w_0) \leftarrow 0$
9: **if** $\text{excess}^{\text{in}}_{v_i} > 0$ for some $i \in \{1, \ldots, k\}$ **then** ▷ root is in V_2, in subtree attached to v_i
10: $\quad \deg^{O'}(w_i) \leftarrow \deg^{O'}(w_i) + 1$
11: $\quad \deg^{I'}(w_i) \leftarrow \deg^{I'}(w_i) - 1$
12: $\quad \deg^{O'}(w_0) \leftarrow \deg^{O'}(w_0) - 1$
13: $\quad \deg^{I'}(w_0) \leftarrow \deg^{I'}(w_0) + 1$

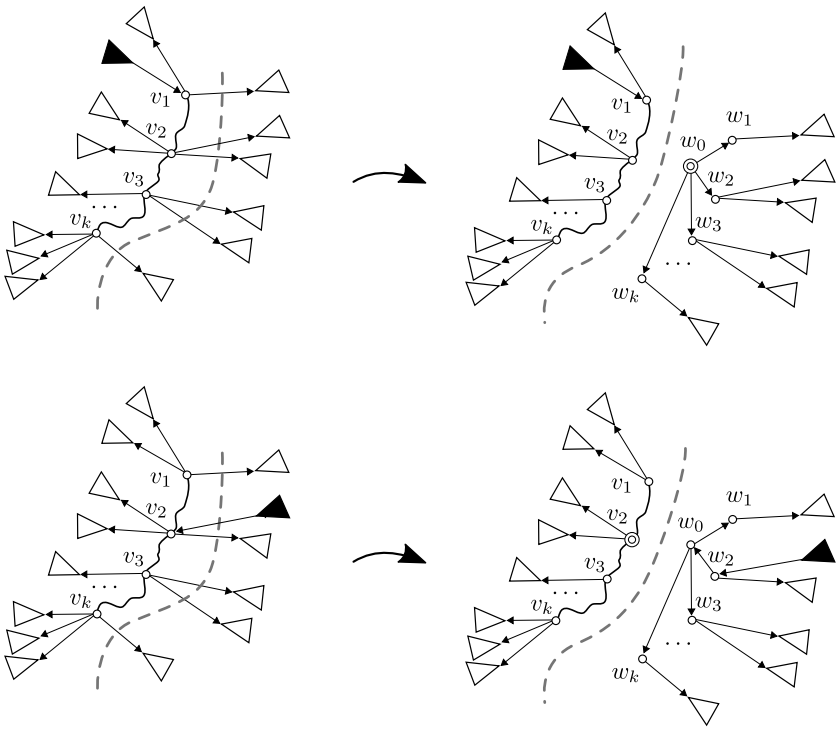


FIGURE 1.7: Illustration of DC-MV2. (left) optimal tree and perfectly balanced partitioning; triangles indicate subtrees, shaded triangle contains root. (right) optimal trees on the two sides of the partitioning, virtual vertices $w_0, w_1, \ldots, w_k$ shown; (above) root falls to the left side of the partition; (below) root falls to the right side of the partition; new roots shown as double circles.

Lines 6, 7 and 9 contain the "guesses", specifically, for the partition of V, for the choice of $v_1, \ldots, v_k$, and for distributing the excess degrees among $v_1, \ldots, v_k$. Line 12 requires a test similar to the corresponding test in Algorithm 1.11, that the guessed values correctly describe a tree on both sides of the partition. Line 11 contains the update of degrees and distances on the two sides of the partition, separated into Algorithm 1.13 for readability.

Analysis of Algorithm 1.12. For a set V of size n we consider at most 2^n partitions (Line 6), at most $n^{O(\log n)}$ choices for the v_i (Line 7), and at most $n^{O(\log n)}$ choices for their excess indegrees and outdegrees (Line 9). We recur on subsets of size at most $\lceil n/2 \rceil + \lfloor \log_2 n \rfloor + 1$.

All remaining operations take $O(n)$ time. The run time $t(n)$, can thus be bounded as:

$$\begin{aligned} t(n) &\leqslant 2^n \cdot n^{O(\log n)} \cdot 2t(\lceil n/2 \rceil + \lfloor \log_2 n \rfloor + 1) \\ &= n^{O(\log^2 n)} \cdot 2^{n\left(\sum_k (1/2)^k\right)+O(\log^2 n)} \\ &= n^{O(\log^2 n)} \cdot O(2^{2n}) \ . \end{aligned}$$

Including the transportation problem, the overall run time of Algorithm 1.9 using DC-MV2 is therefore $\texttt{DSr}(n) \cdot (4^n n^{O(\log n)} + n^3 \log n) + O(n^3 \log r) = O^*(16^{n+o(n)} + \log r)$, with essentially the same space complexity as before.

We remark that in instances of MVTSP where some of the requests are smaller than $n-1$, not all degree sequences of trees are realisable. Pruning out unrealisable degree sequences leads to improvements in efficiency. As a special case, suppose that all requests are equal to 1 (i.e. the standard TSP). Then, the only possible tree is a *path*, with a unique degree sequence, up to the choice of the leaf. In this case the run time of our approach reduces to $4^{n+o(n)}$, matching the Gurevich-Shelah algorithm [51].

1.2 Solving the Many-Visits Path TSP

Although the algorithms in this chapter were presented and analysed with regards to the tour version of the MVTSP, with slight modifications we can obtain identical results to the Many-Visits Path TSP as well. Let us assume there are two special end-vertices s and t given. In the Many-Visits Path TSP, one seeks a minimum cost s-t-walk that visits each vertex v exactly $r(v)$ times. This means a multigraph H with out- and indegree $r(v)$ for all $v \in V - \{s,t\}$, out- and indegree $r(s)$ and $r(s)-1$ for vertex s and out- and indegree $r(t)-1$ and $r(t)$ for vertex t.

Let us make an observation analogous to Lemma 1, for the path version of the MVTSP:

Lemma 1.8. *Let X be a directed multigraph over V with outdegrees $\deg^{\mathrm{O}}_X(\cdot)$ and indegrees $\deg^{\mathrm{I}}_X(\cdot)$. Then X is the edge set of an s-t-walk that visits each vertex $v \in V$ exactly $r(v)$ times if and only if the following conditions hold:*

(i) the underlying graph of X is connected,

(ii) for all $v \in V - \{s,t\}$, we have $\deg^{\mathrm{O}}_X(v) = \deg^{\mathrm{I}}_X(v) = r(v)$,

(iii) for vertex s we have $\deg^{\mathrm{O}}_X(s) = r(s)$ and $\deg^{\mathrm{I}}_X(s) = r(s) - 1$ and

(iv) for vertex t we have $\deg^{\mathrm{O}}_X(t) = r(t) - 1$ and $\deg^{\mathrm{I}}_X(t) = r(t)$.

Proof. By adding an edge ts to X, the claim immediately follows from Lemma 1. □

A tree rooted at $v_0 \in V$ is such that there are no edges ending at v_0 but at least one edge starting at v_0. In a many-visits path, the same is true for the vertex s. Using the same construction that in the proof of Lemma 1.2, we can see that the following is true:

Lemma 1.9. *Let X be the edge set of a many-visits s-t-path on V with arbitrary nonzero requests. Then there is a directed spanning tree T of X rooted at s, and a directed multigraph Z, such that $X = T + Z$.* □

We can thus also split a many-visits path into a directed tree T rooted at s and a multigraph Z.

Lemma 1.10. *Let X be the edge set of an optimal many-visits s-t-path, let T be a directed spanning tree, and let Z be a directed multigraph such that $X = T + Z$. Then, T has the smallest cost among all directed spanning trees with degrees $\deg^{\mathrm{O}}_T(\cdot)$ and $\deg^{\mathrm{I}}_T(\cdot)$, and Z has the smallest cost among all directed multigraphs with degrees $\deg^{\mathrm{O}}_Z(\cdot)$ and $\deg^{\mathrm{I}}_Z(\cdot)$.*

Proof. The proof is analogous to that of Lemma 1.3. □

Due to Lemmas 1.8–1.10, we can use the approaches ENUM-MV, DP-MV, DC-MV and DC-MV2 to solve the Many-Visits Path TSP. The only modifications we need to make in these algorithms concern the degree requirements of special vertices s and t the multigraph Z calculated based on the degrees of the tree T. The degree requirements of $v \in V - \{s, t\}$ are identical to the ones in the tour case in Algorithms 1.1, 1.8 and 1.9, but for s and t we change them the following way:

Algorithm 1.14 Line 2 of Algorithm 1.1, Line 2 of Algorithm 1.8 and Line 3 of Algorithm 1.9 when solving the MANY-VISITS PATH TSP.

Find a minimum-cost directed multigraph Z such that:

$$\begin{aligned}
&\deg^{\mathrm{O}}_Z(v) = r(v), \quad \forall v \in V - \{s, t\},\\
&\deg^{\mathrm{I}}_Z(v) = r(v), \quad \forall v \in V - \{s, t\},\\
&\deg^{\mathrm{O}}_Z(s) = r(s) - \deg^{\mathrm{O}}_T(s),\\
&\deg^{\mathrm{I}}_Z(s) = r(s) - \deg^{\mathrm{I}}_T(s) - 1,\\
&\deg^{\mathrm{O}}_Z(t) = r(t) - \deg^{\mathrm{O}}_T(t) - 1,\\
&\deg^{\mathrm{I}}_Z(t) = r(t) - \deg^{\mathrm{I}}_T(t).
\end{aligned}$$

Similarly to the tour case, we skip calculating the multigraph X if for any vertex $v \in V$ the degree requirement in X would be negative. The time and space complexities of the algorithms also remain unchanged.

Summary

In this chapter we presented a family of exact algorithms that provide an optimal solution to the Many-Visits TSP and Many-Visits Path TSP. All algorithms are deterministic. Algorithm 1.10, under the name DP-MV, provides an optimal solution using $O^*(5^n)$ time and $O^*(5^n)$ space; these complexities asymptotically match the fastest approach to the classical TSP by Bellman [13] and Held and Karp [56], that have $O^*(2^n)$ time

and space complexity. An improved divide & conquer technique, DC-MV2 in Algorithm 1.12 solves the MVTSP using space polynomial in n and $\log r$, in $O^*(16^{n+o(n)})$ time. This approach uses the recursive idea of the best polynomial-space algorithm on the classical TSP by Gurevich and Shelah [51], and asymptotically matches its time complexity of $O^*(4^n)$. Moreover, pruning out unrealisable degree sequences in the implementation of DC-MV2 makes its time complexity $O^*(4^{n+o(n)})$ for single-visit TSP instances, essentially making it as fast as the algorithm of Gurevich and Shelah. Finally, the algorithm DC-MV2 is asymptotically best possible, assuming the Exponential Time Hypothesis.

Our results lead to improvements in applications where MVTSP is solved as a subroutine. For instance, as a corollary of Theorem 1.1*(iii)*, the approximation scheme for Maximum Scatter TSP [79] can now be implemented in space *polynomial* in the error parameter ε.

CHAPTER 2

DEGREE-BOUNDED PROBLEMS AND APPROXIMATIONS

Introduction

This chapter is about combinatorial optimisation problems with degree constraints. An illustrious example is the MINIMUM BOUNDED DEGREE SPANNING TREE PROBLEM, where the goal is to find a minimum cost spanning tree in a graph with lower and upper bounds on the degree at each vertex. Checking the existence of a degree-bounded spanning tree contains the NP-hard Hamiltonian path problem; therefore, efficiently finding spanning trees that only slightly violate the degree constraints, is of interest. Several algorithms were given that were balancing between the cost of the spanning tree and the violation of the degree bounds [25, 26, 45, 76, 77]. Goemans [48] gave a polynomial-time algorithm that finds a spanning tree of cost at most the optimum value that violates each degree bound by at most 2. Singh and Lau [104] improved the additive approximation guarantee to 1 by extending the iterative rounding method of Jain [63] with a relaxation step. Zenklusen [120] considered an extension of the problem where for every vertex v, the edges adjacent to v have to be independent in a given matroid.

Motivated by a problem on binary matroids posed by Frieze, a matroidal generalisation called the MINIMUM BOUNDED DEGREE MATROID BASIS PROBLEM was introduced by Király, Lau and Singh [73] in 2012. The problem takes as input a matroid $M = (S, r)$, a cost function $c : S \to \mathbb{R}$, a hypergraph $H = (S, \mathcal{E})$ and lower and upper bounds $f, g : \mathcal{E} \to \mathbb{Z}_{\geqslant 0}$; the objective is to find a minimum-cost basis B of M such that $f(\varepsilon) \leqslant |B \cap \varepsilon| \leqslant g(\varepsilon)$ for each $\varepsilon \in \mathcal{E}$. For this problem, the authors developed an approximation algorithm that is based on the iterative relaxation method and a clever token-counting argument of Chaudhuri et al. [25] and Singh and Lau [104]. Let Δ denote the maximum degree of the hypergraph H. When both lower bounds and upper bounds are present, their algorithm returns a basis B of cost at most the optimum value such that $f(\varepsilon) - 2\Delta + 1 \leqslant |B \cap \varepsilon| \leqslant g(\varepsilon) + 2\Delta - 1$ holds for each $\varepsilon \in \mathcal{E}$. Based on a technique of Bansal et al. [11], they showed that the additive error can be improved when only lower bounds (or only upper bounds) are present, thus finding a basis B of cost at most the optimum value such that $f(\varepsilon) - \Delta + 1 \leqslant |B \cap \varepsilon|$ (respectively, $|B \cap \varepsilon| \leqslant g(\varepsilon) + \Delta - 1$) for each $\varepsilon \in \mathcal{E}$. Bansal et al. [10] considered extensions of the MINIMUM BOUNDED DEGREE MATROID BASIS PROBLEM to contra-polymatroid intersection and to crossing lattice

polyhedra. In all of these cases, the solution for the problem is a $0-1$ vector defined on the ground set.

Results

In this chapter we consider another combinatorial optimisation problem involving degree bounds. Given is a graph $G = (V, E)$ with edge costs $c(uv)$ for each $u, v \in V$, a positive integer degree requirement $\rho(v)$ for each vertex $v \in V$, nonnegative lower and upper bounds L and U on the edges, and a positive integer k. In the MINIMUM BOUNDED DEGREE k-COMPONENT MULTIGRAPH WITH EDGE BOUNDS problem we seek a minimum cost multigraph Z consisting of at most k components, such that the degree of each vertex $v \in V$ in Z is exactly $\rho(v)$ and the number of copies of each edge vw is between $L(vw)$ and $U(vw)$. This problem contains the MVTSP, therefore it is NP-hard. As one of the main results of this chapter, we show a polynomial-time algorithm that provides a solution to this problem with an additive error:

Theorem 2.1. *There is an algorithm for the* MINIMUM BOUNDED DEGREE k-COMPONENT MULTIGRAPH WITH EDGE BOUNDS *problem that, in time polynomial in n and $\log \sum_v \rho(v)$, returns a multigraph Z consisting of at most k components and $\rho(V)/2$ edges, where each vertex v has degree at least $\rho(v) - 1$ and the cost of Z is at most the cost of $\min\{c^\mathsf{T} x \mid x \in \mathcal{P}_{\mathrm{CG}}(\rho, L, U, k)\}$, which is defined below:*

$$\mathcal{P}_{\mathrm{CG}}(\rho, L, U, k) := \left\{ x \in \mathbb{R}^E_{\geqslant 0} \;\middle|\; \begin{array}{ll} \mathrm{supp}(x) \text{ has at most } k \text{ components} & \\ x(E) = \sum_v \rho(v)/2 & \\ x(\dot{\delta}(v)) \geqslant \rho(v) & \forall v \in V \\ L(vw) \leqslant x(vw) \leqslant U(vw) & \forall v, w \in V \end{array} \right\} . \tag{2.1}$$

Observe that an optimal solution x^* to the MINIMUM BOUNDED DEGREE k-COMPONENT MULTIGRAPH WITH EDGE BOUNDS problem is a minimum cost integral vector from the polytope $\mathcal{P}_{\mathrm{CG}}$. We will use the result of Theorem 2.1 to obtain multigraphs that will serve as building blocks in our approximation algorithms in Chapter 3 and Chapter 4. Note that by choosing $k = 1$ we are seeking a connected multigraph.

In Equation (2.1), it is not immediately clear that the constraint "$\mathrm{supp}(x)$ has at most k components" is a valid polyhedral constraint. We show this in Lemmas 2.9 and 2.10 in §2.3.

The presented MINIMUM BOUNDED DEGREE k-COMPONENT MULTIGRAPH WITH EDGE BOUNDS problem shows a lot of similarities to the MINIMUM BOUNDED DEGREE SPANNING TREE problem. However, neither the result of Singh and Lau [104] nor the more general approach by Király et al. [73] applies directly to our setting, due to the presence of parallel edges and self-loops in a multigraph.

A possible way to overcome these difficulties is as follows: first solve the corresponding linear program, fix the integral parts of the coordinates in the solution, and then work on a reduced problem where the variables are restricted to be between 0

and 1. After fixing the integral parts, the remaining point can be shown to be in a matroid, thus bringing the problem back to the matroid setting. Therefore, one would have to extend the algorithm of Király et al. [73] to handle multiplicities.

Instead of this approach, we develop an extension of the result of Király et al. [73] to generalised polymatroids and element multiplicities, which might be of independent combinatorial interest. Formally, the BOUNDED DEGREE G-POLYMATROID ELEMENT WITH MULTIPLICITIES problem takes as input a g-polymatroid $Q(p, b)$ defined by a paramodular pair $p, b : 2^S \to \mathbb{R}$, a cost function $c : S \to \mathbb{R}$, a hypergraph $H = (S, \mathcal{E})$ with lower and upper bounds $f, g : \mathcal{E} \to \mathbb{Z}_{\geqslant 0}$ and multiplicity vectors $m_\varepsilon : S \to \mathbb{Z}_{\geqslant 0}$ for $\varepsilon \in \mathcal{E}$ satisfying $m_\varepsilon(s) = 0$ for $s \in S - \varepsilon$. The objective is to find a minimum-cost integral element x of $Q(p, b)$ such that $f(\varepsilon) \leqslant \sum_{s\in\varepsilon} m_\varepsilon(s)x(s) \leqslant g(\varepsilon)$ for each $\varepsilon \in \mathcal{E}$. We give a polynomial-time algorithm for finding a solution with cost at most the optimum value with bounds on the violations of the degree prescriptions.

Theorem 2.2. *There is an algorithm for the* BOUNDED DEGREE G-POLYMATROID ELEMENT WITH MULTIPLICITIES *problem which returns an integral element x of $Q(p, b)$ with cost at most the optimum value such that $f(\varepsilon) - 2\Delta + 1 \leqslant \sum_{s\in\varepsilon} m_\varepsilon(s)x(s) \leqslant g(\varepsilon) + 2\Delta - 1$ for each $\varepsilon \in \mathcal{E}$, where $\Delta = \max_{s\in S} \left\{\sum_{\varepsilon\in\mathcal{E}} m_\varepsilon(s)\right\}$. The run time of the algorithm is polynomial in n and $\log \sum_{\varepsilon\in\mathcal{E}} (f(\varepsilon) + g(\varepsilon))$.*

When only lower bounds (or only upper bounds) are present, we call the problem LOWER (UPPER) BOUNDED DEGREE G-POLYMATROID ELEMENT WITH MULTIPLICITIES. Similarly to Király et al. [73] for the matroid setting, we obtain an improved bound on the degree violations when only lower or only upper bounds are present:

Theorem 2.3. *There is an algorithm for* LOWER BOUNDED DEGREE G-POLYMATROID ELEMENT WITH MULTIPLICITIES *which returns an integral element x of $Q(p, b)$ with cost at most the optimum value such that $f(\varepsilon) - \Delta + 1 \leqslant \sum_{s\in\varepsilon} m_\varepsilon(s)x(s)$ for each $\varepsilon \in \mathcal{E}$. An analogous result holds for* UPPER BOUNDED DEGREE G-POLYMATROID ELEMENT WITH MULTIPLICITIES, *where $\sum_{s\in\varepsilon} m_\varepsilon(s)x(s) \leqslant g(\varepsilon) + \Delta - 1$. The run time of these algorithms is polynomial in n and $\log \sum_{\varepsilon\in\mathcal{E}} f(\varepsilon)$ or $\log \sum_{\varepsilon\in\mathcal{E}} g(\varepsilon)$, respectively.*

The results of this chapter are joint work with Kristóf Bérczi, André Berger and Matthias Mnich, and they first appeared on arXiv [14].

2.1 Polyhedral background

In what follows, we make use of some basic notions and theorems of the theory of generalised polymatroids. For background, see for example the paper of Frank and Tardos [41] or Chapter 14 in the book by Frank [40].

Given a ground set S, a set function $b : 2^S \to \mathbb{Z}$ is *submodular* if

$$b(X) + b(Y) \geqslant b(X \cap Y) + b(X \cup Y)$$

holds for every pair of subsets $X, Y \subseteq S$. A set function $p : 2^S \to \mathbb{Z}$ is *supermodular* if $-p$ is submodular. As a generalisation of matroid rank functions, Edmonds introduced

the notion of polymatroids [31]. A set function b is a *polymatroid function* if $b(\emptyset) = 0$, b is non-decreasing, and b is submodular.

We define

$$P(b) := \left\{x \in \mathbb{R}^S_{\geqslant 0} \;\middle|\; x(Y) \leqslant b(Y) \text{ for every } Y \subseteq S\right\} \quad .$$

The set of integral elements of $P(b)$ is called a *polymatroidal set*. Similarly, the *base polymatroid* $B(b)$ is defined by

$$B(b) := \left\{x \in \mathbb{R}^S \;\middle|\; x(Y) \leqslant b(Y) \text{ for every } Y \subseteq S,\, x(S) = b(S)\right\} \quad .$$

Note that a base polymatroid is just a facet of the polymatroid $P(b)$. In both cases, b is called the *border function* of the polyhedron. Although non-negativity of x is not assumed in the definition of $B(b)$, this follows by the monotonicity of b and the definition of $B(b)$: $x(s) = x(S) - x(S-s) \geqslant b(S) - b(S-s) \geqslant 0$ holds for every $s \in S$. The set of integral elements of $B(b)$ is called a *base polymatroidal set*. Edmonds [31] showed that the vertices of a polymatroid or a base polymatroid are integral, thus $P(b)$ is the convex hull of the corresponding polymatroidal set, while $B(b)$ is the convex hull of the corresponding base polymatroidal set. For this reason, we will call the sets of integral elements of $P(b)$ and $B(b)$ simply a polymatroid and a base polymatroid.

Hassin [54] introduced polyhedra bounded simultaneously by a nonnegative, monotone non-decreasing submodular function b over a ground set S from above and by a nonnegative, monotone non-decreasing supermodular function p over S from below, satisfying the so-called *cross-inequality* linking the two functions:

$$b(X) - p(Y) \geqslant b(X - Y) - p(Y - X) \qquad \text{for every pair of subsets } X, Y \subseteq S \quad .$$

We say that a pair (p, b) of set functions over the same ground set S is a *paramodular pair* if $p(\emptyset) = b(\emptyset) = 0$, p is supermodular, b is submodular, and they satisfy the cross-inequality. The slightly more general concept of generalised polymatroids was introduced by Frank [39]. A *generalised polymatroid*, or *g-polymatroid* is a polyhedron of the form

$$Q(p, b) := \left\{x \in \mathbb{R}^S \;\middle|\; p(Y) \leqslant x(Y) \leqslant b(Y) \text{ for every } Y \subseteq S\right\} \quad ,$$

where (p, b) is a paramodular pair. Here, (p, b) is called the *border pair* of the polyhedron. It is known (see e.g. Frank [40]) that a g-polymatroid defined by an integral paramodular pair is a non-empty integral polyhedron.

A base polymatroid is a special g-polymatroid. Indeed, given a polymatroid function b with finite $b(S)$, its *complementary set function* p is defined for $X \subseteq S$ by $p(X) := b(S) - b(S - X)$. It is not difficult to check that (p, b) is a paramodular pair and that $B(b) = Q(p, b)$. Another illustrious example is a box $\beta(L, U) = \{x \in \mathbb{R}^S \mid L \leqslant x \leqslant U\}$ where $L : S \to \mathbb{Z} \cup \{-\infty\}$, $U : S \to \mathbb{Z} \cup \{\infty\}$ with $L \leqslant U$.

Theorem 2.4 (Frank [40, Theorem 14.3.9])**.** *The intersection Q' of a g-polymatroid $Q(p, b)$ and a box $\beta(L, U)$ is a g-polymatroid. If $L(Y) \leqslant b(Y)$ and $p(Y) \leqslant U(Y)$ hold for every $Y \subseteq S$,*

then Q' is non-empty, and its unique border pair (p', b') is given by

$$\begin{aligned} p'(Z) &= \max\{p(Z') - U(Z' - Z) + L(Z - Z') \mid Z' \subseteq S\} \ , \\ b'(Z) &= \min\{b(Z') - L(Z' - Z) + U(Z - Z') \mid Z' \subseteq S\} \ . \end{aligned} \tag{2.2}$$

Let us continue with two basic operations on g-polymatroids. Given a g-polymatroid $Q(p, b)$ and Z, by *deleting* $Z \subseteq S$ from $Q(p, b)$ we obtain a g-polymatroid $Q(p, b)\backslash Z$ defined on set $S - Z$ by the restrictions of p and b to $S - Z$, that is,

$$Q(p, b)\backslash Z := \left\{x \in \mathbb{R}^{S-Z} \;\middle|\; p(Y) \leqslant x(Y) \leqslant b(Y) \text{ for every } Y \subseteq S - Z\right\} \ .$$

In other words, $Q(p, b)\backslash Z$ is the projection of $Q(p, b)$ to the coordinates in $S - Z$.

Extending the notion of contraction from matroids to g-polymatroids is not immediate. A set can be naturally identified with its characteristic vector, that is, in the case of matroids contraction is basically an operation defined on $0-1$ vectors. In our proof, we will need a generalisation of this to the integral elements of a g-polymatroid. However, such an element might have coordinates larger than one as well, hence finding the right definition is not straightforward. In the case of matroids, the most important property of contraction is the following: I is an independent set of M/Z if and only if $F \cup I$ is independent in M for any maximal independent set F of Z.

With this property in mind, we define the g-polymatroid obtained by the contraction of an integral vector $z \in Q(p, b)$ to be the polymatroid $Q(p', b') := Q(p, b)/z$ on the same ground set S with border functions

$$\begin{aligned} p'(X) &:= p(X) - z(X) \\ b'(X) &:= b(X) - z(X) \ . \end{aligned}$$

Observe that p' is obtained as the difference of a supermodular and a modular function, implying that it is supermodular. From similar arguments, the submodularity of b' follows. Moreover, $p'(\emptyset) = b'(\emptyset) = 0$, and

$$\begin{aligned} b'(X) - p'(Y) &= b(X) - z(X) - p(Y) + z(Y) \\ &\geqslant b(X - Y) + p(Y - X) - z(X - Y) + z(Y - X) \\ &= b'(X - Y) - p'(Y - X), \end{aligned}$$

hence (p', b') is indeed a paramodular pair. The main reason for defining the contraction of an element $z \in Q(p, b)$ this way is shown by the following lemma.

Lemma 2.5. *Let $Q(p', b')$ be the polymatroid obtained by contracting $z \in Q(p, b)$. Then $x + z \in Q(p, b)$ for every $x \in Q(p', b')$.*

Proof. Let $x \in Q(p', b')$. By definition, this implies $p'(Y) \leqslant x(Y) \leqslant b'(Y)$ for $Y \subseteq S$. Thus $p(Y) = p'(Y) + z(Y) \leqslant x(Y) + z(Y) \leqslant b'(Y) + z(Y) = b(Y)$, concluding the proof. □

For a collection $\mathcal{T}$ of subsets of S, we call $\mathcal{L} \subseteq \mathcal{T}$ an *independent laminar system* if for any pair $X, Y \in \mathcal{L}$: (i) they do not properly intersect, i.e. either $X \subseteq Y$, $Y \subseteq X$ or $X \cap Y = \emptyset$, and (ii) the characteristic vectors χ_Z of the sets $Z \in \mathcal{L}$ are independent over the real numbers. A *maximal* independent laminar system $\mathcal{L}$ with respect to $\mathcal{T}$ is an independent laminar system in $\mathcal{T}$ such that for any $Y \in \mathcal{T} - \mathcal{L}$ the system $\mathcal{L} \cup \{Y\}$ is not independent laminar. In other words, if we include any set Y from $\mathcal{T} - \mathcal{L}$, it will properly intersect at least one set Y from $\mathcal{L}$, or χ_Y can be given as a linear combination of $\{\chi_Z \mid Z \in \mathcal{L}\}$. Given a laminar system $\mathcal{L}$ and a set $X \subseteq S$, the set of maximal members of $\mathcal{L}$ lying inside X is denoted by $\mathcal{L}^{\max}(X)$, that is, $\mathcal{L}^{\max}(X) = \{Y \in \mathcal{L} \mid Y \subsetneq X, \nexists Y' \in \mathcal{L} \text{ s.t. } Y \subsetneq Y' \subsetneq X\}$.

2.2 An approximation algorithm on bounded degree g-polymatroids

The aim of this section is to prove Theorem 2.2 and Theorem 2.3. These theorems extends the result of Király et al. [73] from matroids to g-polymatroids. However, adapting their algorithm is not immediate due to the following major differences. A crucial step of their approach is to relax the problem by deleting a constraint corresponding to a hyperedge ε with small $g(\varepsilon)$ value. This step is feasible when the solution is a $0-1$ vector, but it is not applicable for g-polymatroids (or even for polymatroids) where an integral element might have coordinates larger than 1. This difficulty is compounded by the presence of multiplicity vectors, that makes both the computations and the tracking of changes after hyperedge deletions more complicated. Finally, in contrast to matroids that are defined by a submodular function (the rank function), g-polymatroids are determined by a pair of supermodular and submodular functions. Thus the structure of the family of tight sets is more complex, which affects the proof of one of the key claims (Claim 2.6).

We start by formulating a linear programming relaxation for the BOUNDED DEGREE G-POLYMATROID ELEMENT PROBLEM:

$$
\begin{aligned}
& & \text{minimise} \quad & \sum_{s \in S} c(s)\, x(s) & & \\
\text{(LP)} \qquad & & \text{subject to} \quad & p(Z) \leqslant x(Z) \leqslant b(Z) & \qquad & \forall Z \subseteq S \\
& & & f(\varepsilon) \leqslant \sum_{s \in \varepsilon} m_\varepsilon(s)\, x(s) \leqslant g(\varepsilon) & & \forall \varepsilon \in \mathcal{E}
\end{aligned}
$$

Although the linear program has an exponential number of constraints, it can be separated in polynomial time using submodular minimisation [62, 87, 99]. Algorithm 2.1 generalises the approach of Király et al. [73]. We iteratively solve the linear program, delete elements which get a zero value in the solution, update the solution values and perform a contraction on the polymatroid, or remove constraints arising from the hy-

pergraph. In the first round, the bounds on the coordinates solely depend on p and b, while in the subsequent rounds the whole problem is restricted to the unit cube.

Algorithm 2.1 Approximation algorithm for the BOUNDED DEGREE G-POLYMATROID ELEMENT WITH MULTIPLICITIES problem.

Input: A g-polymatroid $Q(p,b)$ on ground set S, cost function $c : S \to \mathbb{R}$, a hypergraph $H = (S, \mathcal{E})$, lower and upper bounds $f, g : \mathcal{E} \to \mathbb{Z}_{\geqslant 0}$, multiplicities $m_\varepsilon : S \to \mathbb{Z}_{\geqslant 0}$ for $\varepsilon \in \mathcal{E}$ satisfying $m_\varepsilon(s) = 0$ for $s \in S - \varepsilon$.

Output: $z \in Q(p,b)$ with cost at most $\mathsf{OPT}_{\mathrm{LP}}$, violating the hyperedge constraints by at most $2\Delta - 1$.

1: Initialise $z(s) \leftarrow 0$ for every $s \in S$.

2: **while** $S \neq \emptyset$ **do**

3: Compute a basic optimal solution x for (LP).
(Note: starting from the second iteration, $0 \leqslant x \leqslant 1$.)

a: Delete any element s with $x(s) = 0$.
Update each hyperedge $\varepsilon \leftarrow \varepsilon - s$ and $m_\varepsilon(s) \leftarrow 0$.
Update the g-polymatroid $Q(p,b) \leftarrow Q(p,b)\backslash s$ by deletion.

b: For all $s \in S$ update $z(s) \leftarrow z(s) + \lfloor x \rfloor(s)$.
Apply polymatroid contraction $Q(p,b) \leftarrow Q(p,b)/\lfloor x \rfloor$, that is, redefine $p(Y) := p(Y) - \lfloor x \rfloor(Y)$ and $b(Y) := b(Y) - \lfloor x \rfloor(Y)$ for every $Y \subseteq S$.
Update $f(\varepsilon) \leftarrow f(\varepsilon) - \sum_{s \in \varepsilon} m_\varepsilon(s)\lfloor x \rfloor(s)$ and $g(\varepsilon) \leftarrow g(\varepsilon) - \sum_{s \in \varepsilon} m_\varepsilon(s)\lfloor x \rfloor(s)$ for each $\varepsilon \in \mathcal{E}$.

c: If $m_\varepsilon(\varepsilon) \leqslant 2\Delta - 1$, let $\mathcal{E} \leftarrow \mathcal{E} - \varepsilon$.

d: **if** it is the first iteration **then**
Take the intersection of $Q(p,b)$ and the unit cube $[0,1]^S$, that is, $p(Y) := \max\{p(Y') - |Y' - Y| : Y' \subseteq S\}$ and $b(Y) := \min\{b(Y') + |Y - Y'| : Y' \subseteq S\}$ for every $Y \subseteq S$.

return z

Theorem 2.2. *There is an algorithm for the* BOUNDED DEGREE G-POLYMATROID ELEMENT WITH MULTIPLICITIES *problem which returns an integral element x of $Q(p,b)$ with cost at most the optimum value such that $f(\varepsilon) - 2\Delta + 1 \leqslant \sum_{s \in \varepsilon} m_\varepsilon(s)x(s) \leqslant g(\varepsilon) + 2\Delta - 1$ for each $\varepsilon \in \mathcal{E}$, where $\Delta = \max_{s \in S}\left\{\sum_{\varepsilon \in \mathcal{E}} m_\varepsilon(s)\right\}$. The run time of the algorithm is polynomial in n and $\log \sum_{\varepsilon \in \mathcal{E}} (f(\varepsilon) + g(\varepsilon))$.*

Proof. Our algorithm is presented as Algorithm 2.1.

Correctness. First we show that if the algorithm terminates then the returned solution z satisfies the requirements of the theorem. In a single iteration, the g-polymatroid $Q(p,b)$ is updated to $(Q(p,b)\backslash D)/\lfloor x \rfloor$, where $D = \{s : x(s) = 0\}$ is the set of deleted elements. In the first iteration, the g-polymatroid thus obtained is further intersected with the unit cube. By Lemma 2.5, the vector $x - \lfloor x \rfloor$ restricted to $S - D$ remains a feasible solution for the modified linear program in the next iteration. Note that this vector is contained in the unit cube as its coordinates are between 0 and 1. This remains true when a lower degree constraint is removed in Line 3.c as well, therefore the cost of z plus the cost of an optimal LP solution does not increase throughout the procedure.

Hence the cost of the output z is at most the cost of the initial LP solution, which is at most the optimum.

By Lemma 2.5, the vector $x - \lfloor x \rfloor + z$ is contained in the original g-polymatroid, although it might violate some of the lower and upper bounds on the hyperedges. We only remove the constraints corresponding to the lower and upper bounds for a hyperedge ε when $m_\varepsilon(\varepsilon) \leqslant 2\Delta - 1$. As the g-polymatroid is restricted to the unit cube after the first iteration, these constraints are violated by at most $2\Delta - 1$, as the total value of $\sum_{s\in\varepsilon} m_\varepsilon(s)z(s)$ can change by a value between 0 and $2\Delta - 1$ in the remaining iterations.

It remains to show that the algorithm terminates successfully. The proof is based on similar arguments as in Király et al. [73, proof of Theorem 2].

Termination. Suppose, for sake of contradiction, that the algorithm does not terminate. Then there is some iteration after which none of the simplifications in Lines 3.a–3.c can be performed. This implies that for the current basic LP solution x it holds $0 < x(s) < 1$ for each $s \in S$ and $m_\varepsilon(\varepsilon) \geqslant 2\Delta$ for each $\varepsilon \in \mathcal{E}$. We say that a set Y is *p-tight* (or *b-tight*) if $x(Y) = p(Y)$ (or $x(Y) = b(Y)$), and let $\mathcal{T}^p = \{Y \subseteq S : x(Y) = p(Y)\}$ and $\mathcal{T}^b = \{Y \subseteq S : x(Y) = b(Y)\}$ denote the collections of p-tight and b-tight sets with respect to solution x.

Let $\mathcal{L}$ be a maximal independent laminar system in $\mathcal{T}^p \cup \mathcal{T}^b$.

Claim 2.6. $\operatorname{span}(\{\chi_Z \mid Z \in \mathcal{L}\}) = \operatorname{span}(\{\chi_Z \mid Z \in \mathcal{T}^p \cup \mathcal{T}^b\})$

Proof of Claim 2.6. The proof uses an uncrossing argument. Let us suppose indirectly that there is a set R from $\mathcal{T}^p \cup \mathcal{T}^b$ for which $\chi_R \notin \operatorname{span}(\{\chi_Z \mid Z \in \mathcal{L}\})$. Choose this set R so that it is incomparable to as few sets of $\mathcal{L}$ as possible. Without loss of generality, we may assume that $R \in \mathcal{T}^p$. Now choose a set $T \in \mathcal{L}$ that is incomparable to R. Note that such a set necessarily exists as the laminar system is maximal. We distinguish two cases.

Case 1. $T \in \mathcal{T}^p$. Because of the supermodularity of p, we have

$$\begin{aligned} x(R) + x(T) &= p(R) + p(T) \leqslant p(R \cup T) + p(R \cap T) \leqslant x(R \cup T) + x(R \cap T) \\ &= x(R) + x(T), \end{aligned}$$

hence equality holds throughout. That is, $R \cup T$ and $R \cap T$ are in $\mathcal{T}^p$ as well. In addition, since $\chi_R + \chi_T = \chi_{R\cup T} + \chi_{R\cap T}$ and χ_R is not in $\operatorname{span}(\{\chi_Z \mid Z \in \mathcal{L}\})$, either $\chi_{R\cup T}$ or $\chi_{R\cap T}$ is not contained in $\operatorname{span}(\{\chi_Z \mid Z \in \mathcal{L}\})$. However, both $R \cup T$ and $R \cap T$ are incomparable with fewer sets of $\mathcal{L}$ than R, which is a contradiction.

Case 2. $T \in \mathcal{T}^b$. Because of the cross-inequality, we have

$$\begin{aligned} x(T) - x(R) &= b(T) - p(R) \geqslant b(T - R) - p(R - T) \geqslant x(T - R) - x(R - T) \\ &= x(T) - x(R), \end{aligned}$$

implying $T - R \in \mathcal{T}^b$ and $R - T \in \mathcal{T}^p$. Since $\chi_R + \chi_T = \chi_{T-R} + \chi_{R-T} + 2\,\chi_{R\cap T}$ holds and χ_R is not in $\operatorname{span}(\{\chi_Z \mid Z \in \mathcal{L}\})$, one of the vectors χ_{T-R}, χ_{R-T} and $\chi_{R\cap T}$ is not

contained in $\mathrm{span}(\{\chi_Z \mid Z \in \mathcal{L}\})$. However, any of the sets $T - R$, $R - T$ and $R \cap T$ is incomparable with fewer sets of $\mathcal{L}$ than R, which is a contradiction.

The case when $R \in \mathcal{T}^b$ is analogous to the above. This completes the proof of the Claim. ◊

We say that a hyperedge $\varepsilon \in \mathcal{E}$ is *tight* if either $f(\varepsilon) = \sum_{s\in\varepsilon} m_\varepsilon(s)x(s)$ or $g(\varepsilon) = \sum_{s\in\varepsilon} m_\varepsilon(s)x(s)$ holds. As x is a basic solution, there is a set $\mathcal{E}' \subseteq \mathcal{E}$ of tight hyperedges such that $\{m_\varepsilon \mid \varepsilon \in \mathcal{E}'\} \cup \{\chi_Z \mid Z \in \mathcal{L}\}$ are linearly independent vectors with $|\mathcal{E}'| + |\mathcal{L}| = |S|$.

We derive a contradiction using a token-counting argument. We assign 2Δ tokens to each element $s \in S$, accounting for a total of $2\Delta|S|$ tokens. The tokens are then redistributed in such a way that each hyperedge in $\mathcal{E}'$ and each set in $\mathcal{L}$ collects at least 2Δ tokens, while at least one extra token remains. This implies that $2\Delta|S| > 2\Delta|\mathcal{E}'| + 2\Delta|\mathcal{L}|$, leading to a contradiction.

We redistribute the tokens as follows. Each element s gives Δ tokens to the smallest member in $\mathcal{L}$ it is contained in, and $m_\varepsilon(s)$ tokens to each hyperedge $\varepsilon \in \mathcal{E}'$ it is contained in. As $\sum_{\varepsilon\in\mathcal{E}:s\in\varepsilon} m_\varepsilon(s) \leqslant \Delta$ holds for every element $s \in S$, thus we redistribute at most 2Δ tokens per element and so the redistribution step is valid. Now consider any set $U \in \mathcal{L}$. Recall that $\mathcal{L}^{\max}(U)$ consists of the maximal members of $\mathcal{L}$ lying inside U. Then $U - \bigcup_{W\in\mathcal{L}^{\max}(U)} W \neq \emptyset$, as otherwise $\chi_U = \sum_{W\in\mathcal{L}^{\max}(U)} \chi_W$, contradicting the independence of $\mathcal{L}$. For every set Z in $\mathcal{L}$, $x(Z)$ is an integer, meaning that $x(U - \bigcup_{W\in\mathcal{L}^{\max}(U)} W)$ is an integer. But also $0 < x(s) < 1$ for every $s \in S$, which means that $U - \bigcup_{W\in\mathcal{L}^{\max}(U)} W$ contains at least 2 elements. Therefore, each set U in $\mathcal{L}$ receives at least 2Δ tokens, as required. By assumption, $m_\varepsilon(\varepsilon) \geqslant 2\Delta$ for every hyperedge $\varepsilon \in \mathcal{E}'$, which means that each hyperedge in $\mathcal{E}'$ receives at least 2Δ tokens, as required.

If $\sum_{\varepsilon\in\mathcal{E}':s\in\varepsilon} m_\varepsilon(s) \leqslant \Delta$ holds for any $s \in S$ or $\mathcal{L}^{\max}(S)$ is not a partition of S, then an extra token exists. Otherwise, $\sum_{\varepsilon\in\mathcal{E}'} m_\varepsilon = \Delta \cdot \chi_S = \Delta \cdot \sum_{W\in\mathcal{L}^{\max}(S)} \chi_W$, contradicting the independence of $\{m_\varepsilon \mid \varepsilon \in \mathcal{E}'\} \cup \{\chi_Z \mid Z \in \mathcal{L}\}$.

Time complexity. Solving an LP, as well as removing a hyperedge in Line 3.a or removing an element from a hyperedge in Line 3.c can be done in polynomial time. In Lines 3.b and 3.d, we calculate the value of the current functions p and b for a set Y only when it is needed during the ellipsoid method. We keep track of the vectors $\lfloor x \rfloor$ that arise during contraction steps (there is only a polynomial number of them), and every time a query for p or b happens, it takes into account every contraction and removal that occurred until that point.

Line 3.a can be repeated at most $|S|$ times, while Line 3.c can be repeated at most $|\mathcal{E}|$ times. Starting from the second iteration, we are working in the unit cube. That is, when Line 3.b adds the integer part of a variable $x(s)$ to $z(s)$ and reduces the problem, then the given variable will be 0 in the next iteration and so element s is deleted. This means that the total number of iterations of Line 3.b is at most $O(|S|)$. □

Now we consider case when only lower or only upper bounds are given.

Theorem 2.3. *There is an algorithm for* LOWER BOUNDED DEGREE G-POLYMATROID ELEMENT WITH MULTIPLICITIES *which returns an integral element x of $Q(p,b)$ with cost at most the optimum value such that $f(\varepsilon) - \Delta + 1 \leqslant \sum_{s\in\varepsilon} m_\varepsilon(s)x(s)$ for each $\varepsilon \in \mathcal{E}$. An analogous result holds for* UPPER BOUNDED DEGREE G-POLYMATROID ELEMENT WITH MULTIPLICITIES, *where $\sum_{s\in\varepsilon} m_\varepsilon(s)x(s) \leqslant g(\varepsilon) + \Delta - 1$. The run time of these algorithms is polynomial in n and $\log \sum_{\varepsilon\in\mathcal{E}} f(\varepsilon)$ or $\log \sum_{\varepsilon\in\mathcal{E}} g(\varepsilon)$, respectively.*

Proof. The proof is similar to the proof of Theorem 2.2, the main difference appears in the counting argument. When only lower bounds are present, the condition in Line 3.c changes: we delete a hyperedge ε if $f(\varepsilon) \leqslant \Delta - 1$. Suppose, for the sake of contradiction, that the algorithm does not terminate. Then there is an iteration after which none of the simplifications in Lines 3.a–3.c can be performed. This implies that in the current basic solution $0 < x(s) < 1$ holds for each $s \in S$ and $f(\varepsilon) \geqslant \Delta$ for each $\varepsilon \in \mathcal{E}$. We choose a subset $\mathcal{E}' \subseteq \mathcal{E}$ and a maximal independent laminar system $\mathcal{L}$ of tight sets the same way as in the proof of Theorem 2.2. Recall that $|\mathcal{E}'| + |\mathcal{L}| = |S|$.

Let $Z_1, \ldots, Z_k$ denote the members of the laminar system $\mathcal{L}$. As $\mathcal{L}$ is an independent system, $Z_i - \bigcup_{W\in\mathcal{L}^{\max}(Z_i)} W \neq \emptyset$. Since $x(s) < 1$ for all $s \in S$,

$$x(Z_i - \bigcup_{W\in\mathcal{L}^{\max}(Z_i)} W) < |Z_i - \bigcup_{W\in\mathcal{L}^{\max}(Z_i)} W| \ .$$

As we have integers on both sides of this inequality, we get

$$|Z_i - \bigcup_{W\in\mathcal{L}^{\max}(Z_i)} W| - x(Z_i - \bigcup_{W\in\mathcal{L}^{\max}(Z_i)} W) \geqslant 1 \quad \text{for all } i = 1, \ldots, k \ .$$

Moreover, $\sum_{s\in\varepsilon} m_\varepsilon(s)x(s) \geqslant f(\varepsilon) \geqslant \Delta$ for all hyperedges; therefore,

$$\begin{aligned} |\mathcal{E}'| + |\mathcal{L}| &\leqslant \sum_{\varepsilon\in\mathcal{E}'} \frac{\sum_{s\in\varepsilon} m_\varepsilon(s)x(s)}{\Delta} + \sum_{i=1}^{k} \left[|Z_i - \bigcup_{W\in\mathcal{L}^{\max}(Z_i)} W| - x(Z_i - \bigcup_{W\in\mathcal{L}^{\max}(Z_i)} W) \right] \\ &= \sum_{s\in S} \frac{x(s)}{\Delta} \sum_{\substack{\varepsilon\in\mathcal{E}' \\ s\in\varepsilon}} m_\varepsilon(s) + \sum_{W\in\mathcal{L}^{\max}(S)} |W| - \sum_{W\in\mathcal{L}^{\max}(S)} x(W) \leqslant |S| \ . \end{aligned}$$

In the last line, the first term is at most $x(S)$ since $\sum_{\varepsilon\in\mathcal{E}:s\in\varepsilon} m_\varepsilon(s) \leqslant \Delta$ holds for each element $s \in S$. From $x(S) - \sum_{W\in\mathcal{L}^{\max}(S)} x(W) \leqslant |S| - \sum_{W\in\mathcal{L}^{\max}(S)} |W|$ the upper bound of $|S|$ follows. As $|S| = |\mathcal{L}| + |\mathcal{E}'|$, we have equality throughout. This implies that

$$\sum_{\varepsilon\in\mathcal{E}'} m_\varepsilon = \Delta \cdot \chi_S = \Delta \cdot \sum_{W\in\mathcal{L}^{\max}(S)} \chi_W,$$

contradicting linear independence.

If only upper bounds are present, we remove a hyperedge ε in Line 3.c when $g(\varepsilon) + \Delta - 1 \geqslant m_\varepsilon(\varepsilon)$. Suppose, for the sake of contradiction, that the algorithm does not terminate. Then there is an iteration after which none of the simplifications in Lines 3.a–3.c can be performed. This implies that in the current basic solution $0 < x(s) < 1$ holds

for each $s \in S$ and $m_\varepsilon(\varepsilon) - g(\varepsilon) \geqslant \Delta$ for each $\varepsilon \in \mathcal{E}$. Again, we choose a subset $\mathcal{E}' \subseteq \mathcal{E}$ and a maximal independent laminar system $\mathcal{L}$ of tight sets the same way as in the proof of Theorem 2.2.

Let $Z_1, \ldots, Z_k$ denote the members of the laminar system $\mathcal{L}$. As $\mathcal{L}$ is an independent system, $Z_i - \bigcup_{W \in \mathcal{L}^{\max}(Z_i)} W \neq \emptyset$ and so

$$x(Z_i - \bigcup_{W \in \mathcal{L}^{\max}(Z_i)} W) \geqslant 1 \ .$$

By $\sum_{s\in\varepsilon} m_\varepsilon(s)x(s) \leqslant g(\varepsilon)$, we get $\sum_{s\in\varepsilon} m_\varepsilon(s) - \sum_{s\in\varepsilon} m_\varepsilon(s)x(s) \geqslant m_\varepsilon(\varepsilon) - g(\varepsilon) \geqslant \Delta$. Thus,

$$\begin{aligned}
|\mathcal{E}'| + |\mathcal{L}| &\leqslant \sum_{\varepsilon\in\mathcal{E}'} \frac{\sum_{s\in\varepsilon} m_\varepsilon(s) - \sum_{s\in\varepsilon} m_\varepsilon(s)x(s)}{\Delta} + \sum_{i=1}^{k} x(Z_i - \bigcup_{W\in\mathcal{L}^{\max}(Z_i)} W) \\
&= \sum_{s\in S} \frac{1-x(s)}{\Delta} \sum_{\substack{\varepsilon\in\mathcal{E}' \\ s\in\varepsilon}} m_\varepsilon(s) + \sum_{W\in\mathcal{L}^{\max}(S)} x(W) \\
&\leqslant \sum_{s\in S} \frac{1-x(s)}{\Delta} \sum_{\substack{\varepsilon\in\mathcal{E}' \\ s\in\varepsilon}} m_\varepsilon(s) + x(S) \leqslant |S| \ .
\end{aligned}$$

In the last line, the first term is at most $|S| - x(S)$ since $\sum_{\varepsilon\in\mathcal{E}: s\in\varepsilon} m_\varepsilon(s) \leqslant \Delta$ holds for every element $s \in S$. Therefore, the upper bound of $|S|$ follows. As $|S| = |\mathcal{L}| + |\mathcal{E}'|$, we have equality throughout. This implies that $\sum_{\varepsilon\in\mathcal{E}'} m_\varepsilon = \Delta \cdot \chi_S = \Delta \cdot \sum_{W\in\mathcal{L}^{\max}(S)} \chi_W$, contradicting linear independence. □

We have seen in § 2.1 that base polymatroids are special cases of g-polymatroids. This implies that the results of Theorems 2.2 and 2.3 immediately apply to polymatroids. In the BOUNDED DEGREE POLYMATROID BASIS WITH MULTIPLICITIES problem, we are given a base polymatroid $B(b) = (S, b)$ with a cost function $c : S \to \mathbb{R}$, and a hypergraph $H = (S, \mathcal{E})$ on the same ground set. The input contains lower and upper bounds $f, g : \mathcal{E} \to \mathbb{Z}_{\geqslant 0}$ and multiplicity vectors $m_\varepsilon : S \to \mathbb{Z}_{\geqslant 1}$ for every hyperedge $\varepsilon \in \mathcal{E}$. The objective is to find a minimum-cost element $x \in B(b)$ such that $f(\varepsilon) \leqslant \sum_{s\in\varepsilon} m_\varepsilon(s)x(s) \leqslant g(\varepsilon)$ holds for each $\varepsilon \in \mathcal{E}$.

Corollary 2.7. *There is a polynomial-time algorithm for* BOUNDED DEGREE POLYMATROID BASIS WITH MULTIPLICITIES *problem which returns an integral element x of $B(b)$ with cost at most the optimum value such that $f(\varepsilon) - 2\Delta + 1 \leqslant \sum_{s\in\varepsilon} m_\varepsilon(s)x(s) \leqslant g(\varepsilon) + 2\Delta - 1$ for each $\varepsilon \in \mathcal{E}$.*

Corollary 2.8. *There is a polynomial-time algorithm for* LOWER BOUNDED DEGREE POLYMATROID BASIS WITH MULTIPLICITIES *problem which returns an integral element x of $B(b)$ with cost at most the optimum value such that $f(\varepsilon) - \Delta + 1 \leqslant \sum_{s\in\varepsilon} m_\varepsilon(s)x(s)$ for each $\varepsilon \in \mathcal{E}$. An analogous result holds for* UPPER BOUNDED DEGREE POLYMATROID BASIS WITH MULTIPLICITIES *problem, where $\sum_{s\in\varepsilon} m_\varepsilon(s)x(s) \leqslant g(\varepsilon) + \Delta - 1$.*

2.3 Degree-bounded k-component multigraphs

In this section we show that Algorithm 2.1 can be applied in order to obtain an approximation to the MINIMUM BOUNDED DEGREE k-COMPONENT MULTIGRAPH WITH EDGE BOUNDS problem, as described in Theorem 2.1.

Theorem 2.1. *There is an algorithm for the* MINIMUM BOUNDED DEGREE k-COMPONENT MULTIGRAPH WITH EDGE BOUNDS *problem that, in time polynomial in n and $\log \sum_v \rho(v)$, returns a multigraph Z consisting of at most k components and $\rho(V)/2$ edges, where each vertex v has degree at least $\rho(v) - 1$ and the cost of Z is at most the cost of $\min\{c^\mathsf{T} x \mid x \in \mathcal{P}_{\mathrm{CG}}(\rho, L, U, k)\}$, which is defined below:*

$$\mathcal{P}_{\mathrm{CG}}(\rho, L, U, k) := \left\{ x \in \mathbb{R}^E_{\geqslant 0} \;\middle|\; \begin{array}{ll} \mathrm{supp}(x) \text{ consists of at most } k \text{ components} & \\ x(E) = \rho(V)/2 & \\ x(\dot\delta(v)) \geqslant \rho(v) & \forall v \in V \\ L(vw) \leqslant x(vw) \leqslant U(vw) & \forall v, w \in V \end{array} \right\} . \tag{2.1}$$

Let us consider a MINIMUM BOUNDED DEGREE k-COMPONENT MULTIGRAPH WITH EDGE BOUNDS problem instance (G, c, ρ, L, U, k) on a graph $G(V, E)$, where c, ρ, L, U and k are nonnegative and $\rho(V) = \sum_{v \in V} \rho(v)$ is even.[1] Note that we do not require c to satisfy the triangle inequality.

The vectors x for which $x(E)$ is a given number and the number of components of $\mathrm{supp}(x)$ is at most k form a polymatroid. The additional constraints on the vertices can be regarded as intersecting a polymatroid with boxes, and the result is also a polymatroid, as shown in Theorem 2.4. Therefore, we can use the result in Corollary 2.8 to obtain an approximate solution to the MINIMUM BOUNDED DEGREE k-COMPONENT MULTIGRAPH WITH EDGE BOUNDS problem. We start with defining the specific input variables passed over Algorithm 2.1.

We first set the base set S as the edge set E of our original graph G. In the hypergraph $H = (S, \mathcal{E})$, the elements of S thus correspond to the edges of G. Moreover, there is a hyperedge ε for every vertex in V, defined the following way: $\mathcal{E} := \{\dot\delta(v) \mid v \in V\}$. The multiplicity of an element s in a hyperedge ε is 1, that is, $m_\varepsilon(s) := 1$ if s corresponds to a regular edge $e \in E$, and $m_\varepsilon(s) := 2$ if s corresponds to a self-loop. We set the lower bound f for a hyperedge ε according to the degree requirement of the corresponding vertex v, that is $f(\varepsilon) := \rho(v)$.

We now define the second input of Algorithm 2.1, a g-polymatroid $Q(S, p, b)$. This is done in two steps, by first defining an auxiliary polymatroid $Q'(S, p', b')$, then taking the intersection of the g-polymatroid Q' with a box. From now on we use $\hat\rho = \rho(V)/2 - |V| + k$. We define the border function p' as the zero vector on S, and $b'(Z)$ as follows:

[1] Due to the handshaking lemma, the sum of degrees in a graph is even, therefore $\rho(V)$ being even is necessary.

Lemma 2.9. *Let b' denote the following function defined on sets $Z \subseteq S$:*

$$b'(Z) = \begin{cases} |V(Z)| - \mathrm{comp}(Z) + \hat{\rho} & \text{if } Z \neq \emptyset, \\ 0 & \text{otherwise} \end{cases}. \tag{2.3}$$

Then b' is a polymatroid function.

Proof. By definition, $b'(\emptyset) = 0$ and b' is monotone increasing. It remains to show that b' is submodular. Let $X, Y \subseteq S$. The submodular inequality clearly holds if one of X and Y is empty. If none of X and Y is empty then the submodular inequality follows from the fact that $|V(Z)| - \mathrm{comp}(Z)$ is the rank function of the graphic matroid. □

Consider the g-polymatroid $B(p', b')$ determined by the border functions defined in Equation (2.3). Let us define the set $B = \{x \in \mathbb{Z}^E_{\geqslant 0} \mid x(E) = \rho(V)/2,\ \mathrm{supp}(x)$ *consists of at most k components*$\}$.

Lemma 2.10. $B = B(p', b') \cap \mathbb{Z}^E_{\geqslant 0}$.

Proof. Take an integral element $x \in B(p', b')$. Since an element of B consists of at most k components, let us take a partition $V_1 \uplus \cdots \uplus V_{k+1}$ of V, and denote the set of edges between $V_1, \ldots, V_{k+1}$ by C. Then

$$\begin{aligned} x(C) &= x(E) - x\big(E(V_1) \cup \cdots \cup E(V_{k+1})\big) \\ &\geqslant |V| - k + \hat{\rho} - \left(\sum_{i=1}^{k+1} |V_i| - \mathrm{comp}\big(E(V_1) \cup \cdots \cup E(V_{k+1})\big) + \hat{\rho}\right) \\ &\geqslant 1, \end{aligned} \tag{2.4}$$

therefore $\mathrm{supp}(x)$ consists of at most k components.[2] Since $x(E) = |V| - k + \hat{\rho} = \rho(V)/2$, $x \in B$ follows, showing that $B(p', b') \subseteq B$.

To see the other direction, take an element $x \in B$. Let F be a set of edges such that $F \subseteq E$, and let G/F be the graph where the edges in F are contracted, resulting in each component of F being contracted to a new vertex in G. Since $\mathrm{supp}_{E(G)}(x)$ consists of at most k components, $\mathrm{supp}_{E(G/F)}(x)$ also has to consist of at most k components. As the number of vertices of G/F is $\mathrm{comp}(F) + |V - V(F)|$ and $\mathrm{supp}_{E(G/F)}(x)$ consists of at most k components, necessarily $x(E - F) \geqslant \mathrm{comp}(F) + |V - V(F)| - k$ holds. Therefore

$$\begin{aligned} x(F) &= x(E) - x(E - F) \\ &\leqslant \rho(V)/2 - (|V - V(F)| + \mathrm{comp}(F) - k) \\ &= |V(F)| - \mathrm{comp}(F) + \hat{\rho}, \end{aligned}$$

thus $x(F) \leqslant b'(F)$. As $x(E) = \rho(V)/2 = |V| - k + \hat{\rho}$, we obtain $x \in B(p', b')$, showing $B \subseteq B(p', b')$.

□

[2] Note that considering a partition of V with less than $k+1$ parts, the right hand side of Equation (2.4) is zero or less, therefore allowing $\mathrm{supp}(x)$ to consist of $1, \ldots, k$ components.

So far we proved that the integral points of $Q'(S, p', b')$ correspond to a connected multigraph on V that has $\rho(V)/2$ edges. Let $\beta(L, U)$ be the box defined by

$$\beta(L, U) := \left\{x \in \mathbb{R}^S_{\geqslant 0} \mid L(s) \leqslant x(s) \leqslant U(s) \ \ \forall s \in S\right\} \ .$$

Let us define the polymatroid $Q = (S, p, b)$ as the intersection of the polymatroid Q' and the box β, where the border functions p, b are defined as in Equation (2.2). We now prove that taking $H = (S, \mathcal{E})$ and $Q(S, p, b)$ as input, the output of Algorithm 2.1 corresponds to a multigraph with the properties stated in Theorem 2.1.

Proof of Theorem 2.1. Consider the linear program (LP) that is defined in the iterative rounding method for the G-POLYMATROID ELEMENT WITH MULTIPLICITIES problem. The constraints regarding the bounds on the hyperedges imply $\rho(v) \leqslant x(\dot{\delta}(v))$ for every $v \in V$: note that $m_\varepsilon(s) = 2$ for self loops and 1 for simple edges, and this is equal to the contribution of an edge uv to the value $x(\dot{\delta}(v))$. This, together with Lemma 2.10 and the fact that x is contained in the box $\beta(L, U)$, implies that Algorithm 2.1 returns an integral solution z such that the cost of z is at most the minimum cost element of $\mathcal{P}_{\mathrm{CG}}$.

According to Theorem 2.3, the integral solution z violates the bounds f on the hyperedges by at most $\Delta - 1$, where $\Delta := \max_{s \in S} \left\{\sum_{\varepsilon \in \mathcal{E}: s \in \varepsilon} m_\varepsilon(s)\right\}$. But we defined $m_\varepsilon(s)$ to be equal to 2 if s corresponds to a self-loop in G and 1 otherwise, meaning that the solution z violates the bounds f on the hyperedges and thus the bounds ρ on the vertices by at most 1. The solution z has at most k components, with a total number of edges $\rho(V)/2$ and z satisfies the edge bounds L and U, due to Theorem 2.3. Therefore, the solution z corresponds to a multigraph that admits the properties in the claim of Theorem 2.1. □

Remark 1. *Note that Theorems 2.2 and 2.3 only provide a solution if there exists a (fractional) solution to the underlying linear program in (LP). Consequently, Theorem 2.1 only provides a solution if the polytope $\mathcal{P}_{\mathrm{CG}}$ in Equation (2.1) is not empty.*

Summary

In this chapter we showed a polynomial-time approximation for the BOUNDED DEGREE G-POLYMATROID ELEMENT PROBLEM WITH MULTIPLICITIES. Our algorithm outputs an element with at most the optimal cost, that violates the hyperedge constraints by at most $2\Delta - 1$. If only lower or only upper bounds are present on the hyperedges, the violation on these constraints is at most $\Delta - 1$. These results generalise the results of Király et al. [73], who provided approximations with the same additive error in case of matroids.

As a direct application, we provided a solution to the MINIMUM BOUNDED DEGREE k-COMPONENT MULTIGRAPH WITH EDGE BOUNDS problem, that has cost at most the optimal cost and the degree of each vertex in the solution is at most one short of the required one. Besides being of fundamental interest, this problem arises as a subproblem in our approximation algorithms in Chapter 3 and Chapter 4.

CHAPTER 3

CONSTANT-FACTOR APPROXIMATIONS FOR THE MANY-VISITS PATH TSP

Introduction

The TSP is well-known to be NP-hard even in the case of symmetric costs, that satisfy the triangle inequality, i.e. when $c(uv) = c(vu)$ and $c(uw) \leqslant c(uv) + c(vw)$ holds for every vertex $u, v, w \in V$. For symmetric, metric costs, the most widely known approximation ratio that can be obtained in polynomial time is $3/2$, discovered independently by Christofides [27] and Serdyukov [103]. Approximating the metric PATH TSP has a long history, from the first $5/3$-approximation by Hoogeveen [59], through subsequent improvements [4, 49, 101, 102, 111] to the recent breakthroughs. The latest results eventually closed the gap between the metric TSP and the metric PATH TSP: Traub and Vygen [106] provided a $(3/2 + \varepsilon)$-approximation for any $\varepsilon > 0$, Zenklusen [121] provided a $3/2$-approximation and finally, the three authors showed a reduction from the PATH TSP to the TSP [108].

Throughout this chapter, we make the same assumptions on the edges, and consider the metric MANY-VISITS PATH TSP on symmetric edge costs c. Formally, a graph $G = (V, E)$ is given with a positive integer $r(v)$ for each $v \in V$, and a nonnegative cost $c(uv)$ for every pair of vertices u, v; finally, a departure city s and an arrival city t are specified. We seek a minimum cost s-t-walk that visits each city v exactly $r(v)$ times, where leaving city s as well as arriving to city t counts as one visit. Let us denote by $c^{\min}(v) := \min_{u \in V - v} c(uv)$ the cheapest edge adjacent to v. Besides the inequality $c(uw) \leqslant c(uv) + c(vw)$ for every triplet u, v, w, the cost function being metric means that the cost of the self-loop $c(vv)$ at each vertex $v \in V$ is at most the cost of leaving city v to any other city u and returning[1], that is:

$$c(vv) \leqslant 2 \cdot c^{\min}(v) \qquad \text{for all } v \in V \ .$$

We denote the total number of visits by $r(V) := \sum_{v \in V} r(v)$.

[1] Note that, unlike in most results involving metric costs, we do not assume that the cost of a self-loop is 0.

The assumption of metric costs is necessary, as the TSP, and therefore the MANY-VISITS TSP, does not admit any non-trivial approximation for unrestricted cost functions, assuming that P $\neq$ NP (see e.g. Theorem 6.13 in the book of Garey and Johnson [47]). To the best of our knowledge, no constant-factor approximation algorithms for metric MANY-VISITS TSP[2] or metric MANY-VISITS PATH TSP are currently known.

Results

We start with a simple approximation idea, that leads to a constant-factor approximation in strongly polynomial time:

Theorem 3.1. *There is a polynomial-time $5/2$-approximation for the metric* MANY-VISITS PATH TSP. *The algorithm runs in time polynomial in n and $\log r(V)$.*

The approximation factor $5/2$ in Theorem 3.1 still leaves a gap to the best-known factor $3/2$ for the metric PATH TSP, which is due to Zenklusen [121]. His recent $3/2$-approximation for the metric PATH TSP uses a Christofides-Serdyukov-like construction that combines a spanning tree and a matching, with the key difference that it calculates a constrained spanning tree in order to bound the costs of the tree and the matching by $3/2$ times the optimal value.

The main algorithmic result of this chapter matches this approximation ratio for the metric MANY-VISITS PATH TSP.

Theorem 3.2. *There is a polynomial-time $3/2$-approximation for the metric* MANY-VISITS PATH TSP. *The algorithm runs in time polynomial in n and $\log r(V)$.*

Our approach follows the main steps of Zenklusen's work [121], but differs in many details. The reason for those differences are the presence of requests $r(v)$—which could be exponentially large in n—makes the problem significantly more difficult, and several new ideas are needed to design an algorithm which returns a tour with the correct number of visits and still runs in polynomial time. For instance, whereas the backbone of both Christofides and Zenklusen's algorithm is a spanning tree (with certain properties), the possibly exponentially large number of (parallel) edges in a many-visits TSP solution requires us to work with a structure that is more general than spanning trees.

We therefore consider the problem of finding a minimum cost connected multigraph with lower bounds ρ on the degrees of vertices, and lower and upper bounds L and U, respectively, on the number of occurrences of the edges. This problem is the $k = 1$ case of the MINIMUM BOUNDED DEGREE k-COMPONENT MULTIGRAPH WITH EDGE BOUNDS problem introduced in Chapter 2, to which we also showed an approximation algorithm.

As a direct consequence of Theorem 3.2, we obtain the following:

[2] At the Hausdorff Workshop on Combinatorial Optimization in 2018, Rico Zenklusen brought up the topic of approximation algorithms for the metric version of MANY-VISITS TSP in the context of iterative relaxation techniques; he suggested an approach to obtain a $3/2$-approximation, which is unpublished.

Theorem 3.3. *There is a $3/2$-approximation for the metric* MANY-VISITS TSP *that runs in time polynomial in n and $\log r(V)$.*

The results of this chapter are joint work with Kristóf Bérczi and Matthias Mnich, and they first appeared on arXiv [15].

3.1 Preliminaries

Many-Visits Path TSP Recall that in the MANY-VISITS PATH TSP, we seek for a minimum cost s-t-walk X that visits each vertex $v \in V$ exactly $r(v)$ times. Let $r(V) = \sum_{v \in V} r(v)$. The sequence of the edges of X has length $r(V) - 1$, which is exponential in the size of the input, therefore we always use a compact representation of X that uses $O(n^2 \log r(V))$ space, as discussed in the Introduction.

A vector $z \in \mathbb{Z}^E_{\geqslant 0}$ represents a feasible many-visits path if $\mathrm{supp}(z)$ is a connected multigraph on the vertex set V, $\deg_z(v) = 2 \cdot r(v)$ holds for all $v \in V - \{s,t\}$ and $\deg_z(v) = 2 \cdot r(v) - 1$ for $v \in \{s,t\}$. Recall that each self-loop vv contributes 2 in the value $\deg(v) = |\dot{\delta}(v)|$. Let us point out that the definition of z only makes sense if s and t are distinct vertices, otherwise the total degree of z would need to be odd, which is impossible due to the handshaking lemma. For technical reasons like the one above, from now on we assume that the two endpoints s and t are different.

Denote by $X^\star_{c,r,s,t}$ an optimal solution for an instance (G, c, r, s, t) of the MANY-VISITS PATH TSP. Let us denote by $X^\star_{c,1,s,t}$ an optimal solution for the single-visit counterpart of the problem, i.e. where $r(v) = 1$ for each $v \in V$. Relaxing the connectivity requirement for solutions of the MANY-VISITS PATH TSP yields Hitchcock's transportation problem [58] presented in §1.1.1. Due to the different degree requirements of the special vertices s and t, the supplies corresponding to $v \in V - s$ are defined by $r(v)$, the supply of s by $r(s) - 1$; the demand of each vertex $v \in V - t$ by $r(v)$ and the demand of t by $r(t) - 1$, in the transportation problem. This yields to capacities $r(v)$ for (σ, σ_v) and (τ_v, τ), and $r(s) - 1$ for (σ, σ_s) and $r(t) - 1$ for (τ_t, τ) in the corresponding min-cost max-flow problem; see §1.1.1 for more details. The transportation problem is thus solvable in polynomial time using a minimum cost maximum flow algorithm [32] and we denote the multigraph constructed from the optimal solution by $\mathsf{TP}^\star_{c,r,s,t}$, where s and t denote the special vertices with decreased capacities in the flow formulation.

Lemma 3.4. *Let $\mathsf{TP}^\star_{c,r,s,t}$ be an optimal solution to the Hitchcock transportation problem, where $supply(v) + demand(v)$ is odd for $v \in \{s,t\}$ and it is even otherwise. Then $\mathsf{TP}^\star_{c,r,s,t}$ can be decomposed into cycles and exactly one s-t-path.*

Proof. Any solution Z to the transportation problem is essentially a multigraph that has a positive even degree for vertices $v \in V - \{s,t\}$, and an odd degree for $v \in \{s,t\}$. Hence, because of a parity argument, there has to be an s-t-path U in Z, possibly covering other vertices $W \subset V - \{s,t\}$. Vertices $w \in W$ have an even degree in U. Therefore, deleting the edges of U from Z, all vertices $v \in V$ will have an even degree in the modified multigraph Z'. Thus Z' can be decomposed into a union of cycles (possibly taking multiple copies of the same cycle), and the lemma follows. □

The decomposition provided by the lemma is called a *path-cycle representation*. Such a representation can be stored as a path P_0 and a collection $\mathcal{C}$ of pairs (C, μ_C), where each C is a simple closed walk (cycle) and μ_C is the corresponding integer denoting the number of copies of C. Below we show that one can always calculate a path-cycle decomposition in polynomial time, and such a decomposition takes polynomial space.

Lemma 3.5. *Let $X_{c,r,s,t}$ be a many-visits TSP path with endpoints s, t, and $\mathsf{TP}_{c,r,s,t}$ be a transportation problem solution with special vertices s, t. There is a path-cycle representation of $X_{c,r,s,t}$ and $\mathsf{TP}_{c,r,s,t}$, both of which take space polynomial in n and $\log r(V)$, and can be computed in time polynomial in n and $\log r(V)$.*

Proof. We first show the proof for a many-visits TSP path $X_{c,r,s,t}$. Let us first add an edge ts to $X_{c,r,s,t}$, and denote the resulting multigraph by Z. Observe that Z is a many-visits TSP tour with the same number of visits as $X_{c,r,s,t}$, since it is connected and the degree of every vertex v in Z is $2 \cdot r(v)$. We can now use the procedure **ConvertToSequence** in Algorithm 1, which takes the edge multiplicities of Z, denoted by $\{x(uv)\}_{u,v \in V}$ as input, and outputs a collection $\mathcal{C}$ of pairs (C, μ_C). Here, C is a simple closed walk, and μ_C is the corresponding integer denoting the number of copies of the walk C in Z. Lastly, choose an arbitrary cycle C such that $ts \in C$, and transform one copy of C into a path as follows. Let $C_0 := C$, and remove the edge ts from C_0, resulting in an s-t-path P_0. Update $\mu_C := \mu_C - 1$. Now $(P_0, \mathcal{C})$ is a path-cycle representation of $X_{c,r,s,t}$.

In every iteration, the procedure **ConvertToSequence** looks for a cycle C and removes each of its occurrences from $\{x(uv)\}_{u,v \in V}$. The procedure stops when variables $\{x(uv)\}_{u,v \in V}$ represent a graph without edges. This demonstrates that the input need not represent a connected graph in the first place, as the edge removals possibly make it disconnected during the process. Note that the only structural difference between $\mathsf{TP}_{c,r,s,t}$ and $X_{c,r,s,t}$ is that the underlying multigraph of $\mathsf{TP}_{c,r,s,t}$ might be disconnected. This means that the procedure **ConvertToSequence** can be applied to obtain a path-cycle representation of $\mathsf{TP}_{c,r,s,t}$ the same way as in the case of $X_{c,r,s,t}$.

Finally, the number of cycles C in $\mathcal{C}$ can be bounded by $O(n^2)$ (as removing all occurrences of a cycle C sets at least one variable $x(uv)$ to zero), and the algorithm has a time complexity of $O(n^4)$ [50]. The edge insertion and deletion, and other graph operations during the process, can also be implemented efficiently. This concludes the proof. □

Let $(P_0, \mathcal{C})$ be a path-cycle representation of a many-visits TSP path $X_{c,r,s,t}$. One can obtain the explicit order of the vertices from $(P_0, \mathcal{C})$ the following way: traverse the s-t-path P_0, and whenever a vertex u is reached for the first time, traverse μ_C copies of every cycle C containing u.

From now on, we assume that the path-cycle decompositions appearing in this chapter are stored in space polynomial in n and $\log r(V)$.

3.2 Simple 7/2- and 5/2-approximations for Metric Many-Visits Path TSP

One can obtain a simple 7/2-approximation solution to a MANY-VISITS PATH TSP instance (G, c, r, s, t) as follows. First, calculate a 3/2-approximation to the corresponding single-visit PATH TSP instance $(G, c, 1, s, t)$. The cost of this path is at most 3/2 times the cost of the optimal solution $X^\star_{c,r,s,t}$ to the instance (G, c, r, s, t), due to the metric edge costs. Then, for every $v \in V$, add $r(v) - 1$ copies of the self-loop at vertex v to the path above. The resulting s-t-walk X is clearly a feasible solution to (G, c, r, s, t), since it is connected and the degree of each vertex $v \in V - \{s, t\}$ is $2 \cdot r(v)$, while the degrees or s and t are $2 \cdot r(s) - 1$ and $2 \cdot r(t) - 1$, respectively. The total cost of the self-loops is $\sum_{v \in V}(r(v) - 1) \cdot c(vv)$ and can be bounded from above the following way:

$$\sum_{v \in V}(r(v) - 1) \cdot c(vv) \leqslant \sum_{v \in V}(r(v) - 1) \cdot 2 \cdot c^{\min}(v) \leqslant 2 \cdot c(\mathsf{TP}^\star_{c,r,s,t}) \leqslant 2 \cdot c(X^\star_{c,r,s,t}) \ , \tag{3.1}$$

where the second inequality follows from the fact that ensuring a visit to a vertex v costs at least $c^{\min}(v)$, and one needs $r(v)$ many of such visits. Since the path computed in the first step imposes a cost of $3/2 \cdot c(X^\star_{c,r,s,t})$ and the self-loops cost an additional $4/2 \cdot c(X^\star_{c,r,s,t})$, the approximation ratio 7/2 follows.

Remark 2. *Some clarifications regarding the second inequality are in order. When we talk about the minimum cost of visiting a vertex v in $\mathsf{TP}^\star$ or $X^\star$, we consider a cost of $c^{\min}(v)/2$ arriving to v and likewise a cost of $c^{\min}(v)/2$ departing from v. This adds up to a total cost $c^{\min}(v)$ we incur when visiting v once, and $r(v) \cdot c^{\min}(v)$ when visiting $r(v)$ times. For vertex s, however, $r(s)$ visits corresponds to departing the vertex $r(s)$ times but arriving to it only $r(s) - 1$ times, hence this case it follows that $r(s)$ visits cost at least $(r(s) - 1/2) \cdot c^{\min}(s)$. Similarly, $r(t)$ visits to t cost $(r(t) - 1/2) \cdot c^{\min}(t)$, and thus $c(\mathsf{TP}^\star_{c,r,s,t})$ can be bounded from below by*

$$\sum_{\substack{v \in V \\ v \notin \{s,t\}}} r(v) \cdot c^{\min}(v) + (r(s) - 1/2) \cdot c^{\min}(s) + (r(t) - 1/2) \cdot c^{\min}(t) \geqslant \sum_{v \in V}(r(v) - 1) \cdot c^{\min}(v) \ ,$$

and therefore the second inequality of Equation (3.1) holds.

Now we present Algorithm 3.1, an approximation algorithm for the metric MANY-VISITS PATH TSP, that runs in polynomial time and achieves an improved approximation factor of 5/2.

Theorem 3.1. *There is a polynomial-time 5/2-approximation for the metric* MANY-VISITS PATH TSP. *The algorithm runs in time polynomial in n and $\log r(V)$.*

Proof. The algorithm is presented as Algorithm 3.1. Since $X^\alpha_{c,1,s,t}$ is connected and Z contains all the edges of $X^\alpha_{c,1,s,t}$, P is also connected. Let $(P_0, \mathcal{C})$ be a path-cycle decomposition of $\mathsf{TP}^\star_{c,r,s,t}$. The graph Z thus consists of $X^\alpha_{c,1,s,t}$ and the cycles of $\mathcal{C}$. The edges of $X^\alpha_{c,1,s,t}$ contribute a degree of 1 in case of s and t, and 2 for $v \in V - \{s, t\}$; the cycles of $\mathcal{C}$ contribute a degree of $2 \cdot r(v) - 2$ for $v \in \{s, t\}$, and a degree of $2 \cdot r(v)$ or $2 \cdot r(v) - 2$

Algorithm 3.1 A polynomial-time $(\alpha+1)$-approximation for metric MANY-VISITS PATH TSP.

Input: A complete undirected graph $G = (V, E)$, costs $c : E \to \mathbb{R}_{\geqslant 0}$ satisfying the triangle inequality, requests $r : V \to \mathbb{Z}_{\geqslant 1}$, distinct vertices $s, t \in V$.
Output: An s-t-walk that visits each $v \in V$ exactly $r(v)$ times.

1: Calculate an α-approximate solution $X^{\alpha}_{c,1,s,t}$ for the single-visit metric PATH TSP instance $(G, c, 1, s, t)$.
2: Calculate an optimal solution $\mathsf{TP}^{\star}_{c,r,s,t}$ for the corresponding transportation problem, together with a path-cycle decomposition $(P_0, \mathcal{C})$, where $\mathcal{C}$ is a collection of pairs (C, μ_C).
3: Let Z be the union of $X^{\alpha}_{c,1,s,t}$ and μ_C copies of every cycle $C \in \mathcal{C}$.
4: Do shortcuts in Z and obtain a solution X, such that X visits every city v exactly $r(v)$ times (that is, $\deg_X(v) = 2 \cdot r(v)$ for every vertex $v \in V - \{s, t\}$, and $\deg_X(v) = 2 \cdot r(v) - 1$ otherwise).
return X

for $v \in V - \{s, t\}$. Let us denote the latter set by W, matching the notation used in the proof of Lemma 3.4. The total degree of v in Z is therefore:

$$\begin{aligned} 2 \cdot r(v) - 1 \quad & \text{for } v \in \{s, t\}, \\ 2 \cdot r(v) \quad & \text{for } v \in W, \text{ and} \\ 2 \cdot r(v) + 2 \quad & \text{for the remaining vertices in } V - (W \cup \{s, t\}). \end{aligned}$$

As a direct consequence of the degrees and connectivity, Z is an open walk that starts in s, visits every vertex $v \in V$ either $r(v)$ or $r(v) + 1$ times, and ends in t. Since the edge costs are metric, we can use shortcuts at the vertices $w \in V - (W \cup \{s, t\})$ to reduce their degrees by 2. We describe the procedure below.

Shortcutting. At Line 3, $(X^{\alpha}_{c,1,s,t}, \mathcal{C})$ denotes the compact path-cycle representation of the many-visits path Z. Let us construct an auxiliary multigraph A on the vertex set V by taking the edges of $X^{\alpha}_{c,1,s,t}$ and each cycle C from $\mathcal{C}$ exactly once. Note that parallel edges appear in A if and only if an edge appears in multiple distinct cycles, or in the path $X^{\alpha}_{c,1,s,t}$ and at least one cycle C. Due to the construction, s and t have odd degree, while every other vertex has an even degree in A, which means that there exist an Eulerian trail in A. Moreover, there are $O(n^2)$ cycles [50], hence the total number of edges in A is $O(n^3)$. Consequently, using Hierholzer's algorithm [38, 57], we can compute an Eulerian trail η in A in $O(n^3)$ time. The trail η covers the edges of each cycle C once. Now an implicit order of the vertices in the many-visits TSP path Z is the following. Traverse the vertices of the Eulerian trail η in order. Every time a vertex u appears the first time, traverse all cycles C that contain the vertex μ_C times. Denote this trail by η'. It is easy to see that the sequence η' is a sequence of vertices that uses the edges of $X^{\alpha}_{c,1,s,t}$ once and the edges of each cycle C exactly μ_C times, meaning that this is a feasible sequence of the vertices in the path Z. Moreover, the order itself takes polynomial space, as it is enough to store indices of $O(n^3)$ vertices and $O(n^2)$ cycles.

Denote the surplus of visits of a vertex $w \in V - (W \cup \{s,t\})$ by $\gamma(w) := \deg_P(w)/2 - r(w)$.[3] In Line 4, we remove the last $\gamma(w)$ occurrences of every vertex w from Z by doing shortcuts: if an occurrence of w is preceded by u and superseded by v in Z, replace the edges uw, wv by the single edge uv in the sequence. This can be done by traversing the compact representation of η' backwards, and removing the vertex w from the last $\gamma(w)$ cycles $C^{(w)}_{r(w)-\gamma(w)+1}, \dots, C^{(w)}_{r(w)}$. As $\sum_w \gamma(w)$ can be bounded by $O(n)$, this operation makes $O(n)$ new cycles, keeping the space required by the new sequence of vertices and cycles polynomial. Moreover, since the edge costs are metric, making shortcuts the way described above cannot increase the total cost of the edges in Z. Finally, using a similar argument as in the algorithm of Christofides, the shortcutting does not make the trail disconnected. The resulting graph is therefore an s-t-walk X that visits every vertex v exactly $r(v)$ times, that is, a feasible solution for the instance (G, c, r, s, t).

Costs and complexity. The cost of the path X constructed by Algorithm 3.1 is equal to $c(X) \leqslant c(X^{\alpha}_{c,1,s,t}) + c(\mathsf{TP}^{\star}_{c,r,s,t})$. Since $c(\mathsf{TP}^{\star}_{c,r,s,t})$ is an optimal solution to a relaxation of the MANY-VISITS PATH TSP, its cost is a lower bound to the cost of the corresponding optimal solution, $X^{\star}_{c,r,s,t}$. Since the cost of $X^{\alpha}_{c,1,s,t}$ is at most α times the cost of an optimal single-visit TSP path $X^{\star}_{c,1,s,t}$, and $c(X^{\star}_{c,1,s,t}) \leqslant c(X^{\star}_{c,r,s,t})$ holds for any r, Algorithm 3.1 provides an $(\alpha+1)$-approximation for the MANY-VISITS PATH TSP. Using Zenklusen's recent polynomial-time 3/2-approximation algorithm on the single-visit metric PATH TSP [121] in Line 1 yields the approximation guarantee of 5/2 stated in the theorem.

The transportation problem in Line 2 can be solved in $O(n^3 \log n)$ operations using the approach of Orlin [92] or its extension due to Kleinschmidt and Schannath [74]. Line 3 can also be performed in polynomial time [50], and the number of closed walks can be bounded by $O(n^2)$. Moreover, the total surplus of degrees in Z is at most $n-2$, therefore the number of operations performed during shortcutting in Line 4 is also bounded by $O(n)$. This proves that the algorithm terminates in polynomial time. □

Corollary 3.6. *There is a polynomial-time 5/2-approximation for the metric* MANY VISITS TSP, *running in time polynomial in n and $\log r(V)$.*

Proof. One can obtain a 5/2-approximation for the metric MANY-VISITS TSP by simply running Algorithm 3.1 for every pair $(u,v) \in V \times V$ and setting $s = u$ and $t = v$, then choosing a solution whose cost together with the cost of the edge uv is minimal. However, Algorithm 3.1 can be simplified while maintaining the same approximation guarantee. This approach appeared in an unpublished manuscript [14] and has a simpler proof, as the algorithm does not involve making shortcuts. □

The TSP, as well as the PATH TSP can also be formulated for directed graphs, where the costs c are asymmetric. (Note that c still has to satisfy the triangle inequality, which implies the following bound for the self-loops: $c(vv) \leqslant \min_{u \in V-v} \{c(vu) + c(uv)\}$.) In

[3] Note that by construction, Z is such that the surplus of visits $\gamma(w)$ equals either 0 or 1. However, the same shortcutting procedure is used in Algorithm 3.3 later in this chapter, where $\gamma(w)$ can take higher values as well.

a recent breakthrough, Svensson et al. [105] gave the first constant-factor approximation for the metric ATSP. In a subsequent work, Traub and Vygen [107] improved the constant factor to $22 + \varepsilon$ for any $\varepsilon > 0$. Moreover, Feige and Singh [36] proved that an α-approximation for the metric ATSP yields a $(2\alpha + \varepsilon)$-approximation for the metric PATH-ATSP, for any $\varepsilon > 0$. By combining these results with a suitable modification of Algorithm 3.1, one can obtain the following results:

Corollary 3.7. *There exists a $(23+\varepsilon)$-approximation for the metric asymmetric* MANY-VISITS TSP, *and a $(45 + \varepsilon)$-approximation for any $\varepsilon > 0$ for the metric asymmetric* MANY-VISITS PATH TSP *in polynomial time.* □

3.3 A $3/2$-approximation for the Metric Many-Visits Path TSP

Before we prove the main result of this chapter, as a warm-up we show a $3/2$-approximation for the metric MANY-VISITS TSP, using the result in Theorem 2.1.

Algorithm 3.2 A $3/2$-approximation algorithm for the metric MANY-VISITS TSP

Input: A complete undirected graph $G = (V, E)$, costs $c : E \to \mathbb{R}_{\geqslant 0}$ satisfying the triangle inequality, requests $r : V \to \mathbb{Z}_{\geqslant 1}$.
Output: A MVTSP tour that visits each $v \in V$ exactly $r(v)$ times.
1: Compute a connected multigraph Z on V such that each vertex $v \in V$ has degree at least $2 \cdot r(v) - 1$.
2: Compute a minimum-cost matching M with respect to c on the vertices $\mathrm{odd}(Z)$.
3: Let Z' denote the MVTSP tour $Z + M$.
4: Do shortcuts in Z' and obtain an MVTSP tour X that visits each city v exactly $r(v)$ times.
return X

We show that Algorithm 3.2 is an efficient $3/2$-approximation for the metric MANY-VISITS TSP.

Theorem 3.3. *There is a $3/2$-approximation for the metric* MANY-VISITS TSP *that runs in time polynomial in n and $\log r(V)$.*

Proof. Let $X^\star_{c,r}$ denote an optimal MVTSP tour for the instance (G, c, r).

According to Theorem 2.1, the multigraph Z calculated in Line 1 has cost at most $c(X^\star_{c,r})$. Due to Christofides and Serdyukov [27, 103], the cost of M calculated in Line 2 has cost at most $1/2$ times $c(X^\star_{c,1})$, the cost of an optimal single-visit TSP tour on the instance (G, c), which itself is at most $c(X^\star_{c,r})$ due to c being metric. Again because of c being metric the shortcutting procedure in Line 4 cannot increase the cost of Z, hence the cost of X is at most $3/2 \cdot c(X^\star_{c,r})$.

Note that X is a feasible MVTSP tour for the instance (G, c, r), as every vertex $v \in V$ has a degree at least $2 \cdot r(v)$ in Z, and Line 4 ensures that the degrees are exactly $2 \cdot r(v)$ in X.

Due to Theorem 2.1, Line 1 takes time polynomial in n and $\log r(V)$. The same is true for the graph operations in Lines 2–3. We can perform shortcuts in Line 4 in a similar

fashion than described in he proof of Theorem 3.1; one can simply remove an arbitrary edge of Z' to transform it into a many-visits path. Since these operations also take time polynomial in n and $\log r(V)$, the proof is finished. □

Remark 3. *Note that the approach described in Algorithm 3.2 can be used to obtain a simple 5/3-approximation to the metric* MANY-VISITS PATH TSP. *The arguments of Hoogeveen [59] also hold for the many-visits setting, hence it is possible to bound the cost of M by $2/3 \cdot c(X^\star_{c,r,s,t})$, where $X^\star_{c,r,s,t}$ is an optimal many-visits TSP path. We omit the details here.*

Now we extend Algorithm 3.2 in order to obtain a 3/2-approximation for the metric MANY-VISITS PATH TSP. Our approach follows the general strategy of Zenklusen [121], but we need to make several crucial modifications for the many-visits setting. Instead of calculating a constrained spanning tree, we use the result in Theorem 2.1 to obtain a connected multigraph Z with a sufficiently large number of edges. Then compute a matching M so that all the degrees in $Z + M$ have the correct parity, and the cost of $Z + M$ is at most 3/2 times the optimal cost. In order to ensure this bound, we have to enforce certain restrictions on Z, similarly to the computation of the spanning tree in [121]. However, as we will show, the many-visits setting leads to further challenges.

Let us recall that the two endpoints, s and t are different. First we define the Held-Karp relaxation of the MANY-VISITS PATH TSP as $\min\{c^\mathsf{T} x \mid x \in \mathcal{P}_{\text{HK}}\}$, where $\mathcal{P}_{\text{HK}}$ denotes the following polytope:

$$\text{(3.2)}\quad \mathcal{P}_{\text{HK}} := \left\{ x \in \mathbb{R}^E_{\geqslant 0} \;\middle|\; \begin{array}{ll} x(\delta(C)) \geqslant 2 & \forall C \subset V, C \neq \emptyset, |C \cap \{s,t\}| \in \{0,2\} \\ x(\delta(C)) \geqslant 1 & \forall C \subseteq V, |C \cap \{s,t\}| = 1 \\ x(\dot\delta(v)) = 2 \cdot r(v) & \forall v \in V - \{s,t\} \\ x(\dot\delta(s)) = 2 \cdot r(s) - 1 & \\ x(\dot\delta(t)) = 2 \cdot r(t) - 1 & \end{array} \right\}$$

The *Q-join polytope* (where $Q \subseteq V$ is of even cardinality) is defined as follows:

$$\text{(3.3)}\qquad \mathcal{P}^\uparrow_{Q\text{-join}} := \left\{ x \in \mathbb{R}^E_{\geqslant 0} \;\middle|\; x(\delta(C)) \geqslant 1 \;\; \forall Q\text{-cut } C \subset V \right\},$$

where a *Q-cut* is a set $C \subseteq V$ with $|C \cap Q|$ odd.

In the following, we assume arbitrary but fixed parameters c, r and endvertices s, t, and denote the optimal many-visits TSP path by $X^\star = X^\star_{c,r,s,t}$. Observe that given a solution y of the linear program $\min\{c^\mathsf{T} x \mid x \in \mathcal{P}_{\text{HK}}\}$, the vector $y/2$ is not necessarily in $\mathcal{P}^\uparrow_{Q\text{-join}}$ for every even set $Q \subseteq V$. Indeed, y only needs to have a load of 1 on s-t-cuts, therefore $y/2$ may violate some of the constraints of $\mathcal{P}^\uparrow_{Q\text{-join}}$. This means that calculating a minimum cost perfect matching on an arbitrary even set $Q \subseteq V$ might lead to a matching M with higher cost than $c(X^\star)/2$. Therefore, simply taking a solution Z provided by the algorithm for the MINIMUM BOUNDED DEGREE k-COMPONENT MULTIGRAPH WITH EDGE BOUNDS described in Theorem 2.1, and a minimum cost matching M on the vertices with degrees having incorrect parity, then applying shortcuts would not lead to a 3/2-approximation.

To circumvent this problem, we would like to have a control over the vertices of Z that take part in the minimum-cost matching phase of the algorithm. Similarly to the work of Zenklusen [121], we calculate a point q that is feasible for the Held-Karp relaxation of the MANY-VISITS PATH TSP, and that is only needed for the analysis of the algorithm. Let $\mathrm{odd}(Z)$ denote the vertices v with an odd degree in Z. We need Z and q to meet the following requirements:

(R1) $c(Z) \leqslant c(X^\star)$,

(R2) $c(q) \leqslant c(X^\star)$, and

(R3) $q/2 \in \mathcal{P}^{\uparrow}_{Q_Z\text{-join}}$, where $Q_Z := \mathrm{odd}(Z) \triangle \{s,t\}$,

where $c(q)$ stands for the cost of the vector q with respect to the cost function c, that is, $c(q) = \sum_{e\in E} c(e)q(e)$.

Adding a shortest Q_Z-join J to the multigraph Z results in a multigraph Z' where every vertex $v \in V - \{s,t\}$ has an even degree at least $2 \cdot r(v)$, and every $v \in \{s,t\}$ has an odd degree at least $2 \cdot r(v) - 1$. Due to (R3), the cost of the shortest Q_Z-join J satisfies $c(J) \leqslant c(q)/2$. Therefore, using Wolsey's analysis for Christofides' algorithm [112], the solution X obtained by taking the edges of Z' and applying shortcuts has cost at most $3/2 \cdot c(X^\star)$.

Let x^* be an optimal solution to the Held-Karp relaxation of the MANY-VISITS PATH TSP:

$$\min \left\{ c^\mathsf{T} x \;\middle|\; x \in \mathcal{P}_{\mathrm{HK}} \right\} \quad . \tag{3.4}$$

In order to obtain Z and q that satisfy the conditions (R1)–(R3) above, we will calculate another solution $y \in \mathcal{P}_{\mathrm{HK}}$ with $c(y) \leqslant c(X^\star)$, and set q to be the midpoint between x^* and y, that is, $q = x^*/2 + y/2$. The construction of Z also depends on y, the details are given in Algorithm 3.3 and the reasoning in the proof of Lemma 3.9. Being a convex combination of two points in $\mathcal{P}_{\mathrm{HK}}$, q is in $\mathcal{P}_{\mathrm{HK}}$ as well. We would like to ensure the existence of a multigraph Z such that $q/2 \in \mathcal{P}^{\uparrow}_{Q_Z\text{-join}}$, therefore we need to construct y accordingly.

Let $Q \subseteq V$ be a set of even cardinality. Recall the definition of $\mathcal{P}^{\uparrow}_{Q\text{-join}}$ at Equation (3.3), which requires that the load on Q-odd cuts is at least 1. Since q is in $\mathcal{P}_{\mathrm{HK}}$, $q(\delta(C))/2 \geqslant 1$ holds for any non-s-t-cuts, i.e. for cuts $C \subset V, C \neq \emptyset$ with $|C \cap \{s,t\}| \in \{0,2\}$. However, for s-t-cuts, the property $y \in \mathcal{P}_{\mathrm{HK}}$ only implies $y(\delta(C)) \geqslant 1$. If in addition $x^*(\delta(C)) \geqslant 3$ holds, then we get $q(\delta(C))/2 \geqslant 1$ regardless of our choice of the multigraph Z. If, however, $x^*(\delta(C)) < 3$ holds, we cannot use the same argument. In that case we need to take care of the constraints of $\mathcal{P}^{\uparrow}_{Q_Z\text{-join}}$ that correspond to s-t-cuts where the x^*-load is strictly less than 3. Let us denote the family of these cuts by $\mathcal{B}(x^*)$, that is,

$$\mathcal{B}(x^*) := \{C \subseteq V \mid s \in C, t \notin C, x^*(\delta(C)) < 3\} \quad .$$

For a family $\mathcal{B} \subseteq \{C \subseteq V \mid s \in C, t \notin C\}$ of s-t-cuts, we say that a point $y \in \mathcal{P}_{\mathrm{HK}}$ is *$\mathcal{B}$-good*, if for every $B \in \mathcal{B}$ we have

(i) either $y(\delta(B)) \geqslant 3$,

(ii) or $y(\delta(B)) = 1$, and y is integral on the edges $\delta(B)$.

Therefore, if $y \in \mathcal{P}_{\mathrm{HK}}$ is $\mathcal{B}(x^*)$-good, then $q = x^*/2 + y/2$ satisfies $q(\delta(C))/2 \geqslant 1$ for every Q_Z-cut C. We will refer to a cut B satisfying condition (i) as a *type (i) cut*, and if it satisfies condition (ii) we will refer to it as a *type (ii) cut*. Note that condition (ii) translates to having a single edge $f \in \delta(B)$ with $y(f) = 1$ and $y(e) = 0$ for all other edges e from $\delta(B)$. The notion of $\mathcal{B}$-goodness was introduced by Zenklusen for the elements of the (single-visit version of the) polytope $\mathcal{P}_{\mathrm{HK}}$ in relation to metric PATH TSP [121].

Lemma 3.8. *The characteristic vector χ_U of any many-visits s-t path U is $\mathcal{B}$-good for any family $\mathcal{B}$ of s-t-cuts.*

Proof. The lemma easily follows from the fact that a many-visits s-t path U crosses any s-t-cut an odd number of times. □

We present our algorithm for the metric MANY-VISITS PATH TSP as Algorithm 3.3.

Algorithm 3.3 A 3/2-approximation algorithm for the metric MANY-VISITS PATH TSP

Input: A complete undirected graph $G = (V, E)$, costs $c : E \to \mathbb{R}_{\geqslant 0}$ satisfying the triangle inequality, requests $r : V \to \mathbb{Z}_{\geqslant 1}$, distinct vertices $s, t \in V$.
Output: An s-t-walk that visits each $v \in V$ exactly $r(v)$ times.

1: Calculate an optimal solution x^* to the Held-Karp relaxation of the MANY-VISITS PATH TSP, i.e. $x^* := \operatorname{argmin}\{c^\mathsf{T} x \mid x \in \mathcal{P}_{\mathrm{HK}}\}$.
2: Determine a $\mathcal{B}(x^*)$-good solution $y \in \mathcal{P}_{\mathrm{HK}}$ minimizing $c^\mathsf{T} y$.
3: Let $B_1 \subset \cdots \subset B_k$ denote the type (ii) cuts with respect to y.
4: Compute a connected multigraph Z on $(V, \mathrm{supp}(y))$ such that
   a: each vertex $v \in V - \{s, t\}$ has degree at least $2 \cdot r(v) - 1$,
   b: each vertex $v \in \{s, t\}$ has degree at least $2 \cdot r(v) - 2$, and
   c: Z contains no parallel edges leaving B_i for $i = 1, \ldots, k$.
5: Compute a minimum-cost matching M with respect to c on the vertices $\mathrm{odd}(Z) \triangle \{s, t\}$.
6: Let Z' denote the many-visits path $Z + M$.
7: Do shortcuts in Z' and obtain an s-t-walk X that visits each city v exactly $r(v)$ times.
return X

In Line 4 of the algorithm, we use Theorem 2.1 to obtain a multigraph with additional properties besides the degree requirements. In the single-visit counterpart of the problem, one can show that even though $x^*(\delta(B)) < 3$ and $y(\delta(B)) = 1$ for type (ii) cuts B, the corresponding point $q/2 = x^*/4 + y/4$ is still in $\mathcal{P}^{\uparrow}_{Q_Z\text{-join}}$. However, due to the possible parallel edges in Z, the parity argument given by Zenklusen [121] does not hold (we will provide a more detailed explanation in the proof of Lemma 3.9), therefore we need to treat this case separately. For this reason we make the following distinction. Let E_y denote the set of edges that correspond to type (ii) cuts in y, that is

$$E_y := \{e \in E \mid \exists B \in \mathcal{B} : \mathrm{supp}(y) \cap \delta(B) = e\} \quad .$$

We let $U(e) := 1$ for all $e \in E_y$, $U(e) := +\infty$ for the rest of the edges of $\mathrm{supp}(y)$, and $U(e) := 0$ for edges $e \in E - \mathrm{supp}(y)$. Moreover, we set $L(e) := 0$ for every edge $e \in E$. Finally we let $\rho(v)$ be defined as $2 \cdot r(v)$ for $v \in V - \{s,t\}$ and as $2 \cdot r(v) - 1$ for $v \in \{s,t\}$. The goal is to find a multigraph Z that satisfies the conditions in Lines 4.a–4.c. According to the claim of Theorem 2.1, we can achieve this by using Algorithm 2.1 to approximate the MINIMUM BOUNDED DEGREE k-COMPONENT MULTIGRAPH WITH EDGE BOUNDS problem, with parameters $k = 1$, and L, U and ρ as defined above. The cost of the resulting multigraph Z is at most $\min\{c^\mathsf{T} x \mid x \in \mathcal{P}_{\mathrm{CG}}(\rho, L, U)\}$, where the polytope $\mathcal{P}_{\mathrm{CG}}(\rho, L, U)$ depends on the instance (G, c, r, s, t) and can be written in the following form:[4]

$$\mathcal{P}_{\mathrm{CG}}(\rho, L, U) := \left\{ x \in \mathbb{R}^E_{\geqslant 0} \;\middle|\; \begin{array}{ll} \mathrm{supp}(x) \text{ is connected} & \\ x(E) = r(V) & \\ x(\dot\delta(v)) \geqslant 2 \cdot r(v) & \forall v \in V - \{s,t\} \\ x(\dot\delta(v)) \geqslant 2 \cdot r(v) - 1 & \forall v \in \{s,t\} \\ 0 \leqslant x(e) \leqslant 1 & \forall e \in E_y \\ 0 \leqslant x(e) \leqslant +\infty & \forall e \in \mathrm{supp}(y) - E_y \\ x(e) = 0 & \forall e \in E - \mathrm{supp}(y) \end{array} \right\}$$

It is not difficult to see that $y \in \mathcal{P}_{\mathrm{CG}}(\rho, L, U)$, and thus

$$c(P) \leqslant \min\left\{ c^\mathsf{T} x \;\middle|\; x \in \mathcal{P}_{\mathrm{CG}}(\rho, L, U) \right\} \leqslant c^\mathsf{T} y \ ,$$

therefore $c(Z) \leqslant c^\mathsf{T} y$ holds; this is one of the reasons behind restricting PZ to $\mathrm{supp}(y)$. Moreover, according to Lemma 3.8, the inequality $c^\mathsf{T} y \leqslant c(X^\star)$ holds, hence the bound $c(Z) \leqslant c(X^\star)$ follows. Now we have all the ingredients to prove the correctness of Algorithm 3.3.

Lemma 3.9. *Algorithm 3.3 provides a feasible solution to the* MANY-VISITS PATH TSP*, with cost at most $3/2$ times the optimal cost.*

Proof. Recall that $Q_Z = \mathrm{odd}(Z) \triangle \{s,t\}$. First we prove that $q = x^*/2 + y/2$ implies that $q/2$ is in $\mathcal{P}^{\uparrow}_{Q_Z\text{-join}}$. For that we need to show that we calculated the solution y in a way that q satisfies $q(\delta(C))/2 \geqslant 1$ for all cuts $C \subset V$ for which $|C \cap \mathrm{odd}(Z) \triangle \{s,t\}|$ is odd.

Clearly, $q \in \mathcal{P}_{\mathrm{HK}}$, as q is the midpoint of two points from $\mathcal{P}_{\mathrm{HK}}$. Therefore, for any Q_Z-cut $C \subseteq V$ that is a not an s-t-cut, we have $q(\delta(C))/2 \geqslant 1$ as needed. Moreover, by definition, for any Q_Z-cut $C \subseteq V$ that is an s-t-cut and is not included in $\mathcal{B}(x^*)$, we have $x^*(\delta(C)) \geqslant 3$, and so

$$\frac{1}{2} q(\delta(C)) = \frac{1}{4}\big(x^*(\delta(C)) + y(\delta(C))\big) \geqslant 1,$$

as $y \in \mathcal{P}_{\mathrm{HK}}$, and thus $y(\delta(C)) \geqslant 1$.

[4] The fact that the constraint "$\mathrm{supp}(x)$ is connected" is a valid polyhedral constraint is proven in §2.3 of Chapter 2.

It remains to consider Q_Z-cuts $C \subseteq V$ that are in $\mathcal{B}(x^*)$. Since y is $\mathcal{B}(x^*)$-good by construction, either $y(\delta(C)) \geqslant 3$, or $y(\delta(C)) = 1$ with y being integral on the edges $\delta(C)$. If $y(\delta(C)) \geqslant 3$, then $q(\delta(C))/2 \geqslant 1$ follows from $x^*(\delta(C)) \geqslant 1$ and the definition of q. If $y(\delta(C)) = 1$ and y is integral on the edges $\delta(C)$, it holds that $y(e) = 0$ for all edges of $\delta(C)$ except for one $f \in \delta(C)$ where $y(f) = 1$. It is at this point where we exploit the restrictions imposed on Z. Since $\operatorname{supp}(Z) \subseteq \operatorname{supp}(y)$ and the load on an edge $e \in E_y$ is at most 1 in Z, the only edge of Z with a positive load on $\delta(C)$ is f and that load is at most 1. Moreover, every cut has at least 1 load in Z, which means $|Z \cap \delta(C)| = 1$. But an s-t-cut $C \subseteq V$ with $|\delta_Z(C)|$ odd cannot be a Q_Z-cut because of the following:

$$\begin{aligned} |C \cap \operatorname{odd}(Z)| &\equiv \sum_{v \in C} |\dot{\delta}_Z(v)| \pmod 2 \\ &= 2 \cdot |\{uv \in Z \mid u, v \in C\}| + |\delta_Z(C)| \ . \end{aligned} \tag{3.5}$$

Equation (3.5) implies that $|C \cap \operatorname{odd}(Z)|$ is odd, and hence the quantity $|C \cap Q_Z| = |C \cap (\operatorname{odd}(Z) \triangle \{s,t\})|$ is even because C is an s-t-cut. By the above, any cut of type (ii) partitions the vertices of $\operatorname{odd}(Z) \triangle \{s,t\}$ into two subsets of even cardinality. This means that no cut constraint of $\mathcal{P}^{\uparrow}_{Q_Z\text{-join}}$ requires a load of 1 for $q/2$ on C, and so $q/2 \in \mathcal{P}^{\uparrow}_{Q_Z\text{-join}}$ holds.

The cost of the matching M can therefore be bounded as follows:

$$c(M) \leqslant c\left(\frac{q}{2}\right) = \frac{1}{4} c^{\mathsf{T}} x^* + \frac{1}{4} c^{\mathsf{T}} y \leqslant \frac{1}{2} c(X^\star),$$

since $c^{\mathsf{T}} x^* \leqslant c(X^\star)$. Thus, the multigraph obtained from $P + M$ has cost at most $3/2 \cdot c(X^\star)$, as claimed.

Note that for a cut C with $y(\delta(C)) = 1$ and y being integral on $\delta(C)$, the term $|T \cap \delta(C)|$ in the proof of Theorem 2.1 of Zenklusen [121] corresponds to the term $|\delta_Z(C)|$ in Equation (3.5). Since the spanning tree T computed on $\operatorname{supp}(y)$ in the algorithm of [121] cannot contain parallel edges, $|T \cap \delta(C)|$ has a value of 1 without enforcing an upper bound on the edge $e \in \delta_T(C)$ for $y(e) = 1$. Because of possible parallel edges in our case, we need to apply an upper bound on the edges of E_y.

Taking shortcuts. According to Theorem 2.1, every vertex $v \in V - \{s,t\}$ has degree at least $2 \cdot r(v) - 1$, while vertices s and t have degrees at least $2 \cdot r(s) - 2$ and $2 \cdot r(t) - 2$ respectively, in the multigraph Z. The matching M provides 1 additional degree for vertices with the wrong parity, therefore Z' will have an even degree at least $r(v)$ for all $v \in V - \{s,t\}$ and an odd degree at least $r(v) - 1$ for $v \in \{s,t\}$. This means that Z' corresponds to a many-visits s-t-path that visits each vertex v at least $r(v)$ times, but possibly more. In Line 7 we proceed with taking shortcuts the way described in Algorithm 3.1, so that X is a feasible solution to the MANY-VISITS PATH TSP instance (G, c, r, s, t). □

Before we show how to calculate a $\mathcal{B}(x^*)$-good point $y \in \mathcal{P}_{\text{HK}}$, let us show that the number of cuts in $\mathcal{B}$ is polynomial in n, and that the set $\mathcal{B}$ can be computed efficiently:

Lemma 3.10. *Let $q \in \mathcal{P}_{\text{HK}}$. Then the family $\mathcal{B}(q)$ of s-t-cuts of q-value strictly less than 3 satisfies $|\mathcal{B}(q)| \leqslant n^4$ and can be computed in $O(mn^4)$ time, where $n := |V|$ and $m := \operatorname{supp}(q)$.*

Proof. Let us define an auxiliary graph $H = (V, E')$ whose edge set E' consists of the edges in $\operatorname{supp}(q)$ and an additional st edge. Let $q_H = q + \chi_{st}$. Clearly, for non-s-t-cuts we have $q_H(\delta_H(C)) = q(\delta(C))$, while for s-t-cuts we have $q_H(\delta_H(C)) = q(\delta(C)) + 1 \geqslant 2$ because of the newly added edge st. Therefore, the family $\mathcal{B}(q)$ can be written as

$$\mathcal{B}(q) = \{C \subset V \mid s \in C, t \notin C, q_H(\delta_H(C)) < 4\} \ .$$

The minimum cut has a load of at least 2, and due to Karger [69] the number of cuts with a load less than k times the minimum cut is at most $O(n^{2k})$. Moreover, using an algorithm by Nagamochi et al. [89], we can enumerate the cuts of size at most k times the minimum cut in time $O(m^2 n + n^{2k} m)$. These results prove that the number of cuts in $|B(q)|$ is $O(n^4)$, and that they can be enumerated in time $O(mn^4)$. □

The dynamic program

Given a family $\mathcal{B}$ of s-t-cuts, our goal is to determine a minimum cost $\mathcal{B}$-good point $y \in \mathcal{P}_{\text{HK}}$. We use the dynamic programming approach introduced by Traub and Vygen [107] and improved upon by Zenklusen [121]. More precisely, the goal of the dynamic program is to determine which cuts in $\mathcal{B}$ are of type (i), and which ones are of type (ii). Our approach is constructive as the dynamic program also determines a point y that is $\mathcal{B}(x^*)$-good.

Consider a $\mathcal{B}(x^*)$-good point y. Let $B_1, \dots, B_k$ denote the type (ii) s-t cuts in $\mathcal{B}$ with respect to y, that is, $y(v_i u_i) = 1$ for exactly one edge $v_i u_i \in \delta(B_i)$ and $y(e) = 0$ for $e \in \delta(B_i) - v_i u_i$. It is not difficult to see that these cuts necessarily form a chain (see e.g. [121]), thus we set the indices such that $B_1 \subsetneq \cdots \subsetneq B_k$. The endpoints of $v_i u_i$ are named such that $v_i \in B_i$, $u_i \notin B_i$. Furthermore, we define $B_0 := \emptyset$, $B_{k+1} := V$, $u_0 := s$ and $v_{k+1} := t$ for notational convenience. Note that u_i and v_{i+1} might coincide for some $i = 0, \dots, k+1$.

The work of Zenklusen [121] argues that the 'first' and 'last' cuts are type (ii) cuts, that is, $B_1 = \{s\}$ and $B_k = V - \{t\}$, because the constraints of $\mathcal{P}_{\text{HK}}$ enforce a degree of 1 on vertices s and t. In the many-visits setting, however, this is not necessarily true, as the instance possibly requires more than one visit for s or t.

Assume for a moment that we know the cuts $B_1, \dots, B_k$ and the edges $v_i u_i$, and we are looking for a $\mathcal{B}(x^*)$-good point $y \in \mathcal{P}_{\text{HK}}$ such that among all cuts in $\mathcal{B}$ the cuts $B_1, \dots, B_k$ are precisely those where (a) y is integral, and (b) $y(\delta(B_i)) = 1$ for all $i = 1, \dots, k$. Then the $\mathcal{B}$-good points $y \in \mathcal{P}_{\text{HK}}$ that satisfy these constraints (a) and (b) have the following properties for all $i = 1, \dots, k$:

(P1) $y(v_i u_i) = 1$ and $y(e) = 0$ for all edges $e \in \delta(B_i) - v_i u_i$,

(P2) the restriction of y to the vertex set $B_{i+1} - B_i$ is a solution to the Held-Karp relaxation for the MANY-VISITS PATH TSP with endpoints u_i and v_{i+1}, with the

additional property that $y(\delta(B)) \geqslant 3$ for every cut $B \in \mathcal{B}$ such that $B_i \cup u_i \subseteq B \subseteq B_{i+1} - v_{i+1}$.

The dynamic program thus aims to find cuts $B_1, \ldots, B_k$ while exploiting the properties (P1) and (P2) above. Formally, it is defined to find a shortest path on an auxiliary directed graph. Let us define the auxiliary directed graph $H = (N, A)$ with *node* set N, *arc* set A, and *length* function $d : A \to \mathbb{R}_{\geqslant 0}$. The node set N is defined by $N = N^+ \cup N^-$, where

$$N^+ = \{(B, u) \in \mathcal{B} \times V \mid u \notin B\} \cup \{(\emptyset, s)\}\,, \text{ and}$$
$$N^- = \{(B, v) \in \mathcal{B} \times V \mid v \in B\} \cup \{(V, t)\}\,.$$

The arc set A is given by $A = A_{\text{HK}} \cup A_E$, where

$$A_{\text{HK}} = \left\{\left((B^+, u), (B^-, v)\right) \in N^+ \times N^- \;\middle|\; B^+ \subseteq B^-, u, v \in B^- - B^+\right\}, \text{ and}$$
$$A_E = \left\{\left((B^-, v), (B^+, u)\right) \in N^- \times N^+ \;\middle|\; B^- = B^+\right\}.$$

Finally, the lengths $d : A \to \mathbb{R}_{\geqslant 0}$ are defined as follows:

$$d(a) = \begin{cases} c(vu) & \text{if } a = ((B, v), (B, u)) \in A_E, \\ \text{OPT}(\text{LP}(a)) & \text{if } a \in A_{\text{HK}}, \end{cases}$$

where $\text{OPT}(\text{LP}(a))$ denotes the optimum value of

$$\begin{array}{lll} & \min & c^{\mathsf{T}} x \\ (\text{LP}(a)) \qquad & \text{subject to} & x \in \mathcal{P}_{\text{HK}}(B^- - B^+, u, v) \\ & & x(\delta(B)) \geqslant 3 \qquad\qquad \text{for all } B \in \mathcal{B} \text{ s.t.} \\ & & \qquad\qquad\qquad\qquad B^+ \subseteq B \subseteq B^-, \\ & & \qquad\qquad\qquad\qquad u \in B, v \notin B, \end{array}$$

where $a = ((B^+, u), (B^-, v))$.

In case $u \neq v$, the polytope $\mathcal{P}_{\text{HK}}(W, u, v)$ is defined as follows:

$$\text{(3.6)}\quad \mathcal{P}_{\text{HK}}(W, u, v) := \left\{ x \in \mathbb{R}^E_{\geqslant 0} \;\middle|\; \begin{array}{ll} x(\delta(C)) \geqslant 2 & \forall C \subset W, C \neq \emptyset, \\ & \quad |C \cap \{u, v\}| \in \{0, 2\} \\ x(\delta(C)) \geqslant 1 & \forall C \subset W, |C \cap \{u, v\}| = 1 \\ x(\dot{\delta}(w)) = 2 \cdot r(w) & \forall w \in W - \{u, v\} \\ x(\dot{\delta}(u)) = 2 \cdot r(u) - 1 & \\ x(\dot{\delta}(v)) = 2 \cdot r(v) - 1 & \\ x(e) = 0 & \forall e \in E - E[W] \end{array} \right\}$$

Observe that unlike in the single-visit case [121], we allow u being equal to s or v being equal to t in Equation (3.6), and the corresponding polytopes $\mathcal{P}_{\text{HK}}(B_1, s, v_1)$ and $\mathcal{P}_{\text{HK}}(V - B_k, u_k, t)$ are feasible.

For y-values across the cuts $B \in \mathcal{B}$ so that $B \notin \{B_1, \dots, B_k\}$, we require that $y(\delta(B)) \geqslant 3$ holds. We ensure this by defining modified Held-Karp relaxations of the MANY-VISITS PATH TSP instances between cuts B_i and B_{i+1} for every $i = 0, \dots, k$. More precisely, such an instance is defined on the subgraph of G induced on the vertex set $B_{i+1} - B_i$ with distinguished vertices u_i and v_{i+1}, with the additional property that it has a y-load of at least 3 on each cut $B \in \mathcal{B}$ with $B_i \subset B \subset B_{i+1}$, as shown in (LP($a$)).

Let us now cover the case when for some index $i \in \{0, \dots, k\}$, vertices u_i and v_{i+1} coincide. [5] In the single-visit PATH TSP, the solution is defined to be the all-zero vector if $u_i = v_{i+1}$ is the only vertex in $B_{i+1} - B_i$, and there exists no solution otherwise. However, since we allow for a vertex to be visited more than once (i.e. have a degree more than 2) in a solution to the Held-Karp relaxation for the MANY-VISITS PATH TSP, we use a different extension in our approach. We define the corresponding subproblem as the Held-Karp relaxation for the MANY-VISITS TSP. First assume that $u_i \neq s$ and $u_i \neq t$. Since $y(v_i u_i) = 1$ and $y(u_i u_{i+1}) = 1$ by construction, the degree requirement for u_i in the MANY-VISITS TSP subproblem is *two* less that in $\mathcal{P}_{\text{HK}}$, namely $r(u_i) - 2$. If $u_i = s$ (or $u_i = t$), then due to $y(s\, u_1) = 1$ (or $y(v_k\, t) = 1$) the degree requirement for u_i in the subproblem is *one* less that in $\mathcal{P}_{\text{HK}}$, which also equals $r(u_i) - 2$. Note that if $u_i = v_{i+1}$ there is no cut $B \in \mathcal{B}$ with $u_i \in B$ and $v_{i+1} \notin B$, thus the linear program LP(a) has the form $\min\{c^{\mathsf{T}} x \mid x \in \mathcal{P}_{\text{HK}}(W, u, u)\}$, where:

$$\mathcal{P}_{\text{HK}}(W, u, u) := \left\{ x \in \mathbb{R}^E_{\geqslant 0} \;\middle|\; \begin{array}{ll} x(\delta(C)) \geqslant 2 & \forall C \subset W, C \neq \emptyset, \\ x(\dot{\delta}(w)) = 2 \cdot r(w) & \forall w \in W - u \\ x(\dot{\delta}(u)) = 2 \cdot r(u) - 2 & \\ x(e) = 0 & \forall e \in E - E[W] \end{array} \right\} . \tag{3.7}$$

If the request for u is $r(u) = 1$ and $|W| > 1$, the polytope $\mathcal{P}_{\text{HK}}(W, u, u)$ is empty, and thus the linear program LP(a) has no solution. In this case the cost of the arc a is defined to be infinity. Note however that if $r(u) = 1$ and $W = \{u\}$, the corresponding linear program has a nonzero solution, namely a vector that has value $r(u) - 1$ in the coordinate of the self-loop uu, and 0 otherwise.

To find a $\mathcal{B}$-good point with minimum cost $c^{\mathsf{T}} y$, we compute a shortest $(\emptyset, s)$–(V, t) path with respect to d in H; due to Lemmas 3.8 and 3.11 this path has finite length. Let $(\emptyset, s), (B_1, v_1), (B_1, u_1), (B_2, v_2), \dots, (B_k, u_k), (V, t)$ be the nodes on this shortest path, and similarly as before, define $B_0 := \emptyset, u_0 := s$ and $B_{k+1} := V, v_{k+1} := t$. By construction of H, we have $B_0 \subset B_1 \subset \cdots \subset B_{k+1}$. Let $x^i \in \mathbb{R}^E$ be an optimal solution to

[5] Note that the corresponding arc in H will have the form $\big((B^+, w), (B^-, w)\big) \in A_{\text{HK}}$.

$\mathrm{LP}(a)$ for $a = ((B_i, u_i), (B_{i+1}, v_{i+1}))$. Set

$$y := \sum_{i=0}^{k} x^i + \sum_{i=1}^{k} \chi_{v_i u_i}. \tag{3.8}$$

By the definition of the lengths d in H, $c^\mathsf{T} y$ necessarily equals the length ℓ^* of a shortest $(\emptyset, s)$–(V, t) path in H with respect to d. We now show that y computed in Equation (3.8) is indeed a $\mathcal{B}(x^*)$-good point of minimum cost.

Lemma 3.11. *The length ℓ^* of a shortest $(\emptyset, s)$–(V, t) path in H with respect to d satisfies $\ell^* \leqslant \min\{c^\mathsf{T} z \mid z \in \mathcal{P}_{\mathrm{HK}}, z \text{ is } \mathcal{B}\text{-good}\}$.*

Proof. Let $\mathcal{B}_z \subseteq \mathcal{B}$ be the family of cuts $B \in \mathcal{B}$ such that $z(f) = 1$ for precisely one edge $f \in \delta(B)$, and $z(e) = 0$ for all other edges $e \in \delta(B) - f$. These are the sets in $\mathcal{B}$ that are type (ii) cuts with respect to z, and also $\mathcal{B}_z$ forms a chain: $B_1 \subset \cdots \subset B_k$ holds, where $B_i \in \mathcal{B}_z$ for $i = 1, \ldots, k$. The cuts $\{B_1, \ldots, B_k\}$ defines a partition of V into sets $B'_0 := B_1$, $B'_1 := B_2 - B_1$, $\ldots$, $B'_{k-1} := B_k - B_{k-1}$, $B'_k := V - B_k$. For $i \in \{1, \ldots, k\}$, let $v_i u_i$ be the unique edge in $\delta(B_i)$ where $z(v_i u_i) = 1$, so that $v_i \in B_i$ and $u_i \notin B_i$.

Consider the path along nodes (B_0, u_0), (B_1, v_1), (B_1, u_1), $\ldots$, (B_{k+1}, v_{k+1}). It suffices to show that the length ℓ of the path is at most $c^\mathsf{T} z$. For each $i \in \{0, \ldots, k\}$, the vector $z^i \in \mathbb{R}^E$ is defined to be the restriction of z to $E[B_{i+1} - B_i]$. Assume for a moment that z^i is a feasible solution of $\mathrm{LP}(a)$ with $a = ((B_i, u_i), (B_{i+1}, v_{i+1}))$. Then the total length ℓ is equal to $\sum_{i=0}^{k} c^\mathsf{T} x^i + \sum_{i=1}^{k} c(v_i u_i)$ by definition, which is at most $\sum_{i=0}^{k} c^\mathsf{T} z^i + \sum_{i=1}^{k} c(v_i u_i) = c^\mathsf{T} z$. Since ℓ^* is minimum among all possible ℓ's, we get $\ell^* \leqslant \ell \leqslant c^\mathsf{T} z$.

Since z is $\mathcal{B}$-good, and $z^i(\delta(B)) = z(\delta(B))$ for any cut B with $B_i \subsetneq B \subsetneq B_{i+1}$, that means $z^i(\delta(B)) = z(\delta(B)) \geqslant 3$. It remains to show that $z^i \in \mathcal{P}_{\mathrm{HK}}(B'_i, u_i, v_{i+1})$ or $z^i \in \mathcal{P}_{\mathrm{HK}}(B'_i, u_i, u_i)$ follows for $i = 0, \ldots, k$.

Distinct endpoints. Let us start with the case when $u_i \neq v_{i+1}$. By definition, $z^i(e) = z(e)$ if both endpoints of e are in B'_i and $z^i(e) = 0$ otherwise. Hence, for vertices $w \in B'_i$ such that $w \notin \{u_i, v_{i+1}\}$, $z^i(\dot\delta(w)) = z(\dot\delta(w)) = 2 \cdot r(w)$. First assume that $u_i \neq s$ and $v_{i+1} \neq t$. Both of the endpoints u_i and v_{i+1} have a total z-value of 1 on the edge set $\delta(B'_i)$ due to $z(v_i u_i) = 1$ and $z(v_{i+1} u_{i+1}) = 1$, therefore the z^i-value on the edges incident to the endpoints is $z^i(\dot\delta(u_i)) = z(\dot\delta(u_i)) - 1 = 2 \cdot r(u_i) - 1$ and $z^i(\dot\delta(v_{i+1})) = z(\dot\delta(v_{i+1})) - 1 = 2 \cdot r(v_{i+1}) - 1$. If $u_i = s$ or $v_{i+1} = t$, their z^i-values equal to their z-values, that is $2 \cdot r(s) - 1$ or $2 \cdot r(t) - 1$, respectively. The degree constraints are therefore satisfied.

Finally, we have to show that for a cut $C \subseteq B'_i$, $z^i(\delta(C)) \geqslant 1$ holds if C is a u_i-v_{i+1}-cut, and $z^i(\delta(C)) \geqslant 2$ if C does not separate u_i and v_{i+1}. For the single-visit variant, the proof goes by showing that z^i is in the spanning tree polytope of $G[B'_i]$, using the fact that z is in the spanning tree polytope of G and the degree constraints of $v \in B'_i$ [121]. However, these terms do not immediately generalise to the many-visits setting, so we show that the connectivity of z^i follows from the properties of z.

Non-u_i-v_{i+1}-cuts: First consider cuts that do not separate u_i and v_{i+1}; we prove that the total z^i-value across these cuts is at least 2. We may assume that $C \subset B'_i$ contains

neither u_i nor v_{i+1} in this paragraph. In case $u_i, v_{i+1} \in C$, we can take $B'_i - C$ and we are done, as $z^i(\delta(C)) = z^i(\delta(B'_i - C)) = z(\delta(C)) \geqslant 2$, yielding $z^i(\delta(C)) \geqslant 2$. Assume first that $u_i \neq s$ and $v_{i+1} \neq t$, and let $C \subset B'_i$. Then $z^i(\delta(C)) \geqslant 2$ simply because C is a non-s-t-cut and thus $z(\delta(C)) \geqslant 2$. Now let u_i be equal to s, and let $C \subset B'_i$ be a cut that does not contain either s or v_1. Likewise, $z^0(\delta(C)) \geqslant 2$ because C is a non-s-t-cut and thus $z(\delta(C)) \geqslant 2$. The argument for $v_{i+1} = t$ goes the same way.

u_i-v_{i+1}-cuts: Here we have to prove that if C is a u_i-v_{i+1}-cut, then $z^i(\delta(C)) \geqslant 1$. Assume that $u_i \neq s$ and $v_{i+1} \neq t$. Without the loss of generality, we may assume that $u_i \in C$. Since C is a non-s-t-cut, $z(\delta(C)) \geqslant 2$. If we account for $z(v_iu_i) = 1$, then $z^i(\delta(C)) \geqslant 1$ follows. One can similarly prove the claim if $u_i \notin C$ and $v_{i+1} \in C$. Assume now that $u_i = s$ and $u_i \in C$; then C is an s-t-cut, so $z(\delta(C)) \geqslant 1$. Moreover, $v_{i+1} \notin C$, so $v_{i+1}u_{i+1} \notin \delta(C)$, therefore $z^i(\delta(C)) = z(\delta(C)) \geqslant 1$. If $u_i = s$ and $v_{i+1} \in C$, then $z^i(\delta(C)) \geqslant z(\delta(C)) - z(v_{i+1}u_{i+1}) \geqslant 1$, since C is a non-s-t-cut and thus $z(\delta(C)) \geqslant 2$. The case $v_{i+1} = t$ can be proved similarly.

Same endpoints. Now we cover the case when $u_i = v_{i+1}$. We need to prove that $z^i \in \mathcal{P}_{\text{HK}}(B'_i, u_i, u_i)$, as defined in Equation (3.7). The argument about the degrees is analogous to the case above, the fact that $z^i(\dot{\delta}(w)) = 2 \cdot r(w)$ directly follows for vertices $w \in B'_i - u_i$. Moreover, if $u_i \notin \{s, t\}$, the endpoint u_i has a z-load of 2 on $\delta(B'_i)$ because of $z(v_iu_i) = 1$ and $z(v_{i+1}u_{i+1}) = z(u_iu_{i+1}) = 1$, hence $z^i(\dot{\delta}(u_i)) = z(\dot{\delta}(u_i)) - 2 = 2 \cdot r(u_i) - 2$, as desired. If $u_i \in \{s, t\}$, then $z(\dot{\delta}(u_i)) = 2 \cdot r(u_i) - 1$. Moreover, $z(s\,u_1) = 1$ or $z(v_k\,t) = 1$ if $u_i = s$ or $u_i = t$, respectively. In both cases this means $z^i(\dot{\delta}(u_i)) = z(\dot{\delta}(u_i)) - 1 = 2 \cdot r(u_i) - 2$.

Now turn to the cut constraints, and let $C \subset B'_i$ be a cut. We can assume that $u_i \notin C$, otherwise we take $B'_i - C$, and we are done. If $u_i \notin C$, then $z^i(\delta(C)) = z(\delta(C)) \geqslant 2$ because C is a non-s-t-cut. The proof is complete. □

Lemma 3.12. $y \in \mathcal{P}_{\text{HK}}$.

Proof. To prove this claim, we use the properties of x^i. We again distinguish two cases, based on whether the endpoints of subproblem $\text{LP}(a)$ are the same or different.

Degree constraints. First, consider the indices i with $u_i \neq v_{i+1}$. By definition, the vector x^i satisfies $x^i \in \mathcal{P}_{\text{HK}}(B_{i+1} - B_i, u_i, v_{i+1})$, meaning that it is a solution to the Held-Karp relaxation for MANY-VISITS PATH TSP in the induced subgraph $G[B_{i+1} - B_i]$ with endpoints u_i and v_{i+1}. Recall that $y = \sum_{i=0}^{k} x^i + \sum_{i=1}^{k} \chi_{v_iu_i}$ by definition. For a $v \in B_{i+1} - B_i$, the value $y(\dot{\delta}(v))$ is equal to

1. $x^i(\dot{\delta}(v))$ if $v = u_i = s$ or $v = v_{i+1} = t$,
2. $x^i(\dot{\delta}(v)) + 1 = 2 \cdot r(v)$ if $v \notin \{s, t\}$ and $v = u_i$ or $v = v_{i+1}$ for some i, due to the edge v_iu_i or $v_{i+1}u_{i+1}$, respectively; and
3. $x^i(\dot{\delta}(v)) = 2 \cdot r(v)$ otherwise.

By the above, $y(\dot{\delta}(s)) = 2 \cdot r(s) - 1$, $y(\dot{\delta}(t)) = 2 \cdot r(t) - 1$, and $y(\dot{\delta}(v)) = 2 \cdot r(t)$ for $v \notin \{s, t\}$, therefore the degree constraints are satisfied for all $v \in V$.

If $u_i = v_{i+1}$, the value $y(\dot{\delta}(v))$ is equal to

1. $x^i(\dot{\delta}(v)) + 1$ if $v = s$ or $v = t$, because of the edge vu_1 or $v_k v$, respectively,
2. $x^i(\dot{\delta}(v)) + 2 = 2 \cdot r(v)$ if $v \notin \{s, t\}$ and $v = u_i = v_{i+1}$ for some i, due to the edge $v_i u_i$ and $v_{i+1}u_{i+1}$; and
3. $x^i(\dot{\delta}(v) = 2 \cdot r(v)$ otherwise.

Again, we get $y(\dot{\delta}(s)) = 2 \cdot r(s) - 1$, $y(\dot{\delta}(t)) = 2 \cdot r(t) - 1$, and $y(\dot{\delta}(v)) = 2 \cdot r(t)$ for $v \notin \{s, t\}$, therefore the degree constraints are satisfied for all $v \in V$.

Cut constraints. It remains to show that y satisfies the cut constraints. As in the proof of Lemma 3.11, instead of building on a spanning subgraph polytope, we directly prove that the cut constraints hold. As before, the cuts $\{B_1, \dots, B_k\}$ define a partition of V into sets $B'_0 := B_1$, $B'_1 := B_2 - B_1, \dots, B'_{k-1} := B_k - B_{k-1}$, $B'_k := V - B_k$.

Let us first consider the value of y on s-t-cuts. For $B_i \in \{B_1, \dots, B_k\}$ the y-load on $\delta(B_i)$ equals 1 due to the edge $v_i u_i$, therefore it satisfies the constraint $y(\delta(B_i)) \geqslant 1$.

If C is a s-t-cut such that $C \notin \{B_1, \dots, B_k\}$, then there is at least one index $i \in \{0, \dots, k\}$, such that both $B'_i \cap C$ and $B'_i - C$ are not empty. In other words, there is at least one vertex from B'_i on both sides of the cut C.

If $u_i \neq v_{i+1}$, x^i satisfies the constraints of LP for $a = ((B_i, u_i), (B_{i+1}, v_{i+1}))$, we have $x^i \in \mathcal{P}_{\mathrm{HK}}(B'_i, u_i, v_{i+1})$. That means x^i has a load of at least 1 on edges leaving every proper subset of B'_i, including $B'_i \cap C$, and $x^i(\delta(B'_i \cap C)) \geqslant 1$ implies $y(\delta(B'_i \cap C)) \geqslant 1$, which yields $y(\delta(C)) \geqslant 1$.

If $u_i = v_{i+1}$, then $x^i \in \mathcal{P}_{\mathrm{HK}}(B'_i, u_i, u_i)$, which means that $y(\delta(B'_i - C, B'_i \cap C)) \geqslant 2$, so $y(\delta(C)) \geqslant 1$ follows.

If C is a non-s-t-cut, we distinguish the following three cases. Note that in neither of the cases is $B'_0 \subseteq C$ or $B'_k \subseteq C$ a possibility, as that would make C an s-t-cut.

- If $C \subsetneq B'_i$ for some i, and C is a u_i-v_{i+1}-cut so that $u_i \in C$ (or $v_{i+1} \in C$), then $y(\delta(C))$ has a load of at least 1 from the fact that $x^i \in \mathcal{P}_{\mathrm{HK}}(B'_i, u_i, v_{i+1})$, and 1 load from the edge $v_{i-1}u_i$ (or $v_i u_{i+1}$). If C is *not* a u_i-v_{i+1}-cut, and u_i, v_{i+1} are in C, then $y(\delta(C)) \geqslant 2$ because of the edges $v_i u_i$ and $v_{i+1}u_{i+1}$; while if u_i, v_{i+1} are not in C then $y(\delta(C)) \geqslant 2$ follows from $x^i \in \mathcal{P}_{\mathrm{HK}}(B'_i, u_i, v_{i+1})$. Note that if $u_i = v_{i+1}$, C can only be a non-u_i-v_{i+1}-cut. In that case $x^i(\delta(C)) \geqslant 2$ because $x^i \in \mathcal{P}_{\mathrm{HK}}(B'_i, u_i, u_i)$, and thus $y(\delta(C)) \geqslant 2$ follows.

- If $C = \bigcup_{i \in \mathcal{I}} B'_i$ for some $\mathcal{I} \subset \{0, \dots, k\}$, let us define $i_{\min} := \min\{i \mid B'_i \subset C\}$ and $i_{\max} := \max\{i \mid B'_i \subset C\}$. Then $y(\delta(C)) \geqslant y(v_{i_{\min}-1}u_{i_{\min}}) + y(v_{i_{\max}}u_{i_{\max}+1}) = 2$.

- Else there exists a set B'_i such that $C \cap B'_i \neq \emptyset$, and $C \not\subseteq B'_i$ and $B'_i \not\subseteq C$ hold, then let us define $i_{\min}$ and $i_{\max}$ as follows:

$$i_{\min} := \min\{i \mid C \cap B'_i \neq \emptyset,\ C \not\subseteq B'_i,\ B'_i \not\subseteq C\},$$
$$i_{\max} := \max\{i \mid C \cap B'_i \neq \emptyset,\ C \not\subseteq B'_i,\ B'_i \not\subseteq C\} \ .$$

Suppose that $i_{\min} \neq i_{\max}$. Then, $\delta(C)$ has at least 1 y-load on $\delta(B'_{i_{\min}} \cap C, B'_{i_{\min}} - C)$, as well as at least 1 y-load between on $\delta(B'_{i_{\max}} \cap C, B'_{i_{\max}} - C)$, thus $y(\delta(C)) \geqslant 2$. In case of $i_{\min} = i_{\max}$, there must exist another index $i \neq i_{\min}$ such that $B'_i \subset C$ (as otherwise we are back in one of the previous two cases), in which case there is at least one edge e in $\delta(C \cap B'_i)$ such that $y(e) = 1$ (either $v_i u_i$ or $v_{i+1}u_{i+1}$ or both); in total $y(\delta(C)) \geqslant 2$ holds.

This concludes the proof of Lemma 3.12. □

Lemma 3.13. *y is $\mathcal{B}$-good.*

Proof. The proof follows the lines of the corresponding proof of Lemma 3.3 of Zenklusen [121]: there the claim can be deduced from cut constraints of P_{HK}, while in our case it follows from those of the polytope $\mathcal{P}_{\text{HK}}$. Nevertheless, we include the full proof here for the sake of completeness.

For $i \in \{1, \dots, k\}$, we have by construction of y that $y(v_i u_i) = 1$ and $y(e) = 0$ for other edges $e \in \delta(B_i)$. This means that all cuts B_i satisfy (ii) of the definition of $\mathcal{B}$-goodness, i.e. the y-value is 1 and y is integral. Let us show that for any other cut $B \in \mathcal{B} - \{B_1, \dots, B_k\}$, the y-load satisfies (i) of the definition.

First suppose that $\{B_1, \dots, B_k\} \cup B$ is not a chain, in this case there is some index $j \in \{0, \dots, k\}$, such that neither $B \subseteq B_j$ nor $B_j \subseteq B$ is true. Hence

$$\begin{aligned} y(\delta(B)) + 1 &= y(\delta(B)) + y(\delta(B_j)) \\ &\geqslant y(\delta(B - B_j)) + y(\delta(B_j - B)) \\ &\geqslant 4 \ . \end{aligned}$$

The first line follows from $y(\delta(B_j)) = 1$, this was shown at the beginning of the proof. The first inequality holds by the cut functions $C \to y(\delta(C))$ being symmetric and submodular. Since B and B_j are s-t-cuts, $B - B_j$ and $B_j - B$ are non-s-t-cuts, and the y-load of both of these cuts is at least 2, hence the second inequality follows.

Suppose that $\{B_1, \dots, B_k\} \cup B$ is a chain, then there is an index j such that $B_j \subsetneq B \subsetneq B_{j+1}$. If $u_j \in B$ and $v_{j+1} \notin B$, then $x^j(\delta(B)) \geqslant 3$ because of the constraints of the corresponding linear program $\text{LP}(a)$, where $a = ((B_j, u_j), (B_{j+1}, v_{j+1}))$. Since $y \geqslant x^j$ holds for all j component-wise, $y(\delta(B)) \geqslant 3$ follows. If $u_j \notin B$ and $v_{j+1} \in B$, then both the edges $v_j u_j$ and $v_{j+1}u_{j+1}$ are in $\delta(B)$; moreover $x^j(\delta(B)) \geqslant 1$ since B is a u_j-v_{j+1}-cut, therefore

$$y(\delta(B)) \geqslant x^j(\delta(B)) + y(v_j u_j) + y(v_{j+1}u_{j+1}) \geqslant 3 \ .$$

Finally, if B is not an u_j-v_{j+1}-cut, $x^j(\delta(B)) \geqslant 2$ since $x^j \in \mathcal{P}_{\text{HK}}(B_{j+1} - B_j, u_j, v_{j+1})$. Moreover, either $u_i v_i$ or $u_{i+1}v_{i+1}$ is an edge in $\delta(B)$, depending on whether u_i and v_{j+1} are in B or not; both of the possibilities imply $y(\delta(B)) \geqslant 3$. □

Lemma 3.14. *Let $\mathcal{B} \subseteq \{C \subseteq V \mid s \in C, t \notin C\}$. One can determine in time polynomial in $|\mathcal{B}|$ and the input size of (G, s, t, c) a $\mathcal{B}$-good point $y \in \mathcal{P}_{\text{HK}}$ of minimum cost.*

Proof. The number of nodes and arcs in H are polynomial in $|\mathcal{B}|$. Calculating a shortest path on H takes time polynomial in $|H|$. The feasibility of the linear programs (LP(a)) can be checked in time $\mathrm{poly}(n, \log r, |\mathcal{B}|)$, therefore an optimal solution can also be found, using the ellipsoid method, in time polynomial in n, $\log r$ and $|\mathcal{B}|$ (see the discussion in § 58.5 of Schrijver's book [100]). □

Finally, we can prove our main theorem:

Theorem 3.2. *There is a polynomial-time $3/2$-approximation for the metric* Many-Visits Path TSP. *The algorithm runs in time polynomial in n and $\log r(V)$.*

Proof. The correctness of Algorithm 3.3 follows from Lemma 3.9. Now we turn to the complexity analysis.

The constraints of the Held-Karp relaxation (Equation (3.4)) of the Many-Visits Path TSP can be tested in time polynomial in n and $\log r(V)$, hence calculating x^* takes a $\mathrm{poly}(n, \log r(V))$ time as well [100, § 58.5]. This means Line 1 takes time polynomial in n and $\log r(V)$. According to Lemma 3.14, Line 2 also has a polynomial-time complexity, and from Lemma 3.10 it follows that $\mathcal{B}(x^*)$ contains a polynomial number of cuts, therefore selecting the type (ii) cuts can be done in polynomial time as well. By Theorem 2.1, Line 4 takes polynomial time, while calculating a matching in Line 5 and the graph operations in Line 6 can be done efficiently as well. Finally, since the number of edges in Z is $r(V)$ and the matching M contributes at most $n/2$ edges, we remove at most $n/2$ edges from Z' to obtain our solution X. This means that the number of operations performed in Line 7 can be bounded by $O(n)$. The claimed time complexity follows. □

Remark 4. *It is worth considering how Algorithm 3.3 proceeds when applied to the single-visit TSP, that is, when $r(v) = 1$ for each $v \in V$. The output of Algorithm 2.1 in Line 4 is then a connected multigraph with $r(V) - 1 = n - 1$ edges. This means that each vertex v has degree at least $2 \cdot r(v) - 1 = 1$, which boils down to a connected graph with $n - 1$ edges, therefore a spanning tree on G with the additional properties (R1)–(R3). Thus Algorithm 3.3 performs the same operations as does the algorithm of Zenklusen [121] for the* Path TSP.

3.4 Strongly polynomial-time implementation

The transportation problem in Algorithm 3.1 can be solved in strongly polynomial time [74, 92]. Computing compact path-cycle representations uses the algorithm of Grigoriev and van de Klundert [50]; which, along with computing the Eulerian trail and making shortcuts, can be done in strongly polynomial time. Moreover, the algorithm uses the $3/2$-approximation for the Path TSP by Zenklusen [121] as a black-box, which can also be implemented in strongly polynomial time. Making shortcuts in Algorithm 3.1 can be performed in strongly polynomial time as well, thus we can find a $5/2$-approximation for the metric Many-Visits Path TSP in strongly polynomial time.

Algorithm 3.3 involves solving three types of LPs. According to § 58.5 in Schrijver's book [100], if the feasibility of a linear program for a vector x can be tested in polynomial time, then the ellipsoid method can find a solution in strongly polynomial time.

In Line 1, we calculate an optimal solution to $\mathcal{P}_{\text{HK}}$ as defined in Equation (3.2), while a number of linear programs of form $\text{LP}(a)$ arise throughout the dynamic program in Line 2. The feasibility of the cut constraints can be checked in strongly polynomial time, by solving a minimum cut problem. The number of degree constraints in both types of these LPs and the number of constraints $x(\delta(B)) \geqslant 3$ in (LP(a)) is polynomial in n. Finally, in (LP), the number of constraints involving hyperedges is polynomial in n, and one can check the feasibility of the constraints involving the border functions using submodular minimisation [62]. This means all of the linear programs arising in Algorithm 3.3 can be solved in strongly polynomial time.

According to Lemma 3.10, the number of cuts in $\mathcal{B}$ is polynomial in n, hence Lines 1, 2 and 4 can be performed in strongly polynomial time. This is true for computing a matching in Line 5, as well as all the remaining graph operations, using the same arguments as in case of Algorithm 3.1. Hence, we provided a $3/2$-approximation for the metric MANY-VISITS PATH TSP in strongly polynomial time.

Summary

In this chapter we gave an approximation algorithm for a far-reaching generalisation of the metric PATH TSP, the metric MANY-VISITS PATH TSP where each city v has a (potentially exponentially large) request $r(v) \geqslant 1$. Our algorithm yields a $3/2$-approximation for the metric MANY-VISITS PATH TSP in time polynomial in the number n of cities and the logarithm of the requests $r(v)$. It therefore generalises the recent fundamental result by Zenklusen [121], who obtained a $3/2$-approximation for the metric PATH TSP, finishing a long history of research.

At the heart of our algorithm is a polynomial-time approximation algorithm for the BOUNDED DEGREE G-POLYMATROID ELEMENT WITH MULTIPLICITIES problem. That algorithm yields a solution with cost at most the optimum, which violates the lower bounds only by a constant factor depending on the weighted maximum element frequency Δ. We use this algorithm to calculate a connected multigraph with certain properties, analogous to the constrained spanning tree calculated by Zenklusen for the single-visit Path TSP [121].

We also showed a simple approach, that gives a $5/2$-approximation for the metric MANY-VISITS TSP in strongly polynomial time, and an $O(1)$-approximation for the metric asymmetric MANY-VISITS TSP in polynomial time.

CHAPTER 4

APPROXIMATION ALGORITHMS FOR MULTIPLE-AGENT MVTSP

Introduction

As mentioned in the Chapter Introduction, the MANY-VISITS TSP and the MANY-VISITS PATH TSP can be used to model scheduling problems. One such example is the high-multiplicity job scheduling problem $1|HM, s(ij), p(j)|\sum C_j$, where each job of class j has a processing time $p(j)$ and there is a setup time $s(ij)$ involved when switching from a job of type i to a type j job. The MVTSP corresponds to the periodic variant of this scheduling problem, while the path version of the MVTSP corresponds to the non-periodic variant. This means Theorem 3.3 from Chapter 3 thus provides a $3/2$-approximation for the aforementioned scheduling problems as well, provided that the parameter $c(ij) := s(ij) + p(j)$ satisfies the triangle inequality. Our work aims for generalising the techniques in Chapter 3 in order to provide approximations to scheduling problems on parallel machines. We achieve this by generalising the MANY-VISITS TSP (i.e. the tour version), by allowing for multiple salesmen (or *agents*), we call this problem the MULTIPLE-AGENT MANY-VISITS TSP.

However, giving the precise definition of the MULTIPLE-AGENT MANY-VISITS TSP is not straightforward; one can consider different, but equally valid descriptions when introducing multiple salesmen to the MANY-VISITS TSP. Firstly, there is the choice of the objective function: one possibility is to minimise the *total* cost of the tours, and another is to minimise the cost of the *longest* tour. These objectives translate to minimising the total processing time ($\sum C_j$) and makespan ($C_{\max}$), respectively, in the 'language' of scheduling theory. Observe that in the case of the MANY-VISITS TSP, where there is only one agent, the two objectives are equivalent. One can also impose different constraints on the tours sought in the solution, this includes requiring the tours of different agents to be disjoint or forbidding empty tours, i.e. agents not visiting any city. Finally, one can consider a special city for each salesman called its *depot*, and each tour in the solution must contain exactly one depot; alternatively there are no depots and each agent simply traverses a subset of the cities.

The four aspects mentioned above are independent from each other, therefore all $2^4 = 16$ possible generalisations can be considered. We decided to focus on problem variants with minimising the total cost of the tours, $\sum_i \text{cost}(X_i)$, as their objective. This

still leaves 8 problem variants to consider. In this chapter we first examine these variants, and show which ones are equivalent and which ones are different, with respect to their optimal solution values. Then we provide polynomial-time constant-factor approximation algorithms for all problem variants.

The results of this chapter are joint work with Kristóf Bérczi and Matthias Mnich, and they first appeared on arXiv [16].

Previous work

The generalisation of the (single-visit) TSP to multiple salesmen or agents is commonly referred to as multiple TSP or MTSP. The first constant-factor approximation algorithm for the MTSP is due to Frieze [43], who considered k salesmen starting from a single vertex called *depot*. They extended the Christofides-Serdyukov algorithm for the TSP [27, 103], and provided a $3/2$-approximation.

A more diverse set of works considered a modification of the MTSP, where the salesmen start from different depots. The Multiple-Depot TSP (also referred to as MDMTSP in the literature) seeks k cycles that together cover a set of cities, and have exactly one depot in each cycle. Rathinam et al. [96] provided a 2-approximation using a tree-doubling approach, then Xu et al. [117] showed that a Christofides-like heuristic yields a $(2 - 1/k)$-approximation. Xu et al. first construct a minimum cost constrained spanning forest, where each tree contains exactly one depot, then complete it with a minimum cost matching on the odd degree vertices. They also argue that the cost of the matching cannot be bounded by $\mathsf{OPT}/2$, as in the case of TSP. The reason is that the edges of the matching might go between vertices, that are in different components of the optimal solution. The $3/2$-approximation algorithm by Xu and Rodrigues [114] circumvents this problem by exchanging the edges of the constrained spanning forest, although this edge-exchanging procedure leads to a time complexity of $n^{\Omega(k)}$. It is worth noting that the results so far allow isolated depots in the solution, i.e. a salesman that visits no cities.

In the Generalised Multiple-Depot TSP (referred to as GMDMTSP), more than k depots are available, but only at most k can be selected. The first constant-factor polynomial-time algorithms are 2-approximations by Malik et al. [86] and by Carnes and Schmoys [22]. Later, Xu and Rodrigues [115] provided a $(2 - 1/(2k))$-approximation. The algorithm of Xu and Rodrigues [114] can also be used to obtain a $3/2$-approximation to the Generalised Multiple-Depot TSP in $n^{\Omega(k)}$ time.

Khachay and Neznakhina [70] considered the relevant *k-size cycle cover* problem, that seeks k cycles covering the set of cities with minimum total cost. The authors show a 2-approximation to this problem when the edge costs are asymmetric, satisfy the triangle inequality, and the cost of each loop is zero.

4.1 Problem descriptions

The MULTIPLE-AGENT MANY-VISITS TSP is defined as follows. Given is a complete graph $G = (V, E)$, with nonnegative costs $c(uv)$ for every pair of vertices u, v and a

positive request $r(v)$ for each vertex v. Moreover, an integer k is given, that denotes the (maximum) number of agents, and

a) *there is a set of depots $D = \{d_1, \dots, d_k\}$ / there are no depots;*

the corresponding problems are denoted with a MD / MA prefix, respectively.

The goal is to find a multigraph X, such that

b) *X can be decomposed into at most / exactly k non-empty many-visits TSP tours;*

where *non-empty* means containing at least one vertex for the MA-setting, or a depot and at least one non-depot for the MD-setting.

Finally, the MVTSP tours are edge-disjoint[1], and

c) *vertex-disjoint / not necessarily vertex-disjoint.*

Let us denote the many-visits TSP tour of agent j by X_j, for $j = 1, \dots, k$. Under point b), we denote the variant with the possibly empty tours by the symbol $\emptyset$. Empty tours in problems without depots mean the tour of one or more agents being the empty graph, whereas in problems with depots an empty tour is assigned to agent i if $V(X_i) = \{d_i\}$ and $E(X_i) = \emptyset$. We call the component containing only a depot a *trivial component* of X.

Table 4.1 contains an overview of the problems considered.

		arbitrary tours	**disjoint tours**
$\sum_i \mathsf{cost}(X_i)$	MA-MVTSP	P1	P2
	MA-MVTSP+$\emptyset$	P3	P4
	MD-MVTSP	P5	P6
	MD-MVTSP+$\emptyset$	P7	P8

TABLE 4.1: Overview of the problems considered in this chapter.

Without depots In the Multiple-Agent MVTSP variants (MA-MVTSP), the goal is to obtain a multigraph consisting of k closed walks (MVTSP tours), that has a minimal total cost and visits each vertex $v \in V$ exactly $r(v)$ times. When it comes to precise definitions of this problem, one can set the rules for the k tours in different ways. Some of these differences might seem subtle, but they are crucial in algorithm design. We denote the different possible problems the following way:

(P1) MA-MVTSP with arbitrary tours: the edge set of the resulting multigraph X can be partitioned into k MVTSP tours, meaning the k tours have to be edge-disjoint but not vertex-disjoint;

(P2) MA-MVTSP with disjoint tours: the edge set of the resulting multigraph X can be partitioned into k *vertex-disjoint* MVTSP tours, i.e. X has exactly k components, each of them a MVTSP tour;

[1]The solution can include multiple copies of the same edge, and each copy is traversed exactly once by exactly one agent.

(P3) MA-MVTSP+$\emptyset$ with arbitrary tours: the edge set of the resulting multigraph X can be decomposed into at most k, edge-disjoint but not necessarily vertex-disjoint MVTSP tours;

(P4) MA-MVTSP+$\emptyset$ with disjoint tours: the edge set of the resulting multigraph X can be decomposed into at most k vertex-disjoint MVTSP tours.

Recall that the cost of a self-loop at a vertex, i.e. going from vertex v to itself does not necessarily incur a zero cost. Moreover, if an agent is assigned a single vertex v in a solution, an isolated point does not count as one visit, i.e. the tour of that agent has to consist of a self-loop at v for each visit they ought to make at v.

With depots In the MD-MVTSP problems, in addition to the setting of the MA-MVTSP problems, there is a set D of special vertices called depots, and the aim is to find a minimum cost multigraph consisting of k closed walks, such that each one contains exactly one depot d from D. The problem definitions follow a similar pattern as in the case of MA-MVTSP, but include the notion of depots:

(P5) MD-MVTSP with arbitrary tours: the edge set of the resulting multigraph X can be partitioned into k MVTSP tours with one depot in each tour, but the tours can have common vertices;

(P6) MD-MVTSP with disjoint tours: the edge set of the resulting multigraph X has exactly k components—each of them a MVTSP tour—with one depot in each;

(P7) MD-MVTSP+$\emptyset$ with arbitrary tours: the edge set of the resulting multigraph X can be decomposed into at most k MVTSP tours with one depot in each tour, but the tours can have common vertices;

(P8) MD-MVTSP+$\emptyset$ with disjoint tours: the edge set of the resulting multigraph X has at most k components—each of them a MVTSP tour—with one depot in each.

Note that although the depot set D is defined to be disjoint from the vertex set V, the edge costs $c(uv)$ defined on graphs $G(V, D, E)$ throughout this chapter satisfy the triangle inequality for every triplet u, v, w from $V \cup D$.

Remark 5. *Note that if we know the subsets of cities each agent visits, the problem reduces to the Many-Visits TSP. More precisely, given a cover of cities for P1 and P3, a partition of cities for P2 and P4, or an assignment of cities to agents for problems P5–P8, we can obtain an $3/2$-approximation for the k-agent problem by calculating a tour on each subset of the cities using Algorithm 3.3.*

Scheduling theory implications The problems above were defined having agents in mind, that visit cities a certain number of times. However one can formalise the same problems using scheduling theory terms. Assume there are k identical machines and n job types, and we are given $r(j)$ jobs of type j, where $j = 1, \ldots, n$. Every job of type j has a processing time $p(j)$ and a sequence-dependent preparation time $s(ij)$ that depends

on the type i of the preceding job in the schedule. Moreover, the parameter $c(ij) := s(ij) + p(j)$ is symmetric and satisfies the triangle inequality. The aim is to provide a periodic schedule of jobs on k machines, such that the total processing time of the jobs is minimal.

In case of MA-MVTSP, the disjoint tours variant can be translated as finding a schedule of jobs, such that each job type j is only assigned to one specific machine, and every job of type j has to be processed on that machine. The arbitrary tours variant corresponds to finding a schedule of jobs, where jobs of the same type can be processed on multiple machines, however, we do not allow any machine to be idle (i.e. with no jobs scheduled on it). Finally, the empty tours variant corresponds to the case when we allow for idle machines in the final schedule.

The different variants of the MD-MVTSP correspond to similar scheduling problems, where additionally a special job (that corresponds to the depot) is present on each machine. One can think of this job as preparation that needs to be performed on a machine if one intends to process other jobs on that machine. Note that the visit requirement $r(d) = 0$ for any depot $d \in D$ in the MD-MVTSP variants supports this assumption, as one only needs to pay the preparation cost (that corresponds to the visit of d) if one uses the machine.

Results

The main results of this chapter include polynomial-time constant-factor approximation algorithms for all problem variants. We start with 3-approximations for the MA-MVTSP with empty tours and the MD-MVTSP with arbitrary tours and empty tours variants (Algorithms 4.1–4.3). The common theme in the these approaches is they all compute a minimum cost constrained spanning forest, double its edges, then calculate an optimal transportation problem solution. By bounding the cost of both of these structures, the approximation ratio follows.

We continue by showing improved 2-approximations for the empty tours variant of both MA-MVTSP and MD-MVTSP (Algorithms 4.4 and 4.5). These approaches build on Theorem 2.1 from Chapter 2, that was used to obtain a polynomial-time $3/2$-approximation for the metric Many-Visits Path TSP in Chapter 3. The algorithms calculate degree-bounded multigraphs, then apply an edge-doubling and an efficient shortcutting procedure.

Finally, we show simple 4-approximations for the MA-MVTSP with arbitrary tours and the disjoint tours variant of both the MA-MVTSP and the MD-MVTSP, presented as Algorithms 4.8–4.10. These approaches are similar to the 3-approximations, as they compute a minimum cost constrained spanning forest and double its edges, but they augment the solution with self-loops instead of a transportation problem solution.

We provide an overview of these results in Table 4.2.

	arbitrary tours	**disjoint tours**
MA-MVTSP	4-approx. (Algorithm 4.8)	4-approx. (Algorithm 4.9)
MA-MVTSP+$\emptyset$	3-approximation (Algorithm 4.1) 2-approximation (Algorithm 4.5)	
MD-MVTSP	3-approx. (Algorithm 4.3)	4-approx. (Algorithm 4.10)
MD-MVTSP+$\emptyset$	3-approximation (Algorithm 4.2) 2-approximation (Algorithm 4.4)	

TABLE 4.2: Overview of approximation algorithms for the different variants of Multiple-Agent and Multiple-Depot MVTSP.

The MA-MVTSP with empty tours generalizes the k-cycle cover problem for undirected graphs [70], hence improving the approximation ratio of Algorithm 4.5 for symmetric edge costs would imply an improved algorithm for the k-cycle cover problem. The (single-visit) Multiple-Depot TSP allows some agents to not visit any cities, i.e. having a component in the solution graph consisting of a depot and no other vertices or edges. Thus, the MD-MVTSP with empty tours generalizes the MDMTSP, and the approximation ratio of Algorithm 4.4 matches that of Rathinam et al. [96] and is only slightly worse than that of Xu et al. [117].

Our results do not compare directly to the results by Jansen et al. [64] and Deppert & Jansen [30]. In one aspect, the problem variants considered by the authors can be regarded as easier than the problems considered in this paper, as they assume sequence-independent setup times that only occur before processing a batch of jobs from the same class. On the other hand, the jobs from the same class considered in those works [30, 64] can have *different* processing times, which is a much more general assumption than the cost structure of jobs considered in this work (the edge costs translate to zero processing times and the sum of processing and setup times being metric).

Reductions among problem variants

Below we discuss the similarities and differences of the problems.

Claim 4.1. *A feasible solution for P1, P2, P5 and P6 is a feasible solution for P3, P4, P7 and P8, respectively. A feasible solution for P2, P4, P6 and P8 is a feasible solution for P1, P3, P5 and P7, respectively.*

Proof. Problems P3, P4, P7 and P8 only differ from their respective counterparts of P1, P2, P5 and P6 in one attribute: the former problems allow for empty tours in the solution. Moreover, problems P2, P4, P6 and P8 are relaxations of problems P1, P3, P5 and P7, respectively, with the disjointness constraint lifted. □

Lemma 4.2. *There are polynomial-time reduction to problems P1, P2, P5, P6 from problems P3, P4, P7, P8, respectively.*

Proof. From the problem definitions, an optimal solution $X^\star$ to P3, P4, P7 or P8 consists of k, possibly empty MVTSP tours, denoted by $X_1, \ldots, X_k$. Let us reorder the agents

such that $X_1, \ldots, X_\ell$ denote the empty tours and $X_{\ell+1}, \ldots, X_k$ the non-empty tours. It is easy to see that $X^\star$ is not only an optimal solution to P3, P4, P7 or P8, but their counterparts P1, P2, P5 or P6, with ℓ agents. This means by solving problem instances of P1, P2, P5 or P6 considering ℓ agents, for $\ell := 1, \ldots, k$, the cheapest of the k solutions will be an optimal solution to P3, P4, P7 or P8. □

Certain problems we consider show some similarities, while others differ significantly, in terms of their optimal solutions. Below we will address these similarities, and use the symbol '$\equiv$' to indicate that given the same parameters G, c, r and k, the optimal solution values of two problems in question coincide. Intuitively this means that restricting the tours to be disjoint or forbidding empty tours in the solution might have no effect on the optimal solution value. Similarly, we use the symbol '$\not\equiv$' if there are instances where the cost of optimal solutions of the two problems are different.

In the next two claims we prove that in case we allow the agents to have empty tours, it does not matter whether we require vertex-disjoint tours or allow the tours to overlap.

Claim 4.3. *P3 $\equiv$ P4.*

Proof. We will prove that the optimal solutions to P3 and P4 have the same cost, given the same input variables (G, c, r, k).

Let us start with proving that $\mathsf{OPT}^{\mathrm{P3}} \geqslant \mathsf{OPT}^{\mathrm{P4}}$. Consider an optimal solution to P3, denoted by $X^\star$, then $X^\star$ has cost $\mathsf{OPT}^{\mathrm{P3}}$. Suppose that the tours of agents i and j overlap, that is, $V(X_i^\star) \cap V(X_j^\star) \neq \emptyset$. $X_i^\star$ and $X_j^\star$ are valid MVTSP tours, meaning every vertex $v \in V(X_i^\star)$ has an even degree in $X_i^\star$, and every vertex $v \in V(X_j^\star)$ has an even degree in $X_j^\star$. This means every $v \in V(X_i^\star) \cup V(X_j^\star)$ has an even degree in $X_i^\star \cup X_j^\star$, and moreover, $X_i^\star \cup X_j^\star$ is connected. One can now redefine the tour of agent i as $X_i := X_i^\star \cup X_j^\star$ and the tour of agent j as the empty graph ($X_j := \emptyset$), making the tours of agents i and j disjoint. The resulting multigraph X is also a solution to P4, since it has k components and visit each vertex $v \in V$ exactly $r(v)$ times, therefore $\mathsf{cost}(X) \geqslant \mathsf{OPT}^{\mathrm{P4}}$. Moreover, the cost of X is $\mathsf{OPT}^{\mathrm{P3}}$, since they have the same edge multiset, hence $\mathsf{OPT}^{\mathrm{P3}} \geqslant \mathsf{OPT}^{\mathrm{P4}}$ holds.

Now assume we have an optimal solution to P4, denoted by $Y^\star$, with cost $\mathsf{OPT}^{\mathrm{P4}}$. According to Claim 4.1, the multigraph $Y^\star$ is a feasible solution to P3, thus $\mathsf{OPT}^{\mathrm{P3}} \leqslant \mathsf{OPT}^{\mathrm{P4}}$, and the proof is complete. □

Similar arguments work in the respective problems with depots.

Claim 4.4. *P7 $\equiv$ P8.*

Proof. Proving that the optimal costs of P7 and P8 are equal goes similarly to the proof of Claim 4.3.

Let $X^\star$ be an optimal solution to P7 with cost $\mathsf{OPT}^{\mathrm{P7}}$, and suppose the tours of agents i and j overlap in $X^\star$; i.e. there are $k' < k$ components in P7 with at least one component having more than one depots. We do a similar transformation to the one in the proof of Claim 4.3, resulting in disjoint tours the following way. Let us go through

every vertex pair (u, v), incident to d_j and replace all copies of edges $d_j u$ and $d_j v$ by a copy of uv. Note that we did not require $u \neq v$ to hold, therefore this sequence of operations resulted in a multigraph $X^\star$ where d_j is an isolated vertex. Moreover, because of the triangle inequality, this transformation did not increase the cost of $X^\star$. Repeat this operation until there is exactly one depot in each component of $X^\star$. The resulting multigraph $X^\star$ is now a feasible solution to P8, with cost at most $\mathsf{OPT}^{\mathrm{P7}}$, therefore $\mathsf{OPT}^{\mathrm{P7}} \geqslant \mathsf{cost}(X^\star) \geqslant \mathsf{OPT}^{\mathrm{P8}}$ holds.

Now assume $Y^\star$ is an optimal solution to P8, with cost $\mathsf{OPT}^{\mathrm{P8}}$. Due to Claim 4.1, the multigraph $Y^\star$ is a feasible solution to P7, therefore $\mathsf{OPT}^{\mathrm{P7}} \leqslant \mathsf{OPT}^{\mathrm{P8}}$ holds, completing the proof. □

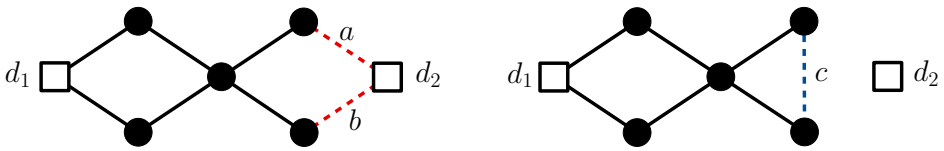


FIGURE 4.1: Illustration for Claim 4.4. Because of the triangle inequality, the blue edge (dashed, right) costs at most as much as the red edges (dashed, left) (the two tours overlapped in the vertex in the middle).

In conclusion, there is an approximation-preserving reduction from problems P3/P4 to both P1 and P2; similarly there is approximation-preserving reduction from problems P7/P8 to both P5 and P6.

Contrary of Claims 4.3–4.4, if we do not allow empty tours, the optimal solutions for the arbitrary tours and the disjoint tours variants can be different:

Claim 4.5. *P1 ≢ P2, P5 ≢ P6.*

Proof. According to Claim 4.1, a solution to problems P2 and P6 are feasible solutions to P1 and P5, respectively. Moreover, the only constraint that problems P2 and P6 impose on the feasible solutions in addition to the constraints of P1 and P5 is that the tours of the agents have to be disjoint. We show that the disjointness requirement is meaningful, i.e. the optimal solution to an instance of problem P2 or P6 can have strictly higher cost than the optimal solution to the corresponding instance of P1 or P5, respectively.

In case of problems P5 and P6, one can see in the example in Figure 4.2 that by appropriately setting the edge costs, the solution on the right has significantly higher cost than the solution on the left. For problems P1 and P2 the same example demonstrates the claim, by assuming that d_1 and d_2 are regular vertices with a requirement of 1. □

Finally, let us show that allowing empty tours in the solutions makes a significant difference:

Claim 4.6. *P1 ≢ P3, P2 ≢ P4.*

Proof. We will prove that by allowing less than k components, the cost of the optimal solution can be strictly smaller than in the case where we require exactly k components.

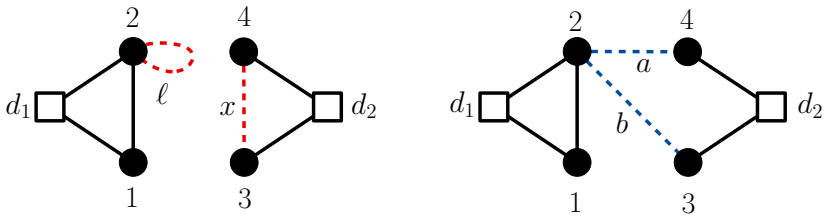


FIGURE 4.2: Illustration for Claim 4.5. Metric costs imply $\ell \leqslant a+b$ and $x \leqslant a+b$. In the case $\ell + x > a + b$ holds, the blue edges (dashed, right) costs less, than the red edges (dashed, left).

This is true for the problems where we allow overlapping tours, and also for problems with disjoint tours.

Consider a graph $\hat{G}$ on the edge set V, with edge set $\hat{E} = \{v_1v_2, v_2v_3, \ldots, v_nv_1\}$, therefore $\hat{G}$ is essentially the graph C_n with $n = |V|$. Now define a complete graph G on the same vertex set, where the edge costs $c(v_iv_j)$ equal to the shortest path distance between v_i and v_j in $\hat{G}$. Moreover, assume $r(v) = 1$ for all $v \in V$.

Clearly, the optimal solution for the MA-MVTSP+$\emptyset$ is the cycle v_1-v_2-$\ldots$-v_n-v_1 with cost n, for any $k \in \mathbb{Z}_{\geqslant 1}$. However, if we require exactly k components in our solution, the optimal solution has to use edges with cost larger than 1, resulting in a solution with a cost larger than n. □

The same is true when considering depots:

Claim 4.7. *P5 ≢ P7, P6 ≢ P8.*

Proof. Assume that the cost of every edge $uv \in E$ and $dv \in E$ is 1, where $u, v \in V$ and $d \in D$. In this case, any optimal solution to the MD-MVTSP+$\emptyset$ consists of one non-trivial components and $k-1$ isolated depots. Therefore the optimal cost (the number of edges in the solution) is $\sum_{v \in V} r(v) + 1$. On the other hand, in an optimal solution to the MD-MVTSP the degree of every depot $d \in D$ is 2, therefore the optimal cost is $\sum_{v \in V} r(v) + k$, which, for $k > 1$ is larger than the optimal cost of MD-MVTSP+$\emptyset$. This also holds if we allow the tours to overlap. □

FIGURE 4.3: Tour on the left has cost 6, tour on the right has cost 5

Now we have full description of the equivalence of problems P1 to P8. Based on Claims 4.3 and 4.4, we do not distinguish between problems P3 and P4, as well as between problems P7 and P8. For this reason we will just refer to these problems as MA-MVTSP with empty tours and MD-MVTSP with empty tours, respectively. We will also simply call problems P1 and P5 as the arbitrary tours variant of MA-MVTSP and MD-MVTSP, respectively; and problems P2 and P6 as the empty tours variant of MA-MVTSP

and MD-MVTSP, respectively. An overview of the reductions among the different problem variants are shown on Figure 4.4.

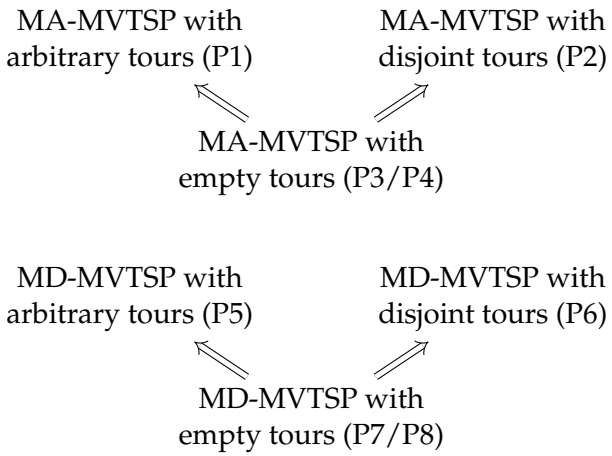


FIGURE 4.4: Reductions among problems P1–P8. An algorithm for a problem at the head of an arrow provides yields an algorithm for the problem at the tail of the arrow.

Although the latter two problems may seem very similar, as feasible solutions to both include multigraphs with less than k components, there is one key difference. In case of MA-MVTSP with arbitrary tours, we have to be able to decompose the components into k MVTSP tours. This condition does not hold for every feasible solution for the MA-MVTSP with empty tours: Suppose $k > 1$ and $r \equiv 1$, then a Hamiltonian cycle on V is a feasible solution to the empty tours case: 1 agent traverses the cycle while $k-1$ agents are idle. However it cannot be decomposed into $k > 1$ MVTSP tours, therefore it is not a valid arbitrary tours solution.

Let us note that the depots have no $r(\cdot)$ values assigned, however, the problem definitions can inherently provide a lower bound for the number of visits: In the case of disjoint and arbitrary tours the number of visits has to be at least one, while in the case of empty tours this lower bound is zero. There is no implicit upper bound for the number of visits, however, there is always an optimal solution where every depot is visited at most once, as in case of more than one visits it is possible to do shortcuts and obtain a tour with smaller or equal cost, due to the edge costs being metric.

Contrary to the MA-MVTSP problems, it can happen that an instance is infeasible. Take the case when there are more depots than cities, then there is no solution for the disjoint tours variant. Moreover, if there are more depots than the total visit requirements $\sum_{v \in V} r(v)$, there is no solution for neither the disjoint nor the arbitrary tour variants (in case we do not allow empty tours in the solutions).

4.2 3-approximations for arbitrary tours and empty tours variants

In this section we provide 3-approximations to the MA-MVTSP with empty tours and both the arbitrary tours and the empty tours variants of the MD-MVTSP. The algorithms share the basic ideas, hence we first present the notions and techniques that are common to all algorithms.

Throughout Lemmas 4.8–4.10 we assume that the edge cost function c satisfies the triangle inequality. Recall the Hitchcock's transportation problem from §1.1.1, and observe that keeping the degree constraints but relaxing the connectivity (similarly to the single-agent case in Chapters 1 and 3) and decomposability constraints in any of the MA-MVTSP variants yields the transportation problem:

Lemma 4.8. *Let $X^\star$ be an optimal solution to any of the MA-MVTSP variants (disjoint, arbitrary or empty tours) on an instance (G, c, r). Let $\mathsf{TP}^\star$ be an optimal transportation problem solution on (G, c, r). Then, $\mathsf{cost}(\mathsf{TP}^\star) \leqslant \mathsf{cost}(X^\star)$.* □

For request vectors $r, r' : V \to \mathbb{Z}_{\geqslant 1}$, let us denote by $r \preccurlyeq r'$ if $r(v) \leqslant r'(v)$ holds for all $v \in V$.

Lemma 4.9. *Let TP and TP' be optimal transportation problem solutions for the instances (G, c, r) and (G, c, r'), respectively. If $r \preccurlyeq r'$ holds, then $\mathsf{cost}(\mathsf{TP}) \leqslant \mathsf{cost}(\mathsf{TP}')$.*

Proof. By definition, $\dot{\delta}_{\mathsf{TP}}(v) = 2 \cdot r(v)$ and $\dot{\delta}_{\mathsf{TP}'}(v) = 2 \cdot r'(v)$ holds. If $\dot{\delta}_{\mathsf{TP}}(\cdot) \equiv \dot{\delta}_{\mathsf{TP}'}(\cdot)$, we are done. Otherwise, there is at least one vertex $v \in V(G)$, such that $\dot{\delta}_{\mathsf{TP}}(v) < \dot{\delta}_{\mathsf{TP}'}(v)$. Apply shortcuts in TP, until $\dot{\delta}_{\mathsf{TP}}(v) = \dot{\delta}_{\mathsf{TP}'}(v)$ holds for every $v \in V(G)$ by applying the following two operations in total of $(\dot{\delta}_{\mathsf{TP}'}(v) - \dot{\delta}_{\mathsf{TP}}(v))/2$ times:

(A) If v is incident to copies of self-loops vv, remove a copy of vv.

(B) If there exist not necessarily distinct vertices u, w adjacent to v, replace a copy of uv and vw by a copy of uw.

Operation (A) can only decrease the cost of TP', as the edge costs are nonnegative. Due to the triangle inequality, operation (B) can also only decrease the cost of TP. Therefore, the cost of the resulting multigraph TP'' will be $\mathsf{cost}(\mathsf{TP}'') \leqslant \mathsf{cost}(\mathsf{TP}')$. Moreover, since both operation (A) and (B) decrease the degree of a vertex by 2, applying one of the operations $(\dot{\delta}_{\mathsf{TP}'}(v) - \dot{\delta}_{\mathsf{TP}}(v))/2$ times will result in v having a degree of $\dot{\delta}_{\mathsf{TP}}(v)$ in TP''.

Perform the operations above for every vertex $v \in V$. Now multigraph TP has the same degree sequence as TP'', moreover, TP is a minimum cost such multigraph. Hence $\mathsf{cost}(\mathsf{TP}) \leqslant \mathsf{cost}(\mathsf{TP}'')$ holds, and the proof follows. □

Now we can show that the claim of Lemma 4.8 also holds for the problem variants with depots.

Lemma 4.10. *Let $X^\star$ be an optimal solution to any of the MD-MVTSP variants (disjoint, arbitrary or empty tours) on an instance (G, c, r, D). Let TP be an optimal transportation problem solution on (G, c, r). Then, $\mathsf{cost}(\mathsf{TP}) \leqslant \mathsf{cost}(X^\star)$.*

Proof. Let us define an auxiliary graph G' on the vertex set $V(G) \cup D$ with edge costs $c'(uv) := c(uv)$ for every vertex $u, v \in V(G) \cup D$. Define the requests of vertices $v \in V(G)$ by $r'(v) := r(v)$, and the requests of the depots $r'(d) := \dot{\delta}^{\star}_{X}(d)$ for $d \in D$. Calculate an optimal transportation problem solution TP' on (G', c', r').

TP' is a minimum cost multigraph with the same degree sequence as $X^{\star}$, while the connectivity and decomposability constraints relaxed. Hence $\mathsf{cost}(\mathsf{TP}') \leqslant \mathsf{cost}(X^{\star})$. Moreover, as the depot set D is not covered by TP, one can think of the requests of $d \in D$ in the instance (G, c, r) as $r(d) = 0$. Since $r(v) = r'(v)$ for vertices $v \in V$ and $r(d) \leqslant r'(d) = \dot{\delta}^{\star}_{X}(d)$ for $d \in D$, $r \preccurlyeq r'$ holds. Due to Lemma 4.9, $\mathsf{cost}(\mathsf{TP}) \leqslant \mathsf{cost}(\mathsf{TP}')$ follows, which together with $\mathsf{cost}(\mathsf{TP}') \leqslant \mathsf{cost}(X^{\star})$ proves the claim. □

As a direct consequence of Lemmas 4.8–4.10, we have the following:

Corollary 4.11. *Let $X^{\star}$ be an optimal solution to any of the MA-MVTSP variants on an instance (G, c, r)* ***or*** *any of the MD-MVTSP variants on an instance (G, c, r, D). Let TP be an optimal solution to the transportation problem on (G, c, r') with $r' \preccurlyeq r$. Then, $\mathsf{cost}(\mathsf{TP}) \leqslant \mathsf{cost}(X^{\star})$.*

4.2.1 Without depots (empty tours)

First, let us consider the problem variant MA-MVTSP with empty tours.

Let $G = (V, E)$ be a complete graph, $c(uv)$ metric, symmetric edge costs and $r(v)$ requirements for every vertex $v \in V$. In all variants of the MA-MVTSP, we seek a minimum cost multigraph $X^{\star}$ that visits each vertex $v \in V$ a total of $r(v)$ times. Recall that the problem MA-MVTSP with arbitrary tours and the MA-MVTSP with empty tours $X^{\star}$ has to consist of at most k components; in addition in the former we require the edges of the multigraph to be partitioned into k MVTSP tours, while in the latter we allow for $k - k'$ agents to have the empty tour, where k' is the number of components in $X^{\star}$.

Let $\mathsf{OPT}^{(k)}_{c,r}$ be the cost of an optimal MA-MVTSP with disjoint tours solution consisting of exactly k components. Moreover, we denote the cost of the optimal MA-MVTSP with empty tours solution by $\mathsf{OPT}^{(\leqslant k)}_{c,r}$, this tour consists of *at most* k components. It is easy to see that

$$\mathsf{OPT}^{(\leqslant k)}_{c,r} = \min_{\kappa \leqslant k} \left\{ \mathsf{OPT}^{(\kappa)}_{c,r} \right\} \quad . \tag{4.1}$$

Theorem 4.12. *Algorithm 4.1 provides a 3-approximation for MA-MVTSP with empty tours in time polynomial in n, k and $\log r(V)$.*

Proof. We first prove that Algorithm 4.1 constructs a feasible solution to the MA-MVTSP with empty tours, and provide an upper bound to its cost. Finally, we show that the time complexity claims hold.

Feasibility First, let us show that the multigraph $X^{(\kappa)}$ constructed by Algorithm 4.1 is a feasible solution to MA-MVTSP with empty tours, for any $\kappa = 1, \ldots, k$. The spanning forest $F^{(\kappa)}$ has exactly κ components, and duplicating its edges does not change

Algorithm 4.1 A 3-approximation for the MA-MVTSP with empty tours

Input: A complete undirected graph $G(V, E)$, costs $c : E \to \mathbb{R}_{\geqslant 0}$ satisfying the triangle inequality, requests $r : V \to \mathbb{Z}_{\geqslant 1}$, number of agents k.

Output: A multigraph consisting of at most k components that visits each $v \in V$ a total of $r(v)$ times.

1: Calculate the solution to the transportation problem defined on G with supply and demand being equal to $r(v) - 1$, denote it by TP.
2: **for** $\kappa = 1, \dots, k$ **do**
3: Calculate a minimum cost spanning forest of G consisting of κ components, denote it by $F^{(\kappa)}$, and its components $F_1^{(\kappa)}, \dots, F_\kappa^{(\kappa)}$.
4: Duplicate the edges of $F^{(\kappa)}$, and apply shortcuts to obtain κ Hamiltonian cycles on the vertex sets $V\big(F_1^{(\kappa)}\big), \dots, V\big(F_\kappa^{(\kappa)}\big)$; denote the resulting graph $H^{(\kappa)}$.
5: Define the multigraph $X^{(\kappa)}$ on vertex set V, such that the edge set of $X^{(\kappa)}$ consists of one copy of $E\big(H^{(\kappa)}\big)$ and one copy of $E(\mathsf{TP})$.

return the cheapest $X^{(\kappa)}$

the number of components. Adding the edges of TP in Line 5 might connect different components, thus the number of components of $X^{(\kappa)}$ is at most κ. Since the algorithm considers κ values up to k, the algorithm returns a graph with at most k components.

In $X^{(\kappa)}$ for any κ, the multigraph TP provides $2 \cdot (r(v) - 1)$ degree to a vertex $v \in V$, whereas $H^{(\kappa)}$ provides a degree of 2. This means $\delta_{X^{(\kappa)}}(v) = 2 \cdot r(v)$ for any $v \in V$ in Line 5.

Cost of solution In the κ^{th} iteration, the minimum cost spanning forest $F^{(\kappa)}$ computed in Line 3 has cost at most $\mathsf{OPT}_{c,r}^{(\kappa)}$, since the optimal solution to MA-MVTSP with κ agents can be turned to a spanning forest with exactly κ components via edge deletion. Moreover, because the costs c are metric, applying shortcuts in Line 4 does not increase the cost of $H^{(\kappa)}$. The cost of the multigraph TP computed in Line 1 has cost at most $\mathsf{OPT}_{c,r}^{(\leqslant k)}$, due to Corollary 4.11.

In conclusion, in the κ^{th} iteration Algorithm 4.1 considers a solution that has cost at most $2 \cdot \mathsf{OPT}_{c,r}^{(\kappa)} + \mathsf{OPT}_{c,r}^{(\leqslant k)}$, therefore the cheapest $X^{(\kappa)}$ has cost at most

$$2 \cdot \min_{\kappa=1,\dots,k} \left\{ \mathsf{OPT}_{c,r}^{(\kappa)} \right\} + \mathsf{OPT}_{c,r}^{(\leqslant k)} = 3 \cdot \mathsf{OPT}_{c,r}^{(\leqslant k)} \ ,$$

due to Equation (4.1).

This proves the approximation ratio of the algorithm.

Complexity analysis One can obtain TP by solving a min cost max flow problem, which takes polynomial time [32, 74, 92], and computing a minimum cost spanning forest can also be done efficiently. The number of iterations is $k \leqslant n$. Throughout the algorithm we use a compact representation of all multigraphs, this way the time and space complexity of graph operations can be bounded by $O(n^2 \log r(V))$. □

4.2.2 With depots (empty tours and arbitrary tours)

Now let us assume that, besides the vertex set V, the graph G comes with a set of special vertices D called depots, such that $D \cap V = \emptyset$ and $k = |D|$. Let us start with the empty tours variant of the MD-MVTSP. The objective is to find k MVTSP tours each containing one depot, that visit each vertex $v \in V$ a total of $r(v)$ times, and have the total cost minimised. The components can be trivial (isolated depot).

Formally, we seek a minimum cost multigraph X with the following properties. Every vertex $v \in V$ has degree $2 \cdot r(v)$ while every vertex $d \in D$ has an even degree (including zero) in X. Moreover, X has exactly k components and there is exactly one vertex d from D in each component.

In this case we denote the cost of the optimal solution of MD-MVTSP with disjoint tours by $\mathsf{OPT}^{(k)}_{c,r,D}$ which consists of k non-trivial components. Moreover, by $\mathsf{OPT}^{(\leqslant k)}_{c,r,D}$ we denote the cost of the optimal MD-MVTSP with empty tours solution consisting of k components, while allowing for isolated depots as components. Similarly to the no-depot case,

$$\mathsf{OPT}^{(\leqslant k)}_{c,r,D} = \min_{\kappa \leqslant k} \left\{ \mathsf{OPT}^{(\kappa)}_{c,r,D} \right\}$$

holds.

We can use the idea of Algorithm 4.1 to design an algorithm for the MD-MVTSP with empty tours. However, due to the restrictions involving depots, we build the spanning forest using the approach of Rathinam et al. [96], that goes as follows. First, assign a zero cost for every edge that connects two depots, denote the new cost function by $\hat{c}$. Now use Kruskal's algorithm [80] to obtain a minimum cost spanning tree $\hat{F}$ on G with respect to $\hat{c}$. Finally, remove the edges with zero cost from $\hat{F}$ and denote the resulting graph F.

Lemma 4.13. *Each component of the forest F calculated above contains exactly one depot from D. Moreover, the total cost of F with respect to c is at most $\mathsf{OPT}^{(\leqslant k)}_{c,r,D}$.*

Proof. Since Kruskal's algorithm chooses the edges of G with increasing cost $\hat{c}$, the tree $\hat{F}$ will contain $k - 1$ edges of zero cost, spanning the depot set D. It is easy to see that removing these edges will result in a forest with k components with one depot in each component.

Take an optimal MD-MVTSP with empty tours solution $X^\star$. The multigraph $X^\star$ clearly contains a spanning forest $F^\star$, that consists of k components with one depot in each component. Since the spanning forest F is a minimum cost of such graphs, $\mathsf{cost}(F) \leqslant \mathsf{cost}(F^\star)$ is true. Additionally, $\mathsf{cost}(F^\star) \leqslant \mathsf{cost}(X^\star)$ holds, hence the bound follows. □

The procedure described above serves as Line 2 of Algorithm 4.2.

Theorem 4.14. *Algorithm 4.2 provides a 3-approximation for MD-MVTSP with empty tours in time polynomial in n, k and $\log r(V)$.*

Proof. The proof goes as follows.

Algorithm 4.2 A 3-approximation for the MD-MVTSP with empty tours

Input: A complete undirected graph $G(V, D, E)$, costs $c : E \to \mathbb{R}_{\geqslant 0}$ satisfying the triangle inequality, requests $r : V \to \mathbb{Z}_{\geqslant 1}$, number of agents $k = |D|$.
Output: A multigraph consisting of k components, with one depot in each, that visits each $v \in V$ a total of $r(v)$ times.

1: Calculate the solution to the transportation problem defined on $G[V]$ with supply and demand being equal to $r(v) - 1$, denote it by TP.
2: Calculate a minimum cost spanning forest of G consisting of k components with exactly one depot in each component [96], denote it by F, and its components $F_1, \dots, F_k$.
3: Duplicate the edges of F, and apply shortcuts to obtain k Hamiltonian cycles on the sets $V(F_1), \dots, V(F_k)$; denote the resulting graph H and its components $H_1, \dots, H_k$.
4: Define the multigraph Z on vertex set V, such that the edge set of Z consists of a copy of $E(H)$ and a copy of $E(\mathsf{TP})$, denote the components of Z by $X_1, \dots, X_{k'}$.
5: If a component X_i contains more than one depot, disconnect all depots but one, using shortcuts on the edges; denote the new trivial components $X_{k'+1}, \dots, X_k$.
return $X := X_1 \cup \dots \cup X_k$

Feasibility First, let us prove that the multigraph X constructed by Algorithm 4.2 is a feasible solution to MD-MVTSP with empty tours. The spanning forest F has exactly k components, and duplicating its edges does not change the number of components. Adding the edges of TP in Line 4 might connect different components, thus the number of components of X is k', which is at most k. Finally, in Line 5 the number of components becomes k with one depot in each component.

The multigraph TP provides $2 \cdot (r(v) - 1)$ degree to a vertex $v \in V$, whereas the Hamiltonian cycles provide a degree of 2 to all vertices in V. In total, the degree of each vertex $v \in V$ in X is $2 \cdot r(v)$.

Cost of solution According to Corollary 4.11, the cost of TP is at most the cost of an optimal MD-MVTSP with empty tours solution, $X^\star$. Due to Lemma 4.13, the cost of F is at most $\mathsf{cost}(X^\star)$ as well. Because of metric edge costs, the shortcutting operations in Lines 3 and 5 does not increase the cost of the multigraph in question, hence both

$$\mathsf{cost}(Z) \leqslant \mathsf{cost}(\mathsf{TP}) + 2 \cdot \mathsf{cost}(F) \leqslant 3 \cdot \mathsf{cost}(X^\star)$$

and $\mathsf{cost}(X) \leqslant \mathsf{cost}(Z)$ hold. This proves that Algorithm 4.2 indeed provides a 3-approximation.

Complexity analysis One can obtain TP by solving a min cost max flow problem, which takes polynomial time [32, 74, 92], and computing a minimum cost spanning forest can also be done efficiently. Throughout the algorithm we use a compact representation of all multigraphs, this way the time and space complexity of graph operations can be bounded by $O(n^2 \log r(V))$. □

Similarly to the problems without depots, we can use the idea of Algorithm 4.2 to obtain an approximation algorithm for the MD-MVTSP with arbitrary tours. Note that the k-component spanning forest algorithm of Rathinam et al. [96] outputs a spanning

forest that may contain trivial components, i.e. isolated points. The MD-MVTSP with arbitrary tours does not allow for empty tours, therefore we cannot compare the cost of an optimal solution with the cost of a spanning forest calculated that way. Given a set K of special vertices, Cerdeira [24] provides a matroid-based algorithm of calculating a minimum cost forest such that every component contains exactly one vertex from K and none of the components are trivial, i.e. they contain at least one vertex from V.

We cannot use the algorithm from [24] directly on the vertex sets V and D because of the following. Note that the MD-MVTSP with arbitrary tours allows the MVTSP tours to overlap in one or more vertices. This means that an optimal solution may not contain a spanning forest with the properties from [24], but only a spanning forest with possibly more depots than 1 in a component. Therefore, we cannot compare the cost of the spanning forest calculated by the algorithm in [24] to the cost of an optimal solution.[2]

For the reasons above, we first build an auxiliary graph $\hat{G}$ in a way that a constrained spanning forest of $\hat{G}$ obtained by the algorithm of Cerdeira [24] yields an appropriate spanning forest in G: one that allows more than one vertices from D in a component but has a cost at most the optimal solution to the MD-MVTSP with arbitrary tours.

Calculating an appropriate spanning forest Let an auxiliary graph $\hat{G}$ consist of the depot set D and $m(v) := \min\{|D|, r(v)\}$ copies of every vertex $v \in V$, denoted by $v_1, \ldots, v_{m(v)}$. Denote the set of the latter vertices by $\hat{V}$. The edges between the same copies of a vertex v incur no cost, i.e. $\hat{c}(v_i, v_j) = 0$ for $i, j \in 1, \ldots, m(v)$; while the other edge costs are inherited from G the natural way: $\hat{c}(d, v_i) := c(d, v)$ and $\hat{c}(v_i, w_j) := c(v, w)$ for all copies $i = 1, \ldots, m(v)$ and $j = 1, \ldots, m(w)$ of v and w, respectively. Now calculate a minimum cost spanning forest $\hat{F}$, such that each component of $\hat{F}$ contains exactly one depot $d \in D$ and at least one vertex $v \in \hat{V}$, using the algorithm by Cerdeira [24]. Transform $\hat{F}$ into a spanning forest F on vertex sets V and D, by identifying all copies $v_1, \ldots, v_{m(v)}$ of v into a single vertex v.

Lemma 4.15. *The graph F calculated above is a minimum cost spanning forest on $G(V, D, E)$, such that:*

1) every component of F contains at least one depot $d \in D$,

2) every component of F contains at least one vertex $v \in V$,

3) no vertex $v \in V$ has more than $r(v)$ depots among its neighbours.

Proof. Every component of $\hat{F}$ contains exactly one depot; transforming $\hat{F}$ to F can merge components but cannot divide them, therefore the components of F will contain at least one depot. The same argument proves that property *2)* holds for a component of F as well. Finally, due to construction, a copy v_i of a vertex $v \in V$ can be adjacent to at most 1 depot in $\hat{F}$, and there are at most $r(v)$ copies of a vertex v. This proves property *3)*.

Now let us turn to the optimality of F. Note that by transforming $\hat{F}$ into F we delete edges with zero cost, hence $\mathsf{cost}(\hat{F}) = \mathsf{cost}(F)$. Suppose indirectly, that there ex-

[2]Moreover, the MD-MVTSP with arbitrary tours always have a feasible solution if $|D| \leqslant \sum_{v \in V} r(v)$, even if the depots outnumber the vertices, i.e. $|D| > |V|$. In these cases, a spanning forest with the aforementioned properties does not even exist.

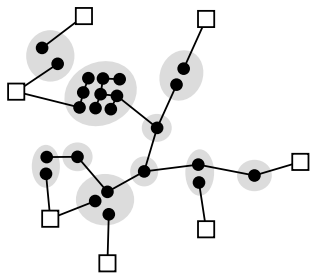
ists a graph F' satisfying properties *1)–3)*, with cost lower than that of F, i.e. $\mathsf{cost}(F') < \mathsf{cost}(F)$. Then, from F' we can construct a constrained spanning forest on $\hat{G}$ with exactly one depot and at least one regular vertex in each component, the following way. Make $m(v) - 1$ additional copies $v_1, \dots, v_{m(v)-1}$ of each vertex $v \in V$, and connect them to v in an arbitrary way, then rename the vertex v into $v_{m(v)}$. The resulting graph, denoted by $\hat{F}'$, is a constrained spanning forest on $\hat{G}$ with the same properties as $\hat{F}$. Moreover, because the transformation from F' into $\hat{F}'$ only added edges with zero cost, $\mathsf{cost}(\hat{F}') = \mathsf{cost}(F')$ holds. From the indirect assumption $\mathsf{cost}(F') < \mathsf{cost}(F)$ it follows, that $\mathsf{cost}(\hat{F}') < \mathsf{cost}(\hat{F})$ holds, which contradicts the optimality of $\hat{F}$. □

We will use the procedure above as a subroutine to get the approximation algorithm in Algorithm 4.3. Moreover, Figure 4.5 illustrates Lines 1–3.

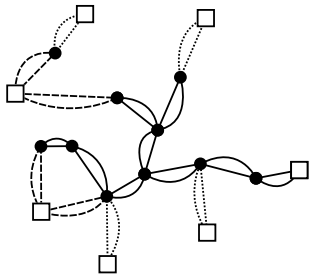
Algorithm 4.3 A 3-approximation for the MD-MVTSP with arbitrary tours

Input: A complete undirected graph $G(V, D, E)$, costs $c : E \to \mathbb{R}_{\geqslant 0}$ satisfying the triangle inequality, requests $r : V \to \mathbb{Z}_{\geqslant 1}$, number of agents $k = |D|$.
Output: A multigraph, such that the edges can be partitioned into k MVTSP tours with one depot and at least one non-depot, that collectively visit each $v \in V$ a total of $r(v)$ times.

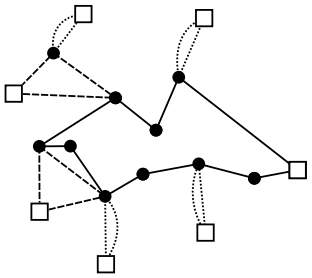
1: Determine a minimum cost spanning forest $\hat{F}$ of $\hat{G}$, as described before Lemma 4.15, that consists of m nontrivial components $\hat{F}_1, \dots, \hat{F}_m$ with exactly one depot in each component [24].
2: Identify all copies $v_1, \dots, v_{m(v)}$ of $v \in \hat{G}$ into a single vertex v, denote the image of $\hat{F}$ by F, and let F_i be the graph arising from $\hat{F}_i$ for $i = 1, \dots, m$.
3: For $i = 1, \dots, m$, duplicate the edges of F_i and apply shortcuts to obtain a Hamiltonian cycle H_i on $V(F_i)$.
4: Let X_i for all $i = 1, \dots, m$ be the MVTSP tours of the agents, with the initial values $X_i := H_i$.
5: Determine an optimal solution TP to the transportation problem defined on $G[V]$ with supply and demand being equal to $r'(v) := r(v) - |\{i \mid v \in V(F_i)\}|$ for $v \in V$.
6: For each component TP_j of TP, take an agent i with $V(\mathsf{TP}_j) \cap V(H_i) \neq \emptyset$, and update $X_i := X_i \cup \mathsf{TP}_j$.
return $X := X_1 \cup \dots \cup X_m$



(A) Spanning forest $\hat{F}$ on the auxiliary graph $\hat{G}$ (Line 1).



(B) Multigraph F, after duplicating the edges (Lines 2–3).



(C) Hamiltonian cycles H_i of the agents (Line 3).

FIGURE 4.5: Illustrating Lines 1–3 of Algorithm 4.3. Shaded ellipses in Figure 4.5a indicate copies of the same vertex $v \in V$. Cycles H_i of different agents marked with different edge styles for clarity (Figures 4.5b–4.5c).

Theorem 4.16. *Algorithm 4.3 provides a 3-approximation for MD-MVTSP with arbitrary tours in time polynomial in n, k and $\log r(V)$.*

Proof. The proof goes as follows.

Feasibility The Hamiltonian cycles constructed in Line 3 contribute a degree of $2 \cdot (r(v) - r'(v))$ to the degree of each vertex $v \in V$. Furthermore, by construction, each v in V is contained in at most $r(v)$ components of $\hat{F}$, therefore $r'(v) \geqslant 0$ holds and the transportation problem has a solution. The solution TP contributes with a degree of $2 \cdot r'(v)$ to the degree of each $v \in V$. Therefore the total degree of each vertex v in $X_1 \cup \cdots \cup X_m$ is $2 \cdot r(v)$.

Cost of solution Let $X^\star$ be an optimal solution for the problem. By Corollary 4.11, $\mathsf{cost}(\mathsf{TP}) \leqslant \mathsf{cost}(X^\star)$ holds. We claim that the cost of F is also bounded from above by the cost of $X^\star$. Indeed, $X^\star$ fulfills the properties *1)–3)* and, by Lemma 4.15, F is a minimum cost spanning multigraph with these properties. Due to metric edge costs, the shortcutting operation in Line 3 cannot increase the cost of the multigraph, hence the approximation ratio follows.

Complexity analysis The number of vertices in $\hat{G}$ is $O(n^2)$, therefore Lines 1 and 2 take polynomial time. The operations in Lines 3 and 6 can also be performed efficiently. Finally, the transportation problem in Line 5 can be solved in polynomial time as well. □

Similarly to the problem variants without depots, due to Lemma 4.2, the claim in Theorem 4.14 is a corollary of Theorem 4.16. Thus, in order to obtain a 3-approximation for the MD-MVTSP with empty tours, it is sufficient to run Algorithm 4.3 with $\ell = 1, \ldots, k$ agents, and choose the cheapest solution.

4.3 Improved 2-approximations for the empty tours variants

4.3.1 2-approximation algorithm for MD-MVTSP with empty tours

Theorem 4.17. *The output of Algorithm 4.4 is a 2-approximation for MD-MVTSP with empty tours.*

Proof. The proof goes as follows.

Feasibility For every $\kappa = 1, \ldots, k$, calculate an approximate solution to the MINIMUM BOUNDED DEGREE 1-COMPONENT MULTIGRAPH problem on $\hat{V}$ with $\rho(v) = 2 \cdot \hat{r}(v)$ for all $v \in \hat{V}$. The resulting multigraph $\hat{X}^{(\kappa)}$ is connected and provides a degree at least $2 \cdot \hat{r}(v) - 1$ for every $v \in \hat{V}$. After doubling the edges, $X^{(\kappa)}$ is still connected. By replacing $\hat{d}$ with $d_1, \ldots, d_\kappa$ the resulting multigraph will have at most κ components.

After taking shortcuts, the number of components will be exactly k, with exactly one depot in each component. In the end, each vertex $v \in V$ is visited exactly $r(v)$ times by the components.

Algorithm 4.4 A 2-approximation for the MD-MVTSP with empty tours

Input: A complete undirected graph $G(V, D, E)$, costs $c : E \to \mathbb{R}_{\geqslant 0}$ satisfying the triangle inequality, requests $r : V \to \mathbb{Z}_{\geqslant 1}$, number of agents $k = |D|$.

Output: k MVTSP tours with total minimal cost such that each tour contains exactly one depot $d \in D$ and that the tours visit each $v \in V$ a total of $r(v)$ times.

1: Identify all depots $d \in D$ into one meta-depot $\hat{d}$. Let $\hat{V} := V \cup \{\hat{d}\}$ and $\hat{E} := \hat{V} \times \hat{V}$. Set $c_{\hat{d},v} = \min_{d \in D} c_{d,v}$ for every vertex $v \in \hat{V} - \{\hat{d}\}$, and let $d(v) := \operatorname{argmin}_{d \in D} c_{d,v}$. Set $\hat{r}(v) = r(v)$ for $v \in \hat{V} - \{\hat{d}\}$.

2: **for** $\kappa = 1, \ldots, k$ **do**

3: Set $\hat{r}(d) := \kappa$.

4: Run Algorithm 2.1 to obtain a connected multigraph $\hat{X}^{(\kappa)}$ on $\hat{V}$ with $\dot{\delta}_{\hat{X}^{(\kappa)}}(v) \geqslant 2 \cdot \hat{r}(v) - 1$ for all $v \in \hat{V}$.

5: Duplicate all edges in $\hat{X}^{(\kappa)}$.

6: Reverse the identification operation: replace $\hat{d}$ with $d_1, \ldots, d_k$ and all copies of edges $(\hat{d}, v)$ with $(d(v), v)$ for $v \in \hat{V} - \{\hat{d}\}$ in $\hat{X}^{(\kappa)}$; denote the resulting multigraph $X^{(\kappa)}$.

7: If there is a component in $X^{(\kappa)}$ with more than one depot $d \in D$, disconnect depots until there is only one depot per component, using shortcuts. Apply shortcuts in each non-trivial component of $X^{(\kappa)}$ until the degree of every node $v \in V$ is $r(v)$, using the method defined in Algorithm 4.6.

return the cheapest $X^{(\kappa)}$

Cost of solution The multigraph $\hat{X}^{(\kappa)}$ obtained by Line 4 has cost at most $\mathsf{OPT}^{(\kappa)}_{c,r,D}$, therefore after doubling all its edges in Line 5 the resulting multigraph $X^{(\kappa)}$ has cost at most $2 \cdot \mathsf{OPT}^{(\kappa)}_{c,r,D}$. Line 6 does not change the cost of $X^{(\kappa)}$, as we are replacing each edge (d', v) with an edge $(d(v), v)$ of identical cost. Finally, the shortcuts in Line 7 cannot increase the cost of $X^{(\kappa)}$, because the edge costs satisfy the triangle inequality.

Since the cost of $X^{(\kappa)}$ is at most $2 \cdot \mathsf{OPT}^{(\kappa)}_{c,r,D}$ for every $\kappa = 1, \ldots, k$; the cost of the cheapest multigraph $X^{(\kappa)}$ has cost at most the cost of $2 \cdot \mathsf{OPT}^{(\leqslant k)}_{c,r,D}$.

Complexity analysis Due to Theorem 2.1, Line 4 can be done in polynomial time. The graph operations in Lines 1, 5 and 6 can be done efficiently. □

4.3.2 2-approximation for MA-MVTSP with empty tours

Theorem 4.18. *The output of Algorithm 4.5 is a 2-approximation for MA-MVTSP with empty tours.*

Proof. The proof goes as follows.

Feasibility In Line 1 we calculate an approximate solution to the MINIMUM BOUNDED DEGREE k-COMPONENT MULTIGRAPH problem on V with $\rho(v) = 2 \cdot r(v)$ for all $v \in V$. The resulting multigraph X has at most k components and provides a degree at least $2 \cdot r(v) - 1$ for every $v \in V$. After doubling the edges, A still has at most k components, and provides an even degree of at least $4 \cdot r(v) - 2$ to every vertex $v \in V$.

Algorithm 4.5 A 2-approximation for the MA-MVTSP with empty tours

Input: A complete undirected graph $G(V, E)$, costs $c : E \to \mathbb{R}_{\geqslant 0}$ satisfying the triangle inequality, requests $r : V \to \mathbb{Z}_{\geqslant 1}$, number of agents k.

Output: A multigraph consisting of at most k components that visits each $v \in V$ a total of $r(v)$ times.

1: Run Algorithm 2.1 to obtain a multigraph X' on V with at most k components, such that $\dot{\delta}_{X'}(v) \geqslant 2 \cdot r(v) - 1$ holds for all $v \in V$.
2: Duplicate all edges in X', denote the resulting multigraph X.
3: Apply shortcuts in each component of X until the degree of every node $v \in V$ is $r(v)$, using the method in Algorithm 4.6.
4: **return** X

After taking shortcuts, the number of components will be exactly k, with exactly one depot in each component.

Each vertex $v \in V$ is visited exactly $r(v)$ times by the components.

Cost of solution The multigraph X obtained by Line 1 has cost at most $\mathsf{OPT}^{(\kappa)}_{c,r,D}$, therefore, after doubling all its edges in Line 2, the resulting multigraph A has cost at most $2 \cdot \mathsf{OPT}^{(\kappa)}_{c,r,D}$. Applying shortcuts in Line 3 cannot increase the cost of A, because the edge costs satisfy the triangle inequality.

Complexity analysis Due to Theorem 2.1, Line 1 can be done in polynomial time. The graph operations in Line 2 can be done efficiently.

□

Algorithm 2.1 constructs a multigraph with $r(V)$ edges in total, hence, after duplicating the edges in Line 5 of Algorithm 4.4 and in Line 2 of Algorithm 4.5, the resulting multigraphs have a total degree surplus of $r(V)$. Therefore, in order to eliminate this surplus and set the degree of each vertex $v \in V$ to $2 \cdot r(v)$, one has to apply shortcuts. However, doing this the greedy way would mean performing $O(r(V))$ operations, which is exponential in the input size. Moreover, one has to make sure not to split a component into two components via shortcutting. Below we show how to decrease the degree of the vertices by the needed amount efficiently.

Applying shortcuts efficiently

Let $X^\star$ be a feasible solution to any MA-MVTSP or MD-MVTSP variant. Then, let us denote the tours of the agents by $X_1, \ldots, X_k$. Depending on the problem variant, a multigraph X_i can be an empty graph or an isolated point, and two tours X_i and X_j for $i \neq j$ might have common vertices. However in all problem variants, the degree of every vertex $v \in V(X_i)$ is even, and X_i is a connected multigraph, for every $i \in 1, \ldots, k$. Therefore, we can consider every X_i as a feasible solution to an MVTSP instance on the vertex set $V(X_i)$ with requests $\dot{\delta}_{X_i}(v)/2$ for $v \in V(X_i)$. If one ought to decrease the number of visits of v by $\gamma(v)$, the surplus of v, one has to decrease the degree of v by $2 \cdot \gamma(v)$. This is possible using operations (A) and (B) in Lemma 4.9, however, arbitrarily

applying these operations might take exponentially many steps or make the graph X_i disconnected. We describe the procedure in Algorithm 4.6.

Algorithm 4.6 An efficient way of performing shortcuts

Input: A connected multigraph X with even degrees $\delta_X(v)$ and an even degree requirement $\delta(v)$ for every $v \in V(X)$.
Output: A connected multigraph X' with degrees $\delta_{X'}(v) = \delta(v)$ for every $v \in V(X)$.
1: Use **CovertToSequence** from [50] to calculate $\mathcal{C} = \{(C, \mu_C)\}$, a cycle decomposition of X.
2: Construct a multigraph A with one copy of each cycle $C \in \mathcal{C}$.
3: Calculate an Eulerian trail η in A.
4: Calculate an implicit Eulerian trail η' in X.
5: Remove the last $\gamma(v) := \delta_X(v) - \delta(v)$ occurrences of v in η', denote the resulting multigraph by X'.
6: **return** X'

The task is the following. Given a connected multigraph X on n nodes and even degrees $\delta_X(v)$ on every vertex $v \in V(X)$, symmetric, metric edge costs c and even requirements $\delta(v)$; find a multigraph X' with cost at most $\text{cost}(X)$, such that the degree of a vertex v in X' is $\delta_{X'}(v) := \delta(v)$.

Lemma 4.19. *Algorithm 4.6 runs in time polynomial in $|V(X)|$ and $\log \sum_v \gamma(v)$, where $\gamma(v) = \delta_X(v) - \delta(v)$.*

Proof. According to Grigoriev and van de Klundert [50], the multigraph X can be decomposed into $m = O(n^2)$ different cycles, with multiplicities. Moreover, their algorithm **ConvertToSequence**, presented as Algorithm 1 calculates the collection of cycles with multiplicities $\mathcal{C} = \{(C, \mu_C)\}$ in $O(n^4)$ steps; taking μ_C copies of each cycle C gives back the edges of X.

The shortcutting procedure goes similarly to the one in proof of Theorem 3.1. However, this time extra attention is needed, as the total surplus is not bounded by a polynomial of the input size. Let us construct a multigraph A on the vertex set $V(X)$ by taking *one* copy of each cycle C from $\mathcal{C}$. Now calculate an Eulerian trail η in A; this is possible to do since the degrees in A are even, moreover, this can be done in polynomial time due to the polynomial size of A using the approach of Hierholzer [38, 57]. One can obtain an (implicit) Eulerian trail η' of X from η the following way. Start at the first vertex of η, denoted by v_1, and traverse the vertices of η in order. Every time a previously unvisited vertex w is reached, traverse every cycle C containing w, μ_C times; in this case we say that a cycle C is *rooted* at w. The Eulerian trail η' of X can be described implicitly by listing they traverse cycles in order, along with the vertices they are rooted at:

$$v_1 - (C_1, \mu_{C_1}) - v_2 - (C_2, \mu_{C_2}) - \cdots - v_m - (C_m, \mu_{C_m}) \ . \tag{4.2}$$

For the sake of simplicity we index the cycles and vertices based on their traverse order in the sequence. Note that the vertices v_i might repeat multiple times, as different cycles can be rooted at the same vertex. Also note that the description in Equation (4.2) con-

sists of $O(n^2)$ elements of size $O(n \log r(V))$, therefore it can be regarded as a compact representation of η'.

Now we turn to taking shortcuts. Denote the visit surplus of a vertex v by $\gamma(v)$, that is, $\gamma(v) := \delta_X(v) - \delta(v)$. Take a vertex $v \in V$, with $\gamma(v) > 0$, and let $\mathcal{C}^{(v)} := \{(C_1^{(v)}, \mu_{C_1^{(v)}}), \ldots, (C_s^{(v)}, \mu_{C_s^{(v)}})\}$ denote the s unique cycles from $\mathcal{C}$ with multiplicities, that contain the vertex v, in the order these cycles appear in η'. Now apply shortcuts to the last $\gamma(v)$ cycles from $\mathcal{C}^{(v)}$: suppose s' is the first index, such that $\mu_{C_{s'+1}^{(v)}} + \cdots + \mu_{C_s^{(v)}} < \gamma(v)$, i.e. the last $\gamma(v)$ cycles of $\mathcal{C}^{(v)}$ consists of *some* copies of cycles $C_{s'}^{(v)}$ and *all* copies of cycles $C_{s'+1}^{(v)}, \ldots, C_s^{(v)}$. Applying one shortcut means either 1) removing self loop vv, or 2) replacing 1-1 copy of the edges uv and vw by uw for some preceding vertex u and superseding vertex w (u and w might be the same vertex). Perform these operations in X, on the edges of all copies of cycles $C_{s'+1}^{(v)}, \ldots, C_s^{(v)}$ and some copies of cycle $C_{s'}^{(v)}$, such that the total number of shortcuts taken is $\gamma(v)$. As the number of unique cycles in $\mathcal{C}$ is $O(n^2)$, decreasing the degree of vertex v by $\gamma(v)$ using operations 1) and 2) can be done by modifying $O(n^2)$ values with a total amount of $O(\log r(v))$. Lastly, calculate an Eulerian trail η' on the updated multigraph X' the way described above. Perform these steps for every vertex $v \in V$ with a positive $\gamma(v)$ value, in a total of $O(n^3 \log \gamma(v))$ steps. □

Theorem 4.20. *Algorithms 4.4 and 4.5 run in time polynomial in n, k and $\log r(V)$.*

Proof. Algorithm 4.6 runs in time polynomial in the number of vertices in the input multigraph, k and $\log \gamma(V)$, where $\gamma(V)$ is the total degree surplus. We can use Algorithm 4.6 to the components of multigraphs calculated in Line 5 of Algorithm 4.4 and in Line 2 of Algorithm 4.5, in order to decrease the total degree of each vertex v to $2 \cdot r(v)$. According to Lemma 4.19, this operation can be done in time polynomial in $|V(X_i)|$, k and $\log \gamma(V(X_i))$; performing this on all components adds up to a time complexity polynomial in n, k and $\log \gamma(V)$, which implies a time complexity polynomial in n and $\log r(V)$, since $\gamma(V) = O(r(V))$.

The other steps also take polynomial time, as shown in proofs of Theorems 4.17 and 4.18, hence the claim follows. □

4.4 Simple 4-approximations

In this section we show simple 4-approximations for the MA-MVTSP with arbitrary tours as well as for the MA-MVTSP with disjoint tours and the MD-MVTSP with disjoint tours.

MA-MVTSP with arbitrary tours

We first present a simple 3-approximation idea for the MA-MVTSP with arbitrary tours. However, during the publication of this thesis, we discovered a flaw in the analysis, as well as a counterexample for the correctness of the algorithm. For transparency, we

include the flawed algorithm, along with an explanation. Then we include a 4-approximation for the MA-MVTSP with arbitrary tours, that uses a simpler technique, and yields a slightly weaker approximation ratio.

Algorithm 4.7 An *incorrect* 3-approximation for the MA-MVTSP with arbitrary tours

Input: A complete undirected graph $G(V, E)$, costs $c : E \to \mathbb{R}_{\geqslant 0}$ satisfying the triangle inequality, requests $r : V \to \mathbb{Z}_{\geqslant 1}$, number of agents k.
Output: A multigraph, such that the edges can be partitioned into k MVTSP tours that visit each $v \in V$ a total of $r(v)$ times.

1: Calculate the solution to the transportation problem defined on G with supply and demand being equal to $r(v) - 1$, denote it by TP.
2: Calculate a minimum cost spanning forest of G consisting of k components, denote it by F, and its components $F_1, \dots, F_k$.
3: Duplicate the edges of F, and apply shortcuts to obtain k Hamiltonian cycles on the vertex sets $V(F_1), \dots, V(F_k)$; denote the resulting graph H and its components $H_1, \dots, H_k$.
4: Let $X_i := H_i$ for all $i = 1, \dots, k$ be the MVTSP tours of the agents.
5: Attach each component TP_j of TP to any cycle X_i with $V(\mathsf{TP}_j) \cap V(X_i) \neq \emptyset$.
return $X := X_1 \cup \dots \cup X_k$

Algorithm 4.7 might output to a multigraph that cannot be decomposed into m tours. Consider the case where all requirements are 1, the number of agents is $m = 2$, and F is the union of two disjoint paths. In this case the odd-degree vertices in F are the four endpoints of the paths, and a minimum cost matching might connect them into a single cycle. Such a cycle cannot be decomposed into more tours than one, therefore it is not a feasible solution to the MA-MVTSP with arbitrary tours.

Below we show a correct approach, yielding a 4-approximation.

Algorithm 4.8 A 4-approximation for the MA-MVTSP with arbitrary tours

Input: A complete undirected graph $G(V, E)$, costs $c : E \to \mathbb{R}_{\geqslant 0}$ satisfying the triangle inequality, requests $r : V \to \mathbb{Z}_{\geqslant 1}$, number of agents k.
Output: k non-empty MVTSP tours that visit each $v \in V$ a total of $r(v)$ times, or NO if no solution exists.

1: If $k > r(V)$, then **return** NO.
2: If $k > n$, then for each $v \in V$ add $r(v)$ copies of the self-loop vv to X. Take a partition $X := X_1 \cup \dots \cup X_k$ of X into k non-empty sets such that X_i contains copies of the same self-loop for each $i = 1, \dots, k$, and **return** $X := X_1 \cup \dots \cup X_k$.
3: Determine a minimum cost spanning forest F of G consisting of k components, and let $F_1, \dots, F_k$ denote its components.
4: For $i = 1, \dots, k$, if $|V(F_i)| \geqslant 2$ then let H_i denote the cycle on $V(F_i)$ obtained by duplicating the edges of F_i and applying shortcuts, otherwise let H_i consist of a single copy of the loop on the vertex in $V(F_i)$.
5: For each $v \in V$, pick an index i with $v \in V(H_i)$, and add $r(v) - 1$ copies of the self-loop vv to H_i. Denote the resulting multigraphs by X_i for $i = 1, \dots, k$.
return $X := X_1 \cup \dots \cup X_k$

Theorem 4.21. *Algorithm 4.8 provides a 4-approximation for the MA-MVTSP with arbitrary tours in time polynomial in n, k and $\log r(V)$.*

Proof. We first prove that Algorithm 4.8 constructs a feasible solution to the MA-MVTSP with arbitrary tours, and provide an upper bound to its cost. Finally, we analyze the time complexity.

Feasibility. The cycles H_i contribute a degree of 2 while the loops contribute a degree of $2 \cdot (r(v) - 1)$ to the degree of each $v \in V$. Therefore the total degree of each vertex v in $X_1 \cup \cdots \cup X_m$ is $2 \cdot r(v)$.

Cost of solution. Let us denote by $k^\star$ the number of *components* (which does not necessarily equal the number of MVTSP tours) in the optimal solution X, where $1 \leqslant k^\star \leqslant k$. An optimal solution X contains a spanning forest $F^\star$ having $k^\star$ components. Moreover, $\mathsf{cost}(F^\star)$ is at least $\mathsf{cost}(\mathrm{MSF}^{k^\star})$, where $\mathrm{MSF}^{k^\star}$ denotes a minimum cost spanning forest having $k^\star$ components. It is easy to see that $\mathsf{cost}(F) \leqslant \mathsf{cost}(\mathrm{MSF}^{k^\star})$, because F is a minimum cost spanning forest with $k \geqslant k^\star$ components. Hence $\mathsf{cost}(F) \leqslant \mathsf{cost}(F^\star) \leqslant \mathsf{cost}(X^\star)$ follows.

Since the edge costs are metric, the shortcutting operations in Line 4 do not increase the cost of the multigraph, therefore the total cost of the 'non-loop H_i's is at most $2 \cdot \mathsf{cost}(X)$. We claim that the total cost of the self-loops added in Lines 4 and 5 is also at most $2 \cdot \mathsf{cost}(X)$, this was shown in Chapter 3:

$$\sum_{v \in V} r(v) \cdot c(vv) \leqslant \sum_{v \in V} r(v) \cdot 2 \cdot c^{\min}(v) \leqslant 2 \cdot \mathsf{cost}(\mathsf{TP}^\star_{c,r}) \leqslant 2 \cdot \mathsf{cost}(X), \tag{3.1}$$

where the second inequality follows from the fact that ensuring a visit to a vertex v costs at least $c^{\min}(v)$, and one needs $r(v)$ many of such visits. This verifies the approximation ratio.

Complexity analysis. Calculating a minimum cost spanning forest consisting of k components can be done in time polynomial in n and $\log r(V)$. Throughout the algorithm, we use a compact representation of all multigraphs, this way the time and space complexity of graph operations can be bounded by $O(n^2 \log r(V))$, hence the remaining graph operations can also be done in polynomial time. □

By Lemma 4.2, an analogous result holds for the MA-MVTSP with empty tours.

Corollary 4.22. *There exists a 4-approximation algorithm for MA-MVTSP with empty tours, with running time polynomial in n, k and $\log r(V)$.*

Disjoint tours variants

The main difficulty in the problem variants with disjoint tours is that the MVTSP tours of the different agents must be vertex-disjoint. This restriction prevents us to use an approach like in Algorithms 4.1 and 4.2, where we calculated k cycles from the k components of a (constrained) spanning forest, and augmented this graph with a transportation problem solution. The transportation problem solution might contain edges with

the two endpoints being in different cycles, and thus the resulting multigraph would consist of less than k components. To prevent that from happening, we can use copies of self-loops instead of the transportation problem solution, albeit this results in a slightly weaker approximation ratio.

Algorithm 4.9 A 4-approximation for the MA-MVTSP with disjoint tours

Input: A complete undirected graph $G(V, E)$, costs $c : E \to \mathbb{R}_{\geqslant 0}$ satisfying the triangle inequality, requests $r : V \to \mathbb{Z}_{\geqslant 1}$, number of agents k.

Output: k vertex-disjoint MVTSP tours that visit each $v \in V$ a total of $r(v)$ times.

1: Calculate a minimum cost spanning forest of G consisting of k components, denote it by F, and its components $F_1, \dots, F_k$.

2: Duplicate the edges of F, and apply shortcuts to obtain k Hamiltonian cycles on the vertex sets $V(F_1), \dots, V(F_k)$; denote the resulting graph H and its components $H_1, \dots, H_k$.

3: **for** $i = 1, \dots, k$ **do**

4: Add $r(v) - 1$ copies of the self-loop vv to H_i, for each $v \in V(H_i)$, and denote the resulting multigraph X_i.

return $X := X_1 \cup \dots \cup X_k$

Theorem 4.23. *Algorithm 4.9 provides a 4-approximation for MA-MVTSP with arbitrary tours in time polynomial in n, k and $\log r(V)$.*

Proof. The number of components of F is k by definition, duplicating all edges in Line 2 and adding the self-loops in Line 4 does not introduce any edges between these components, hence H, and therefore X also consists of k components. The multigraphs $H_1, \dots, H_k$ are vertex-disjoint, and every vertex $v \in V$ has a degree of 2 in the cycle H_i traversing it. Moreover, the self-loops added in Line 4 contribute a degree of $2 \cdot (r(v) - 1)$ to each vertex $v \in V$, adding up to a total degree $2 \cdot r(v)$ in X for every $v \in V$. This means X is a feasible solution.

Let us fix an optimal solution $X^\star$. The cost of F is at most $c(X^\star)$, therefore the cost of H is at most $2 \cdot c(X^\star)$. The total cost of the self-loops added in Line 4 is also $2 \cdot c(X^\star)$, according to Equation (3.1). The approximation ratio follows.

Finally, both calculating a k-component spanning forest, and the rest of the graph operations of Algorithm 4.9 can be done in time polynomial in the input size. □

By replacing Line 1 of Algorithm 4.9 by the algorithm of Cerdeira [24], we can obtain a 4-approximation algorithm for the analogous problem with depots; please refer to Algorithm 4.10.

Theorem 4.24. *Algorithm 4.10 provides a 4-approximation for MD-MVTSP with arbitrary tours in time polynomial in n, k and $\log r(V)$.*

Proof. The feasibility and complexity analysis follows from the same arguments as in the proof of Algorithm 4.9, by adding that each depot $d \in D$ has an even degree in H and therefore in X as well.

Since an optimal solution $X^\star$ contains a spanning forest with the properties of those of F, and F is a minimum cost such spanning forest, the bound $c(F) \leqslant c(X^\star)$ and

Algorithm 4.10 A 4-approximation for the MD-MVTSP with disjoint tours

Input: A complete undirected graph $G(V, D, E)$, costs $c : E \to \mathbb{R}_{\geqslant 0}$ satisfying the triangle inequality, requests $r : V \to \mathbb{Z}_{\geqslant 1}$, number of agents $k = |D|$.
Output: k vertex-disjoint MVTSP tours each containing one depot from D and at least one vertex from V, that visit each $v \in V$ a total of $r(v)$ times.

1: Calculate a minimum cost spanning forest of G consisting of k nontrivial components with exactly one depot in each components. Denote this graph by F, and its components $F_1, \ldots, F_k$.
2: Duplicate the edges of F, and apply shortcuts to obtain k Hamiltonian cycles on the vertex sets $V(F_1), \ldots, V(F_k)$; denote the resulting graph H and its components $H_1, \ldots, H_k$.
3: **for** $i = 1, \ldots, k$ **do**
4: Add $r(v) - 1$ copies of the self-loop vv to H_i, for each $v \in V(H_i)$, and denote the resulting multigraph X_i.

return $X := X_1 \cup \cdots \cup X_k$

thus $c(H) \leqslant 2 \cdot c(X^\star)$ follows. Equation (3.1) also holds, proving that X is indeed a 4-approximation. □

Summary

In this chapter we showed a family of possible generalisations of the Many-Visits TSP introducing multiple agents. We showed that some of these problem variants are equivalent, and provide polynomial-time approximation algorithms for all of them. We started by 3-approximations for the arbitrary tours variant with depots and both empty tours variants, that build on the common idea of first 1) building a constrained spanning forest and then 2) calculating a transportation problem solution. We then provided 2-approximation algorithms for the empty tours variants, that calculate a constrained multigraph using the algorithms in Chapter 2, followed by an edge-doubling and an efficient shortcutting procedure. Finally we gave 4-approximations for arbitrary tours variant without depots and the disjoint tours variants, where we modify the approach of the 3-approximations by adding self-loops instead of a transportation problem solution; these algorithms are both simple and easy to analyse.

Discussion

In this thesis, we introduced the Many-Visits Travelling Salesman Problem, which is a generalisation of the TSP. We first presented exact algorithms for the MVTSP, the fastest of these algorithms use $O^*(5^n)$ time and space, whereas the fastest polynomial-space algorithm has its runtime bounded from above by $O(16^{n+o(n)})$. These algorithms can be adapted to solve the Many-Visits Path TSP as well, with essentially the same time and space complexities.

The Many-Visits Path TSP, with both asymmetric and symmetric edge costs admit constant factor approximations, provided the edge costs satisfy the triangle inequality. In case of asymmetric edge costs, our algorithms use the approximation algorithms for the single visit counterparts of the problems, as black box subroutines. For the symmetric Many-Visits TSP and Many-Visits Path TSP, we designed a $3/2$-approximation algorithm, that can be implemented to run in strongly polynomial time.

Lastly, we introduced and characterised different variants of the k-agent MVTSP, including variants where the k salesmen start in pre-assigned depots. When the objective is minimising the total cost of the tours, we presented constant factor approximations for all problem variants, provided symmetric and metric edge costs. We showed 4-approximations for the problem variants where the tours of the agents have to be disjoint. For the variants where we only require edge-disjointness, we presented a 4-approximation for the variant without depots and a 3-approximation for the variant with depots. We also showed simple 3-approximations for problem variants allowing for empty tours, and finally we presented 2-approximations for these problems.

Faster exact approaches for special edge weights

Recall that the best polynomial-space algorithm from Chapter 1 takes $O^*(16^{n+o(n)} + \log r)$ operations, which gradually decreases to $O^*(4^{n+o(n)})$ when the r values converge to 1. However, we do not know such improvements in efficiency for other special cases, regarding the edge costs c.

Open problem 1.

Can we obtain more efficient exact algorithms for the Many-Visits TSP with symmetric and/or metric edge costs?

In the open problems from now on, we assume that the distances in the arising graphs satisfy the triangle inequality.

Reducing the Many-Visits Path TSP to the MVTSP

A polynomial-time reduction from the TSP to the Path TSP is well-known. In a nutshell, one assumes an edge uv is in the optimal solution, then solves the arising Path TSP with starting vertex v and endvertex u. Performing these steps for every edge uv, then choosing the cheapest of the solutions will yield an optimal solution to the TSP. This reduction is *approximation-preserving*, i.e. an α-approximation for the Path TSP yields an α-approximation for the TSP. In a recent result Traub, Vygen and Zenklusen showed an approximation-preserving reduction from the Path TSP to the TSP [108], in particular they showed that an α-approximation for the TSP yields an $(\alpha + \varepsilon)$-approximation for the Path TSP.

Approximation-preserving polynomial-time reductions from the Many-Visits TSP to the Many-Visits Path TSP are shown in Corollary 3.6 and Theorem 3.3. However, the other direction remains an open question.

Open problem 2.

Is the Many-Visits Path TSP substantially harder to approximate than the Many-Visits TSP?

Improved approximation algorithms for the Multiple-Agent and Multiple-Depot MVTSP

Although we showed a $3/2$-approximation for the Many-Visits Path TSP, the best approximation ratio for any Multiple-Agent or Multiple-Depot MVTSP variant is 2. A natural question arises: can we obtain a polynomial-time α-approximation for $\alpha < 2$? In case of the single-visit counterparts of these problems, the same gap among the current best approximation ratios is present: for TSP, there exist efficient $3/2$-approximations [27, 103], whereas the best approximation ratio for the Multiple-Depot TSP is $2 - 1/k$ by Xu et al. [117], that converges to 2 if k increases.

Let X be an optimal solution to a single-visit TSP instance. The Christofides heuristic for the TSP calculates a minimum cost spanning tree T and a minimum cost perfect matching M on the odd-degree vertices of T. The cost of T can be bounded by $c(X)$, and since X decomposes into two perfect matchings the cost of M is at most $c(X)/2$. For the k-agent case, one may consider replacing T by a minimum cost spanning forest F. However, there is currently no known way of bounding the cost of M, a minimum cost matching on the odd-degree vertices of F. The difficulty lies in the fact that each tour of X might have an odd number of odd-degree vertices in F, as there is no guarantee that the components of F will coincide with those of X.

Open problem 3.

Are there polynomial-time constant-factor approximation algorithms for Multiple-Agent and Multiple-Depot MVTSP variants, with a constant strictly less than 2?

The Multiple-Depot TSP considered in Xu et al. [117] is a special case of the MD-MVTSP with empty tours, which means any improvement on the approximation factor for the latter would carry over for the former.

Xu and Rodrigues [114] provided a $3/2$-approximation for the Multiple-Depot TSP, that runs in $n^{\Omega(k)}$ time, hence it is polynomial for fixed k. Their approach is based on an exchange property of matroids, and they solve the problem of bounding the cost of the matching using an edge exchange algorithm, by swapping the edges of a minimum cost forest F. A natural question is whether a similar technique yields a $3/2$-approximation that is polynomial for a fixed number of agents.

Open problem 4.

Can we generalise the edge exchange algorithm [114] to obtain a $3/2$-approximation for the Multiple-Depot MVTSP with empty tours, that runs in $O(n^k \log r)$ time?

Path versions of Multiple-Agent MVTSP

In the path variant of the Travelling Salesman Problems with multiple agent, referred to as Multiple Depot Multiple Terminal Hamiltonian Path Problem (MDMTHPP), a set of depots D and a set of terminals T are given, and the aim is to find paths of minimal total cost, that start in separate depots from D, end separate terminals from T, and together visit all cities exactly once. For the MDMTHPP there have been 2-approximations [8, 95] and a $(2 - \frac{1}{2k+1})$-approximation [118] when allowing for empty tours.

An interesting direction of future research could be characterising the path versions of the different Multiple-Agent and Multiple-Depot MVTSP variants. The 4-approximations provided in Algorithms 4.9 and 4.10 seems to be simple enough to be adapted to 4-approximations for the Multiple-Agent and Multiple-Depot Many-Visits Path TSP. These modifications would be using Hamiltonian paths instead of cycles in Algorithm 4.9 and finding a constrained spanning forest with exactly one depot and one terminal in each tree using a matroid intersection algorithm [8, 95] in Algorithm 4.10.

However, for the other problem variants, adapting the existing algorithms is not this straightforward. The arguments in Claims 4.3 and 4.4 do not carry over for the path case, so the path versions of problems P3–P4 and P7–P8 might require approaches completely different to the ones presented in Chapter 4, in order to approximate.

Open problem 5.

Can the approaches in Algorithms 4.1–4.5, 4.7, 4.9 and 4.10 be applied for the Multiple-Agent and Multiple-Depot Many-Visits Path TSP variants?

Moreover, there has been no mention of the tour variants reducing to the path variants of these problems. The naive attempt for such a reduction would be defining the terminal set T' *and* a depot set D' of a MDMTHPP instance to be the depot set D of a MDMTSP instance, that is, $D' := D$ and $T' := D$. However, the solution to this MDMTHPP instance might not provide a feasible solution for the underlying MDMTSP

instance, as nothing prevents the paths from starting and ending in vertices that correspond to different depots in D. It is not clear whether a reduction similar to the (Many-Visits) TSP to the (Many-Visits) Path TSP exists.

Open problem 6.

Is there a reduction from some of the tour versions to the corresponding path versions of the Multiple-Agent and Multiple-Depot MVTSP variants? Is there a reduction in the opposite direction?

Minimising the longest tour in Multiple-Agent MVTSP

In Chapter 4 we considered Multiple-Agent MVTSP variants with the objective of minimising the total cost of the tours. It would be equally natural to consider minimising the longest (most expensive) tour in the solution as well. The single-visit counterparts of such problems have been referred to min-max cycle cover and min-max routing problems. The first constant-factor approximation result of the sort is due to Frederickson et al. [42], and many other followed considering problems without depots [7, 71, 113, 119], as well as with depots with different restrictions on the tours sought [35, 65, 113, 116, 119]. The approximation guarantees of these approaches range from $5/2$ to $8 + \varepsilon$.

The many-visits counterparts of these problems also have connections to scheduling theory, in particular, they can be regarded as high-multiplicity scheduling problems with sequence-depending setup times, with the objective of minimising the makespan of the schedule, commonly denoted by $C_{\max}$.

Open problem 7.

Are there constant-factor approximation algorithms for Multiple-Agent and Multiple-Depot MVTSP variants, considering the min-max objective?

BIBLIOGRAPHY

[1] ABASI, H., BSHOUTY, N. H., GABIZON, A., AND HARAMATY, E. On r-simple k-path. In *Proc. MFCS 2014* (2014), vol. 8635 of *Lecture Notes Comput. Sci.*, pp. 1–12.

[2] AGUAYO, M. M., SARIN, S. C., AND SHERALI, H. D. Single-commodity flow-based formulations and accelerated Benders algorithms for the high-multiplicity asymmetric traveling salesman problem and its extensions. *J. Oper. Res. Soc. 69*, 5 (2018), 734–746.

[3] ALLAHVERDI, A., NG, C. T., CHENG, T. C. E., AND KOVALYOV, M. Y. A survey of scheduling problems with setup times or costs. *European J. Oper. Res. 187*, 3 (2008), 985–1032.

[4] AN, H.-C., KLEINBERG, R., AND SHMOYS, D. B. Improving Christofides' algorithm for the s-t path TSP. *J. ACM 62*, 5 (2015), Art. 34, 28.

[5] APPLEGATE, D. L., BIXBY, R. E., CHVÁTAL, V., AND COOK, W. J. *The traveling salesman problem*. Princeton Series in Applied Mathematics. Princeton University Press, Princeton, NJ, 2006. A computational study.

[6] ARKIN, E. M., CHIANG, Y., MITCHELL, J. S. B., SKIENA, S., AND YANG, T. On the maximum scatter traveling salesperson problem. *SIAM J. Comput. 29*, 2 (1999), 515–544.

[7] ARKIN, E. M., HASSIN, R., AND LEVIN, A. Approximations for minimum and min-max vehicle routing problems. *J. Algorithms 59*, 1 (2006), 1–18.

[8] BAE, J., AND RATHINAM, S. Approximation algorithms for multiple terminal, Hamiltonian path problems. *Optim. Lett. 6*, 1 (2012), 69–85.

[9] BANG-JENSEN, J., AND GUTIN, G. Z. *Digraphs - theory, algorithms and applications*. Springer, 2002.

[10] BANSAL, N., KHANDEKAR, R., KÖNEMANN, J., NAGARAJAN, V., AND PEIS, B. On generalizations of network design problems with degree bounds. *Math. Prog. 141*, 1 (2013), 479–506.

[11] BANSAL, N., KHANDEKAR, R., AND NAGARAJAN, V. Additive guarantees for degree-bounded directed network design. *SIAM J. Comput. 39*, 4 (2009), 1413–1431.

[12] BEASLEY, J. E., KRISHNAMOORTHY, M., SHARAIHA, Y. M., AND ABRAMSON, D. Scheduling aircraft landings—the static case. *Transp. Sci. 34*, 2 (2000), 180–197.

[13] BELLMAN, R. Dynamic programming treatment of the travelling salesman problem. *J. Assoc. Comput. Mach. 9* (1962), 61–63.

[14] BÉRCZI, K., BERGER, A., MNICH, M., AND VINCZE, R. Degree-bounded generalized polymatroids and approximating the metric many-visits TSP. Tech. rep., 2019. https://arxiv.org/abs/1911.09890.

[15] BÉRCZI, K., MNICH, M., AND VINCZE, R. A 3/2-approximation for the metric many-visits path TSP. Tech. rep., 2020. https://arxiv.org/abs/2007.11389.

[16] BÉRCZI, K., MNICH, M., AND VINCZE, R. Efficient approximations for many-visits multiple traveling salesman problems. Tech. rep., 2022. https://arxiv.org/abs/2201.02054.

[17] BERGE, C. *Graphs and hypergraphs*. North-Holland Publishing Co., Amsterdam-London; American Elsevier Publishing Co., Inc., New York, 1973.

[18] BERGER, A. A note on the characterization of digraphic sequences. *Discrete Math. 314* (2014), 38–41.

[19] BERGER, A., KOZMA, L., MNICH, M., AND VINCZE, R. A time- and space-optimal algorithm for the many-visits TSP. In *Proc. SODA 2019* (2019), SIAM, Philadelphia, PA, pp. 1770–1782.

[20] BERGER, A., KOZMA, L., MNICH, M., AND VINCZE, R. Time- and space-optimal algorithm for the many-visits TSP. *ACM Trans. Algorithms 16*, 3 (2020), Art. 35, 22.

[21] BIANCO, L., DELL'OLMO, P., AND GIORDANI, S. Minimizing total completion time subject to release dates and sequence-dependent processing times. vol. 86. 1999, pp. 393–415.

[22] CARNES, T., AND SHMOYS, D. B. Primal-dual schema and lagrangian relaxation for the k-location-routing problem. Springer Berlin Heidelberg, 2011, pp. 99–110.

[23] CAYLEY, A. A theorem on trees. *Quart. J. Pure Appl. Math. 23* (1889), 376–378.

[24] CERDEIRA, J. O. Matroids and a forest cover problem. *Math. Programming 66*, 3, Ser. A (1994), 403–405.

[25] CHAUDHURI, K., RAO, S., RIESENFELD, S., AND TALWAR, K. A push-relabel approximation algorithm for approximating the minimum-degree MST problem and its generalization to matroids. *Theoret. Comput. Sci. 410*, 44 (2009), 4489–4503.

[26] CHAUDHURI, K., RAO, S., RIESENFELD, S., AND TALWAR, K. What would Edmonds do? Augmenting paths and witnesses for degree-bounded MSTs. *Algorithmica 55*, 1 (2009), 157–189.

[27] CHRISTOFIDES, N. Worst-case analysis of a new heuristic for the travelling salesman problem. Tech. Rep. 388, Carnegie Mellon University, 1976.

[28] COOK, W. *In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation*. Princeton University Press, 2011.

[29] COSMADAKIS, S. S., AND PAPADIMITRIOU, C. H. The traveling salesman problem with many visits to few cities. *SIAM J. Comput. 13*, 1 (1984), 99–108.

[30] DEPPERT, M. A., AND JANSEN, K. Near-linear approximation algorithms for scheduling problems with batch setup times. In *Proc. SPAA 2019* (2019).

[31] EDMONDS, J. Submodular functions, matroids, and certain polyhedra. In *Combinatorial Structures and their Applications (Proc. Calgary Internat. Conf., Calgary, Alta., 1969)*. Gordon and Breach, New York, 1970, pp. 69–87.

[32] EDMONDS, J., AND KARP, R. M. Theoretical improvements in algorithmic efficiency for network flow problems. In *Combinatorial Structures and their Applications (Proc. Calgary Internat. Conf., Calgary, Alta., 1969)*. Gordon and Breach, New York, 1970, pp. 93–96.

[33] EMMONS, H., AND MATHUR, K. Lot sizing in a no-wait flow shop. *Oper. Res. Lett. 17*, 4 (1995), 159–164.

[34] ERDŐS, P., AND GALLAI, T. Graphs with prescribed degrees of vertices (Hungarian). *Mat. Lapok 11* (1960), 264–274.

[35] EVEN, G., GARG, N., KÖNEMANN, J., RAVI, R., AND SINHA, A. Min-max tree covers of graphs. *Oper. Res. Lett. 32*, 4 (2004), 309–315.

[36] FEIGE, U., AND SINGH, M. Improved approximation ratios for traveling salesperson tours and paths in directed graphs. In *Proc. APPROX/RANDOM 2007*. Springer Berlin Heidelberg, 2007, pp. 104–118.

[37] FELLER, W. *An Introduction to Probability Theory and Its Applications*, vol. 1. Wiley, 1968.

[38] FLEISCHNER, H. *Eulerian graphs and related topics. Part 1. Vol. 2*, vol. 50 of *Annals of Discrete Mathematics*. North-Holland Publishing Co., Amsterdam, 1991.

[39] FRANK, A. Generalized polymatroids. In *Finite and infinite sets, Vol. I, II (Eger, 1981)*, vol. 37 of *Colloq. Math. Soc. János Bolyai*. North-Holland, Amsterdam, 1984, pp. 285–294.

[40] FRANK, A. *Connections in combinatorial optimization*, vol. 38 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, Oxford, 2011.

[41] FRANK, A., AND TARDOS, É. Generalized polymatroids and submodular flows. *Math. Prog. 42*, 1-3 (1988), 489–563.

[42] FREDERICKSON, G. N., HECHT, M. S., AND KIM, C. E. Approximation algorithms for some routing problems. *SIAM J. Comput. 7*, 2 (1978), 178–193.

[43] FRIEZE, A. M. An extension of Christofides heuristic to the k-person travelling salesman problem. *Discrete Appl. Math. 6*, 1 (1983), 79–83.

[44] FULKERSON, D. R. Zero-one matrices with zero trace. *Pacific J. Math. 10* (1960), 831–836.

[45] FÜRER, M., AND RAGHAVACHARI, B. Approximating the minimum-degree Steiner tree to within one of optimal. *J. Algorithms 17*, 3 (1994), 409–423.

[46] GABIZON, A., LOKSHTANOV, D., AND PILIPCZUK, M. Fast algorithms for parameterized problems with relaxed disjointness constraints. In *Proc. ESA 2015* (2015), vol. 9294 of *Lecture Notes Comput. Sci.*, pp. 545–556.

[47] GAREY, M. R., AND JOHNSON, D. S. *Computers and intractability*. W. H. Freeman and Co., San Francisco, Calif., 1979.

[48] GOEMANS, M. X. Minimum bounded degree spanning trees. In *Proc. FOCS 2006* (2006), pp. 273–282.

[49] GOTTSCHALK, C., AND VYGEN, J. Better s-t-tours by Gao trees. *Math. Prog. 172*, 1-2, Ser. B (2018), 191–207.

[50] GRIGORIEV, A., AND VAN DE KLUNDERT, J. On the high multiplicity traveling salesman problem. *Discrete Optim. 3*, 1 (2006), 50–62.

[51] GUREVICH, Y., AND SHELAH, S. Expected computation time for Hamiltonian path problem. *SIAM J. Comput. 16*, 3 (1987), 486–502.

[52] GUTIN, G., AND PUNNEN, A. P., Eds. *The Traveling Salesman Problem and Its Variations*. Springer US, 2007.

[53] HAKIMI, S. L. On realizability of a set of integers as degrees of the vertices of a linear graph. I. *J. Soc. Indust. Appl. Math. 10* (1962), 496–506.

[54] HASSIN, R. Minimum cost flow with set-constraints. *Networks 12*, 1 (1982), 1–21.

[55] HAVEL, V. Eine Bemerkung über die Existenz der endlichen Graphen. *Časopis Pěst. Mat. 80* (1955), 477–480.

[56] HELD, M., AND KARP, R. M. A dynamic programming approach to sequencing problems. *J. Soc. Indust. Appl. Math. 10* (1962), 196–210.

[57] HIERHOLZER, C., AND WIENER, C. Ueber die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren. *Math. Ann. 6*, 1 (1873), 30–32.

[58] HITCHCOCK, F. L. The distribution of a product from several sources to numerous localities. *J. Math. Phys. Mass. Inst. Tech. 20* (1941), 224–230.

[59] HOOGEVEEN, J. A. Analysis of Christofides' heuristic: some paths are more difficult than cycles. *Oper. Res. Lett. 10*, 5 (1991), 291–295.

[60] IMPAGLIAZZO, R., AND PATURI, R. On the complexity of k-SAT. vol. 62. 2001, pp. 367–375. Special issue on the Fourteenth Annual IEEE Conference on Computational Complexity (Atlanta, GA, 1999).

[61] IMPAGLIAZZO, R., PATURI, R., AND ZANE, F. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci. 63*, 4 (2001), 512–530.

[62] IWATA, S., FLEISCHER, L., AND FUJISHIGE, S. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *J. ACM 48*, 4 (2001), 761–777.

[63] JAIN, K. A factor 2 approximation algorithm for the generalized Steiner network problem. *Combinatorica 21*, 1 (2001), 39–60.

[64] JANSEN, K., KLEIN, K.-M., MAACK, M., AND RAU, M. Empowering the configuration-IP—new PTAS results for scheduling with setups times. In *Proc. ITCS 2019*, vol. 124 of *Leibniz Int. Proc. Inform.* 2019, pp. Art. No. 44, 19.

[65] JORATI, A. Approximation algorithms for some min-max vehicle routing problems.

[66] JORDAN, C. Sur les assemblages de lignes. *Journal für die reine und angewandte Mathematik 70* (1869), 185–190.

[67] KANNAN, R. Improved algorithms for integer programming and related lattice problems. In *Proc. STOC 1983* (1983), pp. 193–206.

[68] KAPOOR, S., AND RAMESH, H. Algorithms for enumerating all spanning trees of undirected and weighted graphs. *SIAM J. Comput. 24*, 2 (1995), 247–265.

[69] KARGER, D. R. Global min-cuts in RNC, and other ramifications of a simple min-cut algorithm. In *Proc. SODA 1993* (1993), pp. 21–30.

[70] KHACHAY, M., AND NEZNAKHINA, K. Approximability of the minimum-weight k-size cycle cover problem. *J. Global Optim. 66*, 1 (2016), 65–82.

[71] KHANI, M. R., AND SALAVATIPOUR, M. R. Improved approximation algorithms for the min-max tree cover and bounded tree cover problems. *Algorithmica 69*, 2 (2014), 443–460.

[72] KIM, H., TOROCZKAI, Z., ERDŐS, P. L., MIKLÓS, I., AND SZÉKELY, L. A. Degree-based graph construction. *J. Phys. A 42*, 39 (2009), 392001, 10.

[73] KIRÁLY, T., LAU, L. C., AND SINGH, M. Degree bounded matroids and submodular flows. *Combinatorica 32*, 6 (2012), 703–720.

[74] KLEINSCHMIDT, P., AND SCHANNATH, H. A strongly polynomial algorithm for the transportation problem. *Math. Program. 68* (1995), 1–13.

[75] KLEITMAN, D. J., AND WANG, D. L. Algorithms for constructing graphs and digraphs with given valences and factors. *Discrete Math. 6* (1973), 79–88.

[76] KÖNEMANN, J., AND RAVI, R. A matter of degree: Improved approximation algorithms for degree-bounded minimum spanning trees. *SIAM J. Comput. 31*, 6 (2002), 1783–1793.

[77] KÖNEMANN, J., AND RAVI, R. Primal-dual meets local search: approximating MST's with nonuniform degree bounds. In *Proc. STOC 2003* (2003), pp. 389–395.

[78] KOWALIK, L. U., LI, S., NADARA, W., SMULEWICZ, M., AND WAHLSTRÖM, M. Many-visits TSP revisited. *J. Comput. System Sci. 124* (2022), 112–128.

[79] KOZMA, L., AND MÖMKE, T. Maximum scatter TSP in doubling metrics. In *Proc. SODA 2017* (2017), pp. 143–153.

[80] KRUSKAL, JR., J. B. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proc. Amer. Math. Soc. 7* (1956), 48–50.

[81] LAMPIS, M. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica 64*, 1 (2012), 19–37.

[82] LAWLER, E., SHMOYS, D., KAN, A., AND LENSTRA, J. *The Traveling Salesman Problem*. John Wiley & Sons, 1985.

[83] LENSTRA, JR., H. W. Integer programming with a fixed number of variables. *Math. Oper. Res. 8*, 4 (1983), 538–548.

[84] LIEDER, A., BRISKORN, D., AND STOLLETZ, R. A dynamic programming approach for the aircraft landing problem with aircraft classes. *European J. Oper. Res. 243*, 1 (2015), 61–69.

[85] MACKAY, D. J. C. *Information theory, inference and learning algorithms*. Cambridge University Press, New York, 2003.

[86] MALIK, W., RATHINAM, S., AND DARBHA, S. An approximation algorithm for a symmetric generalized multiple depot, multiple travelling salesman problem. *Oper. Res. Lett. 35*, 6 (2007), 747–753.

[87] MCCORMICK, S. T. Submodular function minimization. *Handbooks in operations research and management science 12* (2005), 321–391.

[88] MOON, J. W. *Counting labelled trees*, vol. 1969 of *From lectures delivered to the Twelfth Biennial Seminar of the Canadian Mathematical Congress (Vancouver*. Canadian Mathematical Congress, Montreal, Que., 1970.

[89] NAGAMOCHI, H., NISHIMURA, K., AND IBARAKI, T. Computing all small cuts in an undirected network. *SIAM J. Discrete Math. 10*, 3 (1997), 469–481.

[90] NIJENHUIS, A., AND WILF, H. S. *Combinatorial algorithms*, second ed. Academic Press, Inc. [Harcourt Brace Jovanovich, Publishers], New York-London, 1978.

[91] NISHIZEKI, T., AND CHIBA, N. *Planar graphs: theory and algorithms*, vol. 140 of *North-Holland Mathematics Studies*. North-Holland Publishing Co., Amsterdam, 1988. Annals of Discrete Mathematics, 32.

[92] ORLIN, J. B. A faster strongly polynomial minimum cost flow algorithm. *Oper. Res. 41*, 2 (1993), 338–350.

[93] PAPADIMITRIOU, C. H., AND YANNAKAKIS, M. On minimal eulerian graphs. *Inf. Proc. Lett. 12*, 4 (1981), 203–205.

[94] PSARAFTIS, H. N. A dynamic programming approach for sequencing groups of identical jobs. *Oper. Res. 28*, 6 (1980), 1347–1359.

[95] RATHINAM, S., AND SENGUPTA, R. Lower and upper bounds for a multiple depot UAV routing problem. In *Proceedings of the 45th IEEE Conference on Decision and Control* (2006), IEEE.

[96] RATHINAM, S., SENGUPTA, R., AND DARBHA, S. A resource allocation algorithm for multivehicle systems with nonholonomic constraints. *IEEE Trans. Autom. Sci. Eng. 4*, 1 (2007), 98–104.

[97] ROTHKOPF, M. Letter to the editor–the traveling salesman problem: On the reduction of certain large problems to smaller ones. *Oper. Res. 14*, 3 (1966), 532–533.

[98] SARIN, S. C., SHERALI, H. D., AND YAO, L. New formulation for the high multiplicity asymmetric traveling salesman problem with application to the Chesapeake problem. *Optim. Lett. 5*, 2 (2011), 259–272.

[99] SCHRIJVER, A. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *J. Combinatorial Theory, Ser. B 80*, 2 (2000), 346–355.

[100] SCHRIJVER, A. *Combinatorial optimization. Polyhedra and efficiency. Vol. B*, vol. 24 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin, 2003. Matroids, trees, stable sets, Chapters 39–69.

[101] SEBŐ, A. Eight-fifth approximation for the path TSP. In *Proc. IPCO 2013*, vol. 7801 of *Lecture Notes Comput. Sci.* 2013, pp. 362–374.

[102] SEBŐ, A., AND VAN ZUYLEN, A. The salesman's improved paths through forests. *J. ACM 66*, 4 (2019), Art. 28, 16.

[103] SERDYUKOV, A. I. Some extremal bypasses in graphs. *Upravlyaemye Sistemy*, 17 (1978), 76–79, 89.

[104] SINGH, M., AND LAU, L. C. Approximating minimum bounded degree spanning trees to within one of optimal. *J. ACM 62*, 1 (2015), Art. 1, 19.

[105] SVENSSON, O., TARNAWSKI, J., AND VÉGH, L. A. A constant-factor approximation algorithm for the asymmetric traveling salesman problem. In *Proc. STOC 2018*. 2018, pp. 204–213.

[106] TRAUB, V., AND VYGEN, J. Approaching $\frac{3}{2}$ for the s-t-path TSP. *J. ACM 66*, 2 (2019), Art. 14, 17.

[107] TRAUB, V., AND VYGEN, J. An improved approximation algorithm for ATSP. In *Proc. STOC 2020* (2020).

[108] TRAUB, V., VYGEN, J., AND ZENKLUSEN, R. Reducing path TSP to TSP. In *Proc. STOC 2020* (2020), ACM.

[109] VAN DER VEEN, J. A. A., WOEGINGER, G. J., AND ZHANG, S. Sequencing jobs that require common resources on a single machine: a solvable case of the TSP. *Math. Prog. 82* (1998), 235–254.

[110] VAN DER VEEN, J. A. A., AND ZHANG, S. Low-complexity algorithms for sequencing jobs with a fixed number of job-classes. *Comput. Oper. Res. 23*, 11 (1996), 1059–1067.

[111] VYGEN, J. Reassembling trees for the traveling salesman. *SIAM J. Discrete Math. 30*, 2 (2016), 875–894.

[112] WOLSEY, L. A. Heuristic analysis, linear programming and branch and bound. *Math. Programming Stud.*, 13 (1980), 121–134.

[113] XU, W., LIANG, W., AND LIN, X. Approximation algorithms for min-max cycle cover problems. *IEEE Trans. Comput. 64*, 3 (2015), 600–613.

[114] XU, Z., AND RODRIGUES, B. A 3/2-approximation algorithm for the multiple TSP with a fixed number of depots. *INFORMS J. Comput. 27*, 4 (2015), 636–645.

[115] XU, Z., AND RODRIGUES, B. An extension of the Christofides heuristic for the generalized multiple depot multiple traveling salesmen problem. *European J. Oper. Res. 257*, 3 (2017), 735–745.

[116] XU, Z., XU, D., AND ZHU, W. Approximation results for a min-max location-routing problem. *Discrete Appl. Math. 160*, 3 (2012), 306–320.

[117] XU, Z., XU, L., AND RODRIGUES, B. An analysis of the extended Christofides heuristic for the k-depot TSP. *Oper. Res. Lett. 39*, 3 (2011), 218–223.

[118] YANG, Y., AND LIU, Z. Approximating the multiple-depot multiple-terminal Hamiltonian path problem. *Discrete Optim. 34* (2019), 100545, 10.

[119] YU, W., AND LIU, Z. Improved approximation algorithms for some min-max and minimum cycle cover problems. *Theoret. Comput. Sci. 654* (2016), 45–58.

[120] ZENKLUSEN, R. Matroidal degree-bounded minimum spanning trees. In *Proc. SODA 2012* (2012), pp. 1512–1521.

[121] ZENKLUSEN, R. A 1.5-Approximation for Path TSP. In *Proc. SODA 2019* (2019), pp. 1539–1549.