Dipl.-Ing. Rostislav Staněk

Problems on tours and trees in combinatorial optimization

PHD THESIS

written to obtain the academic degree of

a Doctor of Technical Sciences; Dr. techn.

submitted at the

Graz University of Technology

Supervisor:
Ao.Univ.-Prof. Dipl.-Ing. Dr.techn. Eranda Dragoti-Çela

Institute of Discrete Mathematics

Co-Supervisor:
Ao.Univ.-Prof. Dipl.-Ing. Dr.techn. Ulrich Pferschy

Department of Statistics and Operations Research
University of Graz

Graz, February 2016

EIDESSTATTLICHE ERKLÄRUNG

AFFIDAVIT

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Dissertation identisch.

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present doctoral dissertation.

Datum/Date	Unterschrift/Signature

Contents

List of Figures

List of Tables

List of Algorithms

Acknowledgements

I would like to thank my supervisor Eranda Dragoti-Çela for her patient support throughout my doctoral studies. She encouraged me to freely choose the topics this thesis is composed of. Moreover, she collaborated on my research and actively discussed my progress in detail.

My special thanks also go to my co-supervisor Ulrich Pferschy. He constantly supported my research work and provided many hints which helped during the creation of this thesis. Furthermore, he contributed to the presented research through active collaboration.

I would also like to thank Joachim Schauer for his tireless co-operation. He also helped me to experience many techniques I could apply in my proofs.

My special thanks are extended to all other co-authors of the publications included in this thesis. In particular I am grateful to Anja Fischer, Oswin Aichholzer, Alexander Pilz and Johannes Fabian Meier.

I would also like to thank the developers of the SCIP MIP-solver from the Konrad-Zuse-Zentrum für Informationstechnik Berlin. Especially Gerald Gamrath provided valuable support.

Special thanks go to Franz Rendl and Abraham P. Punnen for the time they invested into refereeing this thesis.

Last but not least I would like to thank many people who have influenced me on my path to a doctoral degree in mathematics. In particular Zdeněk Koňařík, Milena Wachtlová, Jan Čepička and Johannes Hatzl. Moreover, I would like to thank all people who supported me during my doctoral studies and during the last years including Gerald Senarclens de Grancy and Gernot Lechner.

I am also deeply thankful to my family and to Patricia Kolb who patiently supported me and sacrificed time with me so that I could finish this project.

Finally, I would like to express my special thanks: The research was funded by the Austrian Science Fund (FWF): P23829.

1. Introduction

This PhD thesis focuses on so-called *combinatorial optimization problems*. Given a finite set $E = \{1, 2, \ldots, n\}$, a subset $\mathscr{F} \subseteq 2^E$ and a real function $f\colon 2^E \to \mathbb{R}$, a **combinatorial optimization problem** asks for a **solution** $S \in \mathscr{F}$ that minimizes/maximizes the function f, i.e.

$$\min/\max \quad f(S) \quad \text{subject to } S \in \mathscr{F}. \tag{1.1}$$

The set E is called **ground set** and the set $\mathscr{F}$ is called the **set of feasible solutions**. f is called the **objective function**. This thesis focuses on *graph problems*, especially on the problems dealing with *tours* and *trees*.

Of course, there are many possibilities of categorizing *combinatorial optimization problems*. We can e.g. divide the problems into two groups depending whether the objective function value has to be minimized or maximized. Another classification of *combinatorial optimization problems* is based on the *computational complexity*. The concepts we introduce in the following are well known to the combinatorial optimization community. Therefore we just refer to the famous books of Korte and Vygen [32], of Ausiello et al [7] and of Schrijver [49] for future details and exact definitions. Roughly spoken, there exist problems which are "easily solvable" and others which are most likely "hard to solve". In the terms of mathematics, the "easy" problems belong to the complexity class $\mathcal{P}$ while the hard ones belong to $\mathcal{NP}$*-hard*. The question whether $\mathcal{P} \stackrel{?}{=} \mathcal{NP}$ represents one of the most famous open problems of the last century which still remains unsolved. Of course many complexity subclasses exist that make this classification more specific. One such class are the problems having *constant-factor approximation algorithms*.

This thesis has two main research goals.

(1) On the one hand, it estimates whether special cases of well known and widely studied combinatorial optimization problems differ in their complexity status from the general case.

(2) On the other hand, we solve selected $\mathcal{NP}$-hard problems to optimality by using ILP solvers and try to speed up the solution process by exploiting some combinatorial and polyhedral properties of the underlying

problems. The results are strongly backed by thorough computational experiments.

The PhD thesis is organized as follows. Chapter 2 deals with *"trees"*. The *data arrangement problem on regular trees* (*DAPT*) consists in assigning the vertices of a given graph G, called the *guest graph*, to the leaves of a d-regular tree T, called the *host graph*. This is done such that the sum of the pairwise distances of all pairs of leaves in T which correspond to the edges of G is minimized. The problem was first considered by Luczak and Noble [35] who have shown that the DAPT is $\mathcal{NP}$-hard for every fixed $d \geq 2$. Staněk [51] and later Çela and Staněk [10] examined the problem from a computational point of view. They introduced a lower bound and some heuristics which were tested on (i) adopted *linear arrangement problem* (*LAP*) instances, (ii) random instances and (iii) a class of instances based on easily solvable special cases. The last test instance group was the main motivation for this part of this thesis.

We start by focusing on the special case of the DAPT where both the guest and the host graph are binary regular trees and provide a $\frac{203}{200}$-approximation algorithm for this special case. The solution produced by the algorithm and the corresponding value of the objective function are given in closed form. The analysis of the approximation algorithm involves an auxiliary problem which is interesting on its own, namely the *k-balanced partitioning problem* (*k-BPP*) for binary regular trees and particular choices of k. We find a solution algorithm for the latter problem and provide a formula yielding the objective function value. Moreover, we estimate a lower bound for it and subsequently, we obtain a lower bound for the original problem by solving h_G instances of the k-BPP, where h_G is the height of the host graph G.

Furthermore, we generalize the introduced algorithm to another special case of the DAPT. In this case, both the guest and the host graph are d-regular trees for some fixed $d \geq 2$. Thereafter, we provide a weaker but even more general lower bound. It is based on the *partitioning problem into sets of bounded cardinality* (*PPSBC*) and it leads to a proof of a $\frac{585}{392}$-approximation ratio for the introduced algorithm.

Finally, we show that the DAPT remains $\mathcal{NP}$-hard even if the guest graph is a tree. This issue was posed as an open question by Luczak and Noble [35].

Results of this chapter are based on joint work with Eranda Çela and Joachim Schauer. They were published in arXiv and are submitted to Algorithmica for publication [11].

The rest of this thesis—Chapters 3 and 4—focuses on one of the most prominent combinatorial optimization problems, namely the *traveling salesman problem* (*TSP*), and its variants. All discussed problems consist of finding an optimal *tour* with respect to some pre-specified objective function.

In Chapter 3 we deal with the TSP. Given a complete graph $G = (V, E)$ and non-negative distances d for every edge, the TSP asks for a shortest tour through all vertices with respect to the distances d. The method of choice for solving the TSP to optimality is a *branch-and-cut approach*. Usually the *integrality constraints* are relaxed first and all separation processes to identify violated inequalities are done on *fractional solutions*.

In our approach we try to exploit the impressive performance of current ILP solvers and work only with integer solutions without ever interfering with fractional solutions. We stick to a very simple ILP model. First, we relax the *subtour elimination constraints* only and solve the resulting problem to integer optimality. The obtained solution corresponds to a 2-matching containing one or more cycles. These cycles can be found by a simple scan and we include a subtour elimination constraint for each such cycle and subsequently resolve the enlarged ILP model. This process is repeated until a feasible TSP solution is found.

In order to speed up the algorithm, we pursue several attempts to find as many *relevant* subtours as possible. These attempts are based on the clustering of vertices with additional insights gained from empirical observations and random graph theory. Computational results are performed on test instances taken from the *TSPLIB95* and on *random Euclidean graphs*.

At the end of this chapter some theoretical results and further empirical observations for random Euclidean graphs are presented.

The chapter is based on a joint work with Ulrich Pferschy. The first results were published in proceedings of the MATCOS-13 conference [42]. The final version was accepted for publication in Central European Journal of Operations Research [44]. In addition, an extended version is available in arXiv [43].

The results of Chapter 3 were the motivation for their application on other variants of the TSP which are focused in Chapter 4. The *symmetric quadratic traveling salesman problem* (*SQTSP*) associates a cost value with every three vertices traversed in succession. If the vertices correspond to points in the Euclidean plane and the costs are given as the turning angles of the tour, we speak of the *angular-metric traveling salesman problem* (*AngleTSP*).

In this chapter, we consider the SQTSP mainly from a computational point of view. In particular, we adopt the basic algorithmic idea used for the TSP and perform the separation of the classical subtour elimination constraints on integral solutions only. It turns out that this approach beats the standard fractional separation procedure known from the literature. We also test more advanced subtour elimination constraints introduced by Fischer and Helmberg [21] both for the integral and the fractional separation procedures, but these turn out to slow down the computation. In addition, we provide a completely different, mathematically interesting MILP linearization for the AngleTSP. It introduces only a linear number of additional variables while the standard linearization requires a cubic number. However, this theoretical advantage does not carry over to the computational results.

Finally, we deal with the maximization variant *MaxSQTSP*. In contrast to the minimization counterpart it turns out that introducing some of the stronger subtour elimination constraints by Fischer and Helmberg [21] now outperforms the standard approaches. For the special case of *MaxAngleTSP* we can observe an interesting split: For an odd number of vertices it can be shown that the sum of inner turning angles in an optimal solution always equals π. This implies that the problem can be solved by the standard ILP model without producing any integral subtours. Moreover, we can characterize the structure of an optimal solution and give a simple constructive polynomial time algorithm to find such an optimal solution. If the number of vertices is even, no such result exists.

The chapter is based on a joint work with Oswin Aichholzer, Anja Fischer, Johannes Fabian Meier, Ulrich Pferschy and Alexander Pilz. An extended abstract pointing out the computational results and the new linearization was submitted to *Cologne Twente Workshop 2016*.

At the end of the thesis, some final notes, conclusions and questions for future research are provided in Chapter 5.

2. The data arrangement problem on *d*-regular trees*

Given an undirected graph $G = \big(V(G), E(G)\big)$ with $\big|V(G)\big| = n$, an undirected graph $H = \big(V(H), E(H)\big)$ with $|V(H)| \geq n$ and some subset B of the vertex set of H, $B \subseteq V(H)$ with $|B| \geq n$, the **generic graph embedding problem (GEP**) consists of finding an injective embedding of the vertices of G into the vertices in B such that some prespecified objective function is minimized. Throughout this chapter we will call G **the guest graph** and H **the host graph**. A commonly used objective function maps an embedding $\phi\colon V(G) \to B$ to

$$\sum_{(i,j)\in E(G)} d\big(\phi(i), \phi(j)\big), \tag{2.1}$$

where $d(x, y)$ denotes the length of the shortest path between x and y in H. The host graph H may be a weighted or a non-weighted graph; in the second case the path lengths coincide with the respective number of edges. Given a non-negative number $A \in \mathbb{R}$, the decision version of the GEP asks whether there is an injective embedding $\phi\colon V(G) \to B$ such that the objective function does not exceed A.

Different versions of the GEP have been studied in the literature; the **linear arrangement problem**, where the guest graph is a one dimensional equidistant grid with n vertices is probably the most prominent among them (see Chung [12], Juvan and Mohar [30], Shiloach [50]).

This chapter deals with the version of the GEP where the guest graph G has n vertices, the host graph H is a complete d-regular tree of height $\lceil \log_d n \rceil$ and the set B consists of the leaves of H. From now on we will denote the host graph by T. The height of T as specified above guarantees that the number $|B|$ of leaves fulfills $|B| \geq n$ and that the number of the direct predecessors of the leaves in T is smaller than n. Thus $\lceil \log_d n \rceil$ is the

*Joint work with Eranda Çela and Joachim Schauer [11].

smallest height of a d-regular tree which is able to accommodate an injective embedding of the vertices of the guest graph on its leaves. This problem is originally motivated by real problems in communication systems and was first posed by Luczak and Noble [35].

We call the above described version of the GEP the **data arrangement problem on regular trees (DAPT)**. Luczak and Noble [35] have shown that the DAPT is $\mathcal{NP}$-hard for every fixed $d \geq 2$ and have posed as an open question the computational complexity of the DAPT in the case where the guest graph is a tree. We answer this question and show that this particular case of the problem is $\mathcal{NP}$-hard for every $d \geq 2$. In the special case where both the guest graph G and the host graph T are binary regular trees we give a $\frac{203}{200}$-approximation algorithm. Finally, we generalize the introduced algorithm and prove its $\frac{585}{392}$-approximation ratio for the special case where both the guest graph G and the host graph H are d-regular trees with $d \geq 3$.

This chapter is organized as follows. Section 2.1 discusses some general properties of the problem and introduces the notation used throughout the chapter. Section 2.2 presents an algorithm for the DAPT on binary regular trees, where the guest graph is also a binary regular tree. Section 2.3 deals with the *k-balanced partitioning problem* (k-BPP) in binary regular trees. This version of the k-BPP serves as an auxiliary problem in the sense that it leads to a lower bound for the objective function value of the DAPT on binary regular trees. In Section 2.4 we use the auxiliary problem and the lower bound mentioned above to analyze the algorithm presented in Section 2.2 and show that the latter is an $\frac{203}{200}$-approximation algorithm. Moreover, in Section 2.5 we generalize the introduced approximation algorithm for d-regular trees, where $d \geq 3$. Finally, in Section 2.6 it is proven that the DAPT is $\mathcal{NP}$-hard for every $d \geq 2$ even if the guest graph is a tree.

2.1 Notations and general properties of the DAPT

First, we formally define a ***d*-regular tree** as follows:

Definition 2.1 (***d*-regular tree).** *A tree $T = \big(V(T), E(T)\big)$ is called a **d-regular tree**, $d \in \mathbb{N}$, $d \geq 2$, if*

(1) it contains a specific vertex $v_1 \in V$ of degree d which is called the **root** *of T and is also denoted by $r(T)$,*

(2) every vertex but the leaves and the root has degree $d+1$ and

(3) there is a number $h \in \mathbb{N}$ such that the length $d(l, v_1)$ of the path between the root v_1 and a leaf l equals h for every leaf l of T.

The number h is called the **height** *of the tree T, and is also denoted by $h(T)$. For every vertex $v \in V \backslash \{v_1\}$, i.e. for any vertex v but the root v_1, the unique neighbor of v in the path between v_1 and v in T is called the* **father** *of v. All other neighbors of v (if any) are called the* **children of** *v. The neighbors of the root v_1 are called* **children of** *v_1. The* **level of a vertex** *v, denoted by level(v), is the length (i.e. the number of edges) of the unique path joining v and the root v_1 of the tree. Thus in a d-regular tree of height h the level of each leaf equals h, whereas the level of the root v_1 equals 0. All vertices w, $w \neq v$, of the unique path joining v and the root v_1 of the tree are called* **ancestors of** *v. Given two vertices v and u their* **most recent common ancestor** *w is their common ancestor with the highest level, i.e. $w = arg\,max\{level(t) \colon t$ is a common ancesteor of v and $u\}$.*

A **subtree of k-th order** *of a d-regular tree T is a d-regular subtree T' of T of height $h(T') = h(T) - k$, rooted at some vertex of level k in T. A subtree of first order will be called a* **basic subtree**.

Consider a guest graph $G = (V, E)$ with n vertices, and a host graph T which is a d-regular tree of height h, $h := \lceil \log_d n \rceil$. Let B be the set of leaves of T. Notice that due to the above choice of h we get the following upper bound for the number $b = |B|$ of leaves:

$$b := |B| = d^h = d^{h-1}d < nd. \tag{2.2}$$

Definition 2.2 (data arrangement problem on regular trees). *Given a guest graph $G = (V, E)$ with $|V| = n$ and a host graph T which is a d-regular tree with set of leaves B and height equal to $\lceil \log_d n \rceil$, an* **arrangement** *is an injective mapping $\phi \colon V \to B$. The* **data arrangement problem on regular trees (DAPT)** *asks for an arrangement ϕ that minimizes the objective value $OV(G, d, \phi)$*

$$OV(G, d, \phi) := \sum_{(u,v)\in E} d_T\big(\phi(u), \phi(v)\big), \tag{2.3}$$

where $d_T\big(\phi(u), \phi(v)\big)$ denotes the length of the unique $\phi(u)$-$\phi(v)$-path in the d-regular tree T. Such an arrangement is called an **optimal arrangement**. *The corresponding value of the objective functions is called the* **optimal value** *of the problem. An instance of the DAPT is fully determined by the guest graph and the parameter d of the d-regular tree T which serves as a host graph. Such an instance of the problem will be denoted by $DAPT(G, d)$ and its optimal value will be denoted by $OPT(G, d)$.*

Theorem 2.3. *The DAPT is $\mathcal{NP}$-hard for every fixed $d \geq 2$.*

Proof. See Luczak and Noble [35]. □

Example 2.1. *A guest graph G of height 3 is shown in Figure 2.1. Figure 2.2 represents the same guest graph G, but with another coloring of its vertices; the role of the coloring will be explained below. Figures 2.3 and 2.4 depict a feasible ϕ arrangement and an optimal arrangement ϕ_A of G, yielding the objective function values $OV(G, 2, \phi) = 58$ and $OV(G, 2, \phi_A) = 56$, respectively.*

Note that the labels in the vertices of the guest graphs denote the index of the vertices in the so-called canonical ordering (defined below). The labels of the leaves in the host graphs represent the arrangement: The label of each leaf coincides with the index of the vertex arranged at that leaf (in the canonical ordering).

The colors should help to capture some properties of the arrangement at a glance: The set of vertices of a certain color in the guest graph is arranged at the set of leaves of the same color in the host graph T. Moreover, some of the vertices in the guest graph have a dashed boundary, the others have a solid boundary. The graphical representation of an arrangement preserves the boundary property in the sense that vertices with a dashed boundary in G are arranged at dashed-boundary leaves of the same color in T. The same principle holds for dotted boundaries.

Definition 2.4 (canonical order). *The canonical order of the vertices of the guest graph and the canonical order of the leaves of the host graph are defined recursively as follows.*

(a) The **canonical order of the leaves of a d-regular tree** *T is an arbitrary but fixed order if $h(T) = 1$. If $h(T) > 1$ then an order of the leaves is called canonical if (i) it implies a canonical order of the leaves of every basic subtree of T, and (ii) for an arbitrary but fixed order of the children $ch_1, \ldots, ch_d$ of the root $r(T)$ of T all leaves of the basic subtree rooted at ch_i precede all leaves of the basic subtree rooted at ch_j, for $i < j$, $i, j \in \{1, 2, \ldots, d\}$, in this order.*

(b) A **canonical ordering of the vertices of a d-regular tree** *T is the unique order if $h(T) = 0$. If $h(T) \geq 1$, a canonical order of the vertices of T is an order obtained by extending the canonical order of the vertices of the d-regular tree T' of height $h(T') = h(T) - 1$ obtained from T by removing all of its leaves and fulfilling the following two properties: (i) all vertices of T' precede the leaves of T, and (ii) for*

any two leaves a and b of T', if a precedes b, then all children of a in T precede all children of b in T.

If the leaves of a d-regular tree T are ordered according to the canonical order as above, then the pairwise distances between them are given by a simple formula.

Observation 2.5. *Let T be a d-regular tree of height $h := h(T)$ and let its b leaves be labeled according to the canonical order $b_1 \prec b_2 \prec \ldots \prec b_b$. Then the distances between the leaves in T are given as $d_T(b_i, b_j) = 2l$, where*

$$l := \min\left\{k \in \{1, 2, \ldots, h\} \colon \left\lfloor \frac{i-1}{d^k} \right\rfloor = \left\lfloor \frac{j-1}{d^k} \right\rfloor \right\}, \tag{2.4}$$

for all $i, j \in \{1, 2, \ldots, b\}$. If vertex u is the most recent common ancestor of b_i and b_j, then $h - l = level(u)$.

Proof. See Çela and Staněk [10]. □

In this chapter we deal with the special case where both the guest graph G and the host graph T are d-regular trees, where $d \geq 2$; an instance of this problem is fully specified by the guest graph G and will be denoted by $DAPT(G, d)$. From now on we denote by h_G the height of the guest graph G and by h the height of the host graph T. Moreover we will always use the canonical order $v_1 \prec v_2 \prec \ldots \prec v_n$ of the vertices v_i, $1 \leq i \leq n$, $n := |V(G)|$, of the guest graph, and the canonical order $b_1 \prec b_2 \prec \ldots \prec b_b$ of the b leaves of the host graph T as in the observation above. See e.g. Figure 2.1 for an illustration of the canonical order of the vertices of a regular tree of height 3; for simplicity we specify the indices i, $1 \leq i \leq 15$ instead of the labels v_i, $1 \leq i \leq 15$.

In the following we list some obvious equalities which will be used through the rest of this chapter.

Observation 2.6.

$$n = \frac{d^{h_G+1} - 1}{d - 1} \tag{2.5}$$

$$h = \left\lceil \log_d \left(\frac{d^{h_G+1} - 1}{d - 1} \right) \right\rceil = h_G + 1 \tag{2.6}$$

$$b = d^{h_G+1} = d \cdot d^{h_G} \tag{2.7}$$

2.2 An approximation algorithm for binary trees

Let us assume that $d = 2$ in the following three sections. First, we can simplify the equations in Observation 2.6 and get

$$n = 2^{h_G+1} - 1, \tag{2.8}$$

$$b = 2 \cdot 2^{h_G} = n + 1. \tag{2.9}$$

In this section we describe a recursive approximation algorithm $\mathcal{A}$ for the $DAPT(G, 2)$ where the guest graph G is a binary regular tree. Later in Section 2.4 it will be shown that this is an α-approximation algorithm with $\alpha = \frac{203}{200}$, i.e. $OV(G, 2, \phi_A) \leq \alpha OV(G, 2, \phi_*)$ holds for every binary regular tree G, where ϕ_* denotes the optimal arrangement of $DAPT(G, 2)$ and ϕ_A denotes the arrangement computed by algorithm $\mathcal{A}$ described below.

Require: binary regular tree $G = (V, E)$ of height h_G whose vertices are labeled according to the canonical order
Ensure: arrangement ϕ_A
1: $b := 2^{h_G+1}$;
2: **if** $h_G = 0$ **then**
3: $\quad \phi_A(v_1) := b_1$;
4: **else**$[h_G > 0]$
5: $\quad$ solve the problem for the basic subtrees $\widehat{G_1}$ and $\widehat{G_2}$ of height $\widehat{h_G} = h_G - 1$ and obtain the respective arrangements $\widehat{\phi_A}^{(1)}$ and $\widehat{\phi_A}^{(2)}$;
6: $\quad$ arrange the vertices of the left basic subtree on the leaves b_1, b_2, ..., $b_{\frac{1}{2}b}$ according to the arrangement $\widehat{\phi_A}^{(1)}$ and the vertices of the right basic subtree on the leaves $b_{\frac{1}{2}b+1}$, $b_{\frac{1}{2}b+2}$, ..., b_b according to the arrangement $\widehat{\phi_A}^{(2)}$;
7: $\quad \phi_A(v_1) := b_{\frac{1}{2}b}$;
8: $\quad$ **if** h_G is odd and $h_G \geq 3$ **then**
9: $\quad\quad$ exchange the vertices arranged on the leaves $b_{\frac{1}{4}b-1}$ and $b_{\frac{1}{2}b}$ (*pair-exchange*);
10: $\quad$ **end if**
11: **end if**
12: **return** ϕ_A;

Algorithm 2.1: Approximation algorithm $\mathcal{A}$ which computes the arrangement ϕ_A.

In the following we apply this algorithm on an instamce of $DAPT(G, 2)$ with $h_G = 3$. Observe that the leaf $b_{\frac{1}{2}b}$ is always free prior to the execution

of pseudocode line 7 due to the recursion and due to the assignment in pseudocode line 3.

Example 2.2. *Consider the guest graph $G = (V, E)$ of height $h_G = 3$ depicted in Figure 2.1 and apply algorithm $\mathcal{A}$.*

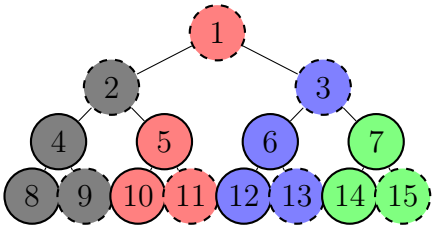

Figure 2.1: A guest graph $G = (V, E)$ (binary regular tree of height $h_G = 3$). The colors are related to the arrangement ϕ depicted in Figure 2.3.

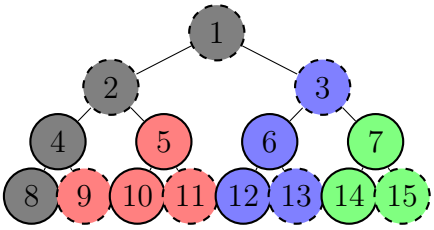

Figure 2.2: A guest graph $G = (V, E)$ (binary regular tree of height $h_G = 3$). The colors are related to the arrangement ϕ depicted in Figure 2.4.

Since $h_G = 3 > 0$, the algorithm executes the else part beginning in pseudocode line 4. In pseudocode lines 5 and 6 the arrangements for both basic subtrees, i.e. for graphs of height $\widehat{h_G} = h_G - 1 = 3 - 1 = 2$ are computed. (The arrangement $\widehat{\phi_A}$ for $\widehat{h_G} = 2$ is depicted in Figure 2.22 in Appendix.) In the next step, the root is arranged at the middle leaf (see pseudocode line 7) and the arrangement ϕ depicted in Figure 2.3 is obtained. The label of each leaf corresponds to the index of the vertex of the guest graph arranged at that leaf. The objective value which corresponds to arrangement ϕ is $OV(G, 2, \phi) = 58$.

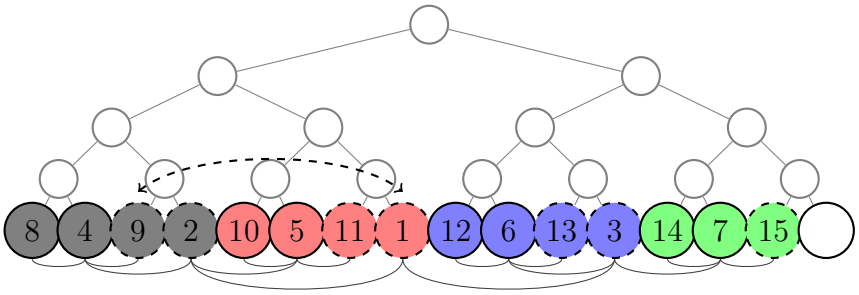

Figure 2.3: Arrangement ϕ obtained from Algorithm 2.1 for the guest graph of height $h_G = 3$ depicted in Figure 2.1. Its objective function value is $OV(G, 2, \phi) = 58$.

Next, consider the condition in pseudocode line 8: Since $h_G = 3$ is odd and $h_G = 3 \geq 3$, the pair-exchange marked in Figure 2.3 by the arrows in the

dashed line is performed. The value of the objective function corresponding to the resulting arrangement ϕ_A in Figure 2.4 is $OV(G, 2, \phi_A) = 56$. The guest graph colored according to this arrangement is depicted in Figure 2.2. In fact, this arrangement is optimal, but in general Algorithm 2.1 does not yield an optimal arrangement.

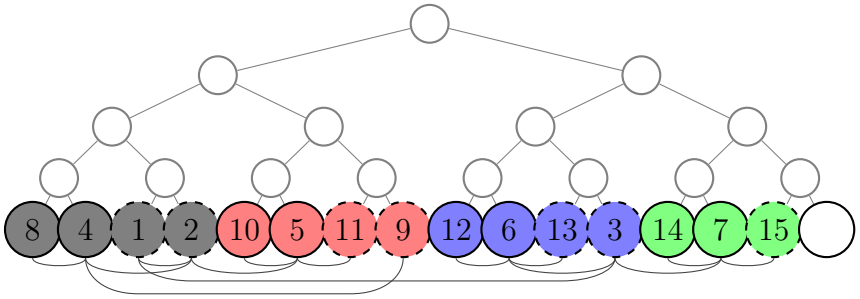

Figure 2.4: Arrangement ϕ_A obtained from Algorithm 2.1 for the guest graph of height $h_G = 3$ depicted in Figure 2.2. Its objective function value is $OV(G, 2, \phi_A) = 56$.

Next we give a closed formula for the objective function value corresponding to the arrangement ϕ_A computed by the algorithm $\mathcal{A}$.

Lemma 2.7. *Let the guest graph $G = (V, E)$ and the host graph T be binary regular trees of heights h_G and $h = h_G + 1$ respectively, where h_G is odd, $h_G \geq 3$. Then the pair-exchange defined in Algorithm 2.1 in pseudocode line 9 decreases by 1 the number of edges which contribute to the objective by 4, increases by 1 the number of edges which contribute to the objective value by 2, and does not change the number of edges which contribute to the objective value by $2i$ for $i \geq 3$. Summarizing such a pair-exchange improves the value of the objective function by 2 as compared to the value corresponding to the arrangement available prior to this pair-exchange.*

Proof. Let ϕ be the arrangement available prior to the pair-exchange steps done in pseudocode line 9 in Algorithm 2.1. Denote the arrangement obtained after the pair-exchange by ϕ_A. Consider the vertices and edges which are affected by the pair-exchange in pseudocode line 9. According to the algorithm (see pseudocode line 7) the root v_1 of G is arranged on the leaf $b_{\frac{1}{2}b}$ and its left and right children v_2 and v_3 are arranged on the leaves $b_{\frac{1}{4}b}$ and $b_{\frac{3}{4}b}$, respectively. (Recall that the pair-exchange is performed only if h_G is odd). Moreover the algorithm places the rightmost leaf, say x, of the basic subtree of G rooted at the child v_2 on leaf $b_{\frac{1}{4}b-1}$ of T. So the pair-exchange involves the vertices v_1 and x of G and $\phi_A(v_1) = \phi(x)$, $\phi_A(x) = \phi(v_1)$, $\phi_A(y) = \phi(y)$,

for $y \in V \setminus \{v_1, x\}$, hold. The change Δ in the value of the objective functions corresponding to ϕ and ϕ_A, respectively, is then given as follows.

$$\Delta := OV(G,2,\phi) - OV(G,2,\phi_A) \tag{2.10}$$

$$= \sum_{\substack{v\in V\setminus\{x\}\\ \{v,v_1\}\in E}} d_T\big(\phi(v),\phi(v_1)\big) + \sum_{\substack{v\in V\setminus\{v_1\}\\ \{v,x\}\in E}} d_T\big(\phi(v),\phi(x)\big) - \sum_{\substack{v\in V\setminus\{x\}\\ \{v,v_1\}\in E}} d_T\big(\phi_A(v),\phi_A(v_1)\big) - \sum_{\substack{v\in V\setminus\{v_1\}\\ \{v,x\}\in E}} d_T\big(\phi_A(v),\phi_A(x)\big) \tag{2.11}$$

$$= \sum_{\substack{v\in V\setminus\{x\}\\ \{v,v_1\}\in E}} \Big[d_T\big(\phi(v),\phi(v_1)\big) - d_T\big(\phi_A(v),\phi_A(v_1)\big)\Big] + \sum_{\substack{v\in V\setminus\{v_1\}\\ \{v,x\}\in E}} \Big[d_T\big(\phi(v),\phi(x)\big) - d_T\big(\phi_A(v),\phi_A(x)\big)\Big] \tag{2.12}$$

$$= \sum_{\substack{v\in V\setminus\{x\}\\ \{v,v_1\}\in E}} \Big[d_T\big(\phi(v),\phi(v_1)\big) - d_T\big(\phi(v),\phi(x)\big)\Big] + \sum_{\substack{v\in V\setminus\{v_1\}\\ \{v,x\}\in E}} \Big[d_T\big(\phi(v),\phi(x)\big) - d_T\big(\phi(v),\phi(v_1)\big)\Big] \tag{2.13}$$

Considering that v_1 has only two neighbors, namely v_2 and v_3, and denoting by y the unique neighbor (i.e. the father) of leaf x in G we get

$$\begin{aligned}\Delta =& d_T\big(\phi(v_2)\phi(v_1)\big) - d_T\big(\phi(v_2)\phi(x)\big) + d_T\big(\phi(v_3)\phi(v_1)\big) - \\ & d_T\big(\phi(v_3)\phi(x)\big) + d_T\big(\phi(y)\phi(x)\big) - d_T\big(\phi(y)\phi(v_1)\big) \\ =& 2h_G - 2 + 2(h_G+1) - 2(h_G+1) + 4 - 2h_G \\ =& 2.\end{aligned} \tag{2.14}$$

□

Lemma 2.8. *Let the guest graph $G = (V, E)$ and the host graph T be binary regular trees of heights h_G and $h = h_G + 1$ respectively. Then the value $OV(G, 2, \phi_A)$ of the objective function of the DAPT corresponding to the arrangement ϕ_A obtained from Algorithm 2.1 is given as follows:*

$$OV(G_{h_G},2,\phi_A) := OV(G,2,\phi_A) = \begin{cases} 0 \ \textit{for } h_G = 0 \\ \frac{29}{3}\cdot 2^{h_G} - 4h_G - 9 + \frac{1}{3}(-1)^{h_G} \\ \quad \textit{for } h_G \geq 1 \end{cases}. \tag{2.15}$$

Proof. The proof is done by induction with respect to h_G. Let us denote G_h a binary regular tree of height h throughout this proof. Clearly we have $OV(G_0, 2, \phi_A) = 0$. For $h_G = 1$ we obviously have $OV(G_1, 2, \phi_A) = 2+4 = 6$ by the construction (see Figures 2.19 and 2.20 in Appendix). Both this equalities are consistent with (2.15).

Assume that (2.15) holds for some $h_G \geq 1$. For $h_G + 1$ we get

$$OV(G_{h_G+1}, 2, \phi_A) = \begin{cases} 2OV(G_{h_G}, 2, \phi_A) + 2(h_G+1) + 2(h_G+2) - 2 \\ \quad \text{for } h_G + 1 \text{ odd} \\ 2OV(G_{h_G}, 2, \phi_A) + 2(h_G+1) + 2(h_G+2) \\ \quad \text{for } h_G + 1 \text{ even} \end{cases}, \tag{2.16}$$

where:

- $2OV(G_{h_G+1}, 2, \phi_A)$ represents the objective function value corresponding to the arrangements of the basic subtrees.
- $2(h_G + 1)$ and $2(h_G + 2)$ represent the contribution of the edges connecting the root v_1 with its left and right child in the objective function value, respectively. (Prior to the pair-exchange step the root v_1 is arranged at the leaf $b_{\frac{1}{2}b}$ while its children, v_2 and v_3 are arranged at the leaves $v_{\frac{1}{4}b}$ and $b_{\frac{3}{4}b}$, respectively.)
- -2 represents the contribution of the pair-exchange step if $h_G + 1$ is odd ($h_G + 1 \geq 3$ since $h_G \geq 1$), according to Lemma 2.7.

According to the induction assumption we substitute $OV(G_{h_G}, 2, \phi_A)$ by the expression on the right hand side of equation (2.15) and after simplifying we get

$$OV(G_{h_G+1}, 2, \phi_A) = \begin{cases} \frac{29}{3} \cdot 2^{h_G+1} - 4(h_G+1) - 9 + \frac{1}{3}(-1)^{h_G+1} \\ \quad \text{if } h_G + 1 \text{ is odd} \\ \frac{29}{3} \cdot 2^{h_G+1} - 4(h_G+1) - 9 + \frac{1}{3}(-1)^{h_G+1} \\ \quad \text{if } h_G + 1 \text{ is even.} \end{cases}. \tag{2.17}$$

□

Finally, notice that this approximation algorithm $\mathcal{A}$ does not solve the problem to optimality as illustrated by the following example.

Example 2.3. *Consider a guest graph $G = (V, E)$ of height $h_G = 6$ depicted in Figure 2.5. The arrangement ϕ_A computed by Algorithm 2.1 is depicted in Figures 2.7 and 2.8; it yields an objective function value of $OV(G, 2, \phi_A) = 586$. Consider now another arrangement ϕ for the same graph yielding an objective function value of $OV(G, 2, \phi) = 584$ and depicted explicitly in Figures 2.6, 2.9 and 2.10. Later in Section 2.4 we will show that the approximation algorithm $\mathcal{A}$ yields an optimal arrangement ϕ_A for $h_G \leq 5$. Thus this is the smallest instance of the $DAPT(G, 2)$ for which the algorithm $\mathcal{A}$ does not compute an optimal arrangement.*

2.3 The *k*-balanced partitioning problem

In this section we introduce the k-balanced partitioning problem and a special case of it which will be involved in the analysis of the approximation algorithm for the $DAPT(G, 2)$ with a binary regular tree G.

Definition 2.9 (*k*-balanced partitioning problem). *Given a graph $G = (V, E)$ with $|V| = n$ and $k \geq 2$, a* ***k*-balanced partition** *is a partition of the vertex set V into k non-empty* **partition sets** *$V_1 \neq \emptyset$, $V_2 \neq \emptyset$, ..., $V_k \neq \emptyset$, where $\cup_{i=1}^{k} V_k = V$, $V_i \cap V_j = \emptyset$ for every $i \neq j$ and $|V_i| \leq \lceil \frac{n}{k} \rceil$ for all $1 \leq i \leq k$. The* ***k*-balanced partitioning problem (*k*-BPP)** *asks for a k-balanced partition $\mathscr{V}$ which minimizes*

$$c(G, \mathscr{V}) := \Big| \big\{ (u, v) \in E | u \in V_i,\, v \in V_j,\, i \neq j \big\} \Big|, \tag{2.18}$$

where $\mathscr{V} := \{V_i | 1 \leq i \leq k\}$.

k-BPP is a well known $\mathcal{NP}$-hard problem (for $k = 2$ we get the *minimum bisection problem* which is $\mathcal{NP}$-hard, see Garey and Johnson [22]). A lot of work has been done focusing on the computational complexity of the k-BPP. Andreev and Räcke proved further complexity results for a generalization allowing near-balanced partitions [5]. Krauthgamer, Naor and Schwartz provide an approximation algorithm achieving an approximation of $O(\sqrt{\log n \log k})$ [33]. And finally, Feldmann and Foschini proved that the k-BPP remains APX-hard even if the graph G is restricted to be an unweighted tree with constant maximum degree [18].

We deal with a special case of this problem where $G = (V, E)$ is a binary regular tree of height $h \geq 1$ and where $k = 2^{k'}$ and $1 \leq k' \leq h$. The following facts are obvious.

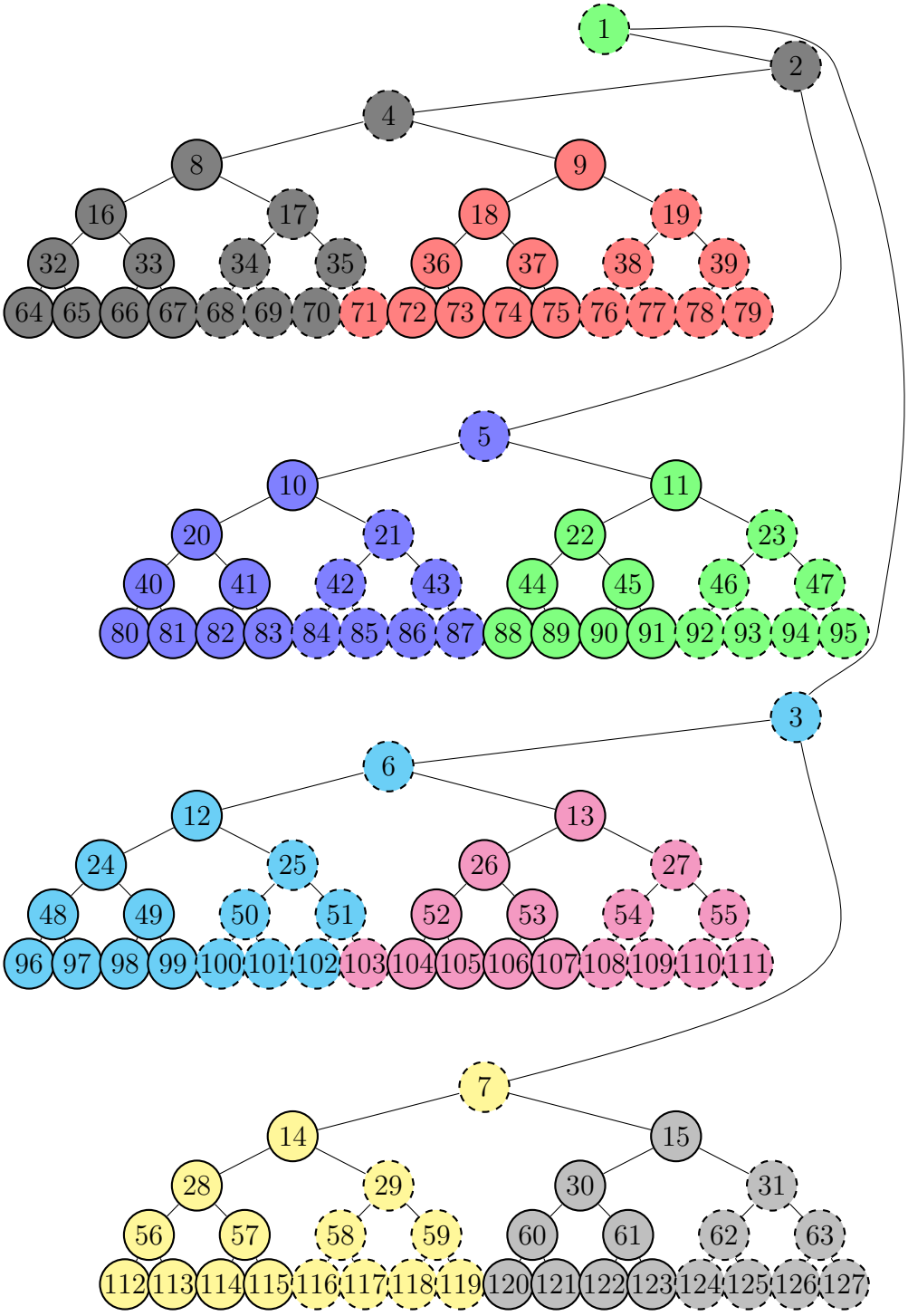

Figure 2.5: Guest graph $G = (V, E)$ (binary regular tree of height $h_G = 6$). The colors are related to the arrangement ϕ_A depicted in Figures 2.7 and 2.8.

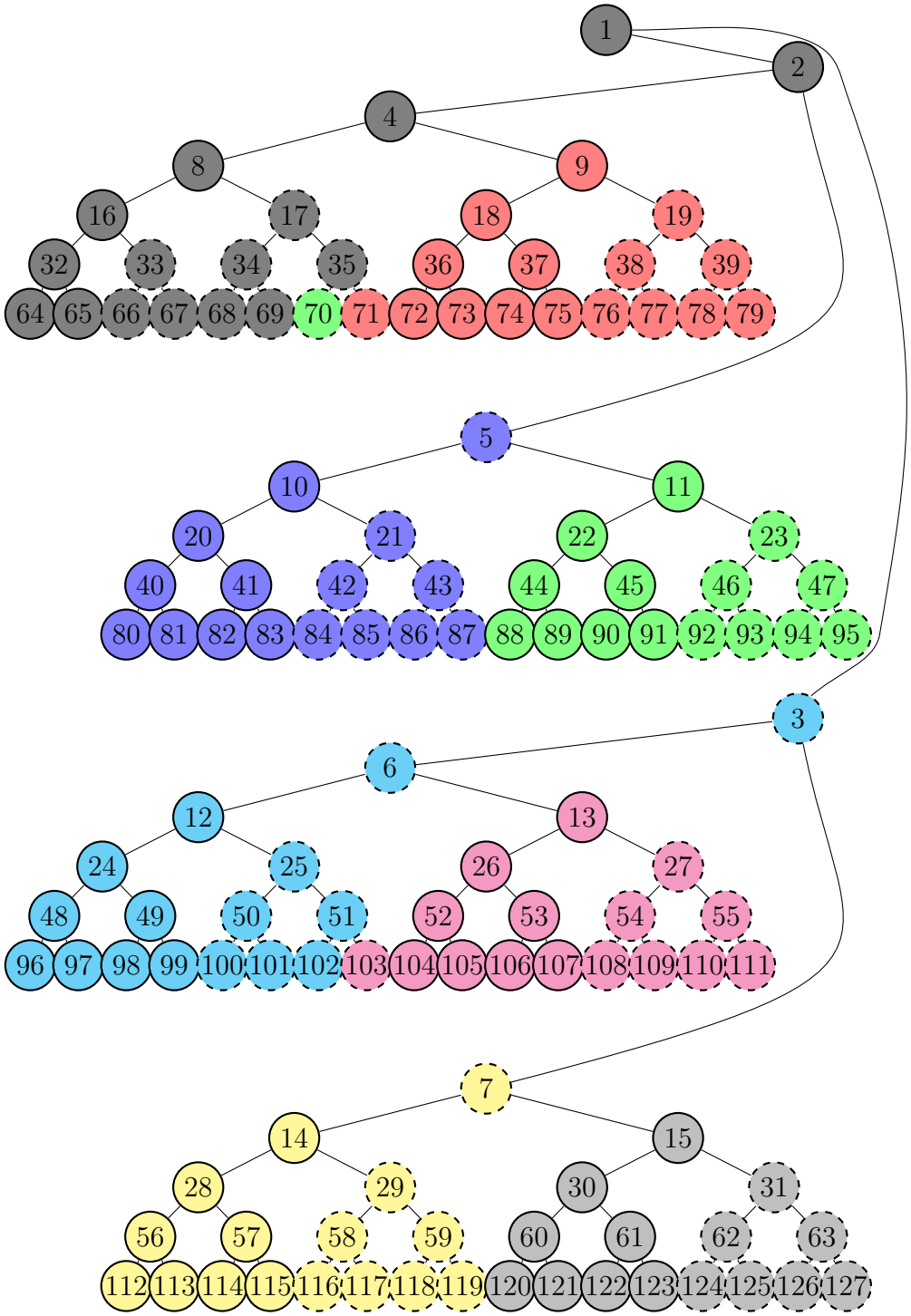

Figure 2.6: Guest graph $G = (V, E)$ (binary regular tree of height $h_G = 6$). The colors are related to the arrangement ϕ depicted in Figures 2.9 and 2.10.

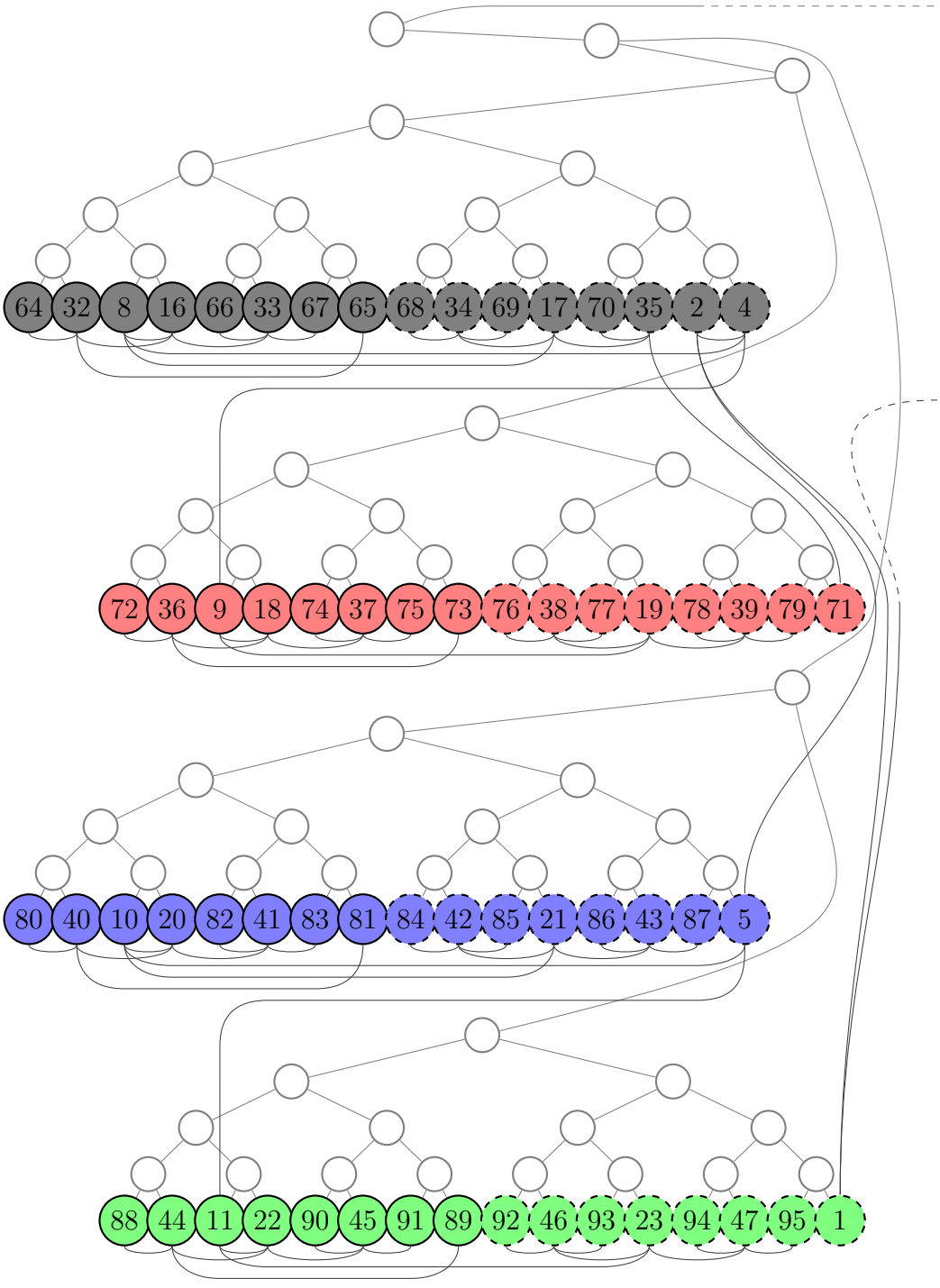

Figure 2.7: Arrangement ϕ_A obtained from Algorithm 2.1 for the guest graph $G = (V, E)$ depicted in Figure 2.5 – first part. Its objective function value is $OV(G, 2, \phi_A) = 586$.

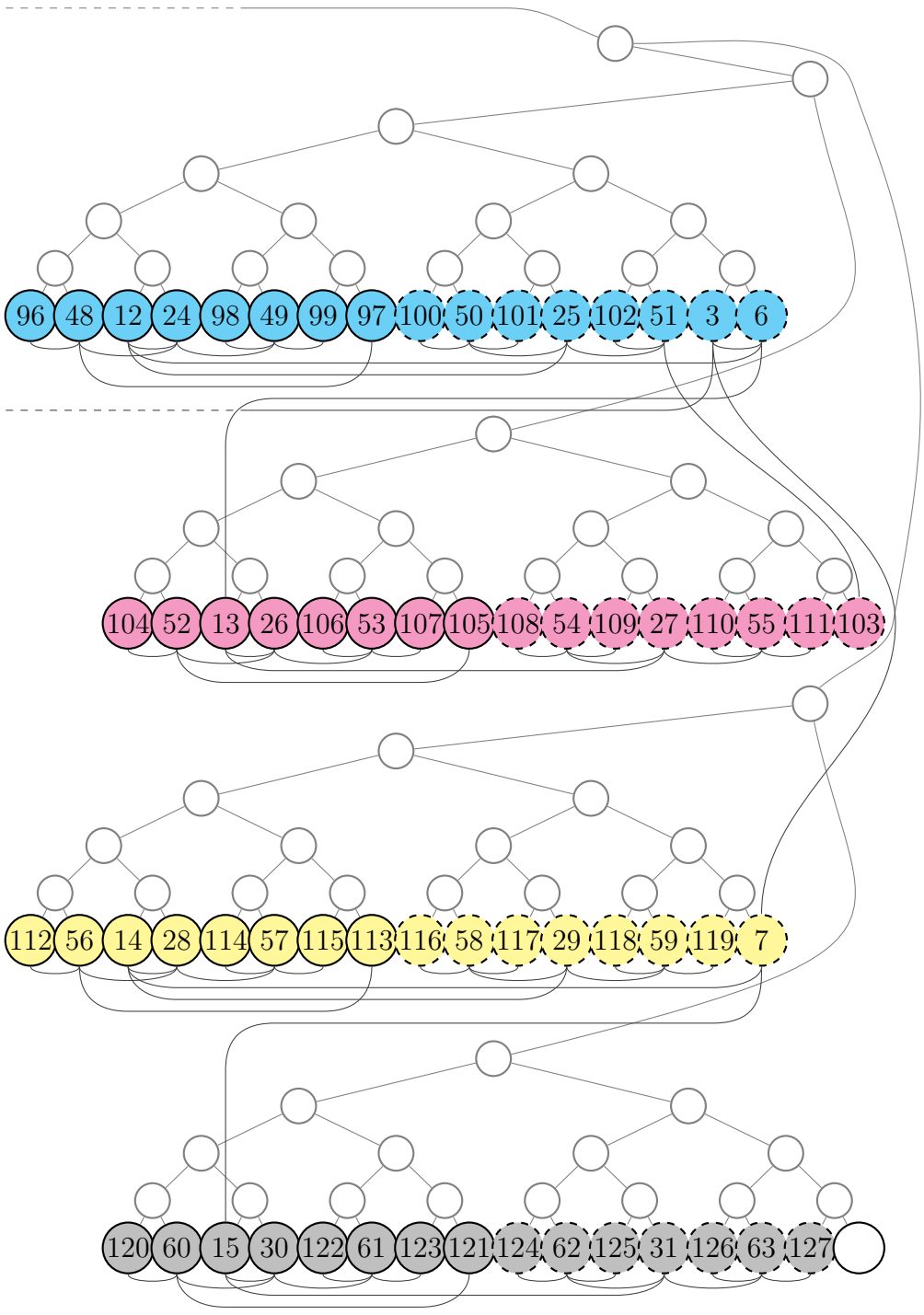

Figure 2.8: Arrangement ϕ_A obtained from Algorithm 2.1 for the guest graph $G = (V, E)$ depicted in Figure 2.5 – second part. Its objective function value is $OV(G, 2, \phi_A) = 586$.

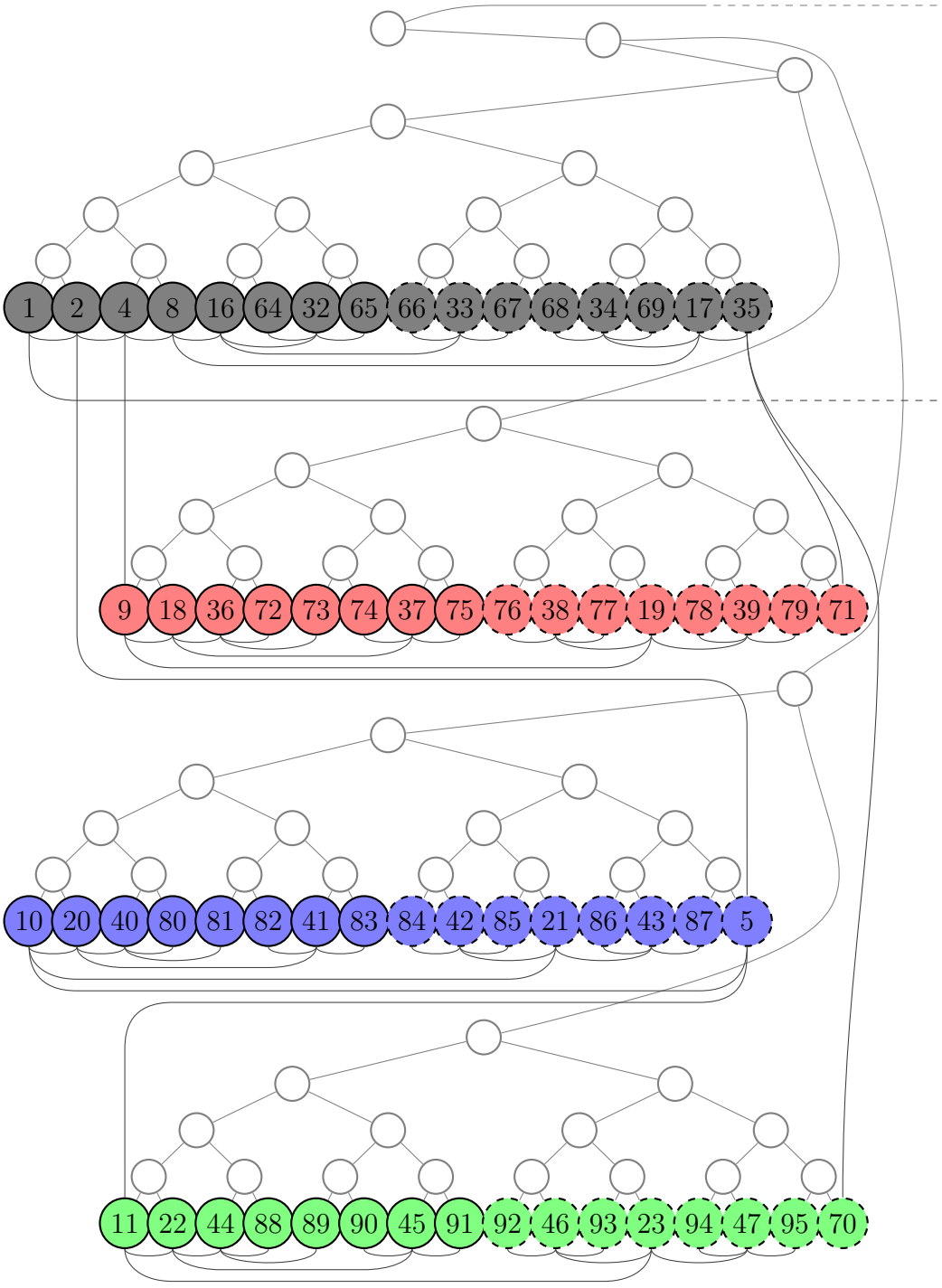

Figure 2.9: Arrangement ϕ for the guest graph $G = (V, E)$ depicted in Figure 2.6 – first part. Its objective function value is $OV(G, 2, \phi_A) = 584$.

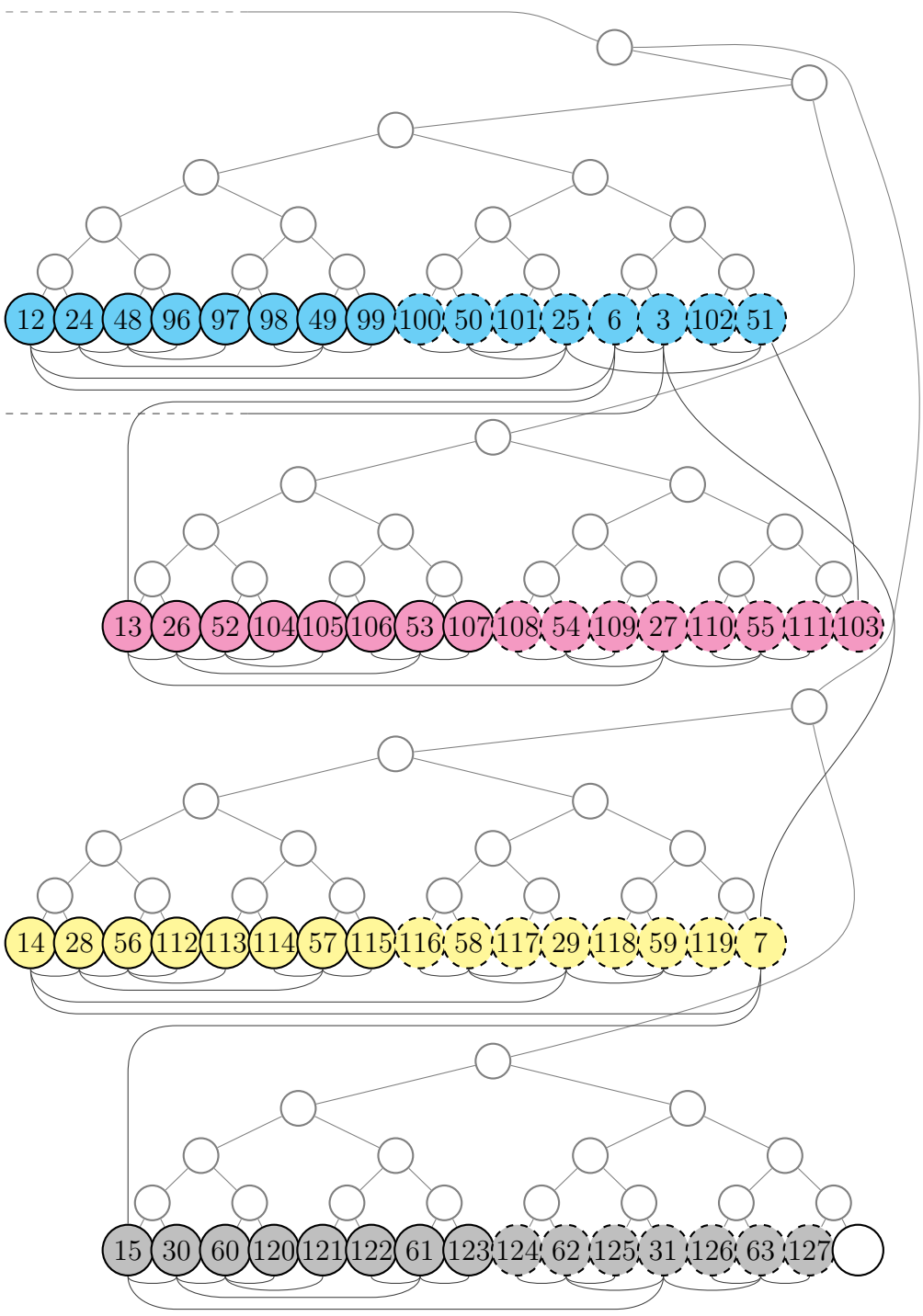

Figure 2.10: Arrangement ϕ for the guest graph $G = (V, E)$ depicted in Figure 2.6 – second part. Its objective function value is $OV(G, 2, \phi_A) = 584$.

Observation 2.10. *Let $G = (V, E)$ be a binary regular tree of height $h \geq 1$ with $n = 2^{h+1} - 1$ vertices. Let $\mathscr{V} = \{V_1, V_2, \ldots, V_k\}$ be a k-balanced partition with $k = 2^{k'}$ and $1 \leq k' \leq h$. Then one of the partition sets in $\mathscr{V}$ has $n_s := \frac{n+1}{k} - 1$ elements and is called* **the small partition set**. *All other partition sets have $n_b := \frac{n+1}{k}$ elements and are called* **big partition sets**. *Moreover the following equalities clearly hold*

$$n_s = 2^{h-k'+1} - 1 \text{ and} \tag{2.19}$$

$$n_b = 2^{h-k'+1}. \tag{2.20}$$

The rest of this section is structured as follows: In Subsection 2.3.1 we introduce an algorithm to construct an optimal $2^{k'}$-balanced partition $\mathscr{V}^*$ in a binary regular tree. The optimality is proven in Subsection 2.3.2.

Subsection 2.3.3 provides a lower bound on the optimal value $c(G, k, \mathscr{V}^*)$ of the objective function of the $2^{k'}$-BPP in a binary regular tree.

2.3.1 A solution algorithm for the $2^{k'}$-BPP on binary regular trees

The algorithm consists of three simple steps. Let $t := h - k' + 2$ and $e := \left\lfloor \frac{h+1}{t} \right\rfloor - 1$.

(1) First, we partition the tree G by cutting all edges $(u, v) \in E$ with $\text{level}(u) = h - it$ and $\text{level}(v) = h - it + 1$, where $1 \leq i \leq e$. Roughly spoken, we separate e horizontal bands of height $t - 1$ from the input tree G, from the bottom to the top. The height of the remaining top part is then $\widehat{h}$ with $t - 1 \leq \widehat{h} \leq 2t - 2$. Let p be the number of binary regular trees of height $t - 1$ contained in these bands.

(2) Next consider the binary regular trees contained in the above mentioned bands and cut all edges connecting their roots with their right children, respectively. After that each root remains connected to the corresponding left basic subtree, thus forming a big partition set, since each root and its left basic subtree tree have $2^{t-2+1} - 1 + 1 = 2^{h-k'+1} = n_b$ vertices altogether.

On the other hand each of the right basic subtrees mentioned above has $2^{t-2+1} - 1 = 2^{h-k'+1} - 1 = n_b - 1$ vertices and needs one more vertex in order to form a big partition set. Let $q := \frac{2^{h-k'+1}-1}{2^{h-k'+1}} p$. We split $p - q$ of the right basic subtrees into isolated vertices, thus obtaining $(p - q)(2^{h-k'+1} - 1) = \left(p - \frac{2^{h-k'+1}-1}{2^{h-k'+1}} p\right)(2^{h-k'+1} - 1) = q$ isolated vertices. Each of them is paired with the remaining q non-split right

basic subtrees in order to obtain further big partition sets. It is not difficult to check that $q \in \mathbb{N}$.

(3) Finally, let us consider the top part consisting of a binary regular tree of height $\widehat{h}$, $t-1 \leq \widehat{h} \leq 2t-2$. We cut all edges $(u,v) \in E$ with $\text{level}(u) = h-(e+1)t+1$ and v being the right child of u. Analogously as above we obtain one big partition set for every vertex $u \in V$ with $\text{level}(u) = h-(e+1)t+1$ together with its corresponding left basic subtree. Moreover, each of the remaining right basic subtrees of the vertices u as above can be paired with one of the vertices $u' \in V$ with $\text{level}(u') < h-(e+1)t+1$, (roughly spoken, these are the vertices lying on the very top of the tree) in order to obtain further big partition sets. Again simple computations show that the number of the right basic subtrees mentioned above exceeds the number of the remaining vertices by exactly one. Hence just one right basic subtree of one vertex $u \in V$ with $\text{level}(u) = h-(e+1)t+1$ remains unpaired; this subtree builds the small partition set.

The following example illustrates this algorithm.

Example 2.4. *Let us consider a binary regular tree $G = (V,E)$ of height $h = 5$ depicted in Figure 2.11 and let $k = 2^4 = 16$, i.e. $k' = 4$.*

We have $t = h-k'+2 = 5-4+2 = 3$ and $e = \left\lfloor \frac{h+1}{t} \right\rfloor - 1 = \left\lfloor \frac{5+1}{3} \right\rfloor - 1 = 1$.

(1) Thus $i = 1$ and we cut all edges $(u,v) \in E$ with level$(u) = h-it = 5-1\cdot 3 = 2$ and level$(v) = h-it+1 = 5-1\cdot 3+1 = 3$, i.e. the edges cut by the two horizontal lines in Figure 2.11.

(2) Now, consider the bottom band consisting of $p = 8$ binary regular trees of height $t-1 = 3-1 = 2$ each and in each of them cut all edges connecting the root with the right child. Each root connected to its corresponding left child form a big partition set; we obtain the big partition sets V_i, $3 \leq i \leq 10$ depicted in Figure 2.11. Notice that in Figure 2.11 a partition set V_i contains the vertices marked by i, $1 \leq i \leq 16$.

Set $q := \frac{2^{h-k'+1}-1}{2^{h-k'+1}}p = \frac{2^{5-4+1}-1}{2^{5-4+1}}8 = 6$, and cut all edges of $p-q = 8-6 = 2$ arbitrarily chosen right basic subtrees. Pair each of the thereby arising isolated vertices with the remaining $q = 6$ right basic subtrees to obtain the big partition sets V_i $11 \leq i \leq 16$.

(3) Finally, notice that $h-(e+1)t+1 = 5-(1+1)3+1 = 0$. Thus we cut the edge connecting the root v_1 (note that level$(v_1) = 0$) with its right child according to the third step of the algorithm. We obtain the big partition sets V_1 and the small partition set V_2 depicted in Figure 2.11.

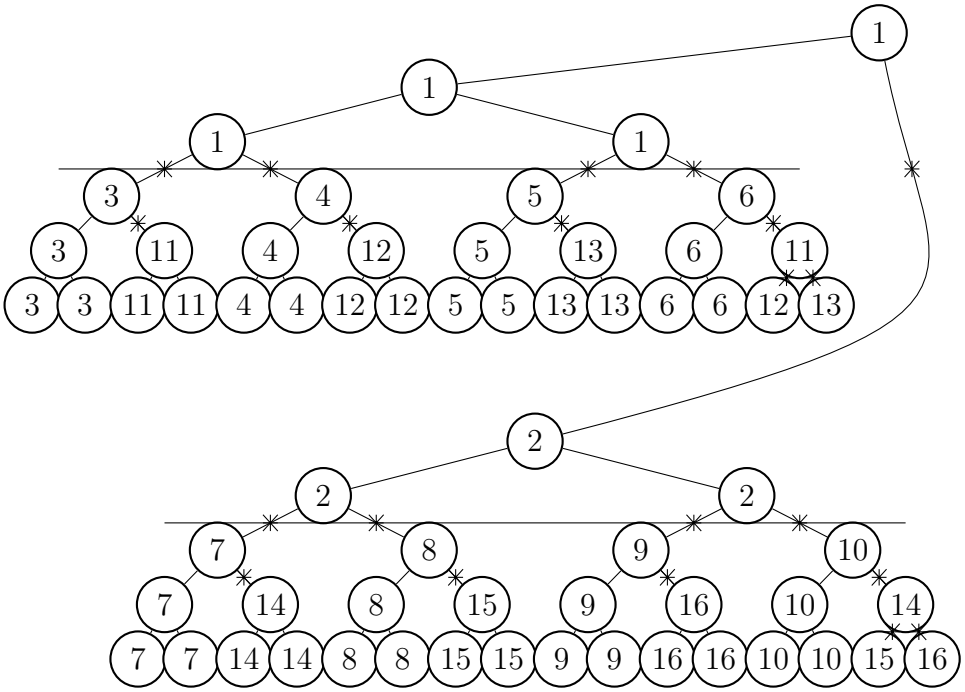

Figure 2.11: 16-balanced partition $\mathscr{V}^*$. Its objective value is $c(G, \mathscr{V}^*) = 21$. The numbers on the vertices indicate the indices of the partition sets to which the corresponding vertices belong.

The objective function value of the obtained 16*-balanced partition* $\mathscr{V}^* = \{V_1, V_2, \ldots, V_{16}\}$*, i.e. the number of the cut edges, equals* $c(G, \mathscr{V}^*) = 21$*. By applying the results of the following subsection we conclude that this is the optimal* 16*-balanced partition of* G*.*

2.3.2 Proof of the optimality for the algorithm described in Subsection 2.3.1

The optimality proof will make use of the following reformulation of the k-BPP.

Consider the input graph $G = (V, E)$ of the k-BPP, a k-balanced partition $\mathscr{V} = \{V_1, V_2, \ldots, V_k\}$, and the respective induced subgraphs $G[V_i]$, $1 \leq i \leq k$. Assume that $G[V_i]$ has l_i connected components $G_{i,j} = (V_{i,j}, E_{i,j})$ for $1 \leq j \leq l_i$, for $1 \leq i \leq k$. Define a new graph $G' = (V', E')$ which contains one representative vertex $\bar{v}_{i,j}$ for each connected component $G_{i,j}$, $1 \leq i \leq k$, $1 \leq j \leq l_i$. Two vertices $\bar{v}_{i_1,j_1}$ and $\bar{v}_{i_2,j_2}$ are connected in G' iff the connected components G_{i_1,j_1}, G_{i_2,j_2} are connected by an edge in G. Observe that if G

is a tree, then G' is also a tree and the following equality holds

$$c(G, \mathscr{V}) = |E'| = |V'| - 1. \tag{2.21}$$

Thus the value of the objective function of the k-BPP corresponding to a k-balanced partition $\mathscr{V}$ of a tree G equals the overall number of the connected components of the subgraphs induced in G by the partition sets of $\mathscr{V}$ minus 1. Hence the goal of the k-BPP can be rephrased as follows: Determine a k-balanced partition $\mathscr{V}^*$ such that the number of the connected components of the subgraphs induced in G by the partition sets is minimized.

Finally, note that the equality (2.21) holds for every (not necessarily balanced) partition $\mathscr{V}$.

Example 2.5. *Let us consider the graph and the partition depicted in Figure 2.11. The tree G' is depicted in Figure 2.12.*

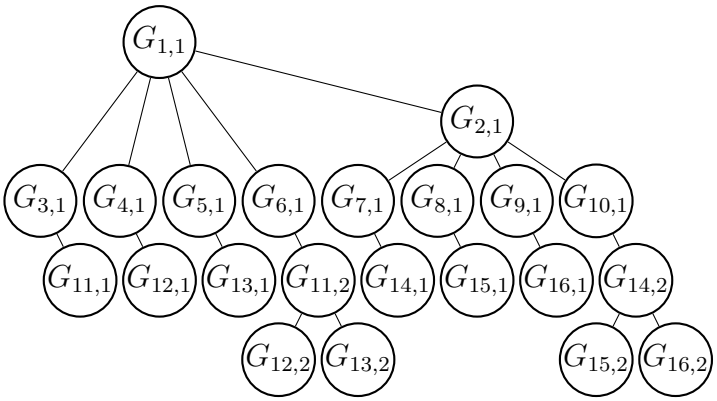

Figure 2.12: The graph G' corresponding to the binary regular tree G and the 16-partition $\mathscr{V}$ depicted in Figure 2.11.

The partition sets V_i, $1 \leq i \leq 10$, generate one connected component each, the partition sets V_i, $11 \leq i \leq 16$ generate two connected components each. Notice that equality (2.21) is fulfilled: the tree G' has 22 vertices and 21 edges and $c(G, \mathscr{V}) = 21$, see Example 2.4.

Denote by $n_i(\mathscr{V})$ be the number of partition sets in $\mathscr{V}$ which induce i connected components each in G, for $i \in \mathbb{N}$. Notice now that the following observation holds.

Observation 2.11. *Consider the $2^{k'}$-BPP for a binary regular tree G of height h, $h \geq k'$, and let $\mathscr{V}^*$ be the k-balanced partition computed by the algorithm described in Subsection 2.3.1. Then*

$$n_1(\mathscr{V}^*) \geq n_1(\mathscr{V}) \tag{2.22}$$

implies

$$c(G, \mathscr{V}^*) \leq c(G, \mathscr{V}^*) \tag{2.23}$$

for any k-balanced partition $\mathscr{V}$ of G.

Proof. As argued above for every k-balanced partition $\mathscr{V}$ the value $c(G, \mathscr{V})$ of the objective function equals the overall number of connected components induced in G by the partition sets of $\mathscr{V}$ minus 1, and hence

$$c(G, \mathscr{V}) = \sum_{i \in \mathbb{N}} in_i(\mathscr{V}) - 1 \tag{2.24}$$

holds. If $n_1(\mathscr{V}^*) \geq n_1(\mathscr{V})$ and since $n_i(\mathscr{V}^*) = 0$ for $i \geq 3$, the following equalities and inequalities hold:

$$\begin{aligned} c(G, \mathscr{V}) &= \sum_{i \in \mathbb{N}} in_i(\mathscr{V}) - 1 && \geq n_1(\mathscr{V}) + 2\big(k - n_1(\mathscr{V})\big) - 1 \\ &= 2k - n_1(\mathscr{V}) - 1 && \geq 2k - n_1(\mathscr{V}^*) - 1 \\ &= n_1(\mathscr{V}^*) + 2(k - n_1(\mathscr{V}^*)) - 1 = n_1(\mathscr{V}^*) + 2n_2(\mathscr{V}^*) - 1 \\ &= c(G, \mathscr{V}^*). \end{aligned} \tag{2.25}$$

□

Theorem 2.12. *Let $G = (V, E)$ be a binary regular tree of height $h \geq 1$ and let $k = 2^{k'}$, where $1 \leq k' \leq h$. Then the algorithm presented in Subsection 2.3.1 yields an optimal k-balanced partition $\mathscr{V}^*$.*

Proof. Due to Observation 2.11 it is enough to show that $n_1(\mathscr{V}^*) \geq n_1(\mathscr{V})$ for any k-balanced partition $\mathscr{V}$ of G.

Let us first show that every k-balanced partition $\mathscr{V} =: \mathscr{V}_0$ can be transformed step by step into a sequence $\mathscr{V} =: \mathscr{V}_0, \mathscr{V}_1, \ldots, \mathscr{V}_l = \mathscr{V}'$, $l \in \mathbb{N}$, of k-balanced partitions with the following properties:

(a) $n_1(\mathscr{V}_t) \geq n_1(\mathscr{V}_{t-1})$ holds for every $t \in \{1, 2, \ldots, l\}$, and

(b) the big partition sets of $\mathscr{V}'$ which induce one connected component in G each coincide with the big partition sets of $\mathscr{V}^*$ which induce one connected component in G each.

In the following the steps of this transformation are explained. Consider a k-balanced partition $\mathscr{V}$ whose big partition sets which induce one connected component in G each do not coincide with the corresponding partition sets of $\mathscr{V}^*$.

For every vertex $v \in V$ let $P_{\mathscr{V}}(v)$ be the uniquely determined partition set in $\mathscr{V}$ such that $v \in P_{\mathscr{V}}(v)$. Consider the vertices $v \in V$ with level$(v) = h - it + 1$, where $1 \leq i \leq e$, $e = \lfloor \frac{h+1}{t} \rfloor - 1$, and choose among them a vertex with the largest level such that $P_{\mathscr{V}}(v) \neq P_{\mathscr{V}^*}(v)$. Perform now the following transformation steps.

Case 1. If the partition set $P_{\mathscr{V}}(v)$ consists of the vertex $v \in V$ together with the binary regular subtree of height $h - k'$ rooted at its right child, then exchange the subtrees rooted at the left and the right child of v, respectively, to obtain a new k-balanced partition $\mathscr{V}'$ for which obviously $n_1(\mathscr{V}) = n_1(\mathscr{V}')$ holds.

Case 2. If Case 1 does not arise, then $P_{\mathscr{V}}(v)$ contains neither the binary regular subtree of height $h - k'$ rooted at the right child of v nor the binary regular subtree of height $h - k'$ rooted at its left child. At least one of these two subtrees does not build a small partition set in $\mathscr{V}$. Let this be the left subtree (otherwise we would apply an exchange of the two subtrees as in Case 1). Denote this subtree by T. If $P_{\mathscr{V}}(v)$ is a big component, then exchange the vertices contained in $P_{\mathscr{V}}(v)$ and the vertices of T (the one by one assignment of the corresponding vertices is done arbitrarily). Denote the resulting balanced partition by $\mathscr{V}'$. Clearly $P_{\mathscr{V}'}(v)$ induces one connected component in G. Moreover $P_{\mathscr{V}}(u)$ induces more than one connected component in G, for every $u \in V(T)$, because T does not build a small partition set in $\mathscr{V}$. Then $n_1(\mathscr{V}') \geq n_1(\mathscr{V})$ holds (no partition sets inducing one connected component in G are destroyed). If $P_{\mathscr{V}}(v)$ is the small component, then again exchange the vertices contained in $P_{\mathscr{V}}(v)$ against all but one of the vertices in the subtree T of height $h - k'$ rooted at the left child of v (the one by one assignment of the corresponding vertices is again done arbitrarily). Then add the remaining vertex of T to the partition set containing v and resulting after that exchange. Denote the resulting balanced partition by $\mathscr{V}'$. Clearly $P_{\mathscr{V}'}(v)$ induces one connected component in G. Further $P_{\mathscr{V}}(u)$ induces more than one connected component in G, for every $u \in V(T)$, because T does not build a small partition set in $\mathscr{V}$. Thus $n_1(\mathscr{V}') \geq n_1(\mathscr{V})$ holds again for the same reason as above.

We repeat the above transformation step as long as there are vertices v for which $P_{\mathscr{V}}(v) \neq P_{\mathscr{V}^*}(v)$, where $\mathscr{V}$ denotes the most recently constructed k-balanced partition. We end up with a k-balanced partition $\mathscr{V}'$ which contains as partition sets all big partitition sets of $\mathscr{V}^*$ which induce one connected component in G, respectively. Moreover $\mathscr{V}'$ fulfills the following inequality

$$n_1(\mathscr{V}') \geq n_1(\mathscr{V}). \tag{2.26}$$

Notice finally that by removing from G all big partition sets of $\mathscr{V}^*$ which induce one connected component in G, respectively, we obtain a graph whose largest connected component contains at most $2^{h-k'}-1$ vertices. So, if $\mathscr{V}'$ contains any partition sets which induce one connected component in G besides the big partition sets of $\mathscr{V}^*$ inducing one connected component, then these partition sets should be small ones. Finally, since in every k-balanced partition there is only one small partition set and the small partition set in $\mathscr{V}^*$ induces one connected component in G we get

$$n_1(\mathscr{V}^*) \geq n_1(\mathscr{V}'). \tag{2.27}$$

By combining (2.26) and (2.27) we get $n_1(\mathscr{V}^*) \geq n_1(\mathscr{V})$ and this completes the proof. □

2.3.3 The optimal value of the $2^{k'}$-BPP on binary regular trees

According to equality (2.25) the optimal value $c(G, \mathscr{V}^*)$ of the $2^{k'}$-BPP on a binary regular tree of height h for $1 \leq k' \leq h$ is given as $c(G, \mathscr{V}^*) = 2k - n_1(\mathscr{V}^*) - 1$. Recall that $n_1(\mathscr{V}^*)$ is the number of partition sets of $\mathscr{V}^*$ which induce exactly one connencted component each in G. Observe that for every $i \in \mathbb{N}$, $1 \leq i \leq e+1$, the algorithm presented in Subsection 2.3.1 constructs exactly 2^{h-it+1} big partition sets inducing one connected component in G, respectively. More precisely, it constructs one such partition set for each vertex of level $h - it + 1$, where the partition sets arise in the i-th bands of height $t-1$ by cutting the edge joining the above mentioned vertices to their right children, respectively. Finally in its last step the algorithm constructs the small partition set which also induces one connected component in G. Thus the following equality holds

$$n_1(\mathscr{V}^*) = 1 + \sum_{i=1}^{e+1} 2^{h-it+1} = 1 + 2^{h+1} \frac{1 - \left(\frac{1}{2^t}\right)^{e+1}}{2^t \left(1 - \frac{1}{2^t}\right)}, \tag{2.28}$$

and this implies

$$c(G, \mathscr{V}^*) = 2k - 2 - 2^{h+1} \frac{1 - \left(\frac{1}{2^t}\right)^{e+1}}{2^t \left(1 - \frac{1}{2^t}\right)}. \tag{2.29}$$

For technical reasons we derive a lower bound for $c(G, \mathscr{V}^*)$ which is given as a closed formula depending just on k.

Lemma 2.13. *Let $G = (V, E)$ be a binary regular tree of height $h \geq 1$ and let $k = 2^{k'}$, where $1 \leq k' \leq h$. Let $\mathscr{V}^*$ be the optimal k-balanced partition in G computed by the algorithm described in Subsection 2.3.1. The optimal value $c(G, \mathscr{V}^*)$ of the k-BPP in G fulfills the following (in)equalities:*

$$c(G, \mathscr{V}^*) \geq \frac{10}{7}k - 2 \qquad \text{if } k' \leq h - 1, \tag{2.30}$$

$$c(G, \mathscr{V}^*) = \frac{4}{3}k - \frac{3}{2} + \frac{1}{6}(-1)^{\log_2 k} \quad \text{if } k' = h \text{ and} \tag{2.31}$$

$$c(G, \mathscr{V}^*) = \frac{3}{2}k - 2 \qquad \text{if } k' \leq \left\lfloor \tfrac{h}{2} \right\rfloor + 1. \tag{2.32}$$

Proof. If $k' \leq h - 1$ we get $t = h - k' + 2 \geq h - (h - 1) + 2 = 3$ and the following inequalities hold

$$\begin{aligned} n_1(\mathscr{V}^*) &\leq 1 + 2^{h+1}\left(\frac{1}{1 - \frac{1}{2^t}} - 1\right) = 1 + \frac{2^{h+1}\frac{1}{2^t}}{1 - \frac{1}{2^t}} \\ &= 1 + \frac{2^{h+1-t}}{1 - \frac{1}{2^t}} \qquad = 1 + \frac{2^{k'-1}}{1 - \frac{1}{8}} \\ &= 1 + \frac{2^{k'}}{\frac{7}{4}} \qquad = 1 + \frac{4}{7}k. \end{aligned} \tag{2.33}$$

The last inequality implies

$$(G, \mathscr{V}^*) = 2k - n_1(\mathscr{V}^*) - 1 \geq \frac{10}{7}k - 2. \tag{2.34}$$

If $k' = h$ we get $t = h - k' + 2 = 2$ and $e = \left\lfloor \frac{h+1}{t} \right\rfloor - 1 = \left\lfloor \frac{h+1}{2} \right\rfloor - 1$. If h is odd, $e = \frac{h+1}{2} - 1$ holds, and if h is even we get $e = \left\lceil \frac{h+1}{2} \right\rceil = \frac{h}{2}$. By setting this values for e and t in equation (2.28) and simplifying we get

$$n_1(\mathscr{V}^*) = \begin{cases} \frac{2}{3}2^h + \frac{2}{3} & \text{if } h \text{ is odd} \\ \frac{2}{3}2^h + \frac{1}{3} & \text{if } h \text{ is even} \end{cases}. \tag{2.35}$$

By plugging these expressions for $n_1(\mathscr{V}^*)$ into equation 2.29 we obtain

$$c(G, \mathscr{V}^*) = 2k - n_1(\mathscr{V}^*) - 1 = \frac{4}{3}k - \frac{3}{2} + \frac{1}{6}(-1)^{\log_2 k} \tag{2.36}$$

in both cases.

If $k' \leq \left\lfloor \frac{h}{2} \right\rfloor + 1$ we get $t = h - k' + 2$ and $e = \left\lfloor \frac{h+1}{t} \right\rfloor - 1 = \left\lfloor \frac{h+1}{h-k'+2} \right\rfloor - 1 \leq \left\lfloor \frac{h+1}{h - \left\lfloor \frac{h}{2} \right\rfloor + 1} \right\rfloor - 1$. By distinguishing the two cases when h is odd and h is

even it can be easily observed that $e = 0$ in both cases. By substituting e by 0 in equation 2.28 we get

$$n_1(\mathscr{V}^*) = 1+2^{h+1}\frac{1}{2^t} = 1+\frac{2^{h+1}}{2^{h-k'+2}} = 1+2^{k'-1} = 1+\frac{2^{k'}}{2} = 1+\frac{k}{2}, \quad (2.37)$$

and then by plugging this into equation 2.29

$$c(G, \mathscr{V}^*) = 2k - 1 - \frac{k}{2} - 1 = \frac{3}{2}k - 2. \quad (2.38)$$

□

2.4 The approximation ratio of Algorithm 2.1

In order to estimate an approximation ratio ρ for Algorithm 2.1 we will exploit the relationship between the *DAPT* on binary regular trees and the *k-BPP* and obtain a lower bound for the objective function value of the *DAPT* in terms of the special case of k-BPP where k is a power of two.

Let the guest graph $G = (V, E)$ and the host graph T be binary regular trees of heights $h_G \geq 1$ and $h = h_G + 1$, respectively, and let ϕ be an arbitrary arrangement with corresponding objective value $OV(G, 2, \phi) = \sum_{(u,v)\in E} d_T\big(\phi(u), \phi(v)\big)$. The length $d_T\big(\phi(u), \phi(v)\big)$ of the unique $\phi(u)$-$\phi(v)$-path in the binary regular tree T is even for any two vertices u and v in G (see and Observation 2.5). Moreover, $2 \leq d_T\big(\phi(u), \phi(v)\big) \leq 2h$ holds for any two vertices u and v in G. Let $a_i(\phi)$, $1 \leq i \leq h$, be the number of edges which contribute to the value of the objective function by $2i$, i.e. the number of edges $(u, v) \in E(G)$ for which $d_T\big(\phi(u), \phi(v)\big) = 2i$ holds. Then

$$\begin{aligned} OV(G, 2, \phi) &= a_h(\phi) \cdot 2h + a_{h-1}(\phi) \cdot 2(h-1) + \ldots + a_1(\phi) \cdot 2 \\ &= 2\sum_{i=1}^{h} a_i(\phi) i. \end{aligned} \quad (2.39)$$

The number of edges which contribute to the objective function value by *at least* $2i$, i.e. the number of edges $(u, v) \in E(G)$ for which $d_T\big(\phi(u), \phi(v)\big) \geq 2i$ holds, is given by the following the **partial sums**

$$s_i(\phi) := \sum_{j=i}^{h} a_j(\phi) \text{ for all } 1 \leq i \leq h. \quad (2.40)$$

Clearly, the following equalities hold

$$a_i(\phi) = \begin{cases} s_i(\phi) - s_{i+1}(\phi) & \text{for } 1 \leq i \leq h-1 \\ s_i(\phi) & \text{for } i = h \end{cases} . \tag{2.41}$$

By plugging this into the objective function in (2.39) we get

$$\begin{aligned} OV(G,2,\phi) &= 2\sum_{i=1}^{h} a_i(\phi) i \\ &= 2\left(\left(\sum_{i=1}^{h-1} i\big(s_i(\phi) - s_{i+1}(\phi)\big)\right) + h s_h(\phi)\right) \\ &= 2\sum_{i=1}^{h} s_i(\phi). \end{aligned} \tag{2.42}$$

Example 2.6. *Let us consider the guest graph in Figure 2.2 and the arrangement ϕ_A obtained by applying Algorithm 2.1; this arrangement is depicted in Figure 2.4. The coefficients $a_i(\phi_A)$, $s_i(\phi_A)$, for $1 \leq i \leq h$ are listed in Table 2.1.*

i	4	3	2	1
$a_i(\phi_A)$	1	3	5	5
$s_i(\phi_A)$	1	4	9	14

Table 2.1: Coefficients $a_i(\phi_A)$ and partial sums $s_i(\phi_A)$ for $1 \leq i \leq h$.

By applying (2.39) and (2.42) we obtain the corresponding objective function value, respectively, as follows:

$$OV(G,2,\phi_A) = 2\sum_{i=1}^{h} a_i(\phi_A) i = 2(5\cdot 1 + 5\cdot 2 + 3\cdot 3 + 1\cdot 4) = 56, \tag{2.43}$$

$$OV(G,2,\phi_A) = 2\sum_{i=1}^{h} s_i(\phi_A) = 2(14 + 9 + 4 + 1) = 56. \tag{2.44}$$

For $h_G = 0$ the arrangement ϕ_A is obviously optimal, so let us assume that $h_G \geq 1$ through the rest of this section. The next lemma and its corollary give closed formulas for the coefficents $a_i(\phi_A)$ and the partial sums $s_i(\phi_A)$, $1 \leq i \leq h$, corresponding to the arrangement computed by Algorithm 2.1.

Lemma 2.14. *Let the guest graph $G = (V, E)$ and the host graph T be binary regular trees of heights $h_G \geq 1$ and $h = h_G + 1$, respectively, and let ϕ_A be the arrangement computed by Algorithm 2.1. Then the coefficients $a_i(\phi_A)$, $1 \leq i \leq h$, are given as follows:*

$$a_i(\phi_A) = \begin{cases} \frac{2}{3}2^{h_G} - \frac{1}{2} - \frac{1}{6}(-1)^{h_G} & \text{for } i = 1 \\ \frac{7}{12}2^{h_G} + \frac{1}{2} + \frac{1}{6}(-1)^{h_G} & \text{for } i = 2 \\ 3 \cdot 2^{h_G - i} & \text{for } 3 \leq i < h \\ 1 & \text{for } i = h \end{cases} \tag{2.45}$$

Proof. Consider first $i = 1$ and determine the number of edges contributing to the objective value $OV(G, 2, \phi_A)$ by exactly 2. Let us first neglect the pair-exchanges done in pseudocode line 9. Then there are only two possibilities how to arrange an edge in Algorithm 2.1 in such a way that it contributes by 2 to the objective value

(1) Either it is taken over from the recursive arrangements in pseudocode line 6

(2) or it is produced by placing the root v_1 in pseudocode line 7.

In the latter case the following property P must hold: (P) A child of the root v_1 and v_1 itself are placed to children vertices of a common father in T. Since the children of the root v_1 are roots in the previous recursion step, and since the roots are always placed on the middle leaf, $h_G = 1$ has to hold in the corresponding recursive run, i.e. in the run when property P is fulfilled. So exactly one edge contributing by 2 to the value of the objective function arises in every such recursive run with $h_G = 1$ (see also Figures 2.19 and 2.20 in Appendix). There are $2^{h_G - 1}$ such runs, one for each vertex with level $h_G - 1$ (playing the role of the root). Thus, if the pair-exchange step is neglected, there are $2^{h_G - 1}$ edges which contribute 1 to the value of the objective function.

Let $pe(h_G)$ be the number of pair-exchanges done in pseudocode line 9 when applying the algorithm on a guest graph of height h_G. We prove that

$$pe(h_G) = \frac{1}{6}2^{h_G} - \frac{1}{2} - \frac{1}{6}(-1)^{h_G} \tag{2.46}$$

by induction on the height h_G. For $h_G = 1$ the formula in (2.46) yields $pe(h_G) = 0$ which is obviously correct (see also Figures 2.19 and 2.20 in Appendix). Analogous arguments as in the proof of Lemma 2.8 show that

the following recursive equations hold for $h_G \geq 1$:

$$pe(h_G+1) = \begin{cases} 2pe(h_G)+1 & \text{for } h_G+1 \text{ odd} \\ 2pe(h_G) & \text{for } h_G+1 \text{ even} \end{cases} . \tag{2.47}$$

So by applying (2.46) and (2.47) we get:

$$\begin{aligned} pe(h_G+1) &= 2pe(h_G)+1 = 2\left(\frac{1}{6}2^{h_G} - \frac{1}{2} - \frac{1}{6}(-1)^{h_G}\right)+1 \\ &= \frac{1}{6}2^{h_G+1} - \frac{1}{2} - \frac{1}{6}(-1)^{h_G} \end{aligned} \tag{2.48}$$

if h_G is odd, and

$$\begin{aligned} pe(h_G+1) &= 2pe(h_G) = 2\left(\frac{1}{6}2^{h_G} - \frac{1}{2} - \frac{1}{6}(-1)^{h_G}\right) \\ &= \frac{1}{6}2^{h_G+1} - \frac{1}{2} - \frac{1}{6}(-1)^{h_G} \end{aligned} \tag{2.49}$$

if h_G is even. Thus also $pe(h_G+1)$ fulfills (2.46), which completes the inductive proof of (2.46).

In Lemma 2.7 it was proven that every pair-exchange done in pseudocode line 9 increases by 1 the number of edges contributing by 2 to the value of the objective function. Thus we get

$$a_1(\phi_A) = 2^{h_G-1} + pe(h_G) = \frac{2}{3}2^{h_G} - \frac{1}{2} - \frac{1}{6}(-1)^{h_G}, \tag{2.50}$$

and hence the claim of the lemma holds for $i = 1$.

Let $i = 2$. We use the same technique: We count vertices with level$(v) = h_G - 1$ and the number of vertices with level$(v) = h_G - 2$ first. Edges which contribute by 4 to the value of the objective function arise only in recursive runs where the guest graph has height 1 or 2 and is rooted at a vertex with level $h_G - 1$ or $h_G - 2$, respectively. In every such recursive run exactly one edge of that kind arises (see also Figures 2.19, 2.20, 2.21 and 2.22). Then we consider the effect of the pair-exchanges done in pseudocode line 9. According to Lemma 2.7 each such pair-exchange reduces by one the number of edges which contribute to the value of the objective function by 4, so we get

$$a_2(\phi_A) = 2^{h_G-1} + 2^{h_G-2} - pe(h_G) = \frac{7}{12}2^{h_G} + \frac{1}{2} + \frac{1}{6}(-1)^{h_G}, \tag{2.51}$$

and hence the claim of the lemma holds for $i = 2$.

Consider now the case $i \geq 3$. According to Lemma 2.7 the pair-exchanges done in pseudocode line 9 have no effect on the number of edges of G which contribute to the value of the objective function by $2i$, if $i \geq 3$. So for $3 \leq i < h$ the pair-exchanges can be neglected and we get

$$a_i(\phi_A) = 2^{h_G-(i-1)} + 2^{h_G-i} = 3 \cdot 2^{h_G-i}, \tag{2.52}$$

in compliance with the claim of the lemma.

Finally, for $i = h$ there is exactly one edge which contributes by $2h$ to the value of the objective function, namely the one joining the root of G with his right child. □

Corollary 2.15. *Let the guest graph $G = (V, E)$ and the host graph T be binary regular trees of heights $h_G \geq 1$ and $h = h_G+1$, respectively, and let ϕ_A be the arrangement computed by Algorithm 2.1. Then the coefficients $s_i(\phi_A)$, $1 \leq i \leq h$, are given as follows:*

$$s_i(\phi_A) = \begin{cases} 2 \cdot 2^{h_G} - 2 & \text{for } i = 1 \\ \frac{4}{3}2^{h_G} - \frac{3}{2} + \frac{1}{6}(-1)^{h_G} & \text{for } i = 2 \\ 6 \cdot 2^{h_G-i} - 2 & \text{for } 3 \leq i \leq h \end{cases}. \tag{2.53}$$

Proof. This corollary is a straightforward consequence of the definition of $s_i(\phi)$, $1 \leq i \leq h$, (see (2.40)) and of Lemma 2.14.

- For $i = h$ we get: $s_h(\phi_A) = a_h(\phi_A) = 1 = 6 \cdot 2^{h_G-(h_G+1)} - 2 = 6 \cdot 2^{h_G-i} - 2$.

- For $3 \leq i < h$ we get by induction on i and starting with $i = h$: $s_i(\phi_A) = a_i(\phi_A) + s_{i+1}(\phi_A) = 3 \cdot 2^{h_G-i} + 6 \cdot 2^{h_G-(i+1)} - 2 = 6 \cdot 2^{h_G-i} - 2$.

- For $i = 2$ we get: $s_2(\phi_A) = a_2(\phi_A) + s_3(\phi_A) = \frac{7}{12}2^{h_G} + \frac{1}{2} + \frac{1}{6}(-1)^{h_G} + 6 \cdot 2^{h_G-3} - 2 = \frac{4}{3}2^{h_G} - \frac{3}{2} + \frac{1}{6}(-1)^{h_G}$.

- For $i = 1$ we get: $s_1(\phi_A) = a_1(\phi_A) + s_2(\phi_A) = \frac{2}{3}2^{h_G} - \frac{1}{2} - \frac{1}{6}(-1)^{h_G} + \frac{4}{3}2^{h_G} - \frac{3}{2} + \frac{1}{6}(-1)^{h_G} = 2 \cdot 2^{h_G} - 2$.

□

Next we give a lower bound for $s_i(\phi)$, where ϕ is an arbitrary arrangement of the vertices of the guest graph G with height h_G into the leaves of the host graph T with height $h := h_G+1$ and where i is some integer between 1 and h. $s_i(\phi)$ is the number of edges of G which contribute by at least $2i$ to the value

of the objective function. Obviously, each such edge joins vertices of G which are arranged at the leaves of *different binary regular subtrees* of T of height $i-1$ and rooted at vertices of level $h-(i-1) = h_G - i + 2 =: k'$. Clearly, there are $2^{k'}$ such subtrees of T. Let us denote them by $T_j^{(i)}$, for $1 \leq j \leq 2^{k'}$. Let $V_j^{(i)} \subset V(G)$ be the set of vertices of G which are arranged at the leaves of $T_j^{(i)}$. Since for all leaves b of T but one there is some vertex $v \in V(G)$ with $\phi(v) = b$, $|V_j^{(i)}| = 2^{i-1}$ holds for all but one index j, $1 \leq j \leq 2^{k'}$, and for the exception, say j_0, $|V_{j_0}^{(i)}| = 2^{i-1} - 1$ holds. Thus $\mathscr{V}^{(i)} := \{V_j^{(i)} | 1 \leq j \leq 2^{k'}\}$ is k-balanced partition of G with $k = 2^{k'}$, $k' = h_G - i + 2$, and $s_i(\phi) = c(G, \mathscr{V}^{(i)})$. Let $\mathscr{V}_i^*$ be the optimal k-balanced partition of G (which can be computed by the algorithm presented in Subsection 2.3.1 and for which lower bounds of the objective function value as in Subsection 2.3.3 are known). Then $s_i(\phi) \geq c(G, \mathscr{V}_i^*)$ holds for all i, $2 \leq i \leq h$ and for every arrangement ϕ. Let us denote the lower bounds $s_i^L := c(G, \mathscr{V}_i^*)$ for $2 \leq i \leq h$, and $s_1^L := |V(G)| - 1 = 2^h - 2$ for $i = 1$.

Thus we get

$$OV(G, 2, \phi) \geq 2 \sum_{i=1}^{h} s_i^L \text{ for all arrangements } \phi. \tag{2.54}$$

Notice that $s_1(\phi) = |E(G)| = |V(G)| - 1 = s_1^L$ holds for every arrangement ϕ of the guest graph G. Notice, moreover, that for $h_G \leq 4$ the bound in (2.54) is tight, in the sense that it matches the optimal value of the objective function of $DAPT(G, 2)$, as illustrated in the following tables.

i	2	1
$s_i(\phi_A)$	1	2
s_i^L	1	2

Table 2.2: Partial sums $s_i(\phi_A)$ and the corresponding lower bounds s_i^L, where $1 \leq i \leq h$, for a guest graph $G = (V, E)$ of height $h_G = 1$.

i	3	2	1
$s_i(\phi_A)$	1	4	6
s_i^L	1	4	6

Table 2.3: Partial sums $s_i(\phi_A)$ and the corresponding lower bounds s_i^L, where $1 \leq i \leq h$, for a guest graph $G = (V, E)$ of height $h_G = 2$.

i	4	3	2	1
$s_i(\phi_A)$	1	4	9	14
s_i^L	1	4	9	14

Table 2.4: Partial sums $s_i(\phi_A)$ and the corresponding lower bounds s_i^L, where $1 \leq i \leq h$, for a guest graph $G = (V, E)$ of height $h_G = 3$.

i	5	4	3	2	1
$s_i(\phi_A)$	1	4	10	20	30
s_i^L	1	4	10	20	30

Table 2.5: Partial sums $s_i(\phi_A)$ and the corresponding lower bounds s_i^L, where $1 \leq i \leq h$, for a guest graph $G = (V, E)$ of height $h_G = 4$.

In general, the lower bound in (2.54) is not tight. Already for $h_G = 5$ there is a gap between the value of the objective function corresponding to the arrangement ϕ_A generated by the algorithm $\mathcal{A}$ and the lower bound, as shown in Table 2.6 below. Moreover, in the following example we prove that ϕ_A is an optimal arrangement for $h_G = 5$. Thus we conclude that the lower bound in (2.54) does not match the optimal value of the objective function of $DAPT(G, 2)$ already for $h_G = 5$.

Example 2.7. *Consider the guest graph $G = (V, E)$ to be a complete binary tree of height $h_G = 5$ ordered according to the canonical ordering. The partial sums $s_i(\phi_A)$ and the lower bounds s_i^L, for $1 \leq i \leq h$, computed according to Corollary 2.15, inequality (2.54) and equation (2.29), are given in Table 2.6 below.*

i	6	5	4	3	2	1
$s_i(\phi_A)$	1	4	10	22	41	62
s_i^L	1	4	10	21	41	62

Table 2.6: Partial sums $s_i(\phi_A)$ and the corresponding lower bounds s_i^L, where $1 \leq i \leq h$, for a guest graph $G = (V, E)$ of height $h_G = 5$.

Thus in the case $h_G = 5$ the bound of inequality (2.54) does not match the objective function value corresponding to ϕ_A because $s_3(\phi_a) > s_3^L$. Now, a natural question arises:

Question: Does the bound in (2.54) match the optimal value of the objective function of the $DAPT(G, 2)$ if $h_G = 5$?

We show that ϕ_A is an optimal arrangement if $h_G = 5$, and hence the answer to the above question is "no". Indeed, assume that ϕ_A is not optimal and let ϕ_* be an optimal arrangement. Observe that $s_6(\phi_*) = 1$ has to hold because $s_6(\phi_*) \geq 2$ leads to a contradiction as follows:

$$\begin{aligned} OV(G,2,\phi_*) - OV(G,2,\phi_A) &= 2\sum_{i=1}^{6}\big(s_i(\phi_*) - s_i(\phi_A)\big) \\ &= 2\sum_{i=1}^{5}\big(s_i(\phi_*) - s_i(\phi_A)\big) + 2\big(s_6(\phi_*) - s_6(\phi_A)\big) \\ &> 2\sum_{i=1}^{5}(s_i^L - s_i(\phi_A)) + 2 = -2 + 2 = 0. \end{aligned} \tag{2.55}$$

By similar arguments we would get $s_i(\phi_*) = s_i^L = s_i(\phi_A)$ for $i \in \{1,2,4,5\}$. Since $s_6(\phi_*) = 1$ there will only be one edge of G such that its endpoints are mapped to leaves of the right and the left basic subtrees of T, respectively. Clearly this edge can only be (v_1, v_2) or (v_1, v_3). Assume w.l.o.g. that this edge is (v_1, v_3) and that ϕ_* arranges the right basic subtree of G (of height 4) at the leaves of the right basic subtree T^r of T. Since algorithm $\mathcal{A}$ yields an optimal arrangement for $h_G \leq 4$, we can than assume w.l.o.g. that ϕ_* and ϕ_A arrange the right basic subtree of G in the same way. Now consider the left basic subtree of G together with the root v_1 and denote this subgraph of G by G_1. ϕ_* arranges G_1 at the leaves of the left basic subtree T^l of T. Let $G_1^{(a)}$ and $G_1^{(b)}$ the two subgraphs of G_1 arranged by ϕ_* at the leaves of the left and the right basic subtrees of T^l, denoted by T^{ll} and T^{lr}, respectively. Since the number of edges which contribute by at least $2 \cdot 5$ to the value $OV(G,2,\phi_*)$ is $s_5(\phi_*) = 4$, there are just two edges e_i of G_1 such that ϕ_* arranges one endpoint of e_i to some leaf of T^{ll} and the other endpoint of e_i to some leaf of T^{lr}, for $i = 1, 2$. Recalling that $\big|V(G_1^{(a)})\big| = \big|V(G_1^{(b)})\big|$ we can easily convince ourselves (in the worst case by using complete enumeration) that one of the edges e_i, $i = 1, 2$, has to coincide with one of the two edges (v_1, v_2) or (v_2, v_5) (or (v_2, v_4), symmetrically). We obtain two cases. (A) If $e_1 = (v_1, v_2)$, then $e_2 = (v_2, v_5)$ must hold. (B) Otherwise, if $e_1 = (v_2, v_5)$ and $e_2 \neq (v_1, v_2)$, then e_2 has to join some leave of the left basic subtree of $G_1 \setminus \{v_1\}$ to its father, e.g. $e_2 = (v_{19}, v_{39})$. The edges e_i, $i = 1, 2$, fully determine the corresponding subgraphs $G_1^{(a)}$ and $G_1^{(b)}$, each of them having 16 vertices. For every realisation of e_i, $i = 1, 2$, the problems $DAPT(G_1^{(a)}, 2)$ and $DAPT(G_1^{(b)}, 2)$ can be solved by complete enumeration to observe that the correpsonding optimal values coincide with the values of the objective function

corresponding to the arrangement of the respective subgraphs according to ϕ_A. Notice that it is enough to do the complete enumeration for the DAPTs resulting in the case A and for the DAPTs resulting in the case B with $e_2 = (v_{19}, v_{39})$; all other possible realisations of e_2 in case B lead to $G_1^{(a)}$, $G_1^{(b)}$ which are isomorphic to $G_1^{(a)}$, $G_1^{(b)}$ obtained for $e_2 = (v_{19}, v_{39})$, respectively.

Notice finally, that the answer "no" to the question posed above is not surprising. In general a collection of optimal 2^k-balanced partitions, $1 \leq k \leq h_G$, of a binary tree G of height h_G, does not need to coincide with the 2^k-balanced partitions of G defined in accordance with some feasible solution of the data arrangement problem in G. The reason is that the collection of 2^k-balanced partitions, $1 \leq k \leq h_G$, which arises in accordance with some arrangement ϕ, is laminar, meaning that the partition sets of the 2^k-balanced partition are obtained as particular partitions of the partition sets of the 2^{k-1}-balanced partition, for $2 \leq k \leq h_G$. More concretely, if $\mathscr{V} = \left\{V_1^{(1)}, V_1^{(2)}\right\}$ is the 2-balanced partition in the laminar collection of balanced partitions, then the 4-balanced partition is obtained by partitioning $V_1^{(1)}$ and $V_1^{(2)}$ into 2-balanced partitions each, and so on, until the partition sets are pairs of vertices, and hence a 2^{h_G}-balanced partition results. On the other side, in general, a collection of optimal 2^k-balanced partitions of a complete binary tree G of height h_G, for $1 \leq k \leq h_G$, is not necessarily laminar.

Now we can state the main result of this section.

Theorem 2.16. *Let the guest graph $G = (V, E)$ and the host graph T be binary regular trees of heights $h_G \geq 1$ and $h = h_G + 1$, respectively. Then Algorithm 2.1 is a $\frac{203}{200}$-approximation algorithm.*

Proof. The cases $h_G = 0$, $h_G = 1$, $h_G = 2$ are obvious: The arrangements ϕ_A obtained form Algorithm 2.1 are optimal in these cases, respectively, as one can convince himself by simple arguments or by full enumeration (see also Figures 2.17 to 2.22 in Appendix). Moreover, we have already proved that Algorithm 2.1 yields an optimal solution for $h_G = 4$ (see Table 2.5) and $h_G = 5$ (see Example 2.7).

Let $h_G \geq 6$. Consider the difference

$$OV(G, 2, \phi) - 2\sum_{i=1}^{h} s_i^L = 2\sum_{i=1}^{h} s_i(\phi_A) - 2\sum_{i=1}^{h} s_i^L, \tag{2.56}$$

and set

$$D := \sum_{i=1}^{h} \left(s_i(\phi_A) - s_i^L\right). \tag{2.57}$$

As mentioned above $s_h^L = 1 = s_h(\phi_A)$.

For $s_2^L = c(G, \mathscr{V}_2^*)$ we have $k' = h_G - 2 + 2 = h_G$ and $k = 2^{k'} = 2^{h_G}$, and thus we can apply Lemma 2.13 and Corollary 2.15 to obtain $s_2^L = \frac{4}{3}2^{h_G} - \frac{3}{2} + \frac{1}{6}(-1)^{h_G} = s_2(\phi_A)$.

Let us consider $s_i^L = c(G, \mathscr{V}_i^*)$, where $3 \leq i \leq h_G$. If $1 \leq k' \leq \left\lfloor \frac{h_G}{2} \right\rfloor + 1$, with $k' = h_G - i + 2$ we obtain the condition $i \geq h_G - \left\lfloor \frac{h_G}{2} \right\rfloor + 1$. For $h_G \geq 3$ the inequality $h_G - \left\lfloor \frac{h_G}{2} \right\rfloor + 1 \geq 3$ holds and hence Corollary 2.15 applies. So by applying Lemma 2.13 we get $s_i^L = \frac{3}{2}2^{h_G - i + 2} - 2 = 6 \cdot 2^{h_G - i} - 2 = s_i(\phi_A)$ if $i \geq h_G - \left\lfloor \frac{h_G}{2} \right\rfloor + 1$.

The remaining indices are $3 \leq i \leq h_G - \left\lfloor \frac{h_G}{2} \right\rfloor$. Apply Corollary 2.15 and Lemma 2.13 to obtain:

$$D \leq \sum_{i=3}^{h_G - \left\lfloor \frac{h_G}{2} \right\rfloor} \left(6 \cdot 2^{h_G - i} - \frac{10}{7} 2^{h_G - i + 2} \right) = \frac{2}{7} 2^{h_G} \sum_{i=3}^{h_G - \left\lfloor \frac{h_G}{2} \right\rfloor} 2^{-i}. \tag{2.58}$$

The sum of the above geometric progression is

$$\sum_{i=3}^{h_G - \left\lfloor \frac{h_G}{2} \right\rfloor} 2^{-i} = \frac{1}{4} - \frac{1}{2} 2^{h_G - \lceil \frac{h_G}{2} \rceil} \leq \frac{1}{4} - \frac{\sqrt{2}}{2} 2^{-\frac{h_G}{2}}. \tag{2.59}$$

Summarizing we get

$$D \leq \frac{2}{7} 2^{h_G} \left(\frac{1}{4} - \frac{\sqrt{2}}{2} 2^{-\frac{h_G}{2}} \right) = \frac{1}{14} 2^{h_G} - \frac{\sqrt{2}}{7} 2^{-\frac{h_G}{2}}. \tag{2.60}$$

By applying Lemma 2.8 we get the following approximation ratio ρ

$$\begin{aligned} \rho(h_G) &:= \frac{OV(G, 2, \phi_A)}{OV(G, 2, \phi_A) - 2D} \\ &= \frac{\frac{29}{3} \cdot 2^{h_G} - 4h_G - 9 + \frac{1}{3}(-1)^{h_G}}{\left(\frac{29}{3} \cdot 2^{h_G} - 4h_G - 9 + \frac{1}{3}(-1)^{h_G} \right) - 2 \left(\frac{1}{14} 2^{h_G} - \frac{\sqrt{2}}{7} 2^{-\frac{h_G}{2}} \right)}. \end{aligned} \tag{2.61}$$

After some standard algebraic transformations we obtain

$$\rho(h_G) := \frac{\frac{29}{3} 2^{h_G} - 4h_G - \frac{26}{3}}{\frac{200}{21} 2^{h_G} - 4h_G + \frac{2\sqrt{2}}{7} 2^{\frac{h_G}{2}} - \frac{28}{3}}. \tag{2.62}$$

Obviously,

$$\rho(h_G) \leq \rho^U(h_G) := \frac{\frac{29}{3} 2^{h_G}}{\frac{200}{21} 2^{h_G} - 4h_G + \frac{2\sqrt{2}}{7} 2^{\frac{h_G}{2}} - \frac{28}{3}}. \tag{2.63}$$

We show that $\rho^U(h_G)$ is a strictly monotonically increasing sequence for $h_G \geq h_G^{(0)}$, where $h_G^{(0)} \geq 6$ is a constant. First, note that the denominator corresponds to a valid positive lower bound and therefore $\frac{200}{21}2^{h_G} - 4h_G + \frac{2\sqrt{2}}{7}2^{\frac{h_G}{2}} - \frac{28}{3} > 0$. The following inequalities are all equivalent.

$$\rho^U(h_G) < \rho^U(h_G+1) \tag{2.64}$$

$$\Leftrightarrow$$

$$\frac{\frac{29}{3}2^{h_G}}{\frac{200}{21}2^{h_G} - 4h_G + \frac{2\sqrt{2}}{7}2^{\frac{h_G}{2}} - \frac{28}{3}} < \frac{\frac{29}{3}2^{h_G+1}}{\frac{200}{21}2^{h_G+1} - 4(h_G+1) + \frac{2\sqrt{2}}{7}2^{\frac{h_G+1}{2}} - \frac{28}{3}} \tag{2.65}$$

$$\Leftrightarrow$$

$$\frac{1}{\frac{200}{21}2^{h_G} - 4h_G + \frac{2\sqrt{2}}{7}2^{\frac{h_G}{2}} - \frac{28}{3}} < \frac{2}{\frac{200}{21}\cdot 2\cdot 2^{h_G} - 4h_G + \frac{2\sqrt{2}}{7}2^{\frac{h_G+1}{2}} - \frac{40}{3}} \tag{2.66}$$

$$\Leftrightarrow$$

$$-4h_G + \frac{2\sqrt{2}}{7}2^{\frac{h_G+1}{2}} - \frac{40}{3} < -8h_G + \frac{4\sqrt{2}}{7}2^{\frac{h_G}{2}} - \frac{56}{3} \tag{2.67}$$

$$\Leftrightarrow$$

$$4h_G + \frac{4}{7}(1-\sqrt{2})\sqrt{2}^{h_G} + \frac{16}{3} < 0 \tag{2.68}$$

Consider a function $f\colon [6,+\infty[\to \mathbb{R}$ defined as follows:

$$f(h_G) = 4h_G + \frac{4}{7}(1-\sqrt{2})\sqrt{2}^{h_G} + \frac{16}{3}. \tag{2.69}$$

Its first derivate is

$$f'(h_G) = 4 + \frac{4}{7}(1-\sqrt{2})\sqrt{2}^{h_G}\ln\sqrt{2}. \tag{2.70}$$

Obviously, there exists only one $h_{G0}^{(f)} = \log_{\sqrt{2}} \frac{4}{\frac{4}{7}(\sqrt{2}-1)\ln\sqrt{2}} \approx 11.215$ with $f'(h_{G0}^{(f)}) = 0$ and since $f'(12) = 4+\ln\sqrt{2}\frac{4}{7}(1-\sqrt{2})2^6 \approx -1.250$, the function $f(h_G)$ is strictly monotonically decreasing for all $h_G \geq 12$. Moreover, $f(17) = 4\cdot 17 + \frac{4}{7}(1-\sqrt{2})\sqrt{2}^{17} + \frac{16}{3} = -\left(\frac{4604}{21} - \frac{1024}{7}\sqrt{2}\right) \approx -12.359$ and thus $f(h_G) = 4h_G + \frac{4}{7}(1-\sqrt{2})2^{\frac{h_G}{2}} + \frac{16}{3} < 0$, i.e. the inequalities (2.64)–(2.68) are fulfilled, for all $h_G \geq 17$.

Consequently $\rho^U(h_G)$ is a strictly monotonically increasing sequence for all $h_G \geq h_G^{(0)} = 17$. We already proved that Algorithm 2.1 yields an optimal

solution for $h_G \in \{1, 2, 3, 4, 5\}$. We can also define

$$\rho \coloneqq \max\left\{\rho(h_G) | h_G \in \{6, 7, \ldots, 17\}, \lim_{h_G \to +\infty} \rho^U(h_G)\right\}, \tag{2.71}$$

where we use the tighter bound in equation (2.62) for all the cases $h_G = 6, 7, \ldots, 17$.

Finally, it is not difficult to evaluate $\rho(h_G)$ for all $h_G = 6, 7, \ldots, 17$ and to estimate $\lim_{h_G \to +\infty} \rho^U(h_G)$. It turns out that

$$\rho = \lim_{h_G \to +\infty} \rho(h_G) = \lim_{h_G \to \infty} \frac{\frac{29}{3} 2^{h_G} - 4h_G - \frac{26}{3}}{\frac{200}{21} 2^{h_G} - 4h_G + \frac{2\sqrt{2}}{7} 2^{\frac{h_G}{2}} - \frac{28}{3}} = \frac{203}{200} = 1.015. \tag{2.72}$$

This completes the proof. □

2.5 A generalized approximation algorithm for *d*-regular trees

In this section we generalize the algorithm introduced in Section 2.2 for all d-regular trees where $d \geq 2$. Subsequently, we provide a weaker but more general lower bound which leads to a constant approximation ratio proof.

The generalization of Algorithm 2.1 cannot be done in a straightforward way. In particular, we need to split it into an (i) *initialization* and an (ii) *improvement* phase. The initialization phase yields an arrangement $\phi_{B'}$ which corresponds to the arrangement we would obtain from Algorithm 2.1 applied without the pair-exchanges performed in pseudocode line 9. The improvement phase then processes the pair-exchanges all at once. The resulting arrangement ϕ_B corresponds to the arrangement ϕ_A obtained from Algorithm 2.1 for binary regular trees.

Let us now concentrate on the arrangement $\phi_{B'}$ constructed in the *initialization phase*. First, additional notation is introduced and some easy observations are done. Then the basic idea behind the algorithm is described and after that a pseudocode and a closed formula yielding the objective value $OV(G, d, \phi_{B'})$ is provided.

We define a function $\iota(b_i) \coloneqq i$ which returns the index of the leaf b_i according to the canonical order. Moreover, let $T^k_\phi(v)$ be the subtree of k-th order of the host graph T such that $\phi(v) \in V\big(T^k_\phi(v)\big)$. We call it **subtree of k-th order containing the leaf $\phi(v)$**.

Observation 2.17. *Let $G = (V, E)$ be a d-regular tree with the vertices v_1, v_2, ..., v_n ordered according to the canonical order for some fixed $d \geq 2$. Then the index of the right most leaf of every level can be expressed in the following way:*

$$\max_{1 \leq i \leq n} \{i | level(v_i) = l\} = \frac{d^{l+1} - 1}{d - 1} \text{ for all } 1 \leq l \leq h_G. \tag{2.73}$$

Proof. The observation follows directly from Observation 2.6. □

Observation 2.18. *Let the guest graph $G = (V, E)$ and the host graph T be d-regular trees of heights h_G and $h = h_G + 1$ respectively. Then*

- *the vertices of the guest graph cannot be arranged on the leaves of one basic subtree of the host graph T;*
- *however, they can be arranged on the leaves of two basic subtrees of the host graph T.*

Proof. The claim obviously holds for $h_G = 0$. Let us also assume that $h_G \geq 1$. According to Observation 2.17 we can define

$$V_L := \left\{ v_i \middle| \frac{d^{h_G} - 1}{d - 1} + 1 \leq i \leq \frac{d^{h_G+1} - 1}{d - 1} \right\} \tag{2.74}$$

as the set of leaves of the guest graph G. Obviously, $|V_L| = d^{h_G}$ and $|V \backslash V_L| = \frac{d^{h_G}-1}{d-1}$. Moreover, every basic subtree of the host graph T has $d^{h-1} = d^{h_G}$ leaves. Thus the first claim holds as there exist enough leaves $v_L \in V_L$ of the guest graph G to occupy all leaves of one basic subtree of the host graph T (note that $V \backslash V_L \neq \emptyset$). Moreover, $\frac{d^{h_G}-1}{d-1} \leq d^{h_G}$ and therefore the remaining vertices $v \in V \backslash V_L$ do not exceed the number of the leaves of another basic subtree of the host graph T. This result leads to the latter claim. □

Observation 2.19. *Let $G = (V, E)$ be a d-regular tree with the vertices v_1, v_2, ..., v_n ordered according to the canonical order for some fixed $d \geq 2$ and let v_i, where $1 \leq i \leq \frac{d^{h_G}-1}{d-1}$, be a vertex which is* not *a leaf. Then this vertex has the children v_{c_1}, v_{c_2}, ..., v_{c_d}, where $c_j = d \cdot i - (d - 1) + j$.*

Proof. We first prove that the index of the $(d-1)$-st child of v_i equals $i' = d \cdot i$ for all $1 \leq i \leq \frac{d^{h_G}-1}{d-1}$. Let $l = \text{level}(v_i)$. The number of the vertices v_j with $\text{level}(v_j) = l$ and $j < i$ is obviously $i - \frac{d^l-1}{d-1} - 1$ due to Observation 2.6 and similarly, the number of vertices v_k with $\text{level}(v_k) = l$ and $k > i$ is $\frac{d^{l+1}-1}{d-1} - i$. Therefore between the vertex v_i and its $(d-1)$-st child $v_{i'}$ lie – according to

the canonical order – all vertices v_k with $\text{level}(v_k) = l$ and $k > i$, all children of the vertices v_j with $\text{level}(v_j) = l$ and $j < i$ and the first $d-2$ children of the vertex v_i. Summed up, we get

$$i' = i + \left(\frac{d^{l+1}-1}{d-1} - i\right) + d\left(i - \frac{d^l-1}{d-1} - 1\right) + (d-2) + 1 = d \cdot i. \quad (2.75)$$

Finally, since the vertex v_i, where $1 \leq i \leq \frac{d^{h_G}-1}{d-1}$ has d children and the $(d-1)$-st child $v_{i'}$ has the index $i' = d \cdot i$ according to the canonical order, we obviously get $c_j = d \cdot i - (d-1) + j$. □

In order to proceed, further notation is required. Let $v \in V$ be a vertex of the guest graph $G = (V, E)$ which is *not* a leaf and let $\widehat{G_1}$, $\widehat{G_2}$, ..., $\widehat{G_d}$ be the subtrees of $(\text{level}(v)+1)$-st order rooted at the children of v and ordered according to the canonical order. Then we define

$$V_v := \left(\cup_{j=1}^{d-1} V(\widehat{G_j})\right) \cup \{v\}. \quad (2.76)$$

Observation 2.20. *Let $G = (V, E)$ be a d-regular tree for some fixed $d \geq 2$ and let $v \in V$ be a vertex which is* not *a leaf. Then*

$$|V_v| = d^{h_G - l}, \quad (2.77)$$

where $l = level(v)$.

Proof. We know that all subtrees $\widehat{G_1}, \widehat{G_2}, \ldots, \widehat{G_d}$ have the height $\widehat{h} = h_G - (l+1) = h_G - l - 1$. According to Observation 2.6 we get $\left|V(\widehat{G_j})\right| = \frac{d^{(h_G-l-1)+1}-1}{d-1} = \frac{d^{h_G-l}-1}{d-1}$ for all $1 \leq j \leq d$. Since we consider only $d-1$ subtrees in our claim, we obtain $\left|\left(\cup_{j=1}^{d-1} V(\widehat{G_j})\right)\right| = (d-1)\frac{d^{h_G-l}-1}{d-1} = d^{h_G-l} - 1$ and after adding the vertex v itself we get the equation (2.77). □

Before going into details let us describe the main idea of our algorithm. As demonstrated in Observation 2.18, we need at least the leaves of two basic subtrees of the host graph T to accommodate all vertices $v \in V$ of the guest graph G. Hence there always exists at least one edge $e \in E$ which contributes $2h$ to the objective function value $OV(G, d, \phi)$ in every arrangement ϕ. Consider the root v_1 of the guest graph G. According to Observation 2.20 $|V_{v_1}| = d^{h_G-l} = d^{h_G-0} = d^{h_G}$ and since the left most basic subtree of the host graph T has exactly $d^{h-1} = d^{h_G}$ leaves, we can arrange all vertices $v \in V_{v_1}$ on the leaves of the first and all remaining vertices $v \in V \backslash V_{v_1}$

on the leaves of the second basic subtree of the host graph T. Let us consider an arrangement $\phi_{B'}$ such that

$$\{\phi_{B'}(v)|v \in V_{v_1}\} = \{b_k|1 \leq k \leq d^{h-1}\} \text{ and} \tag{2.78}$$
$$\{\phi_{B'}(v)|v \in V\backslash V_{v_1}\} = \{b_k|d^{h-1}+1 \leq k \leq 2d^{h-1}\}. \tag{2.79}$$

Thus we have only one edge (v_1, v_{d+1}) contributing $2h$ to the objective function value $OV(G, d, \phi_{B'})$ in the arrangement $\phi_{B'}$. So the problem splits into two subproblems. The induced subgraph $G[V\backslash V_{v_1}]$ corresponds to a d-regular tree of height $h_G - 1$ which has to be arranged on the leaves of a d-regular subtree of height h_G and, therefore, we can solve it recursively. Nevertheless the induced subgraph $G[V_{v_1}]$ has a different structure. We have

$$G\big[V_{v_1}\big] = G\big[(\cup_{j=1}^{d-1}\widehat{G_j}) \cup \{v\}\big] \tag{2.80}$$

according to (2.76) and we would like to reserve one basic subtree of 2nd order of the host graph T for every basic subtree $\widehat{G_j}$ of the guest graph G, where $1 \leq j \leq d-1$. However, we need at least two such subtrees of 2nd order for every subtree $\widehat{G_j}$ of the guest graph G, $1 \leq j \leq d-1$, according to Observation 2.18. We can solve the problem by applying the following idea: Let v_{c_j} be the root vertex of $\widehat{G_j}$ for $1 \leq j \leq d-1$. According to Observation 2.20 $|V_{v_{c_j}}| = d^{h_G-1}$. Thus we can arrange the vertices of every set $V_{v_{c_j}}$, where $1 \leq j \leq d-1$, on the leaves of one separate subtree of 2nd order of the host graph T according to some arrangement which would fulfill an analogon of equation (2.78). We use the $d-1$ left most subtrees of this kind for the vertices of the sets $V_{v_{c_j}}$, where $1 \leq j \leq d-1$, and the d-th subtree of 2nd order, let us call it **cache subtree**, we use to arrange the vertices $v \in V_{v_1}\backslash\big(\cup_{j=1}^{d-1} V_{v_{c_j}}\big)$. As a consequence, every subtree $\widehat{G_j}$ considered in (2.80), where $1 \leq j \leq d-1$, contains an edge $e_j \in E(\widehat{G_j})$ contributing $2(h-1)$ to the objective value $OV(G, d, \phi_{B'})$.

Let us now consider the arrangement of the remaining vertices $v \in V_{v_1}\backslash\big(\cup_{j=1}^{d-1} V_{v_{c_j}}\big)$ on the leaves of the cache subtree. Let v'_{c_j} be the right most child of the root vertex of the trees $\widehat{G_j}$ for all $1 \leq j \leq d-1$. We can assign every vertex set $V_{v'_{c_j}}$ to a subtree of 3rd order contained in the cache subtree according to the same principle as used to arrange the children of v_1, i.e. the vertices v_{c_j}, $1 \leq j \leq d-1$.

In the following the algorithm is defined and considered in detail.

Require: d-regular tree $G = (V, E)$ with $|V| = n$ of height h_G whose vertices are labeled according to the canonical order

Ensure: arrangement $\phi_{B'}$
1: $\xi(v_1) := d - 1$;
2: $\phi_{B'}(v_1) := b_{d^{h_G}}$;
3: **for** $i := 1$ **to** $\frac{d^{h_G}-1}{d-1}$ **do**
4: let $l := \text{level}(v_i)$ be the level of v_i;
5: **for** $j := 1$ **to** $d - 1$ **do**
6: let $c_j := d \cdot i - (d-1) + j$ be the index of the j-th child of v_i;[1]
7: $\xi(v_{c_j}) := j$;
8: $\phi_{B'}(v_{c_j}) := b_{\iota(\phi_{B'}(v_i))-(d-j)d^{h_G-l-1}}$;
9: **end for**
10: let $c_d := d \cdot i + 1$ be the right most child of v_i;[2]
11: $\xi(v_{c_d}) := \xi(v_i)$;
12: $\phi_{B'}(v_{c_d}) := b_{\iota(\phi_{B'}(v_i))+(d-\xi(v_i)-1)d^{h_G-l}+\xi(v_i)d^{h_G-l-1}}$;
13: **end for**

Algorithm 2.2: Algorithm $\mathcal{B}'$ which computes the arrangement $\phi_{B'}$.

Before describing the construction of the arrangement $\phi_{B'}$, let us explain the auxiliary variables i, l, j, c_j and the auxiliary index function $\xi(v_i)$, $1 \leq i \leq n$, used in Algorithm 2.2. In general, the algorithm considers the vertices of the guest graph G one after another according to the canonical order and i always stays for the index of the just processed vertex v_i. We call it **active vertex**. The variable $l = \text{level}(v_i)$ represents the level of the *active vertex* v_i and its usage helps to better understand the assignment statements in pseudocode lines 8 and 12. Moreover, the *for* loop between the pseudocode lines 3 and 13 guarantees that $1 \leq i \leq \frac{d^{h_G}-1}{d-1}$ and thus it captures all vertices of the guest graph G but the leaves, which guarantees that all *active vertices* v_i have children. Consequently, the nested *for* loop between the pseudocode lines 5 and 9 iterates over the children v_{c_j} of every *active vertex* v_i one after another; j represents the child index. The children have the indices $c_1, c_2, \ldots, c_d$ according to the canonical order as demonstrated in Observation 2.19.

Let us now explain the meaning of the auxiliary index function $\xi \colon v_i \to \mathbb{N}$ whose values are assigned in pseudocode lines 7 and 11. For all vertices v_{c_j} which are *not* the right most child of their father v_i, the assignment in pseudocode line 7 is executed and $\xi(v_{c_j}) = j$ equals the child index of the vertex v_{c_j}, i.e. the particular value of the variable j. If v_{c_d} is the d-th (i.e. the right most) child of its father v_i according to the canonical order, pseudocode line 11 is processed and $\xi(v_{c_d}) = \xi(v_i)$ is propagated from the

[1] See Observation 2.19.
[2] See Observation 2.19.

father v_i. Since v_1 has no father and thus no child index, we just define $\xi(v_1) = d - 1$ in pseudocode line 1, but it could be any integer $k < d$.

Let us now provide an example illustrating the definition of the auxiliary index function ξ.

Example 2.8. *Let us consider as guest graph $G = (V, E)$ a 3-regular tree of height $h_G = 3$ depicted in Figure 2.13. The auxiliary index function values*

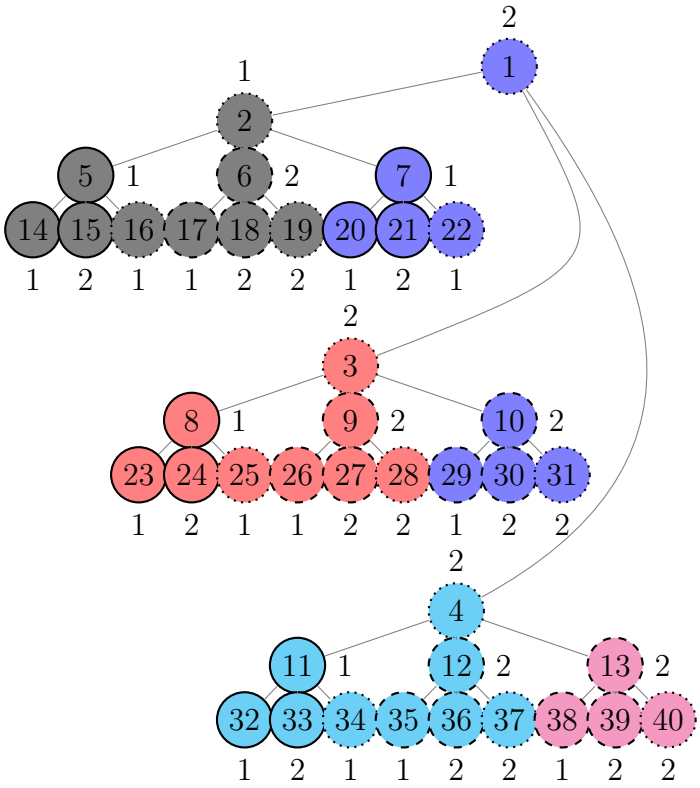

Figure 2.13: Guest graph $G = (V, E)$ (3-regular tree of height $h_G = 3$). The colors are related to the arrangement ϕ depicted in Figure 2.14.

$\xi(v_i)$ are placed next to the vertices v_i for all $1 \leq i \leq n$. We can see that $\xi(v_1) = d - 1 = 3 - 1 = 2$ according to pseudocode line 1. The function values of the first two children of the root v_1, i.e. of the vertices v_2 and v_3, are assigned according to pseudocode line 7 and thus $\xi(v_2) = 1$ and $\xi(v_3) = 2$ (they are the first and the second child of the root vertex v_1, respectively). In contrary, the vertex v_4 is the 3rd (right most) child of the root v_1 according to the canonical order and thus the assignment done in pseudocode line 11 is applied. We get $\xi(v_4) = \xi(v_1) = 2$.

Finally, in order to demonstrate the propagation rule, let us consider the vertex v_{31}. It is the 3rd child of the vertex v_{10} which is again the 3rd child of the vertex v_3. As explained above, $\xi(v_3) = 2$. Thus $\xi(v_{31}) = \xi(v_{10}) = \xi(v_3) = 2$.

Now, we can describe the construction of the arrangement $\phi_{B'}$ performed in pseudocode lines 8 and 12. Let v_i be an *active vertex* (and thus not a leaf), where $1 \leq i \leq \frac{d^{h_G}-1}{d-1}$, let $l = \text{level}(v_i)$ and let v_{c_j}, where $1 \leq j \leq d$, be the children of v_i. Finally, let $V_v := \big(\cup_{j=1}^{d-1} V(\widehat{G_j})\big) \bigcup \{v\}$, where $\widehat{G_1}, \widehat{G_2}, \ldots, \widehat{G_d}$ are the subtrees of (level(v) + 1)-st order rooted at the children of v, be defined as in equation (2.76).

In pseudocode line 8 we use $\iota(\phi_{B'}(v_i))$ as a starting index. According to Observation 2.17 we know that $|V_{c_j}| = d^{h_G-l-1}$ for all $1 \leq j \leq d$ and thus we arrange the children v_{c_j} one after another on the last leaves of the subtrees of $(l+2)$-nd order left to the subtree $T^{l+2}_{\phi_{B'}}(v_i)$ of $(l+2)$-nd order containing the leaf $\phi_{B'}(v_i)$.

In pseudocode line 12 we use $\iota(\phi_{B'}(v_i))$ as a starting index as well, but two positive integers are added to it in this case. In particular, $(d - \xi(v_i) - 1)d^{h_G-l}$ can be understood as a *shift* to the leaves of the *cache subtree* and $\xi(v_i)d^{h_G-l-1}$ specifies the *exact position* among them.

Let us now focus on the shift step. Let v_f be the father of the active vertex v_i and let v_i be its $j^{(f)}$-th child.

(1) If $j^{(f)} < d$, we need to skip over the leaves of $d-j^{(f)}-1$ subtrees of $(l+1)$-st order of the host graph T to reach the *cache subtree*. Since every subtree of $(l+1)$-st order has $d^{h-(l+1)} = d^{h_G+1-l-1} = d^{h_G-l}$ leaves, the resulting shift is $\big(d-\xi(v_i)-1\big)d^{h_G-l}$. Note that $j^{(f)} = \xi(v_i)$ according to the definition of the auxiliary index function ξ in this case.

(2) If $j^{(f)} = d$, we eventually have to skip again over the leaves of some subtrees of $(l+1)$-st order of the host graph T in order to reach the *cache subtree*. However, it is more complicated to determine their number. In particular, the vertex v_i has already been arranged inside of a *cache subtree* in some previous stage of our algorithm. We call it **old cache subtree** now. Let $v_I(v_i)$ be the nearest ancestor of vertex v_i which is not the last child of its father $v_L(v_i)$. We call the vertex $v_L(v_i)$ the **leading vertex** and the vertex $v_I(v_i)$ the **index vertex** of v_i. Then v_i was arranged in the *old cache subtree* with other $d-2$ vertices v with level(v) = level(v_i) having

the same *leading vertex* $v_L(v) = v_L(v_i)$ but different *index vertices* $v_I(v) \neq v_I(v_i)$. Let now $v_L(v_i)$ be the *leading vertex* of v_i and let v_{I_k}, where $1 \leq k \leq d$, be its children. Then the ordering of the vertices v with $\text{level}(v) = \text{level}(v_i)$ having the same *leading vertex* $v_L(v) = v_L(v_i)$ inside the *old cache subtree* was determined by the indices $\xi(v_I(v_k))$, $1 \leq k \leq d$. Exactly this ordering has to be taken into consideration when estimating the number of subtrees of $(l+1)$-st order of the host graph T whose leaves have to be skipped. Since the auxiliary index function ξ propagates the ordering of the *index vertices* v_{I_k}, $1 \leq k \leq d$, to the vertices v with $\text{level}(v) = \text{level}(v_i)$ having the same *leading vertex* $v_L(v) = v_L(v_i)$ we get a shift of $\big(d - \xi(v_i) - 1\big)d^{h_G - l}$.

Now, we have to estimate where to arrange the vertex v_{c_d} inside of the *cache subtree*. Again, analogous to the description above we use either the position of the vertex v_i with respect to its father v_f, if $j^{(f)} < d$, or the position of the *index vertex* $v_I(v_i)$ with respect to its father, the *leading vertex* $v_F(v_i)$, if $j^{(f)} = d$. However, we work with subtrees of $(l+2)$-nd order each having $d^{h_G - l - 1}$ leaves. Thus we obtain the summand $\xi(v_i)d^{h_G - l - 1}$.

Finally, note that we set $\xi(v_1) \coloneqq d - 1$ in pseudocode line 1. Therefore the root v_1 is considered as being a $(d-1)$-st child of its imaginary father. This assignment determines the arrangement of the right most vertices $v = \max_{1 \leq i \leq n} \big\{i | \text{level}(v_i) = l\big\}$ of every level $1 \leq l \leq h_G$.

Next we demonstrate this Algorithm on a 3-regular tree of height $h_G = 3$.

Example 2.9. *Let us consider the guest graph depicted in Figure 2.13. The arrangement $\phi_{B'}$ can be found in Figure 2.14 (the right most basic subtree of the host graph T has not been depicted, because no vertices of the guest graph G are arranged on its leaves). Recall that the set of vertices of a certain color in the guest graph is arranged at the set of leaves of the same color in the host graph T. Moreover, there exist three types of vertex boundaries: a solid one, a dashed one and a dotted one. The graphical representation of an arrangement preserves the boundary property in the sense that e.g. vertices with a solid boundary in G are arranged at solid-boundary leaves of the same color in T. The same principle holds for dashed and for dotted boundaries as well.*

Similarly to other examples, the behavior of Algorithm 2.2 is demonstrated by the coloring and vertex boundaries in Figures 2.13 and 2.14. The root v_1 is arranged according to pseudocode line 2 on the last leaf of the left most basic subtree of the host graph T. Consider the first $d - 1 = 2$ children of

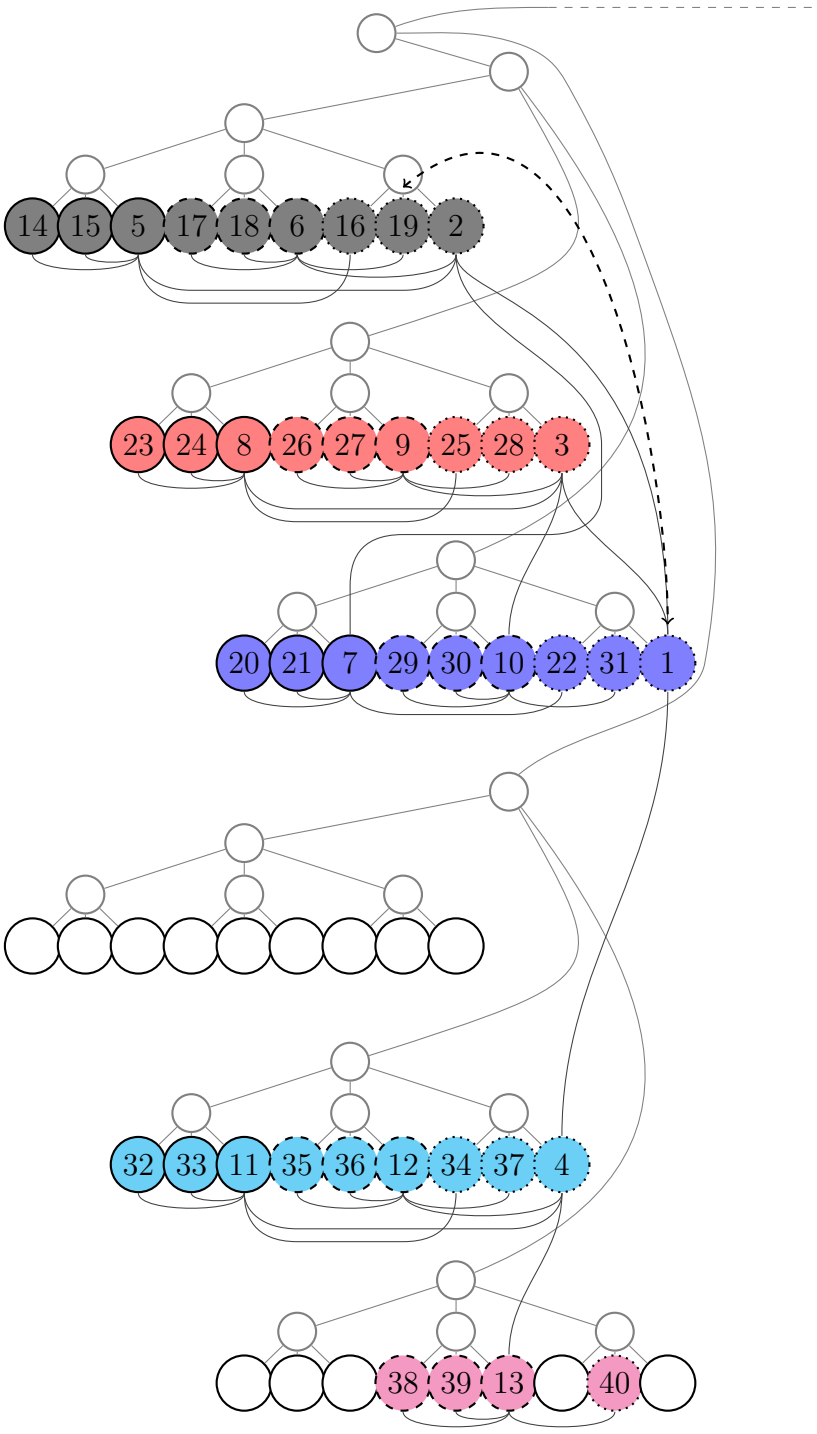

Figure 2.14: Arrangement $\phi_{B'}$ obtained from Algorithm 2.2 for the guest graph $G = (V, E)$ depicted in Figure 2.13. Its objective function value is $OV(G, 3, \phi_{B'}) = 134$.

v_1, *i.e. the vertices* v_2 *and* v_3. *In Figure 2.13 they are arranged on the right most vertices of the subtrees of* 2*nd order of our host graph* T *left to the subtree* $T^2_{\phi_{B'}}(v_1)$ *of* 2*nd order containing the leaf* $\phi_B(v_1)$. *This corresponds to pseudocode line 8 as* $\phi_{B'}(v_2) = b_{\iota(\phi_{B'}(v_i))-(d-j)d^{h_G-l-1}} = b_{27-(3-1)d^{3-0-1}} = b_9$ *and* $\phi_{B'}(v_3) = b_{\iota(\phi_{B'}(v_i))-(d-j)d^{h_G-l-1}} = b_{27-(3-2)d^{3-0-1}} = b_{18}$. *In fact, we can see that all leaves of the first left subtree of* 2*nd order of the host graph* T *are occupied by the vertices* $v \in V_{v_2}$ *(in gray) and similarly, the leaves of the second subtree are occupied by the vertices* $v \in V_{v_3}$ *(in red).*

Let us now focus on the remaining vertices of V_{v_1}, *i.e. the vertices* $v \in V_{v_1} \backslash (V_{v_2} \cup V_{v_3})$. *They are all arranged on the last (third) subtree of* 2*nd order of the host graph* T *(see the blue colored vertices in Figures 2.13 and 2.14). This subtree (in particular the subtree* $T^2_{\phi_{B'}}(v_1)$ *of* 2*nd order containing the leaf* v_1*) remains free for the* cache subtree *if the vertex* v_2 *or* v_3 *becomes active. Let* v_2 *be the* active vertex *now and consider its last child* v_7. *Since* v_2 *was the first child of its father* v_1, *we have to skip over the leaves of* $\xi(v_2) = 1$ *subtrees of* 2*nd order to reach our* cache subtree. *In particular, we skip the leaves of the subtree occupied by the vertices* $v \in V_{v_3}$. *Finally, we have to arrange the vertex* v_7 *on the last leaf of the first subtree of* 3*rd order of the* cache subtree. *Thus we have to add* $d^{h-3} = 3$ *in order to reach the exact position inside the* cache subtree.

Finally, consider e.g. the active vertex v_{10} *with its children* v_{29}, v_{30} *and* v_{31}. *The vertex* v_{10} *has already been arranged on the leaf* b_{24} *in a prior stage of the algorithm. It is not difficult to see that* $\phi_{B'}(v_{29}) = b_{22}$ *and* $\phi_{B'}(v_{30}) = b_{23}$ *according to pseudocode line 8. The case of the last child* v_{31} *is more complicated. The* leading vertex *is* $v_L(v_{10}) = v_1$ *and the* index vertex *is* $v_I(v_{10}) = v_3$. *As demonstrated in Example 2.8,* $\xi(v_{10}) = \xi(v_3) = 2$, *and thus we have to skip over* $d - \xi(v_{10}) - 1 = 3 - 2 - 1 = 0$ *subtrees of* $(l + 1 = 2 + 1 = 3)$*-rd order, i.e. we do not need to consider the shift to the* cache subtree *in this case. In particular, the first leaf of the* cache subtree *is the leaf* $b_{\iota(\phi_{B'}(v_{10}))+1} = b_{24+1} = b_{25}$ *and the* cache subtree *contains the leaves* b_{25}, b_{26} *and* b_{27}. *Nevertheless, we have to move over* $\xi(v_{10}) = 2$ *subtrees of* $(l + 2 = 2 + 2 = 4)$*-th order, i.e. we skip the leaf* b_{25} *and assign* $\phi_{B'}(v_{31}) = b_{\phi_{B'}(v_{10})+0\cdot 3+2\cdot 1} = b_{24+0+2} = b_{26}$.

Before considering the improving phase of our approximation algorithm, we provide an explicite formula on the objective value of the arrangement $\phi_{B'}$. In order to do so, we first have to introduce Lemma 2.21.

Lemma 2.21. *Let* $d, h_G \in \mathbb{N}$ *be integers, where* $d \geq 2$ *and* $h_G \geq 1$. *Then*

$$\sum_{l=0}^{h_G-1} l \cdot d^l = \frac{d^{h_G}}{d-1}(h_G - 1) - \frac{d^{h_G} - d}{(d-1)^2}. \tag{2.81}$$

Proof. The starting point is

$$\sum_{l=0}^{h_G-1} ld^l = 0 \cdot 1 + 1 \cdot d + 2 \cdot d^2 + \ldots + (h_G - 1)d^{h_G-1}. \tag{2.82}$$

This equation can be multiplied by d

$$d \sum_{l=0}^{h_G-1} ld^l = 0 \cdot 1 \cdot d + 1 \cdot d^2 + 2 \cdot d^3 + \ldots + (h_G - 1)d^{h_G}. \tag{2.83}$$

By subtracting (2.82) - (2.83) we get

$$(1-d) \sum_{l=0}^{h_G-1} ld^l = d(1 + d + d^2 + \ldots + d^{h_G-2}) - (h_G - 1)d^{h_G}. \tag{2.84}$$

Now, we can use the formula for the partial sum of a geometry series to obtain

$$(1-d) \sum_{l=0}^{h_G-1} ld^l = d\frac{1 - d^{h_G-1}}{1-d} - (h_G - 1)d^{h_G}. \tag{2.85}$$

Finally, after dividing the last equation by $(1-d)$ we obtain the claim. □

Lemma 2.22. *Let the guest graph $G = (V, E)$ and the host graph T be d-regular trees of heights h_G and $h = h_G + 1$ respectively for some fixed $d \geq 2$. Then the value $OV(G, d, \phi_{B'})$ of the objective function of the DAPT correspondig to the arrangement $\phi_{B'}$ obtained from Algorithm 2.2 is given as follows:*

$$OV(G, d, \phi_{B'}) = \frac{2}{d-1}\left(\frac{d^2 + d - 1}{d-1}\left(d^{h_G} - 1\right) - dh_G\right). \tag{2.86}$$

Proof. This formula directly follows from the description of Algorithm 2.2 above. Let v_i, where $1 \leq i \leq \frac{d^{h_G}-1}{d-1}$, be an *active vertex* and let $l = \text{level}(v_i)$. Moreover, let $v_{c_1}, v_{c_2}, \ldots, v_{c_d}$ be the children of v_i. As shown in the description of Algorithm 2.2, all edges (v_i, v_{c_j}), where $1 \leq j \leq d-1$, contribute $2(h_G - l)$ and the edge (v_i, v_{c_d}) contributes $2(h_G - l + 1)$ to the objective function value $OV(G, d, \phi_{B'})$. As we have d^l vertices v_i with $l = \text{level}(v_i)$, we

obtain

$$\begin{aligned} OV(G,d,\phi_{B'}) &= \sum_{l=0}^{h_G-1} d^l\big((d-1)\cdot 2\cdot(h_G-l)+2\cdot(h_G-l+1)\big) \\ &= 2\sum_{l=0}^{h_G-1} d^l\big(d(h_G-l)+1\big) \\ &= 2\left(d\left(h_G\sum_{l=0}^{h_G-1} d^l - \sum_{l=0}^{h_G-1} l\cdot d^l\right)+\sum_{l=0}^{h_G-1} d^l\right). \end{aligned} \tag{2.87}$$

By using

$$\sum_{l=0}^{h_G-1} d^l = \frac{d^{h_G}-1}{d-1} \tag{2.88}$$

and Lemma 2.21 we get

$$\begin{aligned} &OV(G,d,\phi_{B'}) = \\ &\quad 2\left(d\left(h_G\frac{d^{h_G}-1}{d-1} - \left(\frac{d^{h_G}}{d-1}(h_G-1) - \frac{d^{h_G}-d}{(d-1)^2}\right)\right)+\frac{d^{h_G}-1}{d-1}\right). \end{aligned} \tag{2.89}$$

The lemma follows then by trivial algebraic operations. □

In the following, we focus on the *improvement phase* of the algorithm. The idea behind the particular improvement steps is the same as for binary regular trees. The only difference is that we do the steps all at once. Recall the main principle behind the pair-exchanges done in pseudocode line 9 of Algorithm 2.1 (for a detailed description see Section 2.2, especially the proof of Lemma 2.7): If h_G is odd and $h_G \geq 3$, we arrange the root v_1 together with its left child v_2 on a subtree of h_G-th order by a pair-exchange and thus we guarantee that the edge (v_1, v_2) contributes only 2 instead of $2(h-1)$ to the objective function value $OV(G, 2, \phi_A)$ after the pair-exchange. Moreover, we prove in Lemma 2.7 that the other edges affected by the pair-exchanges do not increase the objective function value as much as decreased by the edge (v_1, v_2).

We can define the *improvement phase* of our generalized algorithm now.

Require: d-regular tree $G = (V, E)$ with $|V| = n$ of height $h_G \geq 3$ whose vertices are labeled according to the canonical order and the arrangement $\phi_{B'}$ obtained from Algorithm 2.2
Ensure: arrangement ϕ_B

1: **for all** vertices $v \in V$ with $h_G - \text{level}(v)$ odd and $h_G - \text{level}(v) \geq 3$ **do**
2: let v_{c_1} be the first child of v;
3: exchange the vertices arranged on the leaves $\phi_{B'}(v)$ and $b_{\iota(\phi_{B'}(v_{c_1}))-1}$ (*pair-exchange*);
4: **end for**

Algorithm 2.3: Approximation algorithm $\mathcal{B}$ which computes the arrangement ϕ_B.

We need two observations in order to analyse the algorithm.

Observation 2.23. *Let the guest graph $G = (V, E)$ and the host graph T be d-regular trees of heights $h_G \geq 3$ and $h = h_G + 1$, respectively, for some fixed $d \geq 2$ and let $\phi_{B'}$ be the arrangement obtained from Algorithm 2.2. Finally, let $B^U = \{\phi_{B'}(v) | v \in V\}$. Then*

$$\{\phi_{B'}(v) | v \in V \text{ is a leaf of } G\} = \{b \in B^U | \iota(b) \not\equiv 0 \mod d\} \tag{2.90}$$

and

$$\{\phi_{B'}(v) | v \in V \text{ is not a leaf of } G\} = \{b \in B^U | \iota(b) \equiv 0 \mod d\}. \tag{2.91}$$

Proof. Consider an *active vertex* $v_i \in V$, where $1 \leq i \leq \frac{d^{h_G}-1}{d-1}$, let $l = \text{level}(v_i)$ and let v_{c_j}, where $1 \leq j \leq d$, be its children ordered according to the canonical order. As already mentioned in the description of Algorithm 2.2, the children v_{c_j}, where $1 \leq j \leq d$, are generally arranged on last leaves of subtrees of $(l+2)$-nd order of the host graph T. Let $\widehat{b}$ be the last leaf of a d-regular tree $\widehat{T}$ of height $\widehat{h} \geq 0$. Then $\iota(\widehat{b}) \not\equiv 0 \mod d$ iff $\widehat{h} = 0$. Every subtree of $(l+2)$-nd order of the host graph T has the height $h-(l+2)$. Thus $h - (l+2) = 0 \Leftrightarrow l = h - 2 = h_G - 1 \Leftrightarrow \text{level}(v_{c_j}) = h_G$ for all $1 \leq j \leq d$. Therefore the children of the *active vertex* v_i are arranged on the leaves b_i with $i \not\equiv 0 \mod d$ iff they are leaves of the guest graph G. □

Let v be a vertex considered in the *for* loop between the pseudocode lines 1 and 4 of Algorithm 2.3 and let v_{c_1} be its first child. By using Observation 2.23 we show that there exists a vertex $x \in G$ arranged on the leaf $\phi_{B'}(x) = b_{\iota(\phi_{B'}(v_{c_1}))-1}$ and moreover, that x is a leaf of the guest graph G. As already argued in the description of Algorithm 2.2, the child v_{c_1} of the vertex v is arranged on the last leaf of a subtree of $(\text{level}(v) + 2)$-nd order of the host graph T in the arrangement $\phi_{B'}$. According to pseudocode line 1 of Algorithm 2.3 we have

$$\text{level}(v) + 2 \leq h_G - 3 + 2 = h_G - 1 \tag{2.92}$$

and thus a subtree of $(\text{level}(v)+2)$-nd order of the host graph T has at least $d^{h_G-(h_G-1)} = d \geq 2$ leaves. Since the vertex v_{c_1} is arranged on the last leaf of a subtree of $(\text{level}(v)+2)$-nd order of the host graph T, we get

$$\iota(\phi_{B'}(v_{c_1})) = a \cdot d \geq 2 \tag{2.93}$$

for some $1 \leq a \leq d^{h-1}$. Thus $\iota(\phi_{B'}(v_{c_1})) \equiv 0 \mod d$ and moreover, there exists a vertex $x \in V$ with $\phi_{B'}(x) = b_{\iota(\phi_{B'}(v_{c_1}))-1}$ and $\phi_{B'}(x) \not\equiv 0 \mod d$. Furthermore, x is a leaf of the guest graph G according to Observation 2.23. Let y be its father. Recall that $\text{level}(x) = h_G$ and $\text{level}(y) = h_G - 1$. As already demonstrated in the proof of Lemma 2.22, the edge (y,x) contributes either $2\big(h_G - \text{level}(y) + 1\big) = 2\big(h_G - (h_G-1) + 1\big) = 4$ or $2\big(h_G - \text{level}(y)\big) = 2\big(h_G - (h_G-1)\big) = 2$ to the objective value $OV(G, d, \phi_{B'})$ depending whether x is the last child of y or not.

Observation 2.24. *Let the guest graph $G = (V,E)$ and the host graph T be d-regular trees of heights $h_G \geq 3$ and $h = h_G + 1$, respectively, for some fixed $d \geq 2$ and let $\phi_{B'}$ be the arrangement obtained from Algorithm 2.2. Finally, let $v_i \in V$, where $1 \leq i \leq \frac{d^{h_G-1}-1}{d-1}$, be a vertex with $h_G - \text{level}(v_i) \geq 3$ and let v_{c_1} be its first child. Then there exists a vertex $x \in V$ arranged on the leaf $\phi_{B'}(x) = b_{\iota(\phi_{B'}(v_{c_1}))-1}$ and x is the last child of its father y.*

Proof. This observation follows from the description of Algorithm 2.2. Consider that $v_i \in V$ is the *active vertex* now. Let $V_v := \big(\cup_{j=1}^{d-1} V(\widehat{G_j})\big) \bigcup \{v\}$, where $\widehat{G_1}, \widehat{G_2}, \ldots, \widehat{G_d}$ are the subtrees of $(\text{level}(v)+1)$-st order rooted at the children of v. As already mentioned in the description of Algorithm 2.2, all vertices $v \in V_{v_i}$ are arranged on the leaves b_k with $\iota\big(\phi_{B'}(v_i)\big) - |V_{v_i}| + 1 \leq k \leq \iota\big(\phi_{B'}(v_i)\big)$. In particular

$$\Big\{\phi_{B'}(v) | v \in V_{v_1}\Big\} = \Big\{b_k | \iota\big(\phi_{B'}(v_i)\big) - |V_{v_i}| + 1 \leq k \leq \iota\big(\phi_{B'}(v_i)\big)\Big\}. \tag{2.94}$$

Consequently, there exists a vertex $x \in V$ arranged on the leaf $\phi_{B'}(x) = b_{\iota(\phi_{B'}(v_{c_1}))-1}$ and moreover, v_i is an ancestor of the vertex x and thus of the vertex y as well (note that y can be the vertex v_{c_1}). Moreover, $\iota(\phi_{B'}(y)) < \iota(\phi_{B'}(x))$ and thus the vertex x was arranged in pseudocode line 12 of Algorithm 2.2, i.e. it is the last child of its father. □

We can also conclude that the edge (y,x) (see the proof above) contributes 4 to the objective function value $OV(G, d, \phi_{B'})$ in the arrangement $\phi_{B'}$.

Lemma 2.25. *Let the guest graph $G = (V,E)$ and the host graph T be d-regular trees of heights $h_G \geq 3$ and $h = h_G + 1$ respectively for some fixed*

$d \geq 2$. Then the pair-exchange defined in Algorithm 2.3 in pseudocode line 3 decreases by 1 the number of edges which contribute to the objective by 4, increases by 1 the number of edges which contribute to the objective value by 2, and does not change the number of edges which contribute to the objective value by $2i$ for $i \geq 3$. Summarizing such a pair-exchange improves the value of the objective function by 2 as compared to the value corresponding to the arrangement available prior to this pair-exchange.

Proof. By using Observations 2.23 and 2.24 the proof is the same as the proof of Lemma 2.7 for binary regular trees given in Section 2.2. □

We can now express the number of pair-exchanges done in Algorithm 2.3.

Lemma 2.26. *Let the guest graph $G = (V, E)$ and the host graph T be d-regular trees of heights h_G and $h = h_G + 1$, respectively, for some fixed $d \geq 2$. Then the number of pair-exchanges p done in Algorithm 2.3 is given by*

$$p = \begin{cases} 0 & \text{for } h_G = 0 \\ \frac{1}{(d^2-1)d} d^{h_G} - \frac{1}{2(d-1)} - \frac{1}{2(d+1)}(-1)^{h_G} & \text{for } h_G > 0 \end{cases}. \tag{2.95}$$

Proof. The lemma obviously holds for $h_G = 0$. Moreover, for $h_G = 1$ and for $h_G = 2$ we obtain $p = 0$ which also corresponds to Algorithm 2.3. For $h_G \geq 3$ we distinguish two cases.

For h_G odd we obviously consider all vertices $v \in V$ with level(v) even and level$(v) \leq h_G - 3$ in the *for* loop in pseudocode line 1 of Algorithm 2.3. Moreover, we have d^l vertices $v \in V$ with level$(v) = l$. Therefore

$$\begin{aligned} p &= d^0 + d^2 + d^4 + \ldots + d^{h_G-3} \\ &= (d^2)^0 + (d^2)^1 + (d^2)^2 + \ldots + (d^2)^{\frac{h_G-3}{2}} \\ &= \sum_{i=0}^{\frac{h_G-3}{2}} (d^2)^l = \frac{d^{h_G-1} - 1}{d^2 - 1}. \end{aligned} \tag{2.96}$$

For h_G even we similarly consider all vertices $v \in V$ with level(v) odd and level$(v) \leq h_G - 3$ in the *for* loop in pseudocode line 1 of Algorithm 2.3. Thus

$$\begin{aligned} p &= d + d^3 + d^5 + \ldots + d^{h_G-3} \\ &= d \cdot (d^2)^0 + d \cdot d^2 + d \cdot d^4 + \ldots + d \cdot (d^2)^{\frac{h_G-4}{2}} \\ &= d \sum_{i=0}^{\frac{h_G-4}{2}} (d^2)^l = d\frac{d^{h_G-2} - 1}{d^2 - 1}. \end{aligned} \tag{2.97}$$

Finally, (2.95) reduces to (2.96) for h_G odd and to (2.97) for h_G even by simple algebraic operations. □

Now, we can express the objective function value $OV(G, d, \phi_B)$ of the arrangement ϕ_B obtained from Algorithm 2.3.

Lemma 2.27. *Let the guest graph $G = (V, E)$ and the host graph T be d-regular trees of heights h_G and $h = h_G + 1$ respectively. Then the value $OV(G, d, \phi_B)$ of the objective function of the DAPT corresponding to the arrangement ϕ_B obtained from Algorithm 2.3 is given as follows:*

$$OV(G,d,\phi_B) = \begin{cases} 0 \ \textit{for } h_G = 0 \\ 2\left(\frac{d^4+2d^3-2d+1}{(d-1)^2d(d+1)}d^{h_G} - \frac{d}{d-1}h_G - \frac{2d^2+d-1}{2(d-1)^2} + \frac{1}{2(d+1)}(-1)^{h_G}\right) \\ \quad \textit{for } h_G > 0 \end{cases} . \tag{2.98}$$

Proof. The case $h_G = 0$ is obvious. Let $h_G \geq 1$ now. We apply Lemma 2.25 and obtain

$$OV(G,d,\phi_B) = OV(G,d,\phi_{B'}) - 2p, \tag{2.99}$$

where p denotes the number of pair-exchanges done in Algorithm 2.3.

Finally, by using Lemmas 2.22 and 2.26 we can write

$$\begin{aligned} &OV(G,d,\phi_B) = \\ &\frac{2}{d-1}\left(\frac{d^2+d-1}{d-1}\left(d^{h_G}-1\right) - dh_G\right) - \\ &2\left(\frac{1}{(d^2-1)d}d^{h_G} - \frac{1}{2(d-1)} - \frac{1}{2(d-1)}(-1)^{h_G}\right). \end{aligned} \tag{2.100}$$

The claim in equation (2.98) follows by trivial algebraic operations.

□

Corollary 2.28. *Let the guest graph $G = (V, E)$ and the host graph T be binary regular trees of heights h_G and $h = h_G+1$ respectively. Let $OV(G, 2, \phi_B)$ and $OV(G, 2, \phi_A)$ be the values of the objective function of the DAPT corresponding to the arrangements ϕ_B and ϕ_A, respectively, where ϕ_B is the arrangement obtained form Algorithm 2.3 and ϕ_A is the arrangement obtained from Algorithm 2.1. Then*

$$OV(G,2,\phi_B) = OV(G,2,\phi_A). \tag{2.101}$$

Proof. By substituting $d = 2$ in equation (2.98) of Lemma 2.27 we get Lemma 2.8 by trivial algebraic operations. Our claim follows obviously. □

Corollary 2.29. *Let the guest graph $G = (V, E)$ and the host graph T be binary regular trees of heights $h_G \geq 1$ and $h = h_G + 1$, respectively. Then Algorithm 2.3 is a $\frac{203}{200}$-approximation algorithm in this special case.*

Proof. Theorem 2.16 guarantees that Algorithm 2.1 is a $\frac{203}{200}$-approximation algorithm. Furthermore, according to Corollary 2.28, if G is a binary regular tree, $OV(G, 2, \phi_B) = OV(G, 2, \phi_A)$, where ϕ_B is the arrangement obtained from Algorithm 2.3 and ϕ_A the arrangement obtained from Algorithm 2.1. Thus if $d = 2$, the $\frac{203}{200}$-approximation ratio holds for Algorithm 2.3 as well. □

In fact, $\phi_B = \phi_A$ for binary regular trees. However, we do not provide a proof of this claim since we need just the objective value, i.e. Corollary 2.29, in our approximation ratio proof later.

Let us now consider an example illustrating the behavior of Algorithm 2.3.

Example 2.10. *Let us again consider the guest graph $G = (V, E)$ of height $h_G = 3$ depicted in Figure 2.13. We have only one vertex $v_1 \in V$ with level$(v_1) = 3$ odd and with $h_G - \text{level}(v_1) = 3 - 3 \geq 0$. This vertex is arranged on the leaf $\phi_{B'}(v_1) = b_{27}$ and its left most child, i.e. the vertex v_2, is arranged on the leaf $\phi_{B'}(v_2) = b_9$ in the arrangement $\phi_{B'}$. Thus we exchange the vertices arranged on the leaves b_{27} and $b_{9-1} = b_8$ according to pseudocode line 1 of Algorithm 2.3 as marked by the arrows in the dashed line in Figure 2.14. In particular, we set $\phi_B(v_1) := b_8$, $\phi_B(v_{19}) := b_{27}$ and $\phi_B(v_i) := \phi_{B'}(v_i)$ for all $1 \leq i \leq n$, $i \neq 1$, $i \neq 19$. The resulting arrangement ϕ_B and the corresponding coloring of the guest graph are depicted in Figures 2.15 and 2.16.*

Finally, this pair-exchange improves the objective function value in accordance with Lemma 2.25: $OV(G, 3, \phi_B) - OV(G, 3, \phi_{B'}) = 2 = 134 - 132 = 2$.

In order to provide an approximation ratio ρ for Algorithm 2.3, we need to generalize the lower bound introduced in Section 2.4. We use the same concepts and arguments as for binary regular trees and thus we refer to Section 2.4 for more details. Let $s_i(\phi)$, where $1 \leq i \leq h$, be the partial sums defined in the same way as for binary regular tree (see equation (2.40)), i.e. $s_i(\phi)$ is the number of edges of the guest graph G which contribute with at least $2i$ to the objective value $OV(G, d, \phi)$, for all $1 \leq i \leq h$. Recall, that $OV(G, 2, \phi) = 2 \sum_{i=1}^{h} s_i(\phi)$ (see equation (2.42)). Now, we bound the partial sums $s_i(\phi)$ by lower bounds s_i^L such that $s_i(\phi) \geq s_i^L$ holds for all $1 \leq i \leq h$ and for all arrangements ϕ. The general lower bound is then given in the

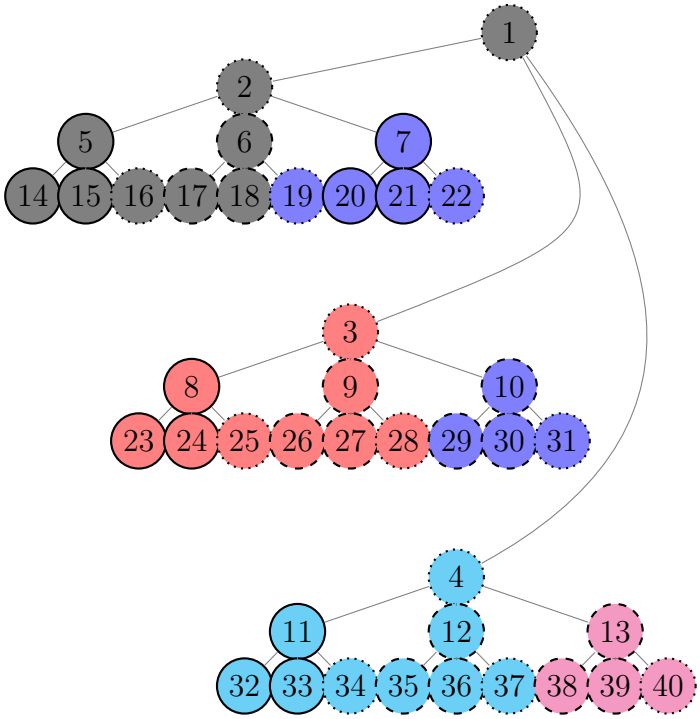

Figure 2.15: Guest graph $G = (V, E)$ (3-regular tree of height $h_G = 3$). The colors are related to the arrangement ϕ depicted in Figure 2.16.

same way as for the binary regular trees, i.e. $OV(G, 2, \phi) \geq 2 \sum_{i=1}^{h} s_i^L$ holds for all arrangements ϕ.

Definition 2.30 (partitioning problem into sets of bounded cardinality). *Given a graph $G = (V, E)$ with $|V| = n$ and an integer $r \geq 1$, a* **partition into sets of bounded cardinality** *is a partition of the vertex set V into k non-empty* **partition sets** *$V_1 \neq \emptyset$, $V_2 \neq \emptyset$, ..., $V_k \neq \emptyset$, where $\lceil \frac{n}{r} \rceil \leq k \leq n$, $\cup_{i=1}^{k} V_k = V$, $V_i \cap V_j = \emptyset$, for all $i \neq j$, and $|V_i| \leq r$ for all $1 \leq i \leq k$. The* **partitioning problem into sets of bounded cardinality (PPSBC)** *asks for a partition into sets of bounded cardinality $\mathscr{U}$ which minimizes*

$$c^{\mathscr{U}}(G, \mathscr{U}) := \Big| \big\{ (u, v) \in E | u \in V_i,\, v \in V_j,\, i \neq j \big\} \Big|, \tag{2.102}$$

where $\mathscr{U} := \{V_i | 1 \leq i \leq k\}$.

This problem plays only an auxiliary role in this thesis and we just refer to Koivisto [31] for details about its complexity. The PPSBC is very similar to the k-BPP. However, not the number but the cardinality of the partition

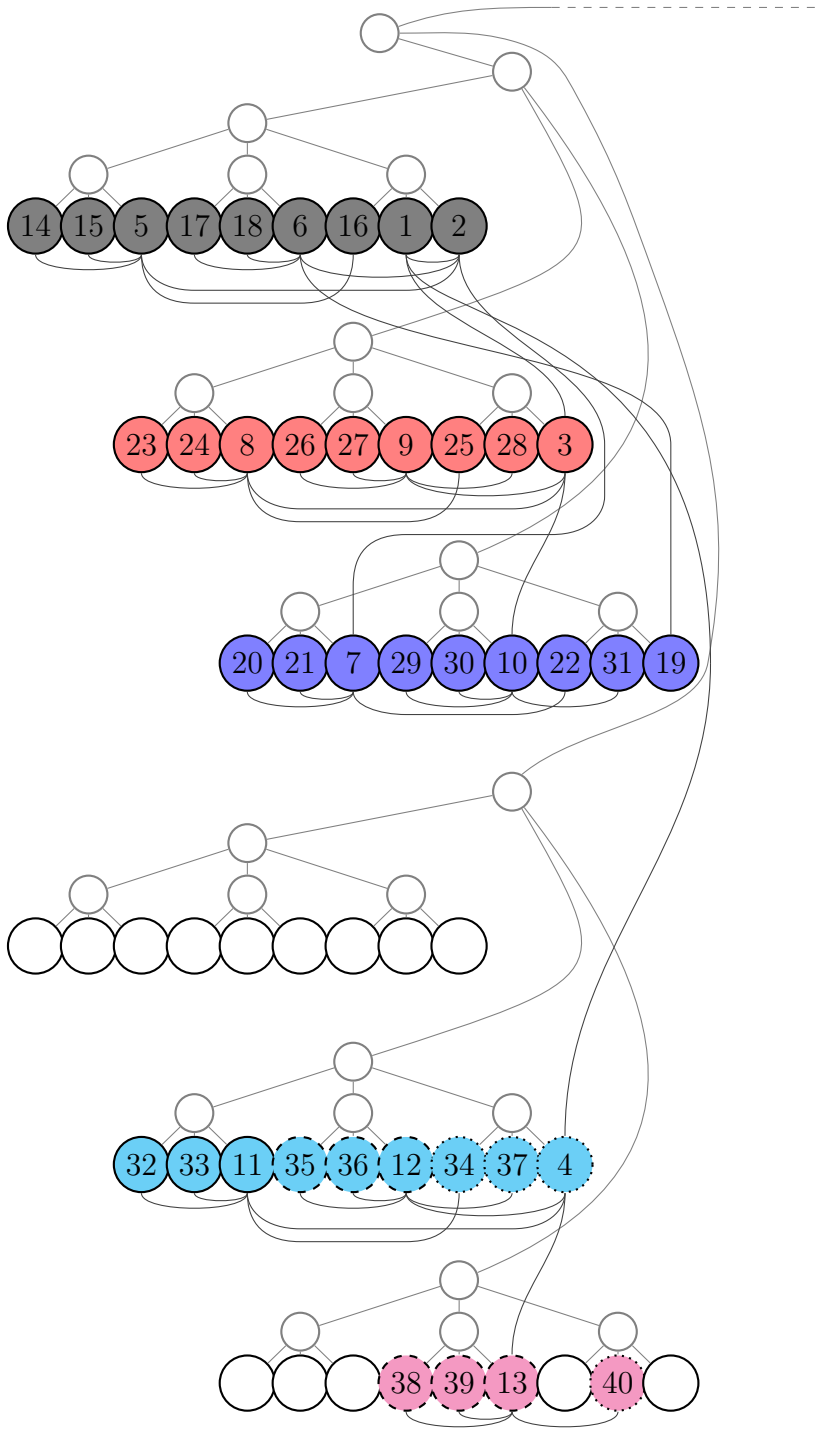

Figure 2.16: Arrangement ϕ_B obtained from Algorithm 2.2 for the guest graph $G = (V, E)$ depicted in Figure 2.15. Its objective function value is $OV(G, 3, \phi_{B'}) = 132$.

sets is bounded. The PPSBC and the k-BPP have one important property in common (cf Subsection 2.3.2). Consider the input graph $G = (V, E)$ of the PPSBC, a partition into sets of bounded cardinality $\mathscr{U} = \{V_1, V_2, \ldots, V_k\}$, and the respective induced subgraphs $G[V_i]$, $1 \leq i \leq k$. Assume that $G[V_i]$ has l_i connected components $G_{i,j} = (V_{i,j}, E_{i,j})$, $1 \leq j \leq l_i$, for every i, $1 \leq i \leq k$. Define a new graph $G' = (V', E')$ which contains one representative vertex $\bar{v}_{i,j}$ for each connected component $G_{i,j}$, $1 \leq i \leq k$, $1 \leq j \leq l_i$. Two vertices $\bar{v}_{i_1,j_1}$ and $\bar{v}_{i_2,j_2}$ are connected in G' iff the connected components G_{i_1,j_1}, G_{i_2,j_2} are connected by an edge in G. Observe that if G is a tree, then G' is also a tree and the following equality holds:

$$c^{\mathscr{U}}(G, \mathscr{U}) = |E'| = |V'| - 1. \tag{2.103}$$

In particular, if G is a tree, the objective function value $c(G, \mathscr{U})$ of a partition $\mathscr{U}$ into sets of bounded cardinality equals the overall number of the connected components of the subgraphs induced in G by the partition sets of V minus 1. The construction of the graph $G' = (V', E')$ is illustrated in Example 2.5 (the 16-balanced partition $\mathscr{V}^*$ of the binary regular tree of height $h = 5$ depicted in Figure 2.11 corresponds to a partition into sets of bounded cardinality with $r = 4$).

We can now use the PPSBC in order to bound the partial sums $s_i(\phi)$, $1 \leq i \leq h$, in a similar way as done for binary regular trees in Section 2.4. Since $s_i(\phi)$ is the number of edges of G which contribute by at least $2i$ to the value of the objective function, each such edge joins vertices of G which are arranged at the leaves of *different d-regular subtrees* of T of order $h-(i-1)$ rooted at vertices v with level$(v) = h-(i-1) = h_G-i+2$. Each such subtree has d^{i-1} leaves. Thus we look for a not necessarily balanced partition into sets of bounded cardinality $\mathscr{U}^{(i)} = \{V_j^{(i)} | 1 \leq j \leq k^{(i)}\}$ with $|V_j^{(i)}| \leq d^{i-1}$, $\left\lceil \frac{n}{d^{i-1}} \right\rceil \leq k^{(i)} \leq n$, which minimizes the objective function value $c^{\mathscr{U}}(G, \mathscr{U}^{(i)})$ for every $1 \leq i \leq h$. Then

$$s_i(\phi) \geq c^{\mathscr{U}}(G, \mathscr{U}^{(i)}) \tag{2.104}$$

is valid for all arrangements ϕ and for every $1 \leq i \leq h$. We do not know the optimal objective function value of the PPSBC in general, but we can use the equation (2.103) and consider the best (hypothetical) case for every $1 \leq i \leq h$. In particular, for every $1 \leq i \leq h$ we would like to have a partition into sets of bounded cardinality $\mathscr{U}^{(i)}$ consisting of the smallest possible number of partition sets each inducing only one connected component. I.e. we would like to have $k^{(i)} = \left\lceil \frac{n}{d^{i-1}} \right\rceil$ and $c^{\mathscr{U}}(G, \mathscr{U}^{(i)}) = k^{(i)} - 1 = \left\lceil \frac{n}{d^{i-1}} \right\rceil - 1$ for every $1 \leq i \leq h$.

Thus we can define

$$s_i^L := \left\lceil \frac{n}{d^{i-1}} \right\rceil - 1 \tag{2.105}$$

for every $1 \leq i \leq h$. However, for $i = h$, we get a better bound by using Observation 2.18; thus we set

$$s_h^L := 1. \tag{2.106}$$

The general lower bound is now given by

$$L := 2 \sum_{i=1}^{h} s_i^L = 2 \left(n - 1 + \sum_{i=2}^{h-1} \left(\left\lceil \frac{n}{d^{i-1}} \right\rceil - 1 \right) + 1 \right) \tag{2.107}$$

and can be approximated as follows

$$L \geq 2 \left(n + \sum_{i=2}^{h-1} \left(\frac{n}{d^{i-1}} - 1 \right) \right) \tag{2.108}$$

$$= 2 \left(n - (h-2) + n \sum_{i=1}^{h-2} \left(\frac{1}{d} \right)^i \right) \tag{2.109}$$

$$= 2 \left(n - (h-2) + n \frac{1 - d^{h-2}}{d^{h-2} - d^{h-1}} \right) \tag{2.110}$$

$$= 2 \left(\frac{d^{h_G+1} - 1}{d-1} + \frac{d^{h_G+1} - 1}{(d-1)^2} \left(1 - d \left(\frac{1}{d} \right)^{h_G} \right) - (h_G - 1) \right) \tag{2.111}$$

Notice that in the last equation we substituted $n = \frac{d^{h_G+1}-1}{d-1}$ and $h = h_G + 1$.

Let us now consider a guest graph $G = (V, E)$ of height $h_G = 1$. The lower bound is tight in this case as illustrated in Table 2.7.

i	2	1
$s_i(\phi_B)$	1	3
s_i^L	1	3

Table 2.7: Partial sums $s_i(\phi_B)$ and the corresponding lower bounds s_i^L, where $1 \leq i \leq h$, for a guest graph $G = (V, E)$ of height $h_G = 1$.

We assume that $h_G \geq 2$ in further computations. Let now $f(x) = 1 - d\left(\frac{1}{d}\right)^x$ be a function of $x \in \mathbb{R}^+$, $x \geq 2$ and let $d \geq 2$ be a constant. Obviously, $f'(x) = -d\left(\frac{1}{d}\right)^x \ln \frac{1}{d} > 0$ and thus the function f is strictly monotonically

increasing. Since $h_G \geq 2$, we get $1 - d\left(\frac{1}{d}\right)^{h_G} \geq 1 - d\left(\frac{1}{d}\right)^2 = \frac{d-1}{d}$ and as $\frac{d^{h_G+1}-1}{(d-1)^2} > 0$, L can be bounded as follows:

$$\begin{aligned} L &\geq 2\left(\frac{d^{h_G+1}-1}{d-1} + \frac{d^{h_G+1}-1}{(d-1)^2}\left(1 - d\left(\frac{1}{d}\right)^2\right) - (h_G - 1)\right) \\ &\geq 2\left(\frac{d+1}{d-1}d^{h_G} - \frac{d+1}{d(d-1)} - (h_G - 1)\right). \end{aligned} \tag{2.112}$$

We can now state the main theorem of this section.

Theorem 2.31. *Let the guest graph $G = (V, E)$ and the host graph T be d-regular trees of heights $h_G \geq 1$ and $h = h_G + 1$ respectively for some fixed $d \geq 2$. Algorithm 2.3 is a $\frac{585}{392}$-approximation algorithm.*

Proof. In Corollary 2.29, we have already proved by other techniques that Algorithm 2.3 (i.e. the algorithm $\mathcal{B}$) is a $\frac{203}{200}$-approximation algorithm if G is a binary regular tree. Thus we can skip the case $d = 2$ since $\frac{203}{200} < \frac{585}{392}$ and prove the claim for $d \geq 3$.

The theorem obviously holds for $h_G = 0$. For $h_G = 1$ Algorithm 2.3 yields an optimal solution as demonstrated in Table 2.7 and thus the claim holds as well. Let $h_G \geq 2$ hold; by using Lemma 2.27 and equation (2.112) we obviously get the approximation ratio

$$\rho(d, h_G) := \frac{2\left(\frac{d^4+2d^3-2d+1}{(d-1)^2d(d+1)}d^{h_G} - \frac{d}{d-1}h_G - \frac{2d^2+d-1}{2(d-1)^2} + \frac{1}{2(d+1)}(-1)^{h_G}\right)}{2\left(\frac{d+1}{d-1}d^{h_G} - \frac{d+1}{d(d-1)} - (h_G - 1)\right)}. \tag{2.113}$$

Both the numerator and the denominator are positive as $OV(G, d, \phi_B) > 0$ and $L > 0$ (for $h_G \geq 2$). Therefore

$$\begin{aligned} \rho(d, h_G) &\leq \frac{\frac{d^4+2d^3-2d+1}{(d-1)^2d(d+1)}d^{h_G} - \frac{d}{d-1}h_G - \frac{2d^2+d-1}{2(d-1)^2} + \frac{1}{2(d+1)}}{\frac{d+1}{d-1}d^{h_G} - \frac{d+1}{d(d-1)} - (h_G - 1)} \\ &= \frac{\frac{d^4+2d^3-2d+1}{(d-1)^2d(d+1)}d^{h_G} + f(d, h_G)}{\frac{d+1}{d-1}d^{h_G} - \frac{d+1}{d(d-1)} - (h_G - 1)}, \end{aligned} \tag{2.114}$$

where

$$f(d, h_G) := -\frac{d}{d-1}h_G - \frac{2d^2+d-1}{2(d-1)^2} + \frac{1}{2(d+1)}. \tag{2.115}$$

Consider now that $f\colon [2,+\infty[\times\mathbb{R}\to\mathbb{R}$ is a function with $d, h_G \in \mathbb{R}$ and let $d \geq 2$ hold. Let $d \in [2,+\infty[$ be fixed. $f(d,h_G)$ is a strictly monotonically decreasing linear function. For $h_G = 0$ we have

$$\begin{aligned} f(d,0) &= -\frac{2d^2+d-1}{2(d-1)^2} + \frac{1}{2(d+1)} \\ &= \frac{-d^3-d^2-d+1}{(d-1)^2(d+1)}. \end{aligned} \tag{2.116}$$

Now, we show that $f(d,0) < 0$, for $d \geq 2$, which implies that $f(d,h_G) < 0$ for all $h_G \geq 0$. Since $d \geq 2$, $(d-1)^2(d+1) > 0$ holds and thus the denominator does not influence the sign of $f(d,0)$. Let

$$g(d) \coloneqq -d^3-d^2-d+1. \tag{2.117}$$

Then

$$g'(d) = -3d^2-2d-1 < 0, \tag{2.118}$$

and thus the function $g(d)$ is a strictly monotonically decreasing cubic function. $g(2) = -13$ and therefore $g(d) < 0$ for all $d \geq 2$. This implies $f(d,0) < 0$, which again implies $f(d,h_G) < 0$ for any $h_G \geq 2$ (recall that $f(d,h_G)$ is monotonically decreasing for every fixed $d \in [2,+\infty[$). Consequently,

$$\rho(d,h_G) \leq \rho^U(d,h_G) \coloneqq \frac{\frac{d^4+2d^3-2d+1}{(d-1)^2d(d+1)}d^{h_G}}{\frac{d+1}{d-1}d^{h_G} - \frac{d+1}{d(d-1)} - (h_G-1)}. \tag{2.119}$$

We show that $\rho^U(d,h_G)$ is a strictly monotonically decreasing sequence for every fixed $d \geq 2$. The denominator is positive since it corresponds to the lower bound $L > 0$. Define a function $r\colon [2,+\infty[\to\mathbb{R}$ such that

$$r(d) \coloneqq \frac{d^4+2d^3-2d+1}{(d-1)^2d(d+1)}d^{h_G}. \tag{2.120}$$

Since $(d-1)^2d(d+1) > 0$ for every $d \geq 2$, the denominator does not influence the sign of $r(d)$. Let $o\colon [2,+\infty[\to\mathbb{R}$ be defined as

$$o(d) \coloneqq d^4+2d^3-2d+1. \tag{2.121}$$

Then

$$o'(d) = 4d^3+6d^2-2 = 4\left(d-\frac{1}{2}\right)(d+1)^2 > 0 \text{ for } d \geq 2. \tag{2.122}$$

Consequently, $o(d)$ is a strictly monotonically increasing function for $d \geq 2$ and $o(2) = 29$ implies that $d^4 + 2d^3 - 2d + 1 > 0$ for all $d \geq 2$. Thus $r(d) > 0$ for all $d \geq 2$.

Next, let us define

$$s(d) \coloneqq \frac{d+1}{d-1} > 0, \tag{2.123}$$

$$t(d) \coloneqq \frac{d+1}{d(d-1)} > 0 \text{ and} \tag{2.124}$$

$$L(d, h_G) \coloneqq \frac{d+1}{d-1} d^{h_G} - \frac{d+1}{d(d-1)} - (h_G - 1) > 0 \tag{2.125}$$

for some $d \geq 2$ and $h_G \geq 2$. Consider the equivalences

$$\begin{aligned}
\rho^U(d, h_G) > \rho^U(d, h_G + 1) &\Leftrightarrow \frac{r(d) d^{h_G}}{L(d, h_G)} > \frac{r(d) d^{h_G+1}}{L(d, h_G + 1)} \\
&\Leftrightarrow \\
L(d, h_G + 1) &> d \cdot L(d, h_G) \\
&\Leftrightarrow \\
s(d) \cdot d^{h_G+1} - t(d) - (h_G + 1 - 1) &> d\big((s(d) \cdot d^{h_G} - t(d) - (h_G - 1)\big) \\
&\Leftrightarrow \\
-t(d) - h_G &> -d \cdot t(d) - d \cdot h_G + d \\
&\Leftrightarrow \\
h_G(d-1) &> d - \frac{d+1}{d(d-1)} d + \frac{d+1}{d(d-1)} \\
&\Leftrightarrow \\
h_G > \frac{d^3 - 2d^2 + 1}{d(d-1)^2} &\Leftrightarrow h_G > \frac{d^2 - d - 1}{d(d-1)}.
\end{aligned} \tag{2.126}$$

Let now $p\colon [2, +\infty[\to \mathbb{R}$ be defined by

$$p(d) \coloneqq \frac{d^2 - d - 1}{d(d-1)}. \tag{2.127}$$

Obviously,

$$p'(d) = \frac{2d - 1}{d^2(d-1)^2} > 0 \text{ for all } d \geq 2. \tag{2.128}$$

The function $p(d)$ is monotonically increasing for $d \geq 2$. Consequently, $h_G > p(d)$ holds for all $d \geq 2$, because

$$h_G \geq 2 > \lim_{d \to +\infty} p(d) = \lim_{d \to +\infty} \frac{d^2 - d - 1}{d(d-1)} = 1. \tag{2.129}$$

Thus the equivalent inequalities (2.126) hold for every $h_G \geq 2$ and $\rho^U(d, h_G)$ is a monotonically decreasing sequence for every fixed $d \geq 2$. We can also define $\rho\colon [2, +\infty[\to \mathbb{R}$ by

$$\rho(d) := \rho^U(d, 2) = \frac{d^6 + 2d^5 - 2d^3 + d^2}{d^6 + d^5 - 2d^4 - d^3 + 1} \tag{2.130}$$

and obtain

$$\rho(d, h_G) \leq \rho^U(d, h_G) \leq \rho^U(d, 2) = \rho(d) \tag{2.131}$$

by considering (2.119). Observe that

$$\rho'(d) = -\frac{d^{10} + 4d^9 + d^8 + 4d^7 + 7d^6 - 10d^5 - 11d^4 + 6d^2 - 2d}{(d^6 + d^5 - 2d^4 - d^3 + 1)^2} < 0, \tag{2.132}$$

because

$$\begin{aligned} &d^{10} + 4d^9 + d^8 + 4d^7 + 7d^6 - 10d^5 - 11d^4 + 6d^2 - 2d \\ &\quad = (d^{10} + 4d^9 + d^8 + 4d^7) + d^4(7d^2 - 10d - 11) + 2d(3d - 1) \\ &\quad > 0 \end{aligned} \tag{2.133}$$

since $d^{10} + 4d^9 + d^8 + 4d^7 > 0$, $d^4 > 0$, $7d^2 - 10d - 11 > 0$, $2d > 0$ and $3d - 1 > 0$ for $d \geq 3$, where we have to verify that $7d^2 - 10d - 11 > 0$ for $d \geq 3$ by solving a quadratic inequality. Consequently, $\rho(d)$, $d \geq 3$, is a monotonically decreasing sequence.

By applying (2.131) we finally obtain

$$\rho(d, h_G) \leq \rho(d) \leq \rho(3) = \frac{585}{392} =: \rho \text{ for } d \geq 3. \tag{2.134}$$

Recall that Algorithm 2.3 is a $\frac{203}{200}$-approximation algorithm according to Corollary 2.29. Thus we get

$$\rho = \frac{585}{392} \approx 1.492 \tag{2.135}$$

as the overall approximation ratio for all $h_G \in \mathbb{N}^0$ and all $d \geq 2$. □

2.6 The complexity of the DAPT with a tree as a guest graph

In this section we show that the DAPT where the guest graph G is a tree on n vertices and the host graph T is a complete d-regular tree of height

$\lceil \log_d n \rceil$, for some fixed $d \in \mathbb{N}$, $d \geq 2$, is $\mathcal{NP}$-hard. This result also settles a more general open question posed by Luczak and Noble [35] about the complexity of the GEP when both input graphs are trees.

First let us state a simple result on the optimal value of the objective function of the $DAPT(G, d)$ in the case where G is a start graph; this result was proven in Çela and Staněk [10].

Lemma 2.32. *Let $G = (V, E)$ be a star graph (i.e. a complete bipartite graph with 1 vertex in one side of the partition and the rest of the vertices in the other side) with n vertices and the central vertex v_1. Let the host graph T be a complete d-regular tree of height $h = \lceil \log_d n \rceil$, with $d \in \mathbb{N}$, $2 \leq d \leq n$. Then the optimal value of the objective function $OPT(G, d)$ is given by*

$$OPT(G, d) = 2 \left(h\, n - \frac{d^h - 1}{d - 1} \right). \tag{2.136}$$

Moreover, an arrangement is optimal if and only if it arranges the central vertex v_1 together with other $d^{h-1} - 1$ arbitrarily selected vertices of G at the leaves of some (arbitrarily selected) basic subtree of T (and the other vertices arbitrarily).

Proof. See Çela and Staněk [10]. □

Next we will consider another very special case where the guest graph G is the disjoint union of three star graphs.

Lemma 2.33. *Let the guest graph G be the disjoint union of three star graphs S_i, $1 \leq i \leq 3$, i.e. $V(G) = \dot{\bigcup}_{i=1}^{3} V(S_i)$ and $E(G) = \dot{\bigcup}_{i=1}^{3} E(S_i)$. Assume that $|V(S_i)| =: n_i$, $1 \leq i \leq 3$, and $n_1 \geq n_2 \geq n_3$, where $V(S_i)$ is the vertex set of S_i, $1 \leq i \leq 3$. Assume that $n := \left|V(G)\right| = n_1 + n_2 + n_3$ is a power of d for some fixed $d \in \mathbb{N}$, $d \geq 2$, and $n_1 \geq \frac{n}{d}$. Let the host graph T be a complete d-regular tree of height $h = \log_d n$. Then the optimal value of the objective function $OPT(G, d)$ is given by*

$$\begin{aligned} OPT(G, d) =& 2 \left(h_1 n_1 - \frac{d^{h_1} - 1}{d - 1} \right) + 2 \left(h_2 n_2 - \frac{d^{h_2} - 1}{d - 1} \right) + \\ & 2 \left(h_3 n_3 - \frac{d^{h_3} - 1}{d - 1} \right), \end{aligned} \tag{2.137}$$

where $h_i = \lceil \log_d n_i \rceil$, $i \in \{1, 2, 3\}$.

Proof. Let $B = \{b_1, b_2, \ldots, b_n\}$ be the set of the leaves of T labeled according to the canonical order. Arrange the central vertex of S_1 together with other $2^{h-1} - 1$ vertices of S_1 at the leaves $b_1, b_2, \ldots, b_{d^{h-1}}$ of the leftmost basic subtree of T. The other vertices of S_1 will be arranged later at other appropriately chosen leaves of T. Notice, however, that independently on the arrangement of these vertices the contribution of the edges of S_1 to the objective function value of $DAPT(G, d)$ will be equal to $2\left(h\, n_1 - \frac{d^h-1}{d-1}\right)$, according to Lemma 2.32.

Arrange the vertices of S_2 to the n_2 leaves $b_{n-n_2+1}, \ldots, b_n$ of T with the largest indices. According to Lemma 2.32 the edges of S_2 will then contribute by $2\left(h_2\, n_2 - \frac{d^{h_2}-1}{d-1}\right)$ to the objective function value of $DAPT(G, d)$. Next, arrange the vertices of S_3 to the n_3 leaves $b_{d^{h-1}+1}, \ldots, b_{n_1+n_3}$ of T (which are still free because $n_1 + n_2 + n_3 = d^h$ and only the leaves with the $d^{h-1} \leq n_1$ smallest indices as well as the leaves with the n_3 largest indices have been occupied already). According to Lemma 2.32 the edges of S_2 will then contribute by $2\left(h_3\, n_3 - \frac{d^{h_3}-1}{d-1}\right)$ to the objective function value of $DAPT(G, d)$. Finally arrange the $n_1 - d^{h-1}$ vertices of S_1 not arranged yet to the remaining $d^h - n_2 - n_3 - d^{h-1} = n_1 - d^{h-1}$ leaves. Summarizing, this arrangement yields an objective function value equal to the expression in (2.137), and is therefore optimal because the following inequality

$$\begin{aligned} OV(G, d, \phi) = \sum_{(u,v)\in E(G)} d_T\big(\phi(u), \phi(v)\big) = \sum_{i=1}^{3} \sum_{(u,v)\in E(S_i)} d_T\big(\phi(u), \phi(v)\big) \geq \\ 2\left(h_1 n_1 - \frac{d^{h_1}-1}{d-1}\right) + 2\left(h_2 n_2 - \frac{d^{h_2}-1}{d-1}\right) + \\ 2\left(h_3 n_3 - \frac{d^{h_3}-1}{d-1}\right) \end{aligned} \tag{2.138}$$

holds for any arrangement ϕ of $DAPT(G, d)$ due to Lemmma 2.32. □

Next we state the main result of this section.

Theorem 2.34. *The DAPT with a host graph T being a complete d-regular tree is $\mathcal{NP}$-hard for every fixed $d \geq 2$ even if the guest graph G is a tree.*

Proof. The problem obviously belongs to $\mathcal{NP}$. The $\mathcal{NP}$-hardness is proven by means of a reduction from the *numerical matching with target sums* (*NMTS*) problem.

The NMTS is $\mathcal{NP}$-hard and is defined as follows (see GAREY and JOHNSON [22]): Let three sets $X = \{x_1, x_2, \ldots, x_n\}$, $Y = \{y_1, y_2, \ldots, y_n\}$ and $Z = \{z_1, z_2, \ldots, z_n\}$ of positive integers, with $\sum_{i=1}^{n} z_i = \sum_{i=1}^{n} x_i + \sum_{i=1}^{n} y_i$ with $n \geq 2$ be given. The goal is to decide whether there exist two permutations $(j_1, j_2, \ldots, j_n)$ and $(k_1, k_2, \ldots, k_n)$ of the indices $\{1, 2, \ldots, n\}$, such that $z_i = x_{j_i} + y_{k_i}$ for all $i = 1, 2, \ldots, n$ hold.

Consider an instance of the NMTS and a given integer $d \geq 2$. We construct an instance $DAPT(G, d)$ of the DAPT as follows. Let $l_y \in \mathbb{N}$ be the smallest natural number such that $y_i \leq d^{l_y - 4}$, for all $1 \leq i \leq n$. Let $l_x \in \mathbb{N}$ be the smallest natural number such that $x_i + y_j + (d-1)d^{l_y - 4} < d^{l_x - 2}$, for all $1 \leq i, j \leq n$, Finally, let l_z be the smallest natural number such that $z_i \leq d^{l_z} - (d-1)d^{l_z - 4} - (d-1)d^{l_z - 2}$, for all $1 \leq i \leq n$. Let then $l \coloneqq \max\{l_x, l_y, l_z\}$, and let $L \in \mathbb{N}$ be the smallest natural number such that $nd^l \leq d^{L-1}$. Define three vertex disjoint star graphs S_i^x, S_i^y and S_i^z, for every $1 \leq i \leq n$, with $|V(S_i^x)| = (d-1)d^{l-2} + x_i$, $|V(S_i^y)| = (d-1)d^{l-4} + y_i$ and $|V(S_i^z)| = d^l - (d-1)d^{l-4} - (d-1)d^{l-2} - z_i$, for $i = 1, 2, \ldots, n$. Notice that $\sum_{i=1}^{n}(|V(S_i^x)| + |V(S_i^y)| + |V(S_i^z)| = nd^l$.

Let $v_1(S_i^x)$, $v_1(S_i^y)$ and $v_1(S_i^z)$ be the central vertices of the stars graphs introduced above for $1 \leq i \leq n$, respectively. Next introduce the vertices $u_1, u_2, \ldots, u_{\widehat{n}}$, where $\widehat{n} = d^{L-1} - 1$ and one vertex v_1. Finally consider a family of stars S_i', $1 \leq i \leq n'$, with d^l vertices each and $n' \coloneqq (d-1)d^{L-1-l} - n \geq 0$. Denote by $v_1(s_i')$, $1 \leq i \leq n'$, their central vertices respectively.

The guest graph $G = (V, E)$ is defined in the following way. The vertex set is given by all vertices defined above, and the edge set contains all edges contained in the stars S_i^x, S_i^y and S_i^z, $1 \leq i \leq n$, and S_i', $1 \leq i \leq n'$, together with edges connecting the vertex v_1 with the central vertices $v_1(S_i^x)$, $v_1(S_i^y)$, $v_1(S_i^z)$, $1 \leq i \leq n$, and $v_1(S_i')$, $1 \leq i \leq n'$, and with all vertices u_1, u_2, $\ldots$, $u_{\widehat{n}}$. Thus we have

$$\begin{aligned} V = &\Big[\bigcup_{i=1}^{n}[V(S_i^x) \cup V(S_i^y) \cup V(S_i^z)]\Big] \bigcup \\ &\Big[\bigcup_{i=1}^{n'} V(S_i')\Big] \bigcup \{u_1, u_2, \ldots, u_{\widehat{n}}, v_1\} \end{aligned} \tag{2.139}$$

and

$$E = E_1 \cup E_2 \cup E_3, \tag{2.140}$$

where

$$E_1 = \Big\{(v_1, u_i) \colon 1 \leq i \leq \widehat{n}\Big\}, \tag{2.141}$$

$$E_2 = \Big[\bigcup_{i=1}^{n'} E(S'_i)\Big] \bigcup \Big\{\big(v_1, v_1(S'_i)\big) \colon 1 \leq i \leq n'\Big\} \tag{2.142}$$

and

$$\begin{aligned} E_3 = \bigcup_{i=1}^{n} \Big[& E(S^x_i) \cup E(S^y_i) \cup E(S^z_i) \cup \\ & \big\{(v_1, v_1(S^x_i)), (v_1, v_1(S^y_i)), (v_1, v_1(S^z_i))\big\}\Big]. \end{aligned} \tag{2.143}$$

The number of vertices in G is given as $|V(G)| = \sum_{i=1}^{n}(|V(S^x_i)| + |V(S^y_i)| + |V(S^z_i)| + \widehat{n} + n'd^l + 1 = d^L$. Thus the host graph is a d-regular tree T of height $h =$
$\lceil \log_d |V(G)| \rceil = L$.

We show that the optimal value $OPT(G, d)$ of the objective function of $DAPT(G, d)$ equals the expression in (2.145) if and only if the corresponding NMTS instance is a YES-instance, and this would complete the $\mathcal{NP}$-hardness proof.

Prior to showing the above if and only if statement, consider the optimal arrangement of the substar G' induced in G by v_1 and its neighbors, i.e. by the following set of vertices

$$\{v_1\} \cup \{u_i \colon 1 \leq i \leq \widehat{n}\} \cup \{v_1(S'_i) \colon 1 \leq i \leq n'\} \cup \big[\cup_{i=1}^{n} \{v_1(S^x_i), v_1(S^y_i), v_1(S^z_i)\}\big].$$

Assume w.l.o.g. that the vertex v_1 is arranged at the left most leaf of T (i.e. the first leaf b_1 of T in the canonical ordering). The vertex v_1 has $3n + \widehat{n} + n' = d^{L-1} - 1 + 3n + n' > d^{L-1} - 1$ neighbors (note that $n' \geq 0$) and hence $|V(G')| = 3n + \widehat{n} + n' + 1 = d^{L-1} + 3n + n'$. According to Lemma 2.32, any arrangement of G' which arranges the $d^{L-1} - 1$ neighbors $\{u_1, u_2, \ldots, u_{\widehat{n}}\}$ of v_1 at the leaves of the leftmost basic subtree of T and the remaining $3n + n'$ neighbors at some other (arbitrarily selected) leaves of T is optimal. In particular such an arrangement would not arrange $v_1(S^x_i)$, $v_1(S^y_i)$, $v_1(S^z_i)$, $i = 1, 2, \ldots, n$, and $v_1(S'_i)$, $1 \leq i \leq n'$, at the leaves of the leftmost basic subtree.

Let us now proof the if and only if statement formulated above.

The "if" statement. Assume that the NMTS instance is a YES-instance. We show that the equality (2.145) holds. Consider two permutations $(j_1, j_2, \ldots, j_n)$ and $(k_1, k_2, \ldots, k_n)$ of the indices $\{1, 2, \ldots, n\}$, such that $z_i = x_{j_i} + y_{k_i}$, for all $i = 1, 2, \ldots, n$. Then for all $i \in \{1, 2, \ldots, n\}$ we

have

$$\begin{aligned}&|V(S^x_{j_i})| + |V(S^y_{k_i})| + |V(S^z_i)| = \\ &(d-1)d^{l-2} + x_{j_i} + (d-1)d^{l-4} + y_{k_i} + \\ &d^l - (d-1)d^{l-4} - (d-1)d^{l-2} - z_i = \\ &d^l.\end{aligned} \tag{2.144}$$

Thus, for every $i \in \{1, 2, \ldots, n\}$ the disjoint stars graphs $S^x_{j_i}$, $S^y_{k_i}$ and S^z_i can be optimally arranged at the leaves of the i-th rightmost subtree of l-th order according to Lemma 2.33. Notice that the assumptions of the lemma are fulfilled because $|V(S^z_i)| > (d-1)d^{l-1} \geq d^{l-1}$. (Indeed, due to $y_{k_i} \leq d^{l-4}$ and $x_{j_i} + y_{k_i} + (d-1)d^{y-4} < d^{l-2}$ we get $|V(S^y_{k_i})| + |V(S^x_{j_i})| = (d-1)d^{l-4} + y_{k_i} + (d-1)d^{l-2} + x_{j_i} < d^{l-1}$ which together with (2.144) implies $|V(S^z_i)| > (d-1)d^{l-1} \geq d^{l-1}$.) Since $nd^l \leq d^{L-1}$ the n rightmost subtrees of the l-th order are subtrees of the rightmost basic subtree (of height $L-1$).

It remains to arrange the vertices $v_1, u_1, u_2, \ldots, u_{\widehat{n}}$ and the stars S'_i, $1 \leq i \leq n'$. This is done by arranging $v_1, u_1, u_2, \ldots, u_{\widehat{n}}$ at the leaves of the leftmost basic subtree and the stars S'_i, $1 \leq i \leq n'$ in one of the n' still free subtrees of l-th order each. These arrangements are cleary optimal for each of the stars S'_i, $1 \leq i \leq n'$ (recall that they have d^l vertices each). Moreover, as mentioned above this is also an optimal arrangement of G'. Thus this arrangement arranges optimally the following subgraphs of G: G', S'_i, $1 \leq i \leq n$, and the disjoint unions of the triples $(S^x_{j_i}, S^y_{k_i}, S^z_i)$, for $1 \leq i \leq n$, respectively. Since the edge sets of the above mentioned graphs yield a partition of the edge set $E(G)$, the arrangement described above is an optimal arrangement of G. The corresponding value $OPT(G,d)$ of the objective function is given as the sum of $OPT(G',d)$, $OPT(S'_i,d)$, $1 \leq i \leq n'$, and $OPT(S^x_{j_i} \cup S^y_{k_i} \cup S^z_i, d)$, for $1 \leq i \leq n$. According to Lemma 2.32 and Lemma 2.33 we get

$$\begin{aligned}OPT(G,d) =& 2\Big(Ln'' - \frac{d^L - 1}{d-1}\Big) + 2n'\Big(ld^l - \frac{d^l - 1}{d-1}\Big) + \\ & 2\sum_{i=1}^{n}\Big[l|V(S^z_i)| - \frac{d^{l-1}}{d-1} + (l-1)|V(S^x_{j_i})| - \frac{d^{l-1}-1}{d-1} + \\ & (l-3)|V(S^y_{k_i})| - \frac{d^{l-3}-1}{d-1}\Big],\end{aligned} \tag{2.145}$$

where $n'' := |V(G')| = \widehat{n} + n' + d^{L-1}$.

The "only if" statement. Assume that $OPT(G,d)$ is given as in (2.145) and consider some optimal arrangement ϕ of G. We show that the corresponding NMTS instance is a YES-instance. Notice that (2.145) implies

$$\begin{aligned} OPT(G,d) =& OPT(G',d) + \sum_{i=1}^{n'} OPT(S'_i,d) + \\ & \sum_{i=1}^{n} \big(OPT(S_i^x,d) + OPT(S_i^y,d) + OPT(S_i^z,d) \big). \end{aligned} \tag{2.146}$$

Thus, in particular, ϕ yields optimal arrangements of G', S'_i, $1 \leq i \leq n'$, and S_i^x, S_i^y and S_i^z, $1 \leq i \leq n$. Assume w.l.o.g. that ϕ arranges v_1 at the leftmost leaf of T. Then according to Lemma 2.32 ϕ arranges neighbors of v_1, i.e. vertices of G', to each leaf of the leftmost basic subtree of T. Thus, none of the neighbors of v_1 arranged at the leaves of the leftmost basic subtree of T can be the central vertex of some of the stars S'_i, $1 \leq i \leq n'$, or S_i^x, S_i^y and S_i^z, $1 \leq i \leq n$, because then according to Lemma 2.32 ϕ would not lead to an optimal arrangement of the corresponding star. It follows that the vertices arranged at the leaves of the leftmost basic subtree of T are u_i, $1 \leq i \leq \widehat{n}$.

According to Lemma 2.32 ϕ arranges each star S'_i, $1 \leq i \leq n'$, to the leaves of some subtree of l-th order. Hence there remain $(d-1)d^{L-1-l} - n' = n$ subtrees of l-th order at the leaves of which ϕ the stars S^{x_i}, S^{y_i} and S^{z_i}, $1 \leq i \leq n$.

Recall now that $(d-1)d^{l-1} < |V(S_i^z)|$, $(d-1)d^{l-2} < |V(S_i^x)| < d^{l-1}$ and $(d-1)d^{l-4} < |V(S_i^y)| < d^{l-3}$ for $1 \leq i \leq n$. Thus in each of the n remaining free subtrees of l-th order can not be arranged more than one of the stars S_i^z, $1 \leq i \leq n$, and since there are n such stars to be arranged in n subtrees of l-th order, exactly one of them will be arranged in each subtree. By analogous arguments we get that exactly one of the stars S_i^x will be arranged in each of the subtrees of l-th order mentioned above, and finally, exactly one of the stars S_i^y will be arranged in each of these subtrees. Thus in each of the n-th subtrees of l-th level exactly one star S_i^z, one star S_i^x and one star S_i^y will be arranged. For all $i \in \{1,2,\ldots,n\}$ denote by j_i and k_i the indices of the stars arranged together with S_i^z in the same subtree, i.e. $S_{j_i}^x$, $S_{k_j}^y$ and S_i^z are arranged in the same subtree of l-th order, $1 \leq i \leq n$. Clearly, $(j_1, j_2, \ldots, j_n)$ and $(k_1, k_2, \ldots, k_n)$ are permutations of $\{1,2,\ldots,n\}$, respectively. Since the stars S_i^x, S_i^y and S_i^z, $1 \leq i \leq n$,

have nd^l vertices altogether, which is the number of leaves of the n subtrees of l-th level, the equality

$$|V(S^x_{j_i})| + |V(S^y_{k_i})| + |V(S^z_i)| = d^l \tag{2.147}$$

must hold, for all $1 \leq i \leq n$. By substituting the cardinalities of the vertex sets of the stars in (2.147) we get $x_{j_i} + y_{k_i} - z_i = 0$, for all $1 \leq i \leq n$, and this completes the proof.

□

2.A Appendix

2.A.1 Arrangements ϕ_A obtained from Algorithms 2.1 and 2.3 for different heights h_G of the guest graph G

Binary trees

Figure 2.17: Guest graph $G = (V, E)$ (binary regular tree of height $h_G = 0$). The colors are related to the arrangement ϕ depicted in Figure 2.18.

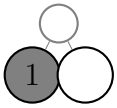

Figure 2.18: Arrangement ϕ_A obtained from Algorithm 2.1 for the guest graph $G = (V, E)$ depicted in Figure 2.17. Its objective function value is $OV(G, 2, \phi_A) = 0$.

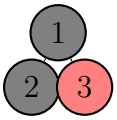

Figure 2.19: Guest graph $G = (V, E)$ (binary regular tree of height $h_G = 1$). The colors are related to the arrangement ϕ depicted in Figure 2.20.

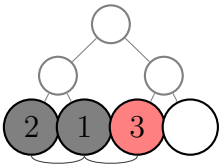

Figure 2.20: Arrangement ϕ_A obtained from Algorithm 2.1 for the guest graph $G = (V, E)$ depicted in Figure 2.19. Its objective function value is $OV(G, 2, \phi_A) = 6$.

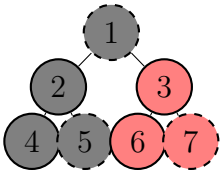

Figure 2.21: Guest graph $G = (V, E)$ (binary regular tree of height $h_G = 2$). The colors are related to the arrangement ϕ depicted in Figure 2.22.

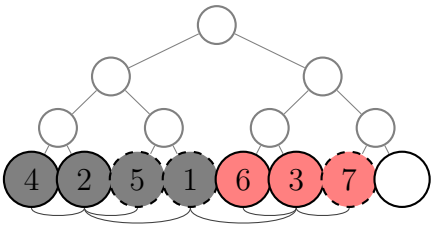

Figure 2.22: Arrangement ϕ_B obtained from Algorithm 2.1 for the guest graph $G = (V, E)$ depicted in Figure 2.21. Its objective function value is $OV(G, 2, \phi_B) = 22$.

3-regular trees

Figure 2.23: Guest graph $G = (V, E)$ (3-regular tree of height $h_G = 0$). The colors are related to the arrangement ϕ depicted in Figure 2.24.

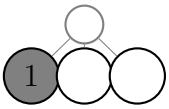

Figure 2.24: Arrangement ϕ_B obtained from Algorithm 2.3 for the guest graph $G = (V, E)$ depicted in Figure 2.23. Its objective function value is $OV(G, 3, \phi_B) = 0$.

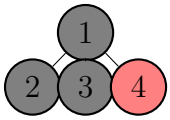

Figure 2.25: Guest graph $G = (V, E)$ (3-regular tree of height $h_G = 1$). The colors are related to the arrangement ϕ depicted in Figure 2.26.

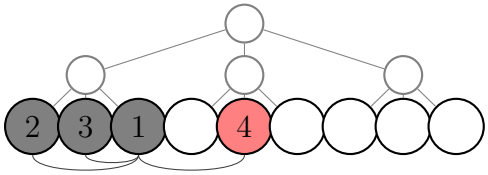

Figure 2.26: Arrangement ϕ_B obtained from Algorithm 2.3 for the guest graph $G = (V, E)$ depicted in Figure 2.25. Its objective function value is $OV(G, 3, \phi_B) = 8$.

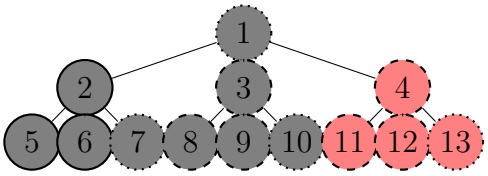

Figure 2.27: Guest graph $G = (V, E)$ (3-regular tree of height $h_G = 2$). The colors are related to the arrangement ϕ depicted in Figure 2.28.

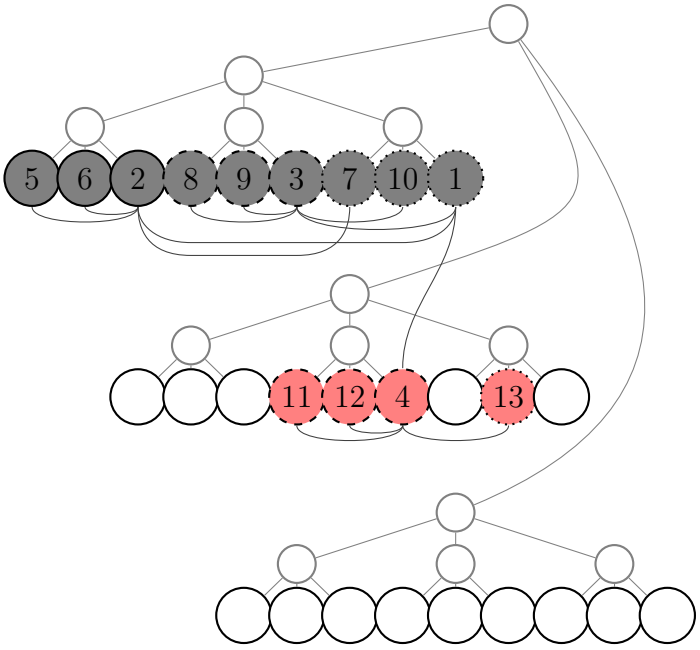

Figure 2.28: Arrangement ϕ_B obtained from Algorithm 2.3 for the guest graph $G = (V, E)$ depicted in Figure 2.27. Its objective function value is $OV(G, 3, \phi_B) = 38$.

3. Generating subtour elimination constraints for the traveling salesman problem from pure integer solutions[†]

The **traveling salesman/salesperson problem** (**TSP**) is one of the best known and most widely investigated combinatorial optimization problems with four famous books entirely devoted to its study (Lawler et al. [34], Reinelt [46], Gutin and Punnen [25], Applegate et al. [6]). Thus, we will refrain from giving extensive references but mainly refer to the treatment in Applegate [6]. Given a complete graph $G = (V, E)$ with $|V| = n$ and $|E| = m = n(n-1)/2$, and nonnegative distances d_e for each $e \in E$, the TSP asks for a shortest tour with respect to the distances d_e containing each vertex exactly once.

Let $\delta(v) := \{e = (v, u) \in E \mid u \in V\}$ denote the set of all edges adjacent to $v \in V$. Introducing binary variables x_e for the possible inclusion of any edge $e \in E$ in the tour we get the following classical ILP formulation:

$$\min \quad \sum_{e \in E} d_e x_e \tag{3.1}$$

$$\text{s.t.} \quad \sum_{e \in \delta(v)} x_e = 2 \qquad \forall v \in V, \tag{3.2}$$

$$\sum_{\substack{e=(u,v) \in E \\ u,v \in S}} x_e \leq |S| - 1 \quad \forall S \subset V,\ S \neq \emptyset, \tag{3.3}$$

$$x_e \in \{0, 1\} \qquad \forall e \in E \tag{3.4}$$

(3.1) defines the *objective function*, (3.2) is the *degree equation* for each vertex, (3.3) are the *subtour elimination constraints*, which forbid solutions con-

[†]Joint work with Ulrich Pferschy [44].

sisting of several disconnected tours, and finally (3.4) defines the *integrality constraints*. Note also that some subtour elimination constraints are redundant: For the vertex sets $S \subset V$, $S \neq \emptyset$, and $S' = V \backslash S$ we get pairs of subtour elimination constraints both enforcing the connection of S and S'.

The established standard approach to solve TSP to optimality, as pursued successfully during the last 30+ years, is a branch-and-cut approach, which solves the LP relaxation obtained by relaxing the integrality constraints (3.4) into $x_e \in [0, 1]$. In each iteration of the underlying branch-and-bound scheme cutting planes are generated, i.e. constraints that are violated by the current fractional solution, but not necessarily by any feasible integer solution. Since there exists an exponential number of subsets $S \subset V$ implying subtour elimination constraints (3.3), the computation starts with a small collection of subsets $S \subset V$ (or none at all), and identifies violated subtour elimination constraints as cutting planes in the so-called separation problem. Moreover, a wide range of other cutting plane families were developed in the literature together with heuristic and exact algorithms to find them (see e.g. SCHRIJVER [49, ch. 58] and APPLEGATE et al. [6]). Also the undisputed champion among all TSP codes, the famous *Concorde* package, is based on this principle (see APPLEGATE et al. [6]).

In this chapter we introduce and examine another concept for solving the TSP. In Section 3.1 we introduce the basic idea of our approach. Some improvement strategies follow in Section 3.2 with our best approach presented in Subsection 3.2.6. Since the main contribution of this chapter are computational experiments, we discuss them in detail in Section 3.3. The common details of all these tests will be given in Subsection 3.3.1. In Section 3.4, we present some theoretical results and further empirical observations. Finally, we provide an Appendix with two summarizing tables (Tables 3.8 and 3.9).

3.1 General solution approach

Clearly, the performance of the above branch-and-cut approach depends crucially on the performance of the used LP solver. Highly efficient LP solvers have been available for quite some time, but also ILP solvers have improved rapidly during the last decades and reached an impressive performance. This motivated the idea of a very simple approach for solving TSP without using LP relaxations explicitly.

The general approach works as follows (see Algorithm 3.1). First, we relax all subtour elimination constraints (3.3) from the model and solve the remaining ILP model (corresponding to a *weighted 2-matching problem*). Then we check if the obtained integer solution contains subtours. If not, the solu-

tion is an optimal TSP tour. Otherwise, we find all subtours in the integral solution (which can be done by a simple scan) and add the corresponding subtour elimination constraints to the model, each of them represented by the subset of vertices in the corresponding subtour. The resulting enlarged ILP model is solved again to optimality. Iterating this process clearly leads to an optimal TSP tour.

Require: TSP instance
Ensure: an optimal TSP tour
1: define current model as (3.1), (3.2), (3.4);
2: **repeat**
3: solve the current model to optimality by an ILP solver;
4: **if** solution contains no subtour **then**
5: set the solution as optimal tour;
6: **else**
7: find all subtours of the solution and add the corresponding subtour elimination constraints into the model;
8: **end if**
9: **until** optimal tour found;

Algorithm 3.1: Main idea of our approach.

Every execution of the ILP solver (see line 3) will be called an *iteration*. We define the *set of violated subtour elimination constraints* as the set of all included subtour elimination constraints which were violated in an iteration (see line 7). Figures 3.1 and 3.2–3.13 illustrate a problem instance and the execution of the algorithm on this instance respectively.

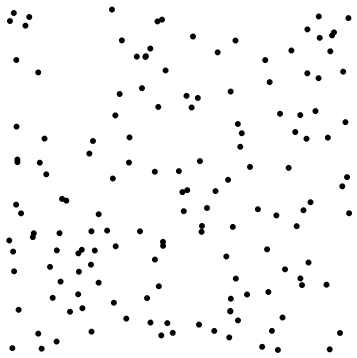

Figure 3.1: Instance *RE_A_150*. Euclidean distances between vertices.

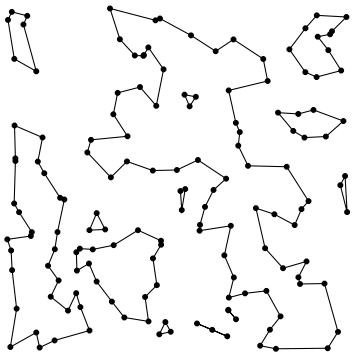

Figure 3.2: Instance *RE_A_150*: Main idea of our approach – iteration 1.

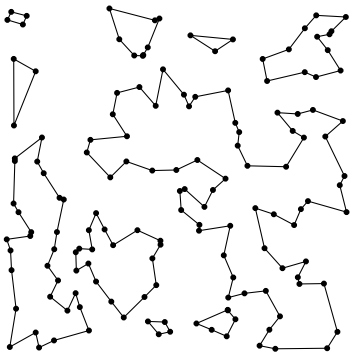

Figure 3.3: *RE_A_150*: iteration 2.

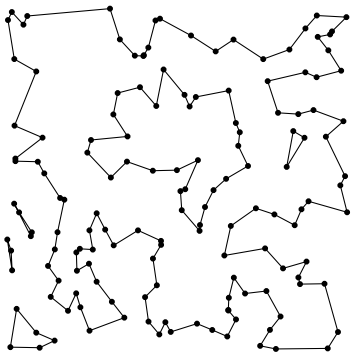

Figure 3.4: *RE_A_150*: iteration 3.

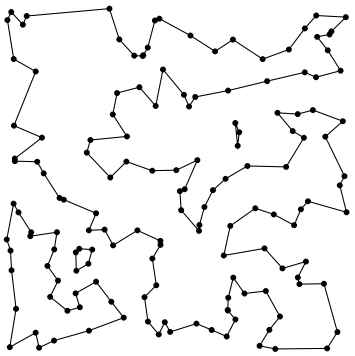

Figure 3.5: *RE_A_150*: iteration 4.

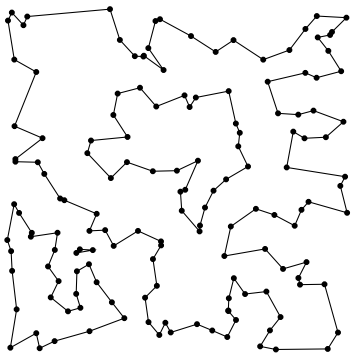

Figure 3.6: *RE_A_150*: iteration 5.

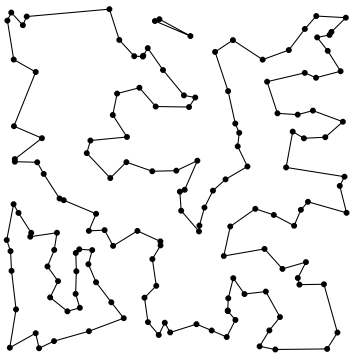

Figure 3.7: *RE_A_150*: iteration 6.

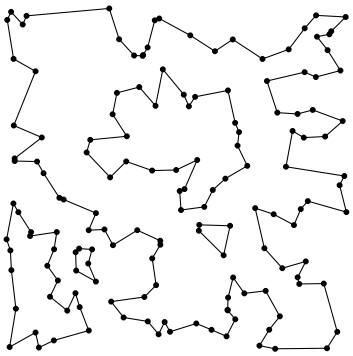

Figure 3.8: *RE_A_150*: iteration 7.

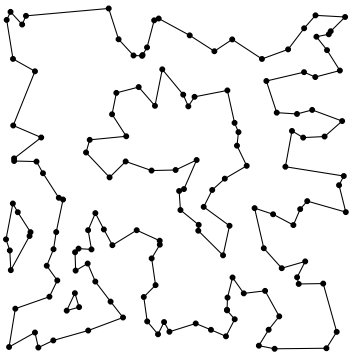

Figure 3.9: *RE_A_150*: iteration 8.

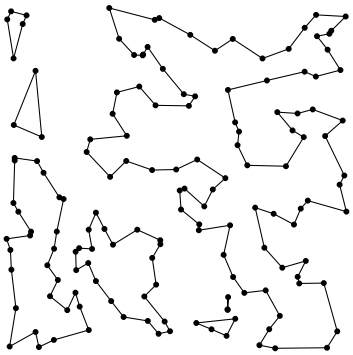

Figure 3.10: *RE_A_150*: iteration 9.

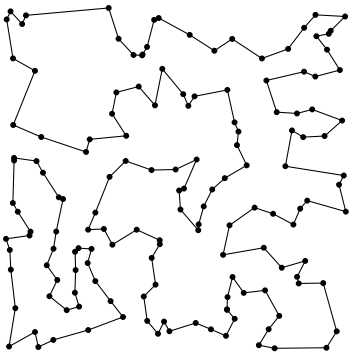

Figure 3.11: *RE_A_150*: iteration 10.

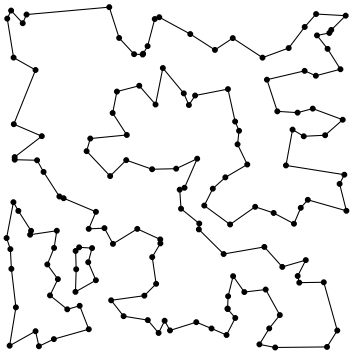

Figure 3.12: *RE_A_150*: iteration 11.

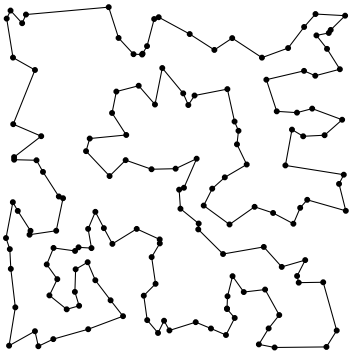

Figure 3.13: *RE_A_150*: iteration 12.

It should be pointed out that the main motivation of this framework is its simplicity. The separation of subtour elimination constraints for fractional solutions amounts to the solution of a max-flow or min-cut problem. Based on the procedure by PADBERG and RINALDI [40], extensive work has been done to construct elaborated algorithms for performing this task efficiently. On the contrary, violated subtour elimination constraints of integer solutions are trivial to find. Moreover, we refrain from using any other additional inequalities known for classical branch-and-cut algorithms, which might also be used to speed up our approach, since we want to underline the strength of modern ILP solvers in connection with a refined subtour selection process (see Subsection 3.2.6).

This approach for solving TSP is clearly not new but was available since the earliest ILP formulation going back to DANTZIG et al. [14] and can be seen as folklore nowadays. Several authors followed the concept of generating integer solutions for some kind of relaxation of an ILP formulation and iteratively adding violated integer subtour elimination constraints. However, it seems that the lack of fast ILP solvers prohibited its direct application in computational studies although it was used in an artistic context (see BOSCH [9]).

MILIOTIS [37] also concentrated on generating integer subtour elimination constraints, but within a fractional LP framework. The classical paper by CROWDER and PADBERG [13] applies the iterative generation of integer subtour elimination constraints as a second part of their algorithm after generating fractional cutting planes in the first part to strengthen the LP relaxation. They report that not more than three iterations of the ILP solver for the strengthened model were necessary for test instances up to 318 vertices. Also GRÖTSCHEL and HOLLAND [23] follow this direction of first improving the LP model as much as possible, e.g. by running preprocessing, fixing certain variables and strengthening the LP relaxation by different families of cutting planes, before generating integer subtours as last step to find an optimal tour. It turns out that about half of their test instances never reach this last phase. In contrast, we stick to the pure ILP formulation without any previous modifications.

From a theoretical perspective, the generation of subtours involves a certain trade-off. For an instance (G, d) there exists a minimal set of subtours $\mathcal{S}^*$, such that the ILP model with only those subtour elimination constraints implied by $\mathcal{S}^*$ yields an overall feasible, and thus optimal solution. However, in practice we can only find collections of subtours larger than $\mathcal{S}^*$ by adding subtours in every iteration until we reach optimality. Thus, we can either collect as many subtours as possible in each iteration, which may decrease

the number of iterations but increases the running time of the ILP solver because of the larger number of constraints. Or we try to control the number of subtour elimination constraints added to the model by trying to judge their relevance and possibly remove some of them later, which keeps the ILP model smaller but may increase the number of iterations. In the following we describe various strategies to find the "right" subtours.

3.1.1 Representation of subtour elimination constraints

The subtour elimination constraints (3.3) can be expressed equivalently by the following cut constraints:

$$\sum_{\substack{e=(u,v)\in E\\ u\in S, v\notin S}} x_e \geq 2 \quad \forall\ S \subset V,\ S \neq \emptyset \tag{3.5}$$

Although mathematically equivalent, the two ways of forbidding a subtour in S may result in quite different performances of the ILP solver.

It was observed that in general the running time for solving an ILP increases with the number of non-zero entries of the constraint matrix. Hence, we also tested a hybrid variant which chooses between (3.3) and (3.5) by picking for each considered set S the version with the smaller number of nonnegative coefficients on the left-hand side as follows:

$$\begin{array}{ll} \sum_{\substack{e=(u,v)\in E\\ u,v\in S}} x_e \leq |S|-1 & \text{if} \quad |S| \leq \frac{2n+1}{3} \\ \sum_{\substack{e=(u,v)\in E\\ u\in S, v\notin S}} x_e \geq 2 & \text{if} \quad |S| > \frac{2n+1}{3} \end{array} \qquad \forall\ S \subset V,\ S \neq \emptyset \tag{3.6}$$

We performed computational tests of our approach to compare the three representations of subtour elimination constraints, namely (3.3), (3.5) and (3.6), and list the results in Table 3.1. Technical details about the setup of the experiments can be found in Subsection 3.3.1.

It turned out that the three versions sometimes (but not always) lead to huge differences in running time (up to a factor of 5). This is an interesting experience that should be taken into consideration also in other computational studies. From our limited experiments it could be seen that version (3.5) was inferior most of the times (with sometimes huge deviations) whereas only a small dominance of the hybrid variant (3.6) in comparison with the standard version (3.3) could be observed. This is due to the small size of most subtours occurring during the solution process (the representation (3.3) equals to the representation (3.6) in these cases). But since also

instance	s.e.c. as in (3.3)			s.e.c. as in (3.5)			s.e.c. as in (3.6)		
	sec.	#i.	#c.	sec.	#i.	#c.	sec.	#i.	#c.
kroA150	89	12	82	75	12	82	**62**	**12**	**82**
kroB150	**52**	**13**	**77**	237	13	77	54	13	77
u159	**9**	**5**	**39**	13	5	39	**9**	**5**	**38**
brg180	62	14	56	**36**	**5**	**29**	64	16	67
kroA200	2153	11	95	**1833**	**11**	**95**	2440	11	95
kroB200	45	7	65	146	7	65	**37**	**7**	**65**
tsp225	**149**	**15**	**102**	376	16	105	155	16	106
a280	**114**	**10**	**59**	249	10	56	132	10	63
lin318	7171	13	177	8201	13	177	**7158**	**13**	**177**
gr431	5973	22	186	19111	22	187	**5925**	**22**	**186**
pcb442	4406	43	215	6186	41	197	**2393**	**43**	**207**
gr666	**33259**	**14**	**216**	189421	14	217	40111	14	216
mean ratio (sec.)				*2.305960*			*0.971694*		
RE_A_150	**23**	**12**	**61**	65	12	61	26	12	61
RE_A_200	81	15	84	139	15	84	**76**	**15**	**84**
RE_A_250	156	14	82	208	14	82	**133**	**14**	**82**
RE_A_300	**534**	**14**	**123**	4819	14	123	692	14	123
RE_A_350	**404**	**9**	**110**	789	9	110	650	9	110
RE_A_400	49234	16	179	247511	16	179	**24619**	**16**	**179**
RE_A_450	4666	8	117	13806	8	117	**3022**	**8**	**117**
RE_A_500	68215	12	167	155977	12	167	**30809**	**12**	**167**
mean ratio (sec.)				*3.390678*			*0.928176*		
mean ratio all				*2.739847*			*0.954287*		

Table 3.1: Comparison of the behavior of the algorithm for different representations of subtour elimination constraints. *Mean ratios* refer to the arithmetic means over ratios between the running times of the approaches using the subtour elimination constraints represented as in (3.5) and (3.6) respectively and the running time of the approach using the subtour elimination constraints represented as in (3.3). "sec." is the time in seconds, "#i." the number of iterations and "#c." the number of subtour elimination constraints added to the ILP before starting the last iteration.

bigger subtours can occur (mostly in the last iterations), we use the representation (3.6) for all further computational tests. For more details about different ILP models see Öncan et al. [39].

3.2 Generation of subtours

As pointed out above, the focus of our attention lies in the generation and selection of a "good" set of subtour elimination constraints, including as many as possible of those required by the ILP solver to determine an optimal solution which is also feasible for TSP, but as few as possible of all others which only slow down the performance of the ILP solver.

Trying to strike a balance between these two goals we followed several

directions, some of them motivated by theoretical results, others by visually studying plots of all subtours generated during the execution of Algorithm 3.1.

3.2.1 Subtour elimination constraints from suboptimal integer solutions

Many ILP solvers report all feasible integer solutions found during the underlying branch-and-bound process. In this case, we can also add all corresponding subtour elimination constraints to the model. These constraints can be considered simply as part of the set of violated subtour elimination constraints. Not surprisingly, these additional constraints always lead to a decrease in the number of iterations for the overall computation and to an increase in the total number of subtour elimination constraints generated before reaching optimality (see Table 3.2). While the time consumed in each iteration is likely to increase, it can also be observed that the overall running time is often decreased significantly by adding all detected subtours to the model. On the other hand, for the smaller number of instances where this is not the case, only relatively modest increases of running times are incurred. Therefore, we stick to adding all detected subtour elimination constraints for the remainder of the chapter. The algorithm in this form will be called *BasicIntegerTSP*.

3.2.2 Subtours of size 3

The next idea we tried was to add subtour inequalities corresponding to some subtours of size 3 into the model before starting the iteration process (i.e. in line 1 of Algorithm 3.1). This idea was motivated by the observations that in many examples smaller subtours (with respect to their cardinality) occur more often than the larger ones. However, there are $\binom{|V|}{3}$ such subtours and thus we should concentrate only on a relevant subset of them. After studying our computational tests we decided to use the shortest ones with respect to their length. Table 3.3 summarizes our computational results and it can be seen that this idea actually tends to slow down our approach. Thus we did not follow it any more.

3.2.3 Subtour selections

As mentioned above, a large number of subtour inequalities which are not really needed only slow down our approach. Thus we also tried not to use all subtour inequalities we are able to generate during one iteration, but to

instance	only subtours from ILP optima sec.	#i.	#c.	all subtours: BasicIntegerTSP sec.	#i.	#c.
kroA150	62	12	82	**19**	**7**	**136**
kroB150	**54**	**13**	**77**	179	8	148
u159	9	5	38	**6**	**4**	**49**
brg180	64	16	67	**44**	**4**	**103**
kroA200	2440	11	95	**677**	**8**	**237**
kroB200	37	7	65	**31**	**5**	**121**
tsp225	**155**	**16**	**106**	178	9	261
a280	**132**	**10**	**63**	157	11	143
lin318	7158	13	177	**6885**	**8**	**357**
gr431	5925	22	186	**2239**	**9**	**453**
pcb442	**2393**	**43**	**207**	2737	11	501
gr666	40111	14	216	**17711**	**8**	**789**
mean ratio (sec.)	*0.946130*					
RE_A_150	26	12	61	**23**	**8**	**100**
RE_A_200	76	15	84	**72**	**7**	**163**
RE_A_250	**133**	**14**	**82**	138	9	186
RE_A_300	**692**	**14**	**123**	866	6	295
RE_A_350	650	9	110	**411**	**5**	**252**
RE_A_400	24619	16	179	**8456**	**8**	**454**
RE_A_450	3022	8	117	**2107**	**5**	**279**
RE_A_500	30809	12	167	**15330**	**6**	**436**
mean ratio (sec.)	*0.786451*					
mean ratio all	*0.882259*					

Table 3.2: Using all constraints generated from all feasible integer solutions found during the solving process vs. using only the constraints generated from the final ILP solutions of each iteration. *Mean ratios* refer to the arithmetic means over ratios between the running times of *BasicIntegerTSP* over the other approach. "sec." is the time in seconds, "#i." the number of iterations and "#c." the number of subtour elimination constraints added to the ILP before starting the last iteration.

make a proper selection. We again used our computational tests in order to identify two general properties which seem to point to such "suitable" subtour inequalities.

- Sort all obtained subtours with respect to their cardinality, choose the smallest ones and add the corresponding subtour inequalities into the model.

- Sort all obtained subtours with respect to their length and proceed as above.

The corresponding results are summarized in Tables 3.4 and 3.5 and it is obvious that this idea does not speed up our approach as intended. Thus we dropped it from our considerations.

instance	$p = 0$			$p = \frac{1}{10000}$			$p = \frac{1}{1000}$		
	sec.	**#i.**	**#c.**	**sec.**	**#i.**	**#c.**	**sec.**	**#i.**	**#c.**
kroA150	**19**	**7**	**136**	**19**	**7**	**97**	40	5	116
kroB150	179	8	148	**71**	**7**	**178**	134	5	105
u159	**6**	**4**	**49**	8	4	46	**6**	**3**	**24**
brg180	44	4	103	**34**	**15**	**108**	82	9	270
kroA200	677	8	237	879	5	157	**504**	**4**	**133**
kroB200	**31**	**5**	**121**	32	5	61	43	5	60
tsp225	178	9	261	**149**	**10**	**224**	167	9	202
a280	157	11	143	**138**	**9**	**98**	156	6	101
lin318	6885	8	357	5360	8	302	**1435**	**8**	**291**
gr431	**2239**	**9**	**453**	3196	10	534	3648	10	571
pcb442	**2737**	**11**	**501**	3483	15	414	3989	14	466
gr666	**17711**	**8**	**789**	–	–	–	–	–	–
mean ratio				*1.002535*			*1.188732*		
RE_A_150	**23**	**8**	**100**	30	7	130	30	6	77
RE_A_200	72	7	163	74	8	135	**57**	**6**	**76**
RE_A_250	**138**	**9**	**186**	155	7	163	140	6	109
RE_A_300	**866**	**6**	**295**	884	6	203	1344	7	211
RE_A_350	**411**	**5**	**252**	642	6	147	879	6	150
RE_A_400	8456	8	454	6623	7	285	**4876**	**8**	**296**
RE_A_450	2107	5	279	**1226**	**4**	**220**	5386	5	215
RE_A_500	15330	6	436	13473	6	366	**6114**	**5**	**237**
mean ratio				*1.035264*			*1.291607*		
mean ratio all				*1.016316*			*1.232048*		

Table 3.3: Using no subtours of size 3 vs. using the shortest subtours of size 3 for generation of subtour constraints before starting the solving process. The parameter p defines the proportion of used subtour constraints. *Mean ratios* refer to the arithmetic means over ratios between the running times of the particular approaches and the running time of the *BasicIntegerTSP* (corresponding to $p = 0$). "sec." is the time in seconds, "#i." the number of iterations and "#c." the number of subtour elimination constraints added to the ILP before starting the last iteration. The entries "–" by TSPLIB instances cannot be computed with 16 GB RAM.

3.2.4 Clustering into subproblems

It can be observed that many subtours have a local context, meaning that a small subset of vertices separated from the remaining vertices by a reasonably large distance will always be connected by one or more subtours, independently from the size of the remaining graph (see also Figures 3.1 and 3.2 to 3.13). Thus, we aim to identify *clusters* of vertices and run the *BasicIntegerTSP* on the induced subgraphs with the aim of generating within a very small running time the same subtours occurring in the execution of the approach on the full graph. Furthermore, we can use the optimal tour from every cluster in order to generate a corresponding subtour elimination constraint for the original instance and therefore enforce a connection between

instance	$p = 1$			$p = \frac{2}{3}$			$p = \frac{1}{3}$		
	sec.	**#i.**	**#c.**	**sec.**	**#i.**	**#c.**	**sec.**	**#i.**	**#c.**
kroA150	**19**	**7**	**136**	34	8	109	69	19	115
kroB150	179	8	148	**51**	**8**	**135**	477	15	134
u159	**6**	**4**	**49**	30	4	52	19	11	56
brg180	44	4	103	**27**	**6**	**77**	59	19	80
kroA200	**677**	**8**	**237**	714	7	171	2846	14	131
kroB200	**31**	**5**	**121**	39	6	98	89	13	77
tsp225	178	9	261	**100**	**14**	**183**	173	34	166
a280	157	11	143	**141**	**12**	**154**	239	27	127
lin318	**6885**	**8**	**357**	7069	12	367	9444	32	392
gr431	**2239**	**9**	**453**	3210	20	522	4924	38	413
pcb442	2737	11	501	**1867**	**18**	**384**	5129	85	386
gr666	17711	8	789	**7643**	**7**	**505**	71594	25	597
mean ratio				*1.252892*			*2.488345*		
RE_A_150	**23**	**8**	**100**	28	9	109	52	17	94
RE_A_200	72	7	163	**69**	**8**	**134**	112	23	98
RE_A_250	138	9	186	**131**	**10**	**149**	208	20	119
RE_A_300	866	6	295	**792**	**10**	**259**	1720	29	293
RE_A_350	**411**	**5**	**252**	715	7	232	849	19	177
RE_A_400	**8456**	**8**	**454**	129380	8	311	107987	26	299
RE_A_450	2107	5	279	**1544**	**7**	**236**	7987	11	238
RE_A_500	15330	6	436	18594	8	324	**13738**	**16**	**308**
mean ratio				*2.878162*			*3.354102*		
mean ratio all				*1.903000*			*2.834648*		

Table 3.4: Using all subtours vs. using only the smallest subtours with respect to their cardinality for generation of subtour constraints. The parameter p defines the proportion of used subtour constraints. *Mean ratios* refer to the arithmetic means over ratios between the running times of the particular approaches and the running time of the *BasicIntegerTSP* (corresponding to $p = 1$). "sec." is the time in seconds, "#i." the number of iterations and "#c." the number of subtour elimination constraints added to the ILP before starting the last iteration.

this cluster and the remainder of the graph.

For our purposes the clustering algorithm should fulfill the following properties:

- *clustering quality:* The obtained clusters should correspond well to the distance structure of the given graph, as in a classical geographic clustering.
- *running time:* It should be low relative to the running time required for the main part of the algorithm.
- *cluster size:* If clusters are too large, solving the TSP takes too much time. If clusters are too small, only few subtour elimination constraints are generated.

instance	$p = 1$			$p = \frac{2}{3}$			$p = \frac{1}{3}$		
	sec.	**#i.**	**#c.**	**sec.**	**#i.**	**#c.**	**sec.**	**#i.**	**#c.**
kroA150	**19**	**7**	**136**	41	10	131	46	16	90
kroB150	**179**	**8**	**148**	495	7	152	250	16	112
u159	**6**	**4**	**49**	14	5	60	23	12	55
brg180	44	4	103	**24**	**13**	**86**	161	8	78
kroA200	**677**	**8**	**237**	862	6	124	1829	13	132
kroB200	**31**	**5**	**121**	59	7	121	79	11	89
tsp225	178	9	261	**112**	**13**	**197**	197	32	159
a280	157	11	143	**94**	**9**	**101**	212	21	96
lin318	**6885**	**8**	**357**	7688	13	355	9593	36	390
gr431	**2239**	**9**	**453**	6091	15	565	9434	45	530
pcb442	2737	11	501	**2365**	**18**	**487**	5913	70	399
gr666	17711	8	789	**14713**	**10**	**735**	–	–	–
mean ratio				*1.478194*			*2.434945*		
RE_A_150	**23**	**8**	**100**	24	9	115	45	22	81
RE_A_200	72	7	163	**60**	**10**	**123**	113	25	108
RE_A_250	**138**	**9**	**186**	**138**	**7**	**117**	209	22	103
RE_A_300	**866**	**6**	**295**	1099	10	321	953	23	201
RE_A_350	**411**	**5**	**252**	876	7	231	934	16	167
RE_A_400	**8456**	**8**	**454**	29625	9	311	301125	27	378
RE_A_450	**2107**	**5**	**279**	2926	7	259	4789	14	237
RE_A_500	**15330**	**6**	**436**	15786	7	329	37460	16	330
mean ratio				*1.524891*			*6.092589*		
mean ratio all				*1.496873*			*3.975006*		

Table 3.5: Using all subtours vs. using only the smallest subtours with respect to their length for generation of subtour constraints. The parameter p defines the proportion of used subtour constraints. *Mean ratios* refer to the arithmetic means over ratios between the running times of the particular approaches and the running time of the *BasicIntegerTSP* (corresponding to $p = 1$). "sec." is the time in seconds, "#i." the number of iterations and "#c." the number of subtour elimination constraints added to the ILP before starting the last iteration. The entries "–" by TSPLIB instances cannot be computed with 16 GB RAM.

Clearly, there is a huge body of literature on clustering algorithms (see e.g. JAIN and DUBES [29]) and selecting one for a given application will never satisfy all our objectives. Our main restriction was the requirement of using a clustering algorithm which works also if the vertices are not embeddable in Euclidean space, i.e. only arbitrary edge distances are given. Simplicity being another goal, we settled for the following approach described in Algorithm 3.2:

Require: complete graph $G = (V, E)$, where $|V| = n$ and $|E| = m = \frac{n(n-1)}{2}$, distance function $d\colon E \to \mathbb{R}_0^+$ and parameter c, where $1 \leq c \leq n$
Ensure: clustering $\mathcal{C} = \{V_1, \ldots, V_c\}$, where $V_1 \cup \ldots \cup V_c = V$
1: sort the edges such that $d_{e_1} \leq \ldots \leq d_{e_m}$;

2: define $G' = (V', E')$ such that $V' = V$ and $E' = \emptyset$;
3: let $i := 1$;
4: define $\mathcal{C} := \{\{v_1\}, \ldots, \{v_n\}\}$;
5: **while** $|\mathcal{C}| > c$ **do**
6: set $E' := E' \cup \{e_i\}$;
7: set $\mathcal{C} := \{V_1, \ldots, V_{|\mathcal{C}|}\}$, where $V_1, \ldots, V_{|\mathcal{C}|}$ are the connected components of graph G';
8: **end while**

Algorithm 3.2: Clustering algorithm.

First, we fix the number of clusters c with $1 \leq c \leq n$ and sort the edges in increasing order of distances (see line 1). Then we start with the empty graph $G' = (V', E')$ (line 2) containing only isolated vertices (i.e. n clusters) and add iteratively edges in increasing order of distances until the desired number of clusters c is reached (see lines 5 and 6). In each iteration the current clustering is implied by the connected components of the current graph (see line 7). We denote this *clustering approach* by $C \mid c$. Note that this clustering algorithm does not make any assumptions about the underlying TSP instance and does not exploit any structural properties of the *Metric TSP* or the *Euclidean TSP*. An example illustrating the behavior of our clustering algorithm on the instance *RE_A_150* for different parameters c can be found in Figures 3.14–3.19.

It was observed in our computational experiments that the performance of the TSP algorithm is not very sensitive to small changes of the cluster number c and thus a rough estimation of c is sufficient. The behavior of the running time as a function of c can be found for particular test instances in Figure 3.33, see Subsection 3.3.2 for further discussion.

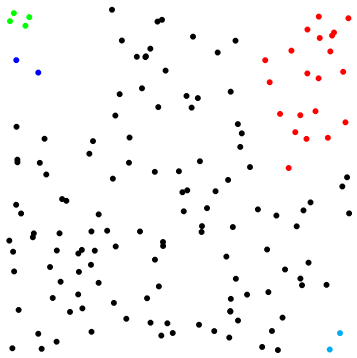

Figure 3.14: Instance *RE_A_150*: Clustering for $c = 5$.

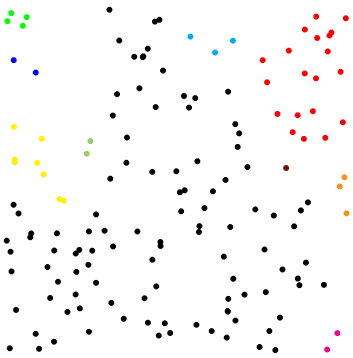

Figure 3.15: Instance *RE_A_150*: Clustering for $c = 10$.

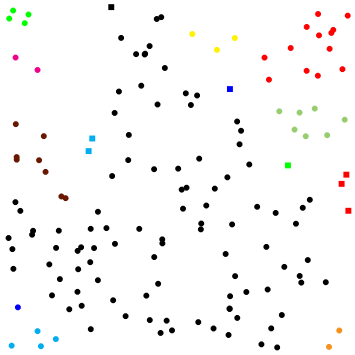

Figure 3.16: Instance *RE_A_150*: Clustering for $c = 15$.

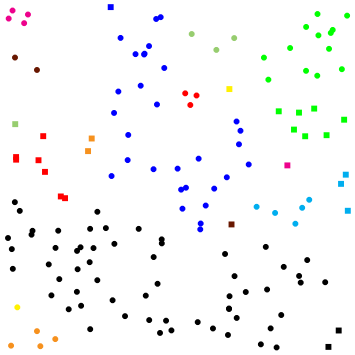

Figure 3.17: Instance *RE_A_150*: Clustering for $c = 20$.

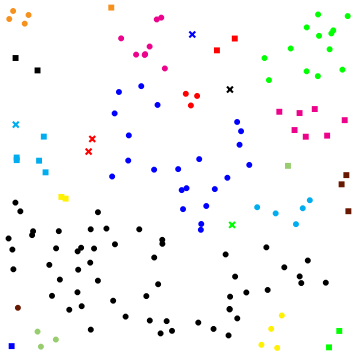

Figure 3.18: Instance *RE_A_150*: Clustering for $c = 25$.

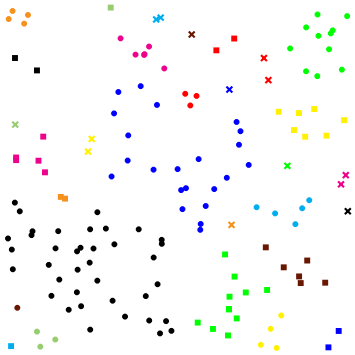

Figure 3.19: Instance *RE_A_150*: Clustering for $c = 30$.

3.2.5 Restricted clustering

Although the clustering algorithm (see Algorithm 3.2) decreases the computational time of the whole solution process for some test instances, we observed a certain shortcoming. There may easily occur clusters consisting of isolated points or containing only two vertices. Clearly, these clusters do not contribute any subtour on their own. Moreover, the degree constraints (3.2) guarantee that each such vertex is connected to the remainder of the graph in any case. The connection of these vertices to some "neighboring" cluster enforced in *BasicIntegerTSP* implies that the clustering yields different subtours for these neighbors and not the violated subtour elimination constraints arising in *BasicIntegerTSP*.

To avoid this situation, we want to impose a minimum cluster size of 3. An easy way to do so is as follows: After reaching the c clusters, continue to add edges in increasing order of distances (as before), but add an edge only, if it is incident to one of the vertices in a connected component (i.e. cluster) of size one or two. This means basically that we simply merge these small clusters to their nearest neighbor with respect to the actual clustering. Note that this is a step-by-step process and it can happen that two clusters of size 1 merge first before merging the resulting pair to its nearest neighboring cluster. The resulting *restricted clustering approach* will be denoted by $RC_3 \mid c$. An example containing restricted clusterings for different values of the parameter c is contained in Figures 3.20–3.25.

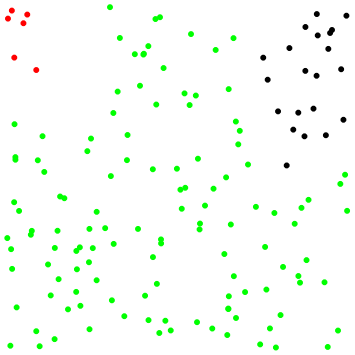

Figure 3.20: Instance *RE_A_150*: Restricted clustering for $c = 5$.

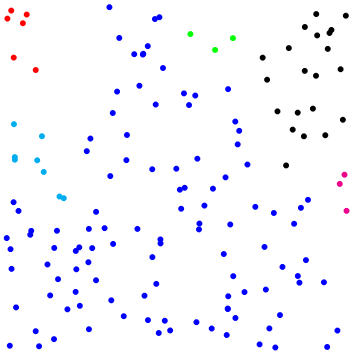

Figure 3.21: Instance *RE_A_150*: Restricted clustering for $c = 10$.

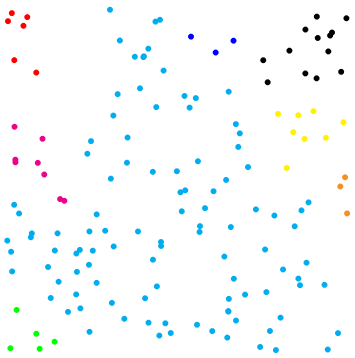

Figure 3.22: Instance *RE_A_150*: Restricted clustering for $c = 15$.

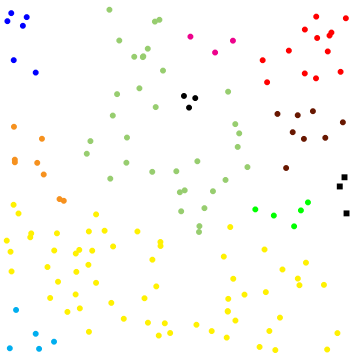

Figure 3.23: Instance *RE_A_150*: Restricted clustering for $c = 20$.

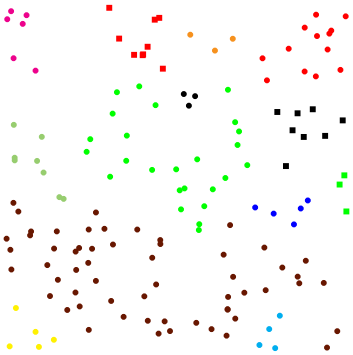

Figure 3.24: Instance *RE_A_150*: Restricted clustering for $c = 25$.

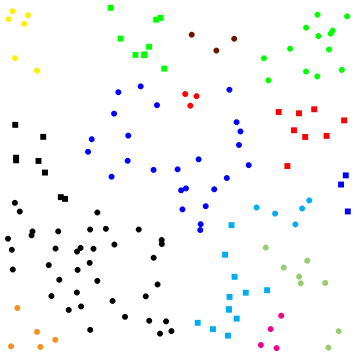

Figure 3.25: Instance *RE_A_150*: Restricted clustering for $c = 30$.

Against our expectations, the computational experiments (see Section 3.3) show that this approach often impacts the algorithm in the opposite way (see also Figure 3.33 and Table 3.9 in Appendix) if compared for the same original cluster size c.

Surprisingly, we could observe an interesting behavior if $c \approx n$. In this case, the main clustering algorithm (see Algorithm 3.2) has almost no effect, but the "post-phase" which enforces the minimum cluster size yields a different clustering on its own. An example depicting such clustering for the instance *RE_A_150* is depicted in Figure 3.26. This variant often beats

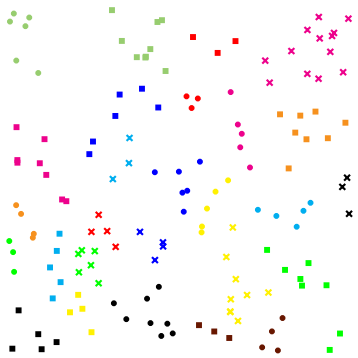

Figure 3.26: Instance *RE_A_150*: Restricted clustering for $c = 150$.

the previous standard clustering algorithm with $c \ll n$ (see Table 3.9 in Appendix). Note that we cannot fix the actual number of clusters c' in this case. But our computational results show that $c' \approx \frac{n}{5}$ usually holds if the

points are distributed relatively uniformly in the Euclidean plane and if the distances correspond to their relative Euclidean distances (see Figure 3.27).

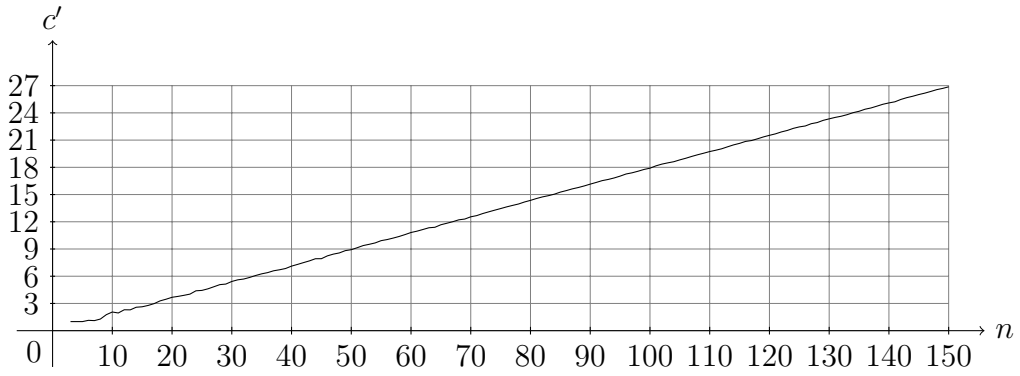

Figure 3.27: Restricted clustering with $c = n$ on random Euclidean graphs with minimum cluster size 3. The number of obtained clusters c' is plotted for every n. For every number of vertices n we created 100000 graphs.

3.2.6 Hierarchical clustering

It was pointed out in Subsection 3.2.4 that the number of clusters c is chosen as an input parameter. The computational experiments in Subsection 3.3.2 give some indication on the behavior of Algorithm 3.2 for different values of c, but fail to provide a clear guideline for the selection of c. Moreover, from graphical inspection of test instances, we got the impression that a larger number of relevant subtour elimination constraints might be obtained by considering more clusters of moderate size. In the following we present an idea that takes both of these aspects into account.

In our *hierarchical clustering* process denoted by HC we do not set a cluster number c, but let the clustering algorithm continue until all vertices are connected (this corresponds to $c = 1$). The resulting clustering process can be represented by a binary *clustering tree* which is constructed in a bottom-up way. The leaves of the tree represent isolated vertices, i.e. the n trivial clusters given at the beginning of the clustering algorithm. Whenever two clusters are merged by the addition of an edge, the two corresponding tree vertices are connected to a new common parent vertex in the tree representing the new cluster. At the end of this process we reach the root of the clustering tree corresponding to the complete vertex set. An example of such a clustering tree is shown in Figures 3.28 and 3.29.

Figure 3.28: Example illustrating the hierarchical clustering: Vertices of the TSP instance. Distances between every two vertices correspond to their respective Euclidean distances in this example.

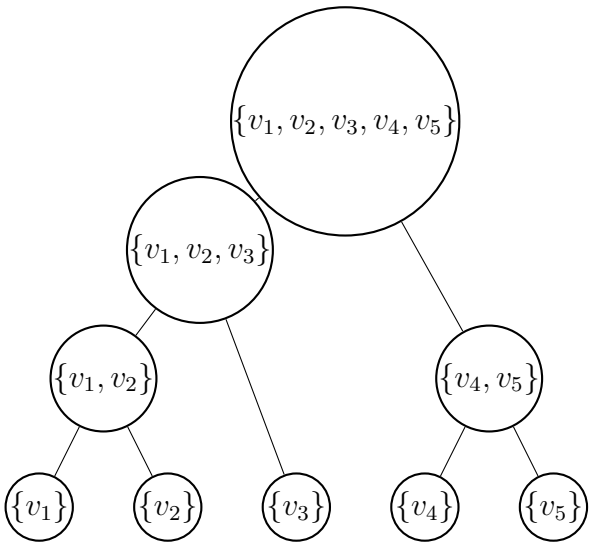

Figure 3.29: Example illustrating the hierarchical clustering: Clustering tree.

Now, we go through the tree in a bottom-up fashion from the leaves to the root. In each tree vertex we solve the TSP for the associated cluster, after both of its child vertices were resolved. The crucial aspect of our procedure is the following: All subtour elimination constraints generated during such a TSP solution for a certain cluster are propagated and added to the ILP model used for solving the TSP instance of its parent cluster. Obviously, at the root vertex the full TSP is solved.

The advantage of this strategy is the step-by-step construction of the violated subtour elimination constraints. A disadvantage is that many constraints can make sense in the local context but not in the global one and thus too many constraints could be generated in this way. Naturally, one pays for the additional subtour elimination constraints by an increase in computation

time required to solve a large number of – mostly small – TSP instances. To avoid the solution of TSPs of the same order of magnitude as the original instance, it makes sense to impose an upper bound u on the maximum cluster size. This means that the clustering tree is partitioned into several subtrees by removing all tree vertices corresponding to clusters of a greater size than u. After resolving all these subtrees we collect all generated subtour elimination constraints and add them to the ILP model for the originally given TSP. This approach will be denoted as $HC \mid u$. Computational experiments with various choices of u indicated that $u = 4\frac{n}{\log_2 n}$ would be a good upper bound.

Let us take a closer look at the problem of including too many subtour elimination constraints which are redundant in the global graph context. Of course the theoretical "best" way would be to check which of the propagated subtour elimination constraints were not used during the runs of the ILP solver and drop them. To do this, it would be necessary to get this information from the ILP solver which often is not possible.

However, we can try to approximately identify subtours which are not only locally relevant in the following way: All subtour elimination constraints generated in a certain tree vertex, i.e. for a certain cluster, are marked as *considered subtour elimination constraints*. Then we solve the TSP for the cluster of its parent vertex in the tree without using the subtours marked as *considered*. If we generate such a considered subtour again during the solution of the parent vertex, we take this as an indicator of global significance and add the constraint permanently for all following supersets of this cluster. If we set the upper bound u, we take also all subtour elimination constraints found in the biggest solved clusters. This approach will be denoted as $HCD \mid u$.

Of course, it is only a heuristic rule and one can easily find examples, where this prediction on a subtour's relevance fails, but our experiments indicate that $HCD \mid 4n/\log_2 n$ is the best approach we considered. A comparison with other hierarchical clustering methods for all test instances can be found in Table 3.8 in Appendix. It can be seen that without an upper bound we are often not able to find the solution at all (under time and memory constraints we made on the computational experiments). In the third and fourth column we can see a comparison between approaches both using the upper bound $u = 4\frac{n}{\log_2 n}$ where the former collects all detected subtour elimination constraints and the latter allows to drop those which seem to be relevant only in a local context. Both these methods beat *BasicIntegerTSP* (for the comparison of this approach with other presented algorithms see the computational experiments in Section 3.3).

3.3 Computational experiments

In the following the computational experiments and their results will be discussed.

3.3.1 Setup of the computational experiments

All tests were run on an *Intel(R) Core(TM) i5-3470 CPU @ 3.20GHz with 16 GB RAM* under *Linux*[1] and all programs were implemented in *C++*[2] by using the *SCIP* MIP-solver (see ACHTERBERG [2]) together with *CPLEX* as LP solver[3]. It has often been discussed in the literature (see e.g. NADDEF and THIENEL [38]) and in personal communications that ILP solvers are relatively unrobust and often show high variations in their running time performance, even if the same instance is repeatedly run on the same hardware and same software environment. Our first test runs also exhibited deviations up to a factor of 2 when identical tests were repeated. Thus we took special care to guarantee the relative reproducibility of the computational experiments: No additional swap memory was made available during the tests, only one thread was used and no other parallel user processes were allowed. This leads to a high degree of reproducibility in our experiments. However, this issue makes a comparison to other simple approaches, which were tested on other computers under other hardware and software conditions, extremely difficult.

We used two groups of test instances: The first group is taken from the well-known TSPLIB95 collected by REINELT [45], which contains the established benchmarks for TSP and related problems. From the collection of instances we chose all those with (i) at least 150 and at most 1000 vertices and (ii) which could be solved in at most 12 hours by our *BasicIntegerTSP*. It turned out that 25 instances of the TSPLIB95 fall into this category (see Table 3.9), the largest having 783 vertices.

We also observed some drawbacks of these instances: Most of them (23 of 25) are defined as point sets in the Euclidean plane with distances corresponding to the Euclidean metric or as a set of geographical cities, i.e. points on a sphere. Moreover, they often contain substructures like meshes or sets of colinear points and finally, since all distances are rounded to the

[1]Precise version: *Linux 3.8.0-29-generic #42~precise1-Ubuntu SMP x86_64 x86_64 x86_64 GNU/Linux.*

[2]Precise compiler version: *gcc version 4.6.3.*

[3]Precise version: *SCIP version 3.0.1 [precision: 8 byte] [memory: block] [mode: optimized] [LP solver: CPLEX 12.4.0.0] [GitHash: 9ee94b7] Copyright (c) 2002-2013 Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB).*

nearest integer, there are many instances which have multiple optimal solutions. These instances are relatively unstable with respect to solution time, number of iterations, and – important for our approach – cardinality of the set of violated subtour elimination constraints. For our approach instances with a mesh geometry (e.g. *ts225* from TSPLIB95) were especially prone to unstable behavior, such as widely varying running times for minor changes in the parameter setting. This seems to be due to the fact that these instances contain many 2-matchings with the same objective function value as illustrated in the following example: Consider a $3 \times (2n + 2)$ mesh graph (see Figure 3.30, left graph). It has 2^n optimal TSP tours (see TOŠIĆ and BODROŽA [52]). If we fix a subtour on the first 6 vertices, we obviously have 2^{n-1} optimal TSP tours on the remaining $3 \times \big((2n + 2) - 2\big)$vertices (see Figure 3.30, right graph) and together with the fixed subtour we have 2^{n-1} 2-matchings having the same objective value as an optimal TSP tour on the original graph. Thus the search process for a feasible TSP tour can vary widely.

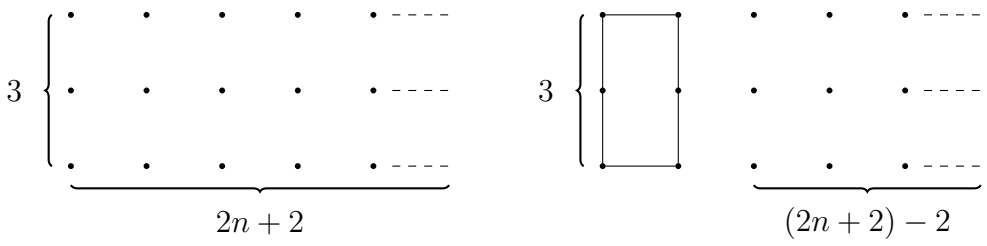

Figure 3.30: Example illustrating the behavior of our approaches by instances based on graphs containing mesh substructures. Distances between every two vertices correspond to their respective Euclidean distances in this example.

In order to provide further comparisons, we also defined a set of instances based on *random Euclidean graphs*: In a unit square $[0, 1]^2$ we chose n uniformly distributed points and defined the distance between every two vertices as their respective Euclidean distance[4]. These *random Euclidean instances* eliminate the potential influence of substructures and always have only one unique optimal solution in all stages of the solving process. We created 40 such instances named *RE_X_n* where $n \in \{150, 200, 250, \ldots, 500\}$ indicates the number of vertices and $X \in \{A, B, C, D, E\}$.

The running times of our test instances, most of them containing between 150 and 500 vertices, were often within several hours. Since we tested many

[4]We represented all distances as integers by scaling with 2^{14} and rounding to the nearest integer.

different variants and configurations of our approach, we selected a subset of these test instances to get faster answers for determining the best algorithm settings for use in the final tests. This subset contains 12 (of the 25) TSPLIB instances and one random instances for every number of vertices n (see e.g. Table 3.1.)

All our running time tables report the name of the instance, the running time (**sec.**) in wall-clock seconds (rounded down to nearest integers), the number of iterations (**#i.**), i.e. the number of calls to the ILP solver in the main part of our algorithm (without the TSP solutions for the clusters) and the number of subtour elimination constraints (**#c.**) added to the ILP model in the last iteration, i.e. the number of constraints of the model which yielded an optimal TSP solution. We often compare two columns of a table by taking the **mean ratio**, i.e. computing the quotient between the running times on the same instance and taking the arithmetic mean of these quotients.

3.3.2 Computational details for selected examples

Let us now take a closer look at two instances in detail. While this serves only as an illustration, we studied lots of these special case scenarios visually during the development of the clustering approach to gain a better insight into the structure of subtours generated by *BasicIntegerTSP*.

We selected instances *kroB150* and *u159* whose vertices are depicted in Figures 3.31 and 3.32. Both instances consist of points in the Euclidean

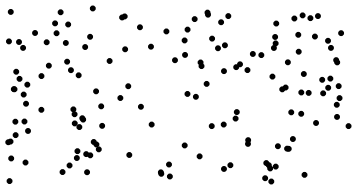

Figure 3.31: Instance *kroB150*. Euclidean distances between vertices.

Figure 3.32: Instance *u159*. Euclidean distances between vertices.

plane and the distances between every two vertices correspond to their respective Euclidean distances, however, they represent two very different instance types: The instance *kroB150* consists of relatively uniformly dis-

tributed points, the instance *u159* is more structured and it contains e.g. mesh substructures which are the worst setting for our algorithm (recall Subsection 3.3.1).

Figure 3.33 illustrates the behavior of the running time t in seconds as a function of the parameter c for the instances *kroB150* and *u159*. The full lines correspond to standard clustering approach $C \mid c$ described in Section 3.2.4 (see Algorithm 3.2), while the dashed line corresponds to the restricted clustering $RC_3 \mid c$ of Subsection 3.2.5 with minimum cluster size 3. The standard *BasicIntegerTSP* without clustering arises for $c = 1$.

Instance *kroB150* consists of relatively uniformly distributed points in the Euclidean plane, but has a specific property: By using Algorithm 3.2 we can observe the occurrence of two main components also for relatively small coefficient c (already for $c = 6$). This behavior is rather atypical for random Euclidean graphs, cf PENROSE [41, ch. 13], but it provides an advantage for our approach since we do not have to solve cluster instances of the same order of magnitude as the original graph but have several clusters of moderate size also for small cluster numbers c.

Considering the standard clustering approach (Algorithm 3.2) in Figure 3.33, upper graph, it can be seen that only a small improvement occurs for c between 2 and 5. Looking at the corresponding clusterings in detail, it turns out in these cases that there exists only one "giant connected component" and all other clusters have size 1. This structure also implies that for the restricted clustering these isolated vertices are merged with the giant component and the effect of clustering is lost completely. For larger cluster numbers c, a considerable speedup is obtained, with some variation, but more or less in the same range for almost all values of $c \geq 6$ (in fact, the giant component splits in these cases). Moreover, the restricted clustering performs roughly as good as the standard clustering for $c \geq 6$.

Instance *u159* is much more structured and has many colinear vertices. Here, we can observe a different behavior. While the standard clustering is actually beaten by *BasicIntegerTSP* for smaller cluster numbers and has a more or less similar performance for larger cluster numbers, the restricted clustering is almost consistently better than the other two approaches. For c between 2 and 10 there exists a large component containing many mesh substructures which consumes as much computation time as the whole instance.

These two instances give some indication of how to characterize "good" instances for our algorithm: They should

- consist of more clearly separated clusters and
- not contain mesh substructures and colinear vertices.

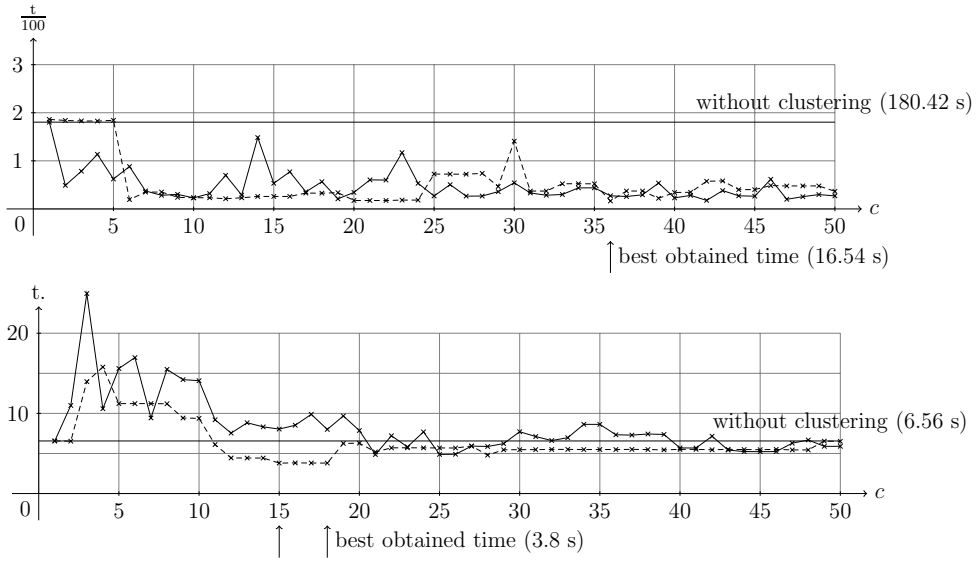

Figure 3.33: Computation time t in seconds depending on the number of clusters c for clustering (full line) and for restricted clustering (dashed). Illustrative instances *kroB150* (upper figure) and *u159* (lower figure).

3.3.3 General computational results

A summary of the computational results for *BasicIntegerTSP* and the most promising variants of clustering based subtour generations can be found in Table 3.9. For random Euclidean instances we report only the mean values of all five instances of the same size. It turns out that $HCD \mid 4\frac{n}{\log_2 n}$, i.e. the hierarchical clustering approach combined with dropping subtour elimination constraints and fixing them only if they are generated again in the subsequent iteration and with the upper bound on the maximum cluster size $u = 4\frac{n}{\log_2 n}$, gives the best overall performance. A different behavior can be observed for instances taken from the TSPLIB and for random Euclidean instances. On the TSPLIB instances this algorithm $HCD \mid 4\frac{n}{\log_2 n}$ is on average about 20% faster than pure *BasicIntegerTSP* and beats the other clustering based approaches for most instances. In those cases, where it is not the best choice, it is usually not far behind.

As already mentioned, best results are obtained with $HCD \mid 4\frac{n}{\log_2 n}$ for instances with a strong cluster structure and without mesh substructures (e.g. *pr299*). For instances with mesh substructures it is difficult to find an optimal 2-matching which is also a TSP tour. For random Euclidean instances the results are less clear but approaches with fixed number of clusters seem to be better then the hierarchical ones.

It was a main goal of this study to find a large number of "good" subtour elimination constraints, i.e. subtours that are present in the last iteration of the ILP model of *BasicIntegerTSP*. Therefore, we show the potentials and limitations of our approach in reaching this goal. In particular, we will report the relation between the set S_1 consisting of all subtours generated by running a hierarchical clustering algorithm with an upper bound u (set as in the computational tests to $u = 4\frac{n}{\log_2 n}$) before solving the original problem (i.e. the root vertex) and the set S_2 containing only the subtour elimination constraints included in the final ILP model of *BasicIntegerTSP*. We tested the hierarchical clustering with and without the dropping of non-repeated subtours.

There are two aspects we want to describe: At first, we want to check whether S_1 contains a relevant proportion of "useful" subtour contraints, i.e. constraints also included in S_2, or whether S_1 contains "mostly useless" subtours. Therefore, we report the *proportion of used subtours* defined as

$$p_{used} := \frac{|S_1 \cap S_2|}{|S_1|}. \tag{3.7}$$

Secondly, we want to find out to what extend it is possible to find the "right"

instance	$HC \mid 4\frac{n}{\log_2 n}$		$HCD \mid 4\frac{n}{\log_2 n}$	
	p_{used}	p_{cov}	p_{used}	p_{cov}
kroA150	0.262712	0.455882	0.476190	0.367647
kroB150	0.222222	0.351351	0.396040	0.270270
u159	0.085271	0.448980	0.153226	0.387755
brg180	0.133929	0.145631	0.714286	0.145631
kroA200	0.209713	0.324895	0.450704	0.270042
kroB200	0.206612	0.413223	0.423423	0.388430
tsp225	0.134752	0.218391	0.297143	0.199234
a280	0.064935	0.314685	0.161943	0.279720
lin318	0.234589	0.383754	0.440273	0.361345
gr431	0.073701	0.209713	0.221053	0.185430
pcb442	0.056759	0.151697	0.133117	0.163673
gr666	0.076048	0.271229	0.220379	0.235741
mean	*0.146770*	*0.307453*	*0.340648*	*0.271243*
RE_A_150	0.179191	0.310000	0.289157	0.240000
RE_A_200	0.122642	0.239264	0.212329	0.190184
RE_A_250	0.120773	0.268817	0.172727	0.204301
RE_A_300	0.191235	0.325424	0.331915	0.264407
RE_A_350	0.151274	0.376984	0.285714	0.333333
RE_A_400	0.170455	0.297357	0.254157	0.235683
RE_A_450	0.148148	0.415771	0.311178	0.369176
RE_A_500	0.165485	0.321101	0.276596	0.268349
mean	*0.156150*	*0.319340*	*0.266722*	*0.263179*
mean of all	*0.150522*	*0.312207*	*0.311078*	*0.268018*

Table 3.6: Proportion of used and proportion of covered subtours for our hierarchical clustering approaches with the upper bound $u = 4\frac{n}{\log_2 n}$ which (i) does not allow ($HC \mid 4\frac{n}{\log_2 n}$) and which (ii) does allow ($HCD \mid 4\frac{n}{\log_2 n}$) to drop the unused subtour elimination constraints.

subtours by our approach. Hence, we define the *proportion of covered subtours* defined as

$$p_{cov} := \frac{|S_1 \cap S_2|}{|S_2|}. \tag{3.8}$$

The values of p_{used} and p_{cov} are given in Table 3.6. It can be seen that empirically there is the chance to find about 26–31% (p_{cov}) of all required violated subtour elimination constraints. If subtour elimination constraints are allowed to be dropped, we are able to find fewer such constraints, but our choice has a better quality (p_{cov} is smaller, but p_{used} is larger), i.e. the solver does not have to work with a large number of constraints which only slow down the solving process and are not necessary to reach an optimal solution.

Furthermore, we can observe a relative big difference between the values of the proportion of used subtour elimination constraints (p_{used}) for the TSPLIB instances and for random Euclidean instances if the dropping of redundant constraints is allowed.

3.3.4 Adding a starting heuristic

Of course, there are many possibilities of adding improvements to our basic approach. Lower bounds and heuristics can be introduced, branching rules can be specified, or cutting planes can be generated. We did not pursue these possibilities since we want to focus on the simplicity of the approach. Moreover, we wanted to take the ILP solver as a "black box" and not interfere with its execution.

Just as an example which immediately comes to mind, we added a starting heuristic to give a reasonably good TSP solution as a starting solution to the ILP solver. We used the improved version of the classical Lin-Kernighan heuristic in the code written by HELSGAUN [26]. The computational results reported in Table 3.7 show that a considerable speedup (roughly a factor of 3, but also much more) can be obtained in this way.

instance	**without starting heuristic**			**with starting heuristic**		
	sec.	**#i.**	**#c.**	**sec.**	**#i.**	**#c.**
kroA150	19	7	136	**16**	**10**	**34**
kroB150	179	8	148	**17**	**8**	**104**
u159	6	4	49	**4**	**5**	**40**
brg180	44	4	103	**0**	**2**	**15**
kroA200	677	8	237	**42**	**8**	**135**
kroB200	31	5	121	**28**	**6**	**124**
tsp225	178	9	261	**73**	**13**	**176**
a280	157	11	143	**32**	**8**	**58**
lin318	6885	8	357	**4941**	**8**	**259**
gr431	2239	9	453	**838**	**10**	**318**
pcb442	2737	11	501	**447**	**18**	**207**
gr666	17711	8	789	**13225**	**11**	**485**
mean ratio (sec.)	*0.432074*					
RE_A_150	23	8	100	**14**	**11**	**65**
RE_A_200	72	7	163	**38**	**11**	**99**
RE_A_250	138	9	186	**63**	**9**	**124**
RE_A_300	866	6	295	**146**	**8**	**173**
RE_A_350	411	5	252	**126**	**6**	**151**
RE_A_400	8456	8	454	**1274**	**6**	**251**
RE_A_450	2107	5	279	**482**	**7**	**197**
RE_A_500	15330	6	436	**1997**	**9**	**241**
mean ratio (sec.)	*0.322231*					
mean ratio all	*0.388137*					

Table 3.7: Results for *BasicIntegerTSP* used without / with the Lin-Kernighan heuristic for generating an initial solution. *Mean ratios* refer to the arithmetic means over ratios between the running times of the approaches. "sec." is the time in seconds, "#i." the number of iterations and "#c." the number of subtour elimination constraints added to the ILP before starting the last iteration.

3.4 Some theoretical results and further empirical observations

Although our work mainly aims at computational experiments, we also tried to analyze *mainApproach* from a theoretical point of view. In particular we studied the expected behavior on random Euclidean instances and tried to characterize the expected cardinality of the minimal set of required subtours $\mathcal{S}^*$ as defined in Section 3.1. It is well known that no polynomially bounded representation of the TSP polytope can be found and there also exist instances based on a mesh-structure for which $\mathbb{E}[|S^*|]$ has exponential size, but the question for the expected size of $|S^*|$ for random Euclidean instances and thus for the expected number of iterations of our solution algorithm remains an interesting open problem.

We started with extensive computational tests, some of them presented in Figures 3.34 and 3.36, to gain empirical evidence on this aspect.

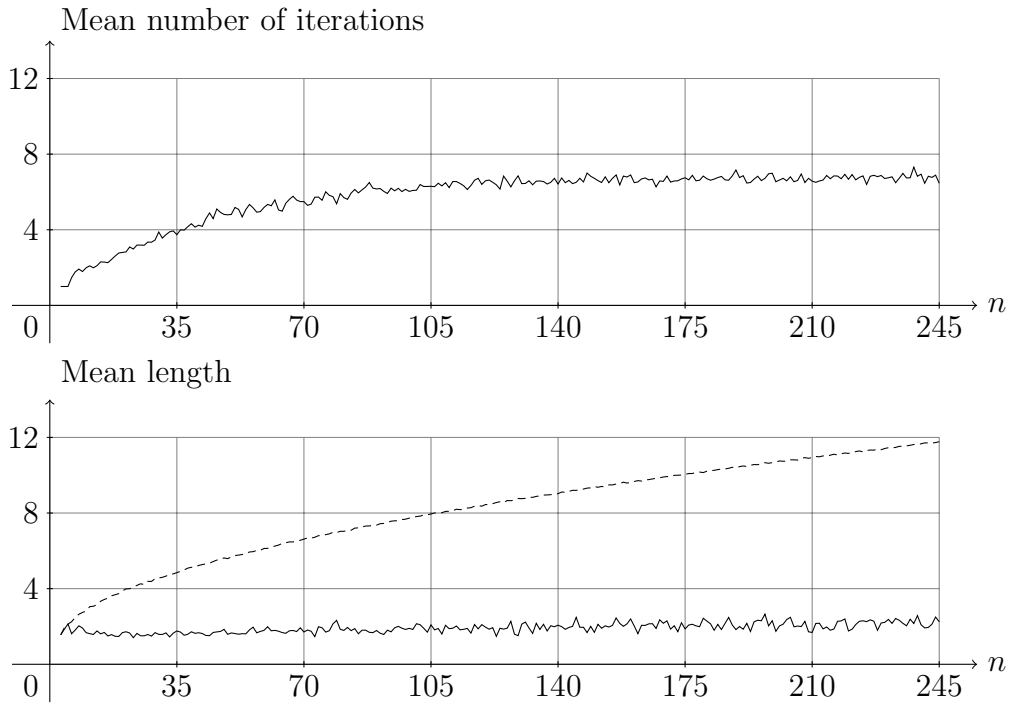

Figure 3.34: Mean number of iterations used by the *mainApproach* (upper figure), mean length of an optimal TSP tour (lower figure, dashed) and mean length of an optimal weighted 2-matching (lower figure, full line) in random Euclidean graphs. For every number of vertices n we created 100 graphs.

The upper graph in Figure 3.34 illustrates the mean number of iterations needed by *mainApproach* to reach optimality for different numbers of vertices n (we evaluated 100 random Euclidean instances for every value n). The lower graph of Figure 3.34 shows the mean length of the optimal TSP tour and of the optimal 2-matching (i.e. the objective value after solving the ILP in the first iteration) by using the same setting.

It was proven back in 1959 that the expected length of an optimal TSP tour is asymptotically $\beta\sqrt{n}$, where β is a constant (see BEARDWOOD et al. [8]). This approach was later generalized for other settings and other properties of the square root asymptotic were identified (RHEE [47], YUKICH [53]. We used these properties to prove the square root asymptotic also for the 2-matching problem (cf Figure 3.34, lower graph, dashed).

We need some definitions, lemmas and theorems originally introduced by RHEE [47] and summarized by YUKICH [53] first in order to prove this result.

Definition 3.1 ((2-)matching functional and boundary (2-)matching functional)**.**

Let $\mathfrak{F} := \mathfrak{F}(dim)$ *denote the* finite subsets of $\mathbb{R}^{dim}$ *and let* $\mathfrak{R} := \mathfrak{R}(dim)$ *denote the dim-*dimensional rectangles of $\mathbb{R}^{dim}$.

Furthermore, let $F \in \mathfrak{F}$ *be a point set in* $\mathbb{R}^{dim}$ *and let* $R \in \mathfrak{R}$ *be a dim-dimensional rectangle in* $\mathbb{R}^{dim}$ *where* $dim \geq 2$.

And finally, let $d\colon R \times R \to \mathbb{R}_0^+$ *be a metric and let* $G = G(F,R) = \big(V(G), E(G)\big)$ *be a complete graph with the vertex set* $V(G) = F \cap R$ *and with the distances* $d(e)$ *between every two vertices* $u, v \in V(G)$ *where* $e = (u,v)$.

Then we will denote

$$M(F,R) := M(F \cap R) := \min_m \{OV(G,m) | m \text{ is a matching in } G\} \quad (3.9)$$

the matching functional.

Furthermore, we will denote

$$\begin{aligned} M_B(F,R) &:= M_B(F \cap R) \\ &:= \min \left\{ M(F,R), \inf_{\substack{(F_i)_{i\geq 1} \in \mathcal{F} \\ \{a_i,b_i\}_{i\geq 1} \in \partial\mathcal{R}}} \left\{ \sum_i M(F_i \cup \{a_i,b_i\}, R) \right\} \right\} \end{aligned} \quad (3.10)$$

the boundary matching functional. $\mathcal{F}$ *stays for the set of all partitions* $(F_i)_{i\geq 1}$ *of* F *and* $\partial\mathcal{R}$ *stays for the set of all sequences of pairs of points* $\{a_i,b_i\}_{i\geq 1}$ *belonging to the* boundary of the rectangle R *denoted by* ∂R. *Additionally, we set* $d(a,b) = 0$ *for all* $a, b \in \partial R$.

Similarly, we define the 2-matching functional L *and the* boundary 2-matching functional L_B*.*

$$L(F,R) \coloneqq L(F \cap R) \coloneqq \min_x \left\{ OV(G,x) | x \text{ is a 2-matching in } G \right\} \quad (3.11)$$

$$\begin{aligned} L_B(F,R) &\coloneqq L_B(F \cap R) \\ &\coloneqq \min \left\{ L(F,R), \inf_{\substack{(F_i)_{i\geq 1} \in \mathcal{F} \\ \{a_i,b_i\}_{i\geq 1} \in \partial\mathcal{R}}} \left\{ \sum_i L(F_i \cup \{a_i,b_i\}, R) \right\} \right\} \end{aligned} \quad (3.12)$$

If it is obvious which rectangle R is considered, we also write $M(F)$ for $M(F,R)$, $M_B(F)$ for $M_B(F,R)$, $L(F)$ for $L(F,R)$ and $L_B(F)$ for $L_B(F,R)$.

Finally, we define $L(F,R) = L_B(F,R) = 0$ if $|F \cap R| < 3$

Definition 3.2 (simple subadditivity, geometric subadditivity and superadditivity)**.** *Let R be a rectangle defined as $[0,t]^{dim}$ for some positive constant $t \in \mathbb{R}^+$ partitioned into two rectangles R_1 and R_1 ($R_1 \cup R_2 = R$). Furthermore, let F, G be finite sets in $[0,t]^{dim}$ and let $P(F,R)\colon \mathfrak{F} \times \mathfrak{R} \to \mathbb{R}_0^+$ be a function.*

The function P is simple subadditive *if the following inequality is satisfied:*

$$P(F \cup G, R) \leq P(F,R) + P(G,R) + C_1 t \quad (3.13)$$

where $C_1 \coloneqq C_1(dim)$ is a finite constant.

If

$$P(F,R) \leq P(F,R_1) + P(F,R_2) + C_2 diam(R) \quad (3.14)$$

is fulfilled, we will call the function P geometric subadditive*. $diam(R)$ denotes the diameter of the rectangle R and $C_2 \coloneqq C_2(dim)$ is a finite constant.*

If

$$P(F,R) \geq P(F,R_1) + P(F,R_2) \quad (3.15)$$

is satisfied, we will call the function P superadditive*.*

Definition 3.3 (subadditive and superadditive Euclidean functional)**.** *Let $P(F,R)\colon \mathfrak{F} \times \mathfrak{R} \to \mathbb{R}_0^+$ be a function satisfying*

$$\forall R \in \mathfrak{R},\ P(\emptyset, R) = 0\ , \quad (3.16)$$

$$\forall y \in \mathbb{R}^{dim},\ R \in \mathfrak{R},\ F \subset R\colon P(F,R) = P(F+y, R+y)\ , \quad (3.17)$$

$$\forall \alpha > 0,\ R \in \mathfrak{R},\ F \subset R\colon\ P(\alpha F, \alpha R) = \alpha P(F, R)\ . \tag{3.18}$$

Then we will say that P is translation invariant *(condition (3.17)) and* homogeneous *(condition (3.18)).*

We will call P an Euclidean functional*.*

Let R be a rectangle defined as $[0,t]^{dim}$ for some positive constant $t \in \mathbb{R}^+$ partitioned into two rectangles R_1 and R_1 ($R_1 \cup R_2 = R$). If P satisfies the geometric subadditivity (3.14), we will say that P is a subadditive Euclidean functional*. If P is supperadditive (3.15), we will say that P is a* superadditive Euclidean functional*.*

Lemma 3.4 (Yukich [53], originally Rhee [47])**.** *The matching functional M and the boundary matching functional M_B are subadditive Euclidean functionals.*

Proof. See Yukich [53]. □

Lemma 3.5 (growth bounds – Yukich [53], originally Rhee [47])**.** *Let P be a subadditive Euclidean functional. Then there exists a finite constant $C_4 := C_4(dim)$ such that for all dim-dimensional rectangles of $\mathbb{R}^{dim}$ and for all $F \subset R$ we have*

$$P(F, R) \leq C_4 diam(R) |F|^{\frac{dim-1}{dim}}\ . \tag{3.19}$$

Proof. See Yukich [53]. □

Definition 3.6 (smoothness)**.** *An Euclidean functional P is* smooth *if there is a finite constant $C_3 := C_3(dim)$ such that for all sets $F, G \in [0,1]^{dim}$ we have*

$$\left|P(F \cup G) - P(F)\right| \leq C_3 |G|^{\frac{d-1}{d}}\ . \tag{3.20}$$

Definition 3.7 (pointwise closeness)**.** *Say that Euclidean functionals P and P_B are* pointwise close *if for all subsets $F \subset [0,1]^{dim}$ we have*

$$\left|P\left(F, [0,1]^{dim}\right) - P_B\left(F, [0,1]^{dim}\right)\right| = o\left(|F|^{\frac{d-1}{d}}\right)\ . \tag{3.21}$$

Definition 3.8 (complete convergence)**.** *A sequence of random variables X_n, $n \geq 1$,* converges completely (c.c.) *to a constant C if and only if for all $\varepsilon > 0$ we have*

$$\sum_{n=1}^{\infty} \mathbb{P}\left[|X_n - C| > \varepsilon\right] < \infty\ . \tag{3.22}$$

Theorem 3.9 (basic limit theorem for Euclidean functionals – YUKICH [53], originally RHEE [47])**.** *Let X_i, $1 \leq i \leq n$, be independent and identically distributed random variables with values in the unit dim-dimensional rectangle $[0,1]^{dim}$, $dim \geq 2$.*

If P_B is a smooth superadditive Euclidean functional on $\mathbb{R}^{dim}$, $dim \geq 2$, then

$$\lim_{n\to\infty} \frac{P_B(X_1, X_2, \ldots, X_n)}{n^{\frac{dim-1}{dim}}} = \alpha(P_B, dim) \quad c.c. \ , \tag{3.23}$$

where $\alpha(P_B, dim)$ is a positive constant.

If P is an Euclidean functional on $\mathbb{R}^{dim}$, $dim \geq 2$, which is pointwise close to P_B, then

$$\lim_{n\to\infty} \frac{P(X_1, X_2, \ldots, X_n)}{n^{\frac{dim-1}{dim}}} = \alpha(P_B, dim) \quad c.c. \tag{3.24}$$

Proof. See YUKICH [53]. □

We can prove our result now.

Lemma 3.10. *The 2-matching functional L and the boundary 2-matching functional L_B fulfill the conditions of Theorem 3.9.*

Proof. The proof is a modification of similar proofs for other combinatorial optimization problems contained in YUKICH [53].

(1) First, we show that the boundary 2-matching functional L_B is a superadditive Euclidean functional. Equalities (3.16), (3.17) and (3.18) are fulfilled obviously.

Let us now show the superadditivity. We can distinguish 2 cases in general:

(a) Either the solution over the whole rectangle R does not cross the boundary between the rectangles R_1 and R_2 at all or

(b) at least one subtour crosses the boundary between the rectangles R_1 and R_2.

$L_B(F, R) = L_B(F, R_1) + L_B(F, R_2)$ obviously holds in the first case.

Let us now consider a subtour crossing the boundary between the rectangles R_1 and R_2 (for an example see Figure 3.35). W.l.o.g. we can assume that the boundary is crossed between the points v_1 and v_2,

and v_3 and v_4 and that the crossing points are x and y respectively. Furthermore, w.l.o.g. we can assume that $v_1, v_3 \in R_1$. Then the new subtour, containing the vertices v_1 and v_3, and lying in the rectangle R_1, consists of the following parts:

- the same path between the vertices v_1 and v_3 belonging to the rectangle R_1 as in the whole rectangle R,
- the orthogonal connections between the vertices v_1 and v_3 and the boundary between the both rectangles, and finally
- a piece of this boundary (see also Figure 3.35).

We have to choose the vertices a and b on this boundary in such a way that $a = \underset{\alpha \in \partial R_1 \cap \partial R_2}{\arg\min} \{d(v_1, \alpha)\}$ and $b = \underset{\beta \in \partial R_1 \cap \partial R_2}{\arg\min} \{d(v_3, \beta)\}$ hold in order to achieve the minimality. Due to this choice of the vertices a and b we can write $d(v_1, a) \leq d(v_1, x)$ and $d(v_3, b) \leq d(v_3, y)$ and since $d(a, b) = 0$ and the remaining part of the subtour belonging to the rectangle R_1 yield the same contribution to the objective value, we can claim that the contribution of this new subtour to the objective value is smaller or equal to the contribution of the part of the original subtour lying in the rectangle R_1. The same argument can be used for the second rectangle R_2 and for all other subtours crossing the boundary between the rectangles R_1 and R_2.

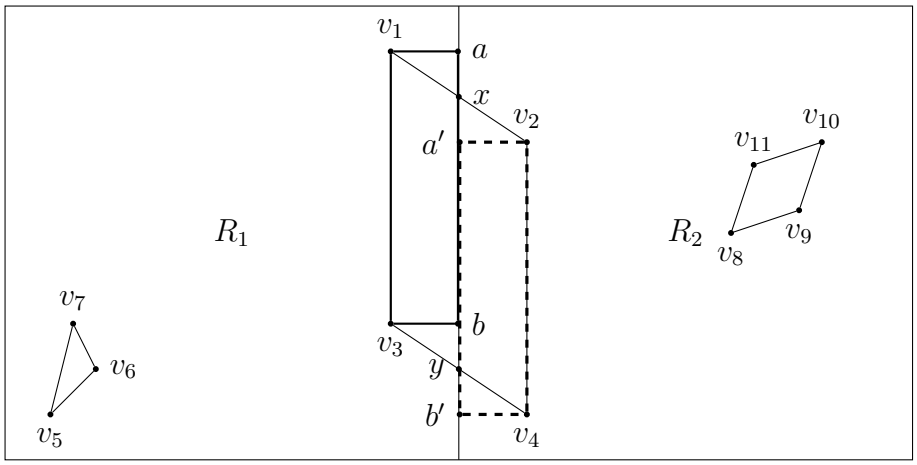

Figure 3.35: Example illustrating the superadditivity of the boundary 2-matching functional L_B.

(2) Next, we check some properties of the 2-matching functional L. Equalities (3.16), (3.17) and (3.18) obviously hold.

Further, it is easy to see that the 2-matching functional L fulfill the geometric subadditivity. Since we minimize, the minimum weighted 2-matching over the whole rectangle R can have only a smaller objective value than the sum of the objective values for the rectangles R_1 and R_2 taken separately.

As a next step, we have to prove the pointwise closeness of the 2-matching functional L to the boundary 2-matching functional L_B.

First, note that $L_B\left(F, [0,1]^{dim}\right) \leq L\left(F, [0,1]^{dim}\right)$ always hold (see (3.12)). Thus it suffices to show

$$L\left(F, [0,1]^{dim}\right) \leq L_B\left(F, [0,1]^{dim}\right) + C_7|F|^{\frac{d-1}{d}} \tag{3.25}$$

where $C_7 := C_7(dim)$ is a finite constant.

Let $F^* \subseteq F$ be the set of vertices which are connected with the boundary $\partial[0,1]^{dim}$ by a path. Now, we remove all edges incident with the vertices contained in the vertex set F^*. If $|F^*| < 3$, we can just put these vertices to an arbitrary subtour (if such a subtour exists) and get the above inequality (the increase of the objective value can be easily bounded by $4\sqrt{\text{dim}}$ in this case). If $|F^*| \geq 3$, we can construct a minimum weighted 2-matching on this vertex set and obtain

$$L\left(F, [0,1]^{dim}\right) \leq L_B\left(F, [0,1]^{dim}\right) + L\left(F^*, [0,1]^{dim}\right) \ . \tag{3.26}$$

And since $|F^*| \leq |F|$, we can use Lemma 3.5 and get inequality (3.25).

(3) Finally, we prove the smoothness of the boundary 2-matching functional L_B. We will show the simple and geometric subadditivity of this functional first in order to be ably to show the smoothness.

Let F and G be finite sets in $[0,t]^{dim}$. If the minimum weighted 2-matching for the vertex set $F \cup G$ equals to the minimum weighted 2-matchings for the vertex sets F and G joined together, inequality (3.13) holds with equality. Since we can always construct such a solution for the vertex set $F \cup G$, the objective value can be only smaller in the other case (note that we minimize it).

Let us now prove the geometric subadditivity in order to fulfill the conditions of Lemma 3.5. We know that the 2-matching functional L is geometric subadditive. Now, it is easy to see that the proof of inequality (3.25) can be easily modified in order to obtain

$$L(F,R) \leq L_B(F,R) + C_7 diam(R)|F|^{\frac{d-1}{d}} \tag{3.27}$$

for an arbitrary *dim*-dimensional rectangle. Since $L_B(F,R) \leq L(F,R)$, we obtain inequality (3.14) immediately.

Using the simple subadditivity and Lemma 3.5 we get for all finite sets $F, G \subset [0,1]^{dim}$

$$\begin{aligned} L_B(F \cup G) &\leq L_B(F) + \left(C_1 + C_4\sqrt{dim}\right)|G|^{\frac{dim-1}{dim}} \\ &\leq L_B(F) + C_5|G|^{\frac{dim-1}{dim}} \end{aligned} \tag{3.28}$$

where $C_5 := C_5(dim)$ denotes a finite constant. This completes this part of the proof if $L_B(F \cup G) - L_B(F) \geq 0$. Hence we just need to show the following inequality

$$L_B(F \cup G) \geq L_B(F) - C_6|G|^{\frac{dim-1}{dim}} \tag{3.29}$$

for some finite constant $C_6 := C_6(dim)$.

Consider the global minimum weighted 2-matching on the vertex set $G \cup F$ and remove all edges from all subtours incident with a vertex $g \in G$. This yield at most $|G|$ paths of a length of at least 1 containing only vertices from the vertex set F and some isolated points $F' \subseteq F$. Let F^* denote the set of all endpoints of those paths. Clearly, we have $|F'| \leq |G|$ and $|F^*| \leq 2|G|$. Consider now the boundary matching functional $M_B(F^*)$ and the corresponding matching m. This matching together with the disconnected paths and with parts of the boundary of the *dim*-dimensional rectangle $[0,1]^{dim}$ yield a set of subtours $\{\tilde{F}_i\}_{i=1}^N$ for some particular positive integer N. Furthermore, we can construct an minimum weighted 2-matching on the vertex set F' and get a feasible minimum weighted 2-matching. We can write

$$L_B(F) \leq L_B(F \cup G) + M_B(F^*) + L_B(F') \ . \tag{3.30}$$

By using Lemmas 3.4 and 3.5 we obtain

$$L_B(F) \leq L_B(F \cup G) + C_6\left(|F^*|^{\frac{dim-1}{dim}} + |F'|^{\frac{dim-1}{dim}}\right) \ . \tag{3.31}$$

And since $|F'| \leq |G| \leq 2|G|$ and $|F^*| \leq 2|G|$, we get

$$\begin{aligned} L_B(F) &\leq L_B(F \cup G) + C_6\left((2|G|)^{\frac{dim-1}{dim}} + 2\,(|G|)^{\frac{dim-1}{dim}}\right) \\ &\leq L_B(F \cup G) + C_6\,(|G|)^{\frac{dim-1}{dim}} \ . \end{aligned} \tag{3.32}$$

This is exactly inequality (3.29).

□

Theorem 3.11. *Let $G = (V, E)$ be a random Euclidean graph with $n = |V|$ vertices and let $d\colon E \to \mathbb{R}_0^+$ be the Euclidean distance function. Furthermore, let $M_2(G, d)$ be the length of an optimal 2-matching. Then*

$$\lim_{n\to\infty} \frac{M_2(G,d)}{\sqrt{n}} = \alpha \quad c.c., \text{ where } \alpha > 0. \tag{3.33}$$

Proof. The theorem immediately follows from Theorem 3.9 and Lemma 3.10. □

Based on these results the following idea might lead to a proof that the expected cardinality $\mathcal{S}^*$ is polynomially bounded: After the first iteration of the algorithm we have a solution possibly consisting of several separate subtours of total asymptotic length $\alpha\sqrt{n} = \alpha_1\sqrt{n}$. If there are subtours, we add subtour elimination constraints (in fact at most $\lfloor \frac{n}{3} \rfloor$), resolve the enlarged ILP and get another solution whose asymptotic length is $\alpha_2\sqrt{n}$. By proving that the expected length of the sequence $\alpha = \alpha_1, \ldots, \alpha_{\#i} = \beta$ is polynomially bounded in n, one would obtain that also $\mathbb{E}\,[|S^*|]$ is polynomially bounded since only polynomially many subtours are added in each iteration. Our intuition and computational tests illustrated in Figure 3.34, upper graph, indicate that the length of this sequence could be proportional to $\sqrt{n}$ as well. Unfortunately, we could not find the suitable techniques to show this step.

A different approach is illustrated by Figure 3.36, where we examine the mean number of subtours contained in every iteration. In particular, we chose $n = 60$, generated 100000 random Euclidean instances and sorted them by the number of iterations #i. required by *mainApproach*. The most frequent number of ILP solver runs was 7 (dotted line), but we summarize the results for 5 (full line), 6 (dashed), 8 (loosely dashed) and 9 (loosely dotted) necessary runs in this figure as well. For every iteration of every class (with respect to the number of involved ILP runs) we compute the mean number of subtours contained in the respective solutions. As can be expected these numbers of subtours are decreasing (in average) over the number of iterations. To allow a better comparison of this behavior for different numbers of iterations we scaled the iteration numbers into the interval $[0, 10]$ (horizontal axis of Figure 3.36). It can be seen that the average number of subtours contained in an optimal 2-matching (first iteration) is about 9.2 while in the last iteration we trivially have only one tour. Between these endpoints we

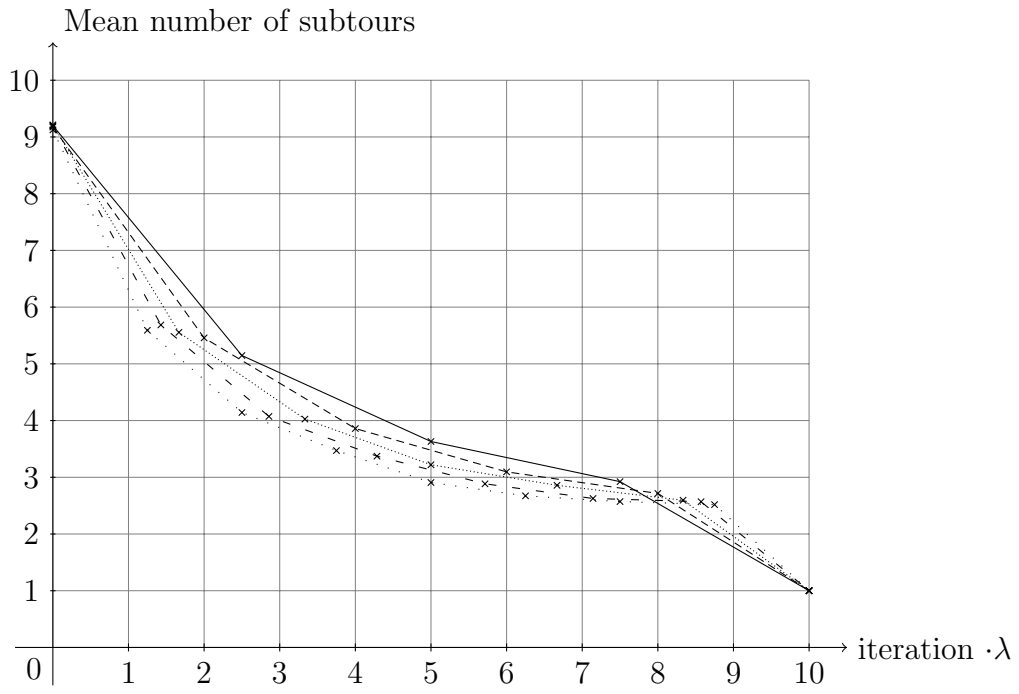

Figure 3.36: Mean number of subtours during the *mainApproach* in random Euclidean graphs for $n = 60$ sorted according to the number of iterations ($\lambda = 4/10$ (full line), 5/10 (dashed), 6/10 (dotted), 7/10 (loosely dashed), 8/10 (loosely dotted)). We created 100000 graphs.

can first observe a mostly convex behavior, only in the last step before reaching the optimal TSP tour a sudden drop occurs. It would be interesting to derive an asymptotic description of these curves. An intuitive guess would point to an exponential function, but so far we could not find a theoretical justification of this claim.

3.A Appendix

instance	BasicIntegerTSP			$HC \mid n$			$HC \mid 4n/\log_2 n$			$HCD \mid 4n/\log_2 n$		
	sec.	#i.	#c.	sec.	#i.	#c.	sec.	#i.	#c.	sec.	#i.	#c.
ch150	13	7	74	150	2	435	**9**	**5**	**223**	14	6	129
kroA150	19	7	136	11	2	268	**8**	**2**	**245**	11	4	130
kroB150	179	8	148	72	2	301	34	4	315	**21**	**4**	**168**
pr152	16	13	184	**5**	**3**	**214**	**5**	**3**	**205**	9	4	174
u159	**6**	**4**	**49**	29	3	292	14	4	303	11	4	140
si175	52	10	183	99	6	494	**40**	**8**	**415**	44	7	263
brg180	44	4	103	54	2	185	102	18	359	**24**	**2**	**27**
rat195	347	6	274	**241**	**3**	**491**	272	4	419	267	5	322
d198	10986	10	301	**483**	**7**	**894**	1094	10	582	3986	9	326
kroA200	677	8	237	**177**	**2**	**362**	941	3	353	690	5	238
kroB200	31	5	121	37	3	292	**23**	**3**	**269**	31	4	164
gr202	**39**	**11**	**77**	2430	3	1216	61	8	330	60	8	217
tsp225	178	9	261	1113	3	981	**138**	**6**	**551**	151	6	341
pr226	5183	10	409	18	1	593	**13**	**1**	**585**	59	3	357
gr229	239	6	311	2984	4	1056	**172**	**7**	**490**	173	8	324
gil262	179	7	268	1052	2	807	**169**	**3**	**564**	217	4	368
a280	157	11	143	–	–	–	**124**	**3**	**733**	181	7	352
pr299	9263	9	413	4051	2	782	1998	5	745	**1716**	**5**	**455**
lin318	6885	8	357	457	2	756	**274**	**5**	**660**	275	5	355
rd400	2401	9	467	9329	5	1494	**983**	**6**	**1018**	1579	8	539
gr431	**2239**	**9**	**453**	–	–	–	4748	9	1734	4214	10	833
pcb442	2737	11	501	–	–	–	3830	16	1796	**2277**	**15**	**888**
u574	17354	6	423	–	–	–	18050	4	1290	**8664**	**4**	**629**
gr666	**17711**	**8**	**789**	–	–	–	23212	4	3104	18031	7	1408
rat783	**30156**	**6**	**457**	–	–	–	–	–	–	–	–	–
mean ratio				*6.016088*			*0.890955*			*0.804727*		
RE_A_150	**23**	**8**	**100**	103	5	363	36	5	238	28	6	162
RE_B_150	**13**	**7**	**78**	99	1	424	**13**	**3**	**255**	14	4	146
RE_C_150	9	5	70	21	1	235	**7**	**3**	**195**	**7**	**3**	**98**
RE_D_150	8	6	60	50	2	274	**7**	**4**	**197**	**7**	**4**	**112**
RE_E_150	**9**	**7**	**55**	76	1	339	22	4	274	17	5	149
mean RE_150	***12.4***	***6.6***	***72.6***	*69.8*	*2*	*327*	*17*	*3.8*	*231.8*	*14.6*	*4.4*	*133.4*
RE_A_200	**72**	**7**	**163**	560	3	613	77	4	376	157	6	304
RE_B_200	125	7	148	452	2	582	**76**	**6**	**348**	205	5	250
RE_C_200	84	8	178	107	1	373	**35**	**4**	**341**	43	4	220

instance	BasicIntegerTSP			$HC \mid n$			$HC \mid 4n/\log_2 n$			$HCD \mid 4n/\log_2 n$		
	sec.	#i.	#c.	sec.	#i.	#c.	sec.	#i.	#c.	sec.	#i.	#c.
RE_D_200	**29**	**5**	**102**	173	2	425	40	3	336	44	5	190
RE_E_200	65	9	139	411	2	561	29	4	301	**21**	**2**	**151**
mean RE_200	*75*	*7.2*	*146*	*340.6*	*2*	*510.8*	***51.4***	***4.2***	***340.4***	*94*	*4.4*	*223*
RE_A_250	**138**	**9**	**186**	1154	4	923	158	6	540	163	7	334
RE_B_250	642	7	263	689	3	599	**198**	**5**	**497**	533	5	359
RE_C_250	156	6	219	1545	1	846	**57**	**3**	**333**	136	4	275
RE_D_250	273	6	220	501	2	542	**192**	**6**	**479**	199	5	292
RE_E_250	103	5	156	339	1	675	**70**	**4**	**511**	105	4	252
mean RE_250	*262.4*	*6.6*	*208.8*	*845.6*	*2.2*	*717*	***135***	***4.8***	***472***	*227.2*	*5*	*302.4*
RE_A_300	866	6	295	3574	2	1142	575	5	648	**460**	**4**	**357**
RE_B_300	1411	8	348	4297	3	1059	**627**	**4**	**672**	865	6	402
RE_C_300	1071	8	339	2071	3	848	**236**	**6**	**567**	687	7	474
RE_D_300	**229**	**6**	**290**	2419	4	962	320	5	544	339	5	416
RE_E_300	577	7	272	1543	3	726	**283**	**6**	**526**	436	6	344
mean RE_300	*830.8*	*7*	*308.8*	*2780.8*	*3*	*947.4*	***408.2***	***5.2***	***591.4***	*557.4*	*5.6*	*398.6*
RE_A_350	411	5	252	3186	2	904	332	3	657	**286**	**4**	**377**
RE_B_350	1021	8	339	11818	2	1102	1234	7	691	**985**	**5**	**463**
RE_C_350	248	6	207	1243	2	936	**232**	**4**	**750**	358	5	390
RE_D_350	1718	9	412	4271	3	1087	**529**	**3**	**691**	957	5	428
RE_E_350	556	5	261	4560	3	1208	485	4	695	**323**	**4**	**408**
mean RE_350	*790.8*	*6.6*	*294.2*	*5015.6*	*2.4*	*1047.4*	***562.4***	***4.2***	***696.8***	*581.8*	*4.6*	*413.2*
RE_A_400	8456	8	454	82054	3	1328	10463	5	980	**8245**	**5**	**594**
RE_B_400	88849	7	438	1043519	2	1661	57469	5	879	**39759**	**6**	**589**
RE_C_400	780	6	312	53580	3	1755	**779**	**6**	**858**	875	4	450
RE_D_400	2052	8	451	122357	2	1998	1546	4	796	**1081**	**4**	**434**
RE_E_400	2847	7	332	222902	2	1409	2450	4	660	**2151**	**4**	**515**
mean RE_400	*20596.8*	*7.2*	*397.4*	*304882.4*	*2.4*	*1630.2*	*14541.4*	*4.8*	*834.6*	***10422.2***	***4.6***	***516.4***
RE_A_450	**2107**	**5**	**279**	–	–	–	2947	3	872	3595	4	535
RE_B_450	68338	8	413	–	–	–	**57921**	**8**	**906**	94135	6	575
RE_C_450	46360	10	596	–	–	–	33505	6	1166	**13425**	**7**	**723**
RE_D_450	**1212**	**6**	**368**	–	–	–	1718	3	902	1644	4	520
RE_E_450	1539	8	391	–	–	–	2438	5	1098	**1345**	**7**	**637**
mean RE_450	*23911.2*	*7.4*	*409.4*	–	–	–	***19705.8***	***5***	***988.8***	*22828.8*	*5.6*	*598*
RE_A_500	15330	6	436	–	–	–	**7531**	**4**	**1000**	10323	5	629
RE_B_500	**16883**	**6**	**352**	–	–	–	33464	4	1558	79362	5	727

instance	BasicIntegerTSP			$HC \mid n$			$HC \mid 4n/\log_2 n$			$HCD \mid 4n/\log_2 n$		
	sec.	#i.	#c.	sec.	#i.	#c.	sec.	#i.	#c.	sec.	#i.	#c.
RE_C_500	3724	5	428	–	–	–	**1337**	**4**	**955**	1437	5	585
RE_D_500	322951	9	567	–	–	–	339694	7	1236	**307921**	**6**	**743**
RE_E_500	243378	9	679	–	–	–	**110212**	**4**	**1113**	134563	9	889
mean RE_500	*120453.2*	*7*	*492.4*	–	–	–	***98447.6***	***4.6***	***1172.4***	*106721.2*	*6*	*714.6*
mean ratio				*12.132531*			*0.898420*			*1.040018*		
mean ratio all				*9.760849*			*0.895621*			*0.951784*		

Table 3.8: Results for *BasicIntegerTSP* and for different variants of the approach which uses the hierarchical clustering. *Mean ratios* refer to the arithmetic means over ratios between the running times of the particular approaches and the running time of the *BasicIntegerTSP*. "sec." is the time in seconds, "#i." the number of iterations and "#c." the number of subtour elimination constraints added to the ILP before starting the last iteration. The entries "–" by TSPLIB instances cannot be computed with 16 GB RAM or would take more than 12 hours.

- **BasicIntegerTSP**
- $\boldsymbol{HC \mid n}$ – hierarchical clustering; the constraints cannot be dropped and the maximum size of a solved cluster is $u = n$ (i.e. in fact, there is no upper bound)
- $\boldsymbol{HC \mid 4n/\log_2 n}$ – hierarchical clustering; the constraints cannot be dropped and the maximum size of a solved cluster is $u = 4\frac{n}{\log_2 n}$
- $\boldsymbol{HCD \mid 4n/\log_2 n}$ – hierarchical clustering; the constraints can be dropped and the maximum size of a solved cluster is $u = 4\frac{n}{\log_2 n}$

instance	BasicIntegerTSP			$C \mid \lfloor n/5 \rfloor$			$RC_3 \mid \lfloor n/5 \rfloor$			$RC_3 \mid n$			$HCD \mid 4n/\log_2 n$		
	sec.	#i.	#c.	sec.	#i.	#c.	sec.	#i.	#c.	sec.	#i.	#c.	sec.	#i.	#c.
ch150	13	7	74	12	6	114	**9**	**6**	**109**	16	7	117	14	6	129
kroA150	19	7	136	25	6	187	43	6	166	33	5	185	**11**	**4**	**130**
kroB150	179	8	148	53	4	219	138	5	215	44	5	202	**21**	**4**	**168**
pr152	16	13	184	17	12	181	17	11	204	18	12	181	**9**	**4**	**174**
u159	6	4	49	6	4	149	5	5	151	**3**	**3**	**70**	11	4	140
si175	52	10	183	**31**	**10**	**213**	55	13	250	35	9	196	44	7	263
brg180	44	4	103	**17**	**3**	**81**	19	8	102	121	11	316	24	2	27
rat195	347	6	274	246	4	268	275	6	315	**114**	**6**	**257**	267	5	322
d198	10986	10	301	4253	11	315	–	–	–	4762	9	321	**3986**	**9**	**326**
kroA200	677	8	237	332	6	214	350	5	190	**287**	**4**	**171**	690	5	238
kroB200	31	5	121	29	5	148	**21**	**5**	**147**	32	4	123	31	4	164
gr202	39	11	77	50	8	233	36	6	174	**25**	**6**	**143**	60	8	217
tsp225	178	9	261	100	9	223	**84**	**10**	**235**	100	8	300	151	6	341
pr226	5183	10	409	3614	6	363	36744	5	403	12944	9	415	**59**	**3**	**357**
gr229	239	6	311	335	6	289	**152**	**6**	**256**	311	7	341	173	8	324
gil262	179	7	268	250	8	250	**133**	**7**	**268**	152	6	274	217	4	368
a280	157	11	143	**61**	**4**	**299**	196	11	350	117	9	221	181	7	352
pr299	9263	9	413	6376	7	387	7410	6	416	16059	6	414	**1716**	**5**	**455**
lin318	6885	8	357	537	7	331	386	6	364	1560	6	391	**275**	**5**	**355**
rd400	2401	9	467	**1212**	**7**	**420**	1827	7	438	1522	8	398	1579	8	539
gr431	**2239**	**9**	**453**	3098	9	626	3384	9	647	2496	10	704	4214	10	833
pcb442	2737	11	501	3868	16	770	**1815**	**17**	**567**	2626	16	594	2277	15	888
u574	17354	6	423	11702	4	498	35204	5	580	13722	5	572	**8664**	**4**	**629**
gr666	17711	8	789	**11756**	**7**	**919**	14223	7	1001	13573	7	1002	18031	7	1408
rat783	**30156**	**6**	**457**	184381	5	701	37805	5	735	38630	6	779	–	–	–
mean ratio				*1.014009*			*1.170299*			*0.983280*			*0.804727*		
RE_A_150	23	8	100	**18**	**5**	**142**	26	6	141	36	7	155	28	6	162
RE_B_150	13	7	78	8	4	117	13	5	129	**7**	**4**	**86**	14	4	146
RE_C_150	9	5	70	**6**	**4**	**63**	8	4	111	9	5	89	7	3	98
RE_D_150	8	6	60	7	4	100	8	5	97	**6**	**4**	**78**	7	4	112
RE_E_150	9	7	55	9	6	103	**8**	**4**	**103**	10	4	114	17	5	149
mean RE_150	*12.4*	*6.6*	*72.6*	***9.6***	***4.6***	***105***	*12.6*	*4.8*	*116.2*	*13.6*	*4.8*	*104.4*	*14.6*	*4.4*	*133.4*
RE_A_200	72	7	163	**54**	**7**	**218**	69	6	237	81	6	223	157	6	304
RE_B_200	125	7	148	97	6	217	64	6	213	**58**	**5**	**199**	205	5	250
RE_C_200	84	8	178	39	6	181	**30**	**6**	**140**	54	7	159	43	4	220

instance	BasicIntegerTSP			$C \mid \lfloor n/5 \rfloor$			$RC_3 \mid \lfloor n/5 \rfloor$			$RC_3 \mid n$			$HCD \mid 4n/\log_2 n$		
	sec.	#i.	#c.	sec.	#i.	#c.	sec.	#i.	#c.	sec.	#i.	#c.	sec.	#i.	#c.
RE_D_200	**29**	**5**	**102**	52	4	204	34	3	194	36	4	147	44	5	190
RE_E_200	65	9	139	36	5	217	26	6	144	54	6	193	**21**	**2**	**151**
mean RE_200	*75*	*7.2*	*146*	*55.6*	*5.6*	*207.4*	***44.6***	***5.4***	***185.6***	*56.6*	*5.6*	*184.2*	*94*	*4.4*	*223*
RE_A_250	138	9	186	160	8	258	338	9	287	**119**	**7**	**242**	163	7	334
RE_B_250	642	7	263	**306**	**6**	**295**	542	5	313	366	6	259	533	5	359
RE_C_250	156	6	219	**104**	**4**	**175**	110	6	229	135	5	211	136	4	275
RE_D_250	273	6	220	**186**	**6**	**262**	377	6	316	403	7	293	199	5	292
RE_E_250	103	5	156	**66**	**5**	**207**	68	4	271	110	5	239	105	4	252
mean RE_250	*262.4*	*6.6*	*208.8*	***164.4***	***5.8***	***239.4***	*287*	*6*	*283.2*	*226.6*	*6*	*248.8*	*227.2*	*5*	*302.4*
RE_A_300	866	6	295	1233	7	343	576	6	324	467	5	311	**460**	**4**	**357**
RE_B_300	1411	8	348	1139	6	391	1100	7	431	1146	7	372	**865**	**6**	**402**
RE_C_300	1071	8	339	608	6	392	**331**	**6**	**314**	458	6	312	687	7	474
RE_D_300	**229**	**6**	**290**	276	6	321	268	7	307	396	7	374	339	5	416
RE_E_300	577	7	272	**353**	**7**	**322**	**353**	**5**	**320**	464	6	334	436	6	344
mean RE_300	*830.8*	*7*	*308.8*	*721.8*	*6.4*	*353.8*	*525.6*	*6.2*	*339.2*	*586.2*	*6.2*	*340.6*	*557.4*	*5.6*	*398.6*
RE_A_350	411	5	252	695	5	275	513	5	277	375	4	268	**286**	**4**	**377**
RE_B_350	1021	8	339	**793**	**7**	**363**	1027	8	362	900	7	353	985	5	463
RE_C_350	248	6	207	**196**	**5**	**232**	326	7	280	296	5	310	358	5	390
RE_D_350	1718	9	412	**749**	**8**	**385**	1047	5	381	781	6	428	957	5	428
RE_E_350	556	5	261	471	6	356	364	4	368	352	4	339	**323**	**4**	**408**
mean RE_350	*790.8*	*6.6*	*294.2*	*580.8*	*6.2*	*322.2*	*655.4*	*5.8*	*333.6*	***540.8***	***5.2***	***339.6***	*581.8*	*4.6*	*413.2*
RE_A_400	8456	8	454	16648	6	471	24941	7	489	28803	7	516	**8245**	**5**	**594**
RE_B_400	88849	7	438	72010	7	496	77325	7	503	59499	7	497	**39759**	**6**	**589**
RE_C_400	**780**	**6**	**312**	1198	6	430	831	5	406	1095	7	453	875	4	450
RE_D_400	2052	8	451	1639	5	436	**591**	**5**	**454**	1595	6	452	1081	4	434
RE_E_400	2847	7	332	**1602**	**6**	**434**	3724	5	390	1608	6	408	2151	4	515
mean RE_400	*20596.8*	*7.2*	*397.4*	*18619.4*	*6*	*453.4*	*21482.4*	*5.8*	*448.4*	*18520*	*6.6*	*465.2*	***10422.2***	***4.6***	***516.4***
RE_A_450	2107	5	279	2921	4	333	2915	5	456	**1385**	**4**	**383**	3595	4	535
RE_B_450	68338	8	413	15587	7	439	**12828**	**5**	**442**	24941	8	494	94135	6	575
RE_C_450	46360	10	596	38930	9	697	35388	6	632	22898	7	647	**13425**	**7**	**723**
RE_D_450	1212	6	368	**948**	**6**	**460**	2175	6	429	2120	7	388	1644	4	520
RE_E_450	1539	8	391	2210	8	480	1786	7	434	1901	7	432	**1345**	**7**	**637**
mean RE_450	*23911.2*	*7.4*	*409.4*	*12119.2*	*6.8*	*481.8*	*11018.4*	*5.8*	*478.6*	***10649***	***6.6***	***468.8***	*22828.8*	*5.6*	*598*
RE_A_500	15330	6	436	10907	6	576	14786	5	543	**6118**	**6**	**531**	10323	5	629
RE_B_500	16883	6	352	**12299**	**5**	**453**	19681	4	483	186708	5	535	79362	5	727

instance	BasicIntegerTSP			$C \mid \lfloor n/5 \rfloor$			$RC_3 \mid \lfloor n/5 \rfloor$			$RC_3 \mid n$			$HCD \mid 4n/\log_2 n$		
	sec.	#i.	#c.	sec.	#i.	#c.	sec.	#i.	#c.	sec.	#i.	#c.	sec.	#i.	#c.
RE_C_500	3724	5	428	3063	6	519	2643	4	471	2339	5	440	**1437**	**5**	**585**
RE_D_500	322951	9	567	514403	8	701	**231961**	**6**	**618**	314232	9	684	307921	6	743
RE_E_500	243378	9	679	167194	8	718	125303	9	685	**82051**	**9**	**671**	134563	9	889
mean RE_500	*120453.2*	*7*	*492.4*	*141573.2*	*6.6*	*593.4*	***78874.8***	***5.6***	***560***	*118289.6*	*6.8*	*572.2*	*106721.2*	*6*	*714.6*
mean ratio				*0.898743*			*0.964008*			*1.180427*			*1.040018*		
mean ratio all				*0.943076*			*1.041367*			*1.104601*			*0.951784*		

Table 3.9: Comparison between different variants of our approach. *Mean ratios* refer to the arithmetic means over ratios between the running times of the particular approaches and the running time of the *BasicIntegerTSP*. "sec." is the time in seconds, "#i." the number of iterations and "#c." the number of subtour elimination constraints added to the ILP before starting the last iteration. The entries "–" by TSPLIB instances cannot be computed with 16 GB RAM.

- **BasicIntegerTSP**
- $\boldsymbol{C \mid \lfloor n/5 \rfloor}$ – clustering for $c = \lfloor \frac{n}{5} \rfloor$
- $\boldsymbol{RC_3 \mid \lfloor n/5 \rfloor}$ – restricted clustering for $c = \lfloor \frac{n}{5} \rfloor$; the minimum size of a cluster is 3
- $\boldsymbol{RC_3 \mid n}$ – restricted clustering for $c = n$; the minimum size of a cluster is 3
- $\boldsymbol{HCD \mid 4n/\log_2 n}$ – hierarchical clustering; the constraints can be dropped and the maximum size of a solved cluster is $u = 4\frac{n}{\log_2 n}$

4. Minimization and maximization versions of the quadratic traveling salesman problem[‡]

4.1 Introduction

In Chapter 3 we focused the *traveling salesman problem* (*TSP*), which asks for a shortest tour through all vertices of a graph with respect to the distances of the edges. In this chapter we consider an extension of the TSP concerning its cost structure. While we are still looking for a *tour*, we do not simply sum up the distances of the edges of the tour in the objective function, but we consider the transition in each vertex, i.e. for each vertex i we take a cost coefficient depending both on the predecessor and on the successor of i on the tour into account. Thus, we can model transition costs such as the effort of turning in path planning (see Aggarwal et al. [3]), changing the equipment in scheduling or the transportation means in logistic networks from one edge to another (see Amaldi et al. [4]).

Mathematically, this can be modeled via a quadratic objective function. The resulting optimization problem is known as *symmetric quadratic traveling salesman problem* (*SQTSP*). Due to its quadratic cost structure, it is in general computationally much more difficult than the classical (linear) TSP.

4.1.1 Formal problem definition and related literature

In the SQTSP we associate costs with every pair of adjacent edges, represented by the corresponding triple of vertices. So, using an edge (i, j) followed

[‡]Joint work with Oswin Aichholzer, Anja Fischer, Johannes Fabian Meier, Ulrich Pferschy and Alexander Pilz.

by edge (j,k) in the tours incurs a certain cost value which is assigned to the ordered triple denoted by $\langle i,j,k\rangle$. Since we deal with the symmetric case, the direction of traversal of the tour is irrelevant and so the costs for $\langle i,j,k\rangle$ and $\langle k,j,i\rangle$ are identical. This allows us to half the number of objects considered and so of variables below.

Our notation follows FISCHER and HELMBERG [21]. Let $V = \{1,2,\ldots,n\}$ be a vertex set. An **edge** $e := (i,j) \in V^{\{2\}} := \big\{(i,j) = (j,i) \colon i,j \in V, i \neq j\big\}$ consists of an undirected pair of vertices and a **2-edge** $e^{\langle 3\rangle} := \langle i,j,k\rangle \in V^{\langle 3\rangle} := \big\{\langle i,j,k\rangle = \langle k,j,i\rangle \colon i,j,k \in V, |\{i,j,k\}| = 3\big\}$ is defined as a sequence of three distinct vertices where the reverse sequence is regarded as identical. If there is no danger of confusion, we simply write ij instead of (i,j) and ijk instead of $\langle i,j,k\rangle$. Furthermore, we define a **2-graph** $G = (V,A)$ as a pair of a vertex set and a set of 2-edges $A \subseteq V^{\langle 3\rangle}$. A 2-graph is called **complete** if $A = V^{\langle 3\rangle}$. Finally, a **tour** $T = \big(\sigma(1),\sigma(2),\ldots,\sigma(n)\big)$ is a permutation σ of the vertices 1, 2, …, n.

Given a complete 2-graph $G = (V, V^{\langle 3\rangle})$ with $n \geq 3$ and non-negative 2-edge costs $c_{e^{\langle 3\rangle}} \in \mathbb{R}_0^+$ for every 2-edge $e^{\langle 3\rangle} \in V^{\langle 3\rangle}$, the **symmetric quadratic traveling salesman problem** (**SQTSP**) asks for a shortest tour with respect to the sum of the corresponding 2-costs $c_{e^{\langle 3\rangle}}$ of the 2-edges contained in the tour, i. e. with respect to the objective value

$$OV(G,T) := \left(\sum_{i=1}^{n-2} c_{\sigma(i)\sigma(i+1)\sigma(i+2)}\right) + c_{\sigma(n-1)\sigma(n)\sigma(1)} + c_{\sigma(n)\sigma(1)\sigma(2)}. \quad (4.1)$$

In this chapter we will also consider the maximization version of that problem, which has not been studied in the literature before. Formally, we introduce the **maximum symmetric quadratic traveling salesman problem** (**MaxSQTSP**) which asks for the *longest* tour T w.r.t. (4.1). For the complexity of MaxSQTSP one can easily derive the following result:

Theorem 4.1. *MaxSQTSP is Max-$\mathcal{SNP}$-hard.*

Proof. We know that the *maximum symmetric traveling salesman problem* (MaxTSP) with distances d_{ij} is Max-$\mathcal{SNP}$-hard (see GUTIN and PUNNEN [25, ch. 12]). We can now define 2-edge costs $c_{ijk} := d_{jk}$ for all $ijk \in V^{\langle 3\rangle}, i < k$. Solving the resulting MaxSQTSP instance and the original MaxTSP instance is obviously equivalent. □

The quadratic TSP was first introduced by JÄGER and MOLITOR [28] in its more general asymmetric version with a motivation from biology (see also FISCHER et al. [19]). The authors provide seven different heuristics and two

exact solution approaches for the asymmetric version. Polyhedral studies on the SQTSP were done by Fischer and Helmberg [21] who also derived several classes of strengthened subtour elimination constraints and proved that many of them are facet defining. They also provide computational comparisons based on an ILP linearization and on the standard separation approach known from the TSP literature (see e.g. Reinelt [46]).

An important application of the SQTSP arises in robotics. Aggarwal et al. [3] discussed the situation of a robot where changing the driving directions is more energy consuming for larger turning angles. Thus one would prefer a tour which keeps the movement of the robot as closely as possible to a straight line. Formally, the following **angular-metric traveling salesman problem** (**AngleTSP**) was introduced in the same paper [3]. We assume in this case that the vertices correspond to points in the Euclidean plane and that the costs c_{ijk} are given by the **turning angles** α_{ijk} defined as

$$c_{ijk} = \alpha_{ijk} := \arccos_{[0,\pi]} \left(\frac{j-i}{\|j-i\|} \cdot \frac{k-j}{\|k-j\|} \right) \tag{4.2}$$

where the dot $\cdot$ denotes the scalar product (for illustration see Figure 4.1). In slight abuse of notation we will sometimes also use the notation $\langle i, j, i \rangle, i, j \in V, i \neq j$, for going from i to j and immediately back to i. Then (4.2) gives $\alpha_{iji} = \arccos(-1) = \pi$.

The same authors proved the $\mathcal{NP}$-hardness of the AngleTSP and provided a polynomial time approximation algorithm guaranteeing an approximation ratio within $O(\log n)$ in [3].

In analogy to MaxSQTSP we will also consider the **maximum angular-metric traveling salesman problem** (**MaxAngleTSP**) where the sum of the turning angles should be *large*, i.e. the sum of the angles contained between two adjacent edges should be as small as possible. Thus we can consider the problem as a minimization problem where the costs $\widehat{c}_{ijk}$ are given by the **inner angles** as illustrated in Figure 4.1 and defined by

$$\widehat{c}_{ijk} = \widehat{\alpha}_{ijk} := \pi - \alpha_{ijk}. \tag{4.3}$$

The corresponding **reverse objective value** is defined as

$$\widehat{OV}(G,T) := n \cdot \pi - OV(G,T). \tag{4.4}$$

Our considerations of the MaxAngleTSP are related to Fekete and Woeginger [17] and Dumitrescu et al. [16] where angle-restricted tours are studied. Given points in the Euclidean plane the authors investigate in which cases there exist tours such that all inner angles defined on $(-\pi, +\pi]$

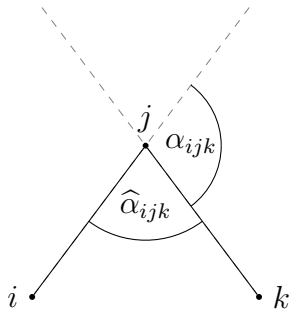

Figure 4.1: Illustration of the *turning angle* α_{ijk} and of the *inner angle* $\widehat{\alpha}_{ijk}$.

belong to some set $A \subseteq (-\pi, +\pi]$ and consider the complexity of the corresponding decision problems. Fekete and Woeginger [17] prove that for at least five vertices there always exists a so called "pseudo-convex" tour, i.e., a tour having only clockwise or only counterclockwise turns in all vertices. Our result on the MaxAngleTSP for an odd number of vertices will imply the theorem by Fekete and Woeginger [17] in this case. Furthermore, Dumitrescu et al. [16] show that there always exists a tour such that the largest inner angle is at most $\frac{2\pi}{3}$ if the number of vertices is even.

4.1.2 Our contribution

The previous ILP-based solution approaches done by Fischer and Helmberg [21] perform all separation processes to identify the violated subtour elimination constraints on fractional solutions of a linearized model. On contrary, motivated by the impressive performance of today's ILP solvers we will pursue the strategy to do all separation processes only on integral solutions. The same idea was tested with very limited success for the standard TSP in the previous chapter but turned out to be much more promising for the SQTSP, as we will show in Section 4.2. In Section 4.4 we will combine this approach with different variants of the classical subtour elimination constraints (see e.g. Dantzig et al. [14]) and the strengthened variants introduced by Fischer and Helmberg [21]. We also give experimental results for MaxSQTSP which was never treated in the literature before and turns out to be very challenging from a computational point of view. These computational studies belong to the main contributions of this paper and show that in many cases our *integral approach* is faster in total and that sophisticated separation strategies do not pay off in the minimization case. The benchmark instances and the computational test environment of all our experiments are described in Section 4.3.

Second, we provide a completely different, mathematically interesting MILP linearization for the AngleTSP and related problems combining angle change and distances in Subsection 4.4.2. This model has the advantage that only linearly many new variables are introduced. For instances with up to 55 vertices, where the angle change is weighted against the distances, the running times can be improved. However, for larger instances or classical AngleTSP instances our computation tests will illustrate that for large n the branch-and-cut approaches based on the linearization by Fischer and Helmberg [21] are faster.

The third main contribution concerns the theoretical solution structure of the MaxAngleTSP. It is shown in Subsection 4.5.1 that there is a surprising split of the problem: For n odd, it can be shown that the reverse optimal solution value is always π, i. e. 180 degrees and the problem can be solved by the standard ILP model without producing any subtours. Nevertheless, even without subtour elimination constraints the solution of the remaining ILP, a 2-*matching problem* (also called *cycle cover problem*), requires very large running times. Fortunately, we can bypass this difficulty since we can characterize the structure of an optimal solution and derive a simple constructive algorithm to find such an optimal solution.

For n even, no such statement is possible and we can show that the reverse objective function is bounded by 2π, i.e. 360 degrees.

4.2 Fractional vs. integral approach

The quadratic traveling salesman problem can be written as the following quadratic integer program with binary edge variables $x_e = x_{ij}$ for $e = (i, j) \in V^{\{2\}}$, and $\delta(i) := \{e \colon e = (i, j) \in V^{\{2\}}\}$ denoting the set of all edges adjacent with $i \in V$.

$$\min/\max \sum_{\substack{e^{\langle 3\rangle}=\langle i,j,k\rangle \in V^{\langle 3\rangle} \\ e=(i,j),\ f=(j,k)}} c_{e^{\langle 3\rangle}} x_e x_f \tag{4.5}$$

$$\text{s.t.} \quad \sum_{e \in \delta(i)} x_e = 2 \qquad \forall\, i \in V, \tag{4.6}$$

$$\sum_{\substack{e=(i,j)\in V^{\{2\}} \\ i,j \in S}} x_e \leq |S| - 1 \quad \forall\, S \subset V,\ S \neq \emptyset, \tag{4.7}$$

$$x_e \in \{0, 1\} \quad \forall\, e \in V^{\{2\}}. \tag{4.8}$$

In the objective function (4.5) for determining the *objective function value* a cost $c_{e^{\langle 3\rangle}}$ for some 2-edge $e^{\langle 3\rangle} \in V^{\langle 3\rangle}$ is taken into account if both edges $e = (i, j)$ and $f = (j, k)$ are contained in the tour. Equations (4.6) are the *degree constraint* ensuring that each vertex is visited once, (4.7) are the well-known *subtour elimination constraints* and, finally, (4.8) are the integrality constraints on the edge variables. In comparison to the standard model for the TSP introduced in Chapter 3 (see equations (3.1) – (3.4)) we just changed the objective function.

This quadratic integer program can easily be linearized by introducing a cubic number of additional integer variables $y_{e^{\langle 3\rangle}} = y_{ijk}$ for all 2-edges $e^{\langle 3\rangle} = \langle i, j, k\rangle \in V^{\langle 3\rangle}$, where $y_{ijk} = 1$ if and only if the vertices i, j and k are visited in the tour in consecutive order. This linearization was first introduced and extensively studied by Fischer and Helmberg [21].

$$\min/\max \sum_{e^{\langle 3\rangle}=\langle i,j,k\rangle \in V^{\langle 3\rangle}} c_{e^{\langle 3\rangle}} y_{e^{\langle 3\rangle}} \tag{4.9}$$

$$\text{s.t.} \quad x_e = \sum_{k \in V\setminus\{i,j\}} y_{ijk} = \sum_{k \in V\setminus\{i,j\}} y_{kij} \quad \forall\, e = (i, j) \in V^{\{2\}}, \tag{4.10}$$

$$\sum_{e \in \delta(i)} x_e = 2 \qquad \forall\, i \in V, \tag{4.11}$$

$$\sum_{\substack{e=(i,j)\in V^{\{2\}} \\ i,j\in S}} x_e \leq |S| - 1 \qquad \forall\, S \subset V,\ S \neq \emptyset, \tag{4.12}$$

$$x_e \in \{0, 1\} \qquad \forall\, e \in V^{\{2\}}, \tag{4.13}$$

$$y_{e^{\langle 3\rangle}} \in \{0, 1\} \qquad \forall\, e^{\langle 3\rangle} \in V^{\langle 3\rangle}. \tag{4.14}$$

The x-variables have to correspond to a tour. Apart from that this model has a linear objective function (4.9). Constraints (4.10) couple the x- and the y-variables. Finally, conditions (4.14) ensure the integrality of the y-variables.

This ILP can be used to solve the SQTSP by using the "standard" TSP techniques which were also applied by Fischer and Helmberg [21] and by Fischer et al. [19] in their computational studies. In particular, we can separate the subtour elimination constraints (4.7) in the same way as described in the TSP literature (see e.g. Reinelt [46]), i. e. by identifying the violated constraints on fractional solutions during the branch-and-cut solution process solving appropriate min-cut problems.

In this paper we focus on a different strategy which was already described in Chapter 3: We relax all subtour constraints (4.7) first and then solve the remaining model to integral optimality using an ILP solver. We get a 2-matching (a cycle cover) usually containing more than one cycle. These cycles can be found by a simple scan. Now, we can include a subtour elimination constraint for each such cycle and resolve the enlarged ILP model. This process is repeated and in each iteration additional subtour elimination constraints are added until we get a solution consisting of only one cycle, i.e. containing only an optimal SQTSP/MaxSQTSP tour.

A short pseudocode description of this approach is given in Algorithm 4.1 (which corresponds to Algorithm 3.1 in Chapter 3). An example illustrating its execution is given in Figures 4.2–4.4: We can see that we need 3 iterations (ILP solver runs) and 4 subtour elimination constraints to solve the problem to optimality.

Require: SQTSP/MaxSQTSP instance
Ensure: an optimal SQTSP/MaxSQTSP tour
1: define current model as (4.6), (4.8), (4.9), (4.10), and (4.14);
2: **repeat**
3: solve the current model to optimality by an ILP solver;
4: **if** solution contains no subtour **then**
5: **return** the solution as optimal tour;
6: **else**
7: find all subtours of the solution and add the corresponding subtour elimination constraints of type (4.7) to the current model;
8: **end if**
9: **until** optimal tour found;

Algorithm 4.1: Main idea of our elementary integral approach.

Let us concentrate on the subtour elimination constraints (4.7) first. They can be expressed equivalently by the following cut constraints:

$$\sum_{\substack{e=(i,j)\in V^{\{2\}} \\ i\in S,\ j\in V\setminus S}} x_e \geq 2 \quad \forall\ S \subset V,\ S \neq \emptyset. \tag{4.15}$$

Although mathematically equivalent, it was observed that the two versions of forbidding a subtour may result in quite different performances of the ILP solver in the TSP context (see Chapter 3). Thus we tested these two variants

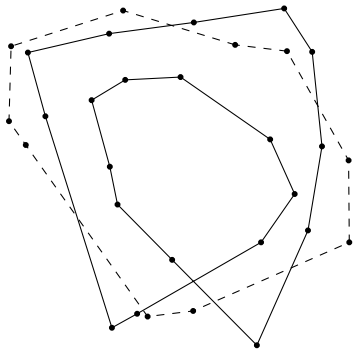

Figure 4.2: Angle-instance with $n = 30$: Iteration 1.

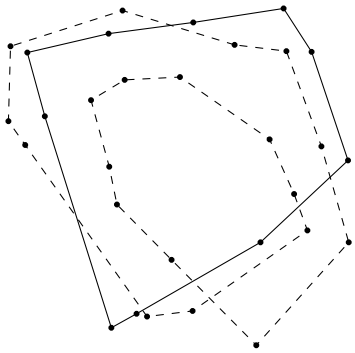

Figure 4.3: Angle-instance with $n = 30$: Iteration 2.

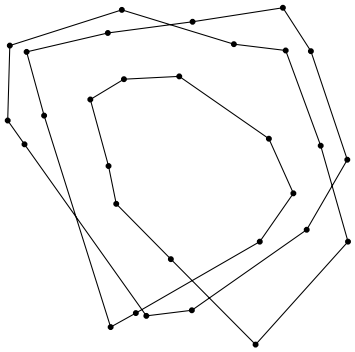

Figure 4.4: Angle-instance with $n = 30$: Optimal AngleTSP tour.

together with the combined variant

$$\begin{array}{lll} \displaystyle\sum_{\substack{e=(i,j)\in V^{\{2\}}\\ i,j\in S}} x_e \;\leq\; |S|-1 & & \text{if} \quad |S| \leq \dfrac{2n+1}{3} \\ & \forall\, S\subset V,\ S\neq\emptyset & \\ \displaystyle\sum_{\substack{e=(i,j)\in V^{\{2\}}\\ i\in S,\ j\in V\setminus S}} x_e \;\geq 2 & & \text{if} \quad |S| > \dfrac{2n+1}{3} \end{array} \quad , \tag{4.16}$$

where we chose the variant which adds less non-zero entries to the overall constraint matrix (see also Section 1 in Dey et al. [15] for some comments on constraints with large support). The computational results are summarized in Table 4.1 (see Section 4.3 for detailed description of the benchmark instances and of the test environment). Our tests will show that the com-

Instance	I (constr. (4.7))	I (constr. (4.15))	I (constr. (4.16))
Angle_25	1.3	1.11	1.11
Angle_30	3.3	1.01	1.01
Angle_35	6.6	1.08	1.08
Angle_40	57.9	1.06	1.06
Angle_45	129.1	1.12	1.12
Angle_50	321.3	1.00	1.00
Angle_55	1099.8	1.07	1.07
mean		*1.07*	*1.07*
Angle-Distance_50	19.2	0.98	0.98
Angle-Distance_55	80.6	1.02	1.02
Angle-Distance_60	68.5	1.01	1.01
Angle-Distance_65	104.3	1.10	1.08
Angle-Distance_70	543.4	1.01	1.01
Angle-Distance_75	773.2	0.98	0.98
Angle-Distance_80	1719.5	0.95	0.95
mean		*1.00*	*1.00*
Random_20	3.8	0.97	1.05
Random_25	34.2	1.12	1.05
Random_30	331.4	0.82	0.83
Random_35	2979.1	1.03	0.92
mean		*0.98*	*0.96*

Table 4.1: Minimization case: Comparing the running times of the approaches I (subtour elimination constraints as in (4.7)), I (subtour elimination constraints as in (4.15)) and the elementary integral approach I (subtour elimination constraints as in (4.16)).

bined variant (4.16) tends to slightly outperform the other two variants for randomly generated instances and for some AngleTSP instances, where the angle changes are weighted against the distances for larger n. Since this representation of subtour elimination constraints was also used in Chapter 3 for computational tests, we stick to using (4.16) for the basic reference methods F and I, i. e. for the *elementary fractional* and *elementary integral approach*, respectively, in the rest of this chapter.

4.3 Computational experiments

Since a major part of this chapter relies on computational experiments, we describe in this section the test setup.

4.3.1 Benchmark instances

Our benchmark instances are mainly based on the instance specification developed by Fischer [20] and by Fischer and Helmberg [21], however, we

usually do not round the costs to integers since the second MILP linearization we will introduce (see Section 4.4.2) requires the exact turning angle values α for instances of the AngleTSP or variants thereof. We test three instance classes:

Angle-instances are based on points in the Euclidean plane: First, we choose n points discrete uniformly at random out of $\{0, \ldots, 500\}^2$ and then compute the *turning angles* α which we multiply by 1000 first and then we round them to 12 decimal places. The multiplicative constant of 1000 is introduced to allow a comparison of our programs with the work by FISCHER [20].

Instances of this type are named *Angle_n* where n denotes the number of points/vertices.

Angle-Distance-instances extend the above Angle-instances by combining them with the Euclidean distances between the points in a weighted sum. Taking the identical point sets in the plane as generated for *Angle_n* (to allow a better comparison) we denote the Euclidean distances between vertices i and j as d_{ij}. For a parameter $\rho \in \mathbb{R}_0^+$ we construct SQTSP-costs as follows:

$$c_{ijk} := 100 \left(\rho \cdot \alpha_{ijk} + \frac{d_{ij} + d_{jk}}{2} \right) \tag{4.17}$$

We again round all costs to 12 decimal places.

This kind of instances was introduced by SAVLA et al. [48] for an approximate solution of the *TSP for Dubins vehicle*, which has applications in robotics. These instances are between Angle-instances (for $\rho \to \infty$) and standard TSP instances (for $\rho = 0$). For compatibility with the literature we set $\rho = 40$ for all our tests as done by FISCHER [20].

Instances of this type are named *Angle-Distance_n*.

Random-instances assign random costs $c_{e^{\langle 3 \rangle}}$ to all 2-edges $e^{\langle 3 \rangle} \in V^{\langle 3 \rangle}$, in particular the costs are chosen discrete uniformly at random from $\{0, \ldots, 10000\}$.

The resulting instances are named *Random_n*.

For each type of instance and each size n we generated 10 test instances.

4.3.2 Layout of test results

The tables containing computational results are all created in the following way: The first column always contains the instance type and size. The second column contains the running times for the first method which acts as a reference method in the particular table. All further columns report the ratios between the running times of the particular approaches and the reference method. The only exception concerns the *root node ratios* in Table 4.2 where the ratios between the values of the LP relaxation in the root node and the optimal solution values are reported. We generated 10 instances for every instance type and every size n and report the mean values over these 10 instances, in particular *arithmetic means* for running times and *geometric means* for all ratios. Entries for instances where a particular method cannot be used are marked by "–" and, finally, entries which could be expected to cause excessive running times were omitted from the tests and are marked by "Ø".

4.3.3 Test environment

All tests were run on an *Intel(R) Core(TM) i5-3470 CPU @ 3.20GHz with 16 GB RAM* under *Linux*[1] and all programs were implemented in *C++*[2] by using *CPLEX*[3] as the ILP solver. Moreover, in order to guarantee the relative reproducibility of our computational results, we (i) allowed no additional swap memory and (ii) ran all tests separately without other user processes in background.

4.4 Minimization problem

The minimization problem SQTSP and its special case AngleTSP were exhaustively studied by Fischer and Helmberg [21] from a theoretical point of view: After introducing the ILP linearization (4.9)–(4.14), stronger forms of subtour elimination constraints which also involve the y-variables are described and analyzed. Their paper also contains a small computational study which is based on a larger set of computational experiments reported by Fischer [20]. However, the authors only deal with the "standard" fractional approach and its variants.

[1]Precise version: *Linux 3.5.0-23-generic #35~precise1-Ubuntu SMP x86_64 x86_64 x86_64 GNU/Linux.*

[2]Precise compiler version: *gcc version 4.8.1.*

[3]Precise version: *IBM(R) ILOG(R) CPLEX(R) Interactive Optimizer 12.6.1.0.*

One of the main goals of this paper is to provide an exhaustive computational comparison of different solution strategies. First, we compare the elementary fractional approach with the elementary integral approach and then we combine both approaches with the stronger subtour elimination constraints from [21] in Section 4.4.1. In this section we also examine some weaker types of subtour elimination constraints. Finally, in Section 4.4.2 we introduce a new and completely different MILP linearization with only a linear number of additional variables. It is based on a geometric argument and works only for AngleTSP and Angle-Distance-instances.

Instance	**F**	**F$^{\mathbf{L}}$**	**I**	**I$^{\mathbf{L}}$**	**ratio**	**ratio$^{\mathbf{L}}$**
Angle_25	1.6	1.55	0.91	1.30	0.91	0.89
Angle_30	3.7	1.98	0.95	1.72	0.92	0.90
Angle_35	7.7	3.39	0.92	2.77	0.91	0.88
Angle_40	63.9	5.30	0.98	4.03	0.91	0.86
Angle_45	148.1	10.02	0.84	9.12	0.90	0.86
Angle_50	443.9	Ø	0.81	Ø	0.91	Ø
Angle_55	2183.3	Ø	0.86	Ø	0.91	Ø
mean		*3.53*	*0.89*	*2.96*	*0.91*	*0.88*
Angle-Distance_30	0.8	0.59	1.04	0.56	0.98	0.97
Angle-Distance_35	2.3	0.53	0.96	0.43	0.97	0.96
Angle-Distance_40	3.4	0.60	0.90	0.53	0.97	0.96
Angle-Distance_45	11.8	0.75	0.95	0.51	0.96	0.95
Angle-Distance_50	22.4	0.76	0.84	0.47	0.95	0.95
Angle-Distance_55	85.7	0.93	0.92	0.61	0.95	0.94
Angle-Distance_60	80.3	1.36	0.87	1.00	0.95	0.94
Angle-Distance_65	126.7	2.07	0.84	1.36	0.95	0.94
Angle-Distance_70	909.1	1.91	0.67	1.28	0.95	0.94
Angle-Distance_75	1035.8	4.55	0.77	3.11	0.95	0.93
Angle-Distance_80	3742.4	6.46[a]	0.60	3.91	0.95	0.93
mean		*1.27*	*0.84*	*0.92*	*0.96*	*0.95*
Random_20	4.6	–	0.90	–	0.66	–
Random_25	41.1	–	0.87	–	0.64	–
Random_30	306.1	–	0.90	–	0.63	–
Random_35	2990.1	–	0.81	–	0.62	–
mean		–	*0.87*	–	*0.64*	–

Table 4.2: Minimization case: Comparing the running times of the elementary fractional approach F, the approach I$^{\mathrm{L}}$, the elementary integral approach I and the approach I$^{\mathrm{L}}$. Moreover, the table compares the respective root node ratios (i. e. the ratio between root node value of the LP relaxation and optimal solution value) of the two linearizations in columns *ratio* and *ratio*$^{\mathrm{L}}$.

Our first computational results are given in Table 4.2. The first column (F) contains the results for the *elementary fractional* and the third column

[a] Mean of 9 instances (one instance did not fit into 16 GB RAM).

(I) the results for the *elementary integral approach*. We can see that the elementary integral approach outperforms the elementary fractional approach significantly and, moreover, this trend seems to increase for larger n by all classes of test instances. This behavior is quite surprising since the analogous integral solution strategies are clearly outperformed by the fractional ones for the standard TSP (cf Chapter 3). To find an explanation for this difference we analyzed numerous test instances individually and could observe one main difference between SQTSP and TSP (cf Chapter 3, especially Table 3.9): The SQTSP instances usually produce significantly less subtour elimination constraints than the TSP instances of similar size and thus solving them involves much less ILP solver runs. In fact, we obtained less than 10 subtour elimination constraints for most of our test instances. By studying the particular stages of the solution process for the Angle- and Angle-Distance-instances we could also observe that the structure of the generated subtours is different: While the subtours in the TSP case are typically small and often consist of only a few vertices and thus we usually have many of them (see e.g. the example of Figures 3.2–3.13 in Chapter 3), for the SQTSP instances we obtained large subtours. For the AngleTSP the subtours have the form of large "circles" or "spiral" shapes. This behavior was also illustrated in the example of Figures 4.2, 4.3 and 4.4, where only 4 subtour elimination constraints were required for determining an optimal SQTSP tour.

4.4.1 Extending the subtour elimination constraints

In this section, we extend the elementary fractional and the elementary integral approach by including the subtour elimination constraints introduced by Fischer and Helmberg [21]. We also examine some variants of these constraints. More detailed descriptions and illustrations of the underlying geometric ideas can be found in [20]. All computational results for the introduced variants are summarized in Tables 4.3 and 4.4.

F(I)/I(I): We try to reduce the number of unnecessary subtour elimination constraints in this variant. In fact, one subtour elimination constraint can be skipped in every iteration as the corresponding tour is implicitly excluded by the other ones. Thus we always omit the constraint for the largest subtour (i. e. for the subtour which causes the most non-zero entries in the constraint matrix).

Since we introduce in every iteration only one subtour elimination constraint less than in the *elementary fractional / elementary integral* approach, we cannot expect huge computational time improvements. Indeed, the methods F(I)/I(I) perform similarly to F/I for the Angle-

Instance	F	F(I)	F(II)	F(III)	F(IV)	F(V)
Angle_25	1.6	1.07	1.03	1.03	1.14	1.02
Angle_30	3.7	1.00	1.01	1.04	1.00	1.08
Angle_35	7.7	0.99	1.01	1.07	0.98	1.14
Angle_40	63.9	1.03	1.07	1.10	1.08	1.14
Angle_45	148.1	0.99	0.90	1.00	1.04	1.03
Angle_50	443.9	0.97	1.14	1.03	1.05	1.14
Angle_55	2183.3	1.03	1.06	1.06	1.00	1.12
mean		*1.01*	*1.03*	*1.05*	*1.04*	*1.09*
Angle-Distance_50	22.4	1.00	1.03	0.97	0.93	0.90
Angle-Distance_55	85.7	1.02	0.95	1.30	1.04	1.03
Angle-Distance_60	80.3	1.06	0.98	1.13	1.02	0.98
Angle-Distance_65	126.7	1.02	1.02	1.13	0.89	1.04
Angle-Distance_70	909.1	1.02	1.07	1.13	0.98	1.05
Angle-Distance_75	1035.8	1.05	1.15	1.43	1.21	1.13
Angle-Distance_80	3742.4	0.93	0.90	1.22	0.93	0.99
mean		*1.01*	*1.01*	*1.18*	*0.99*	*1.01*
Random_20	4.6	0.86	0.79	0.89	0.92	0.87
Random_25	41.1	0.76	0.94	0.92	0.89	0.82
Random_30	306.1	1.12	1.10	1.09	0.87	1.01
Random_35	2990.1	0.98	1.11	1.13	1.11	1.07
mean		*0.92*	*0.98*	*1.00*	*0.94*	*0.94*

Table 4.3: Minimization case: Comparing the running times of the elementary fractional approach F and other approaches using different variants of subtour elimination constraints.

and Angle-Distance-instances. We can observe a larger variance of the ratios for the Random-instances, but this trend does not seem to hold for larger values of n.

F(II)/I(II): In this variant, we make the constraints (4.7) weaker by including the x-variables only for pairs of vertices i, j which are connected with an edge in the particular subtour. So we have

$$\sum_{\substack{e=(i,j)\in V^{\{2\}} \\ i,j\in S \\ x^*_{ij}=1}} x_e \leq |S| - 1 \quad \forall\ S \subset V,\ S \neq \emptyset \tag{4.18}$$

where $x^*_{ij} = 1$ if the vertices i and j are connected by an edge in the current solution. Since there does not exist a cut version of such subtour elimination constraint, we always use (4.18) independently of the subtour size $|S|$.

The performance of this variant is mostly similar to our *elementary approaches*. This is a bit surprising, because this kind of subtour elimination constraints is significantly weaker from a theoretical point of view if $|S| > 3$.

Instance	I	I(I)	I(II)	I(III)	I(IV)	I(V)
Angle_25	1.4	1.08	1.01	1.10	1.13	1.03
Angle_30	3.4	1.01	1.03	1.04	1.04	1.07
Angle_35	6.8	1.04	1.04	1.06	1.00	1.17
Angle_40	59.5	0.95	0.98	0.95	0.99	1.09
Angle_45	125.7	1.04	1.02	1.07	1.04	1.15
Angle_50	323.1	1.11	1.12	0.96	1.09	1.23
Angle_55	1299.2	0.94	0.92	1.00	1.05	1.22
mean		*1.02*	*1.02*	*1.02*	*1.05*	*1.13*
Angle-Distance_50	19.1	1.11	1.26	1.02	1.10	1.14
Angle-Distance_55	78.2	0.94	1.13	1.12	1.09	1.25
Angle-Distance_60	70.4	0.92	1.04	0.90	0.97	0.97
Angle-Distance_65	102.6	1.01	1.14	1.09	1.13	1.08
Angle-Distance_70	444.7	1.13	1.55	1.17	1.40	1.26
Angle-Distance_75	784.1	0.98	0.94	1.07	0.96	1.10
Angle-Distance_80	1699.2	0.99	0.96	1.00	0.98	1.03
mean		*1.01*	*1.13*	*1.05*	*1.08*	*1.11*
Random_20	3.9	0.99	0.97	0.90	0.88	0.91
Random_25	37.0	0.76	0.88	0.82	0.92	0.79
Random_30	235.3	0.97	1.15	1.35	1.05	1.27
Random_35	2887.0	1.07	1.04	1.07	1.07	1.37
mean		*0.94*	*1.00*	*1.01*	*0.98*	*1.06*

Table 4.4: Minimization case: Comparing the running times of the elementary integral approach I and other approaches using different variants of subtour elimination constraints.

F(III)/I(III): We use the strengthened variant of the subtour elimination constraints introduced by FISCHER and HELMBERG [21]. They are based on the idea that a y-variable $y_{ikj}, ikj \in V^{\langle 3\rangle}$, almost acts like an x-variable $x_{ij}, ij \in V^{\{2\}}$. If they are one in a solution both express that the vertices i and j are close in the tour. So, in the following formulas we do not only count the number of direct connections between the vertices of some set S, but also the connections that leave S but immediately return.

We get

$$\begin{aligned}
\sum_{\substack{e=(i,j)\in V^{\{2\}}\\ i,j\in S}} x_e + \sum_{\substack{e^{\langle 3\rangle}=\langle i,k,j\rangle\in V^{\langle 3\rangle}\\ i,j\in S,\ k\in V\setminus S}} y_{e^{\langle 3\rangle}} &\le |S|-1 && \text{if} \quad |S| < \frac{n}{2}\\
&\qquad \forall\ S\subset V,\ S\neq\emptyset &&\\
\sum_{\substack{e=(i,j)\in V^{\{2\}}\\ i,j\in S}} x_e + \sum_{\substack{e^{\langle 3\rangle}=\langle i,k,j\rangle\in V^{\langle 3\rangle}\\ i,j\in S,\ k\in V(\setminus S\cup\{\widehat{t}\})}} y_{e^{\langle 3\rangle}} &\le |S|-1 && \text{if} \quad |S| \ge \frac{n}{2}
\end{aligned} \tag{4.19}$$

as a stronger form of (4.7) and, similarly,

$$\begin{aligned}
\sum_{\substack{e=(i,j)\in V^{\{2\}}\\ i\in S,\ j\in V\setminus S}} x_e - 2 \sum_{\substack{e^{\langle 3\rangle}=\langle i,j,k\rangle\in V^{\langle 3\rangle}\\ i,k\in S,\ j\in V\setminus S}} y_{e^{\langle 3\rangle}} &\geq 2 && \text{if} \quad |S| < \frac{n}{2}\\
&\qquad \forall\ S\subset V,\ S\neq\emptyset\\
\sum_{\substack{e=(i,j)\in V^{\{2\}}\\ i\in S,\ j\in V\setminus S}} x_e - 2 \sum_{\substack{e^{\langle 3\rangle}=\langle i,j,k\rangle\in V^{\langle 3\rangle}\\ i,k\in S,\ j\in V\setminus (S\cup\{\widehat{t}\})}} y_{e^{\langle 3\rangle}} &\geq 2 && \text{if} \quad |S| \geq \frac{n}{2}
\end{aligned} \tag{4.20}$$

as a stronger form of the cut variant (4.15), which is equivalent to (4.19). These can be interpreted in the following way. We only count those edges leaving S that do not immediately reenter S. Note, if $|S| \geq \frac{n}{2}$ then we have to exclude one vertex $\widehat{t} \in V\setminus S$ in summation in order to not forbid tours in both variants (for detailed proofs see Fischer and Helmberg [21]). Of course there are many possibilities how to determine this vertex; we just chose $\widehat{t} = \arg\max_{k\in V\setminus(S\cup\{\widehat{t}\})} \min_{i,j\in S,\ i\neq j} d_{ikj}$ as the influence of this choice seems to be rather limited. Finally, we keep the case distinction introduced in (4.16) and thus use (4.19) if $|S| \leq \frac{2n+1}{3}$ and (4.20) otherwise.

Although these strengthened subtour elimination constraints are facet defining for the SQTSP polytope as long as $2 \leq |S| \leq n-3$ and the constraints are much stronger than (4.7) from a theoretical point of view, we cannot observe any computational time improvements in Tables 4.3 and 4.4. On the contrary, these kinds of subtour elimination constraints increase the computational times for almost all test instance groups and this trend becomes even stronger for larger values of n. Additionally, we want to note that determining a maximally violated inequality (4.19) in the case $|S| < \frac{n}{2}$ is an $\mathcal{NP}$-hard problem (see Fischer and Helmberg [21]), the complexity of the separation problems is currently unknown.

F(IV)/I(IV): This variant combines the subtour elimination constraints used in F(III) and I(III) with the idea introduced for F(II) and I(II), however, this idea is adopted only for the y-variables. So we add the associated y-variable into the set of constraints only if $x^*_{ij} = 1$ in (4.19) or $x^*_{ik} = 1$ in (4.20), where $x^*_{ij} = 1$ if the vertices i and j are connected by an edge in the current solution. In particular, we get

$$\begin{aligned}
&\sum_{\substack{e=(i,j)\in V^{\{2\}}\\ i,j\in S}} x_e + \sum_{\substack{e^{\langle 3\rangle}=\langle i,k,j\rangle\in V^{\langle 3\rangle}\\ i,j\in S,\ k\in V\setminus S\\ x^*_{ij}=1}} y_{e^{\langle 3\rangle}} \leq |S|-1 && \text{if } |S|<\frac{n}{2}\\
&\qquad\qquad \forall\ S\subset V,\ S\neq\emptyset\\
&\sum_{\substack{e=(i,j)\in V^{\{2\}}\\ i,j\in S}} x_e + \sum_{\substack{e^{\langle 3\rangle}=\langle i,k,j\rangle\in V^{\langle 3\rangle}\\ i,j\in S,\ k\in V(\setminus S\cup\{\widehat{t}\})\\ x^*_{ij}=1}} y_{e^{\langle 3\rangle}} \leq |S|-1 && \text{if } |S|\geq\frac{n}{2}
\end{aligned} \tag{4.21}$$

as a variant of (4.19). Subsequently, we get

$$\begin{aligned}
&\sum_{\substack{e=(i,j)\in V^{\{2\}}\\ i\in S,\ j\in V\setminus S}} x_e - 2\sum_{\substack{e^{\langle 3\rangle}=\langle i,j,k\rangle\in V^{\langle 3\rangle}\\ i,k\in S,\ j\in V\setminus S\\ x^*_{ik}=1}} y_{e^{\langle 3\rangle}} \geq 2 && \text{if } |S|<\frac{n}{2}\\
&\qquad\qquad \forall\ S\subset V,\ S\neq\emptyset\\
&\sum_{\substack{e=(i,j)\in V^{\{2\}}\\ i\in S,\ j\in V\setminus S}} x_e - 2\sum_{\substack{e^{\langle 3\rangle}=\langle i,j,k\rangle\in V^{\langle 3\rangle}\\ i,k\in S,\ j\in V\setminus(S\cup\{\widehat{t}\})\\ x^*_{ik}=1}} y_{e^{\langle 3\rangle}} \geq 2 && \text{if } |S|\geq\frac{n}{2}
\end{aligned} \tag{4.22}$$

as a variant of (4.20).

This idea tries to reduce the number of non-zero entries in the constraint matrix and can be seen as the compromise between the *elementary approaches* and variants F(III)/I(III). Obviously, we lose the facetness of the added inequalities.

Looking at the running times, we can expect that these fluctuate between F/I and F(III)/I(III) as well. Indeed, in our tests the results for the methods F(IV)/I(IV) are a bit worse than the results for F/I and a bit better than the results for F(III)/I(III).

F(V)/I(V): We use an equivalent version of (4.19) or (4.20), respectively, that only uses y-variables.

$$\begin{aligned}
&\sum_{\substack{e^{\langle 3\rangle}=\langle i,j,k\rangle\in V^{\langle 3\rangle}\\ i\in S,\ j,k\in V\setminus S}} y_{e^{\langle 3\rangle}} \geq 2 && \text{if } |S|<\frac{n}{2}\\
&\qquad\qquad \forall\ S\subset V,\ S\neq\emptyset\\
&\sum_{\substack{e^{\langle 3\rangle}=\langle i,j,k\rangle\in V^{\langle 3\rangle}\\ i\in S,\ j,k\in V\setminus S}} y_{e^{\langle 3\rangle}} + 2\sum_{\substack{e^{\langle 3\rangle}=\langle i,j,k\rangle\in V^{\langle 3\rangle}\\ i,k\in S,\ j=\widehat{t}}} y_{e^{\langle 3\rangle}} \geq 2 && \text{if } |S|\geq\frac{n}{2}\ ,
\end{aligned}$$

(4.23)

where $\widehat{t} = \arg\max_{k \in V \setminus S \setminus \{\widehat{t}\}} \min_{i,j \in S, i \neq j} d_{ikj}$. They can be interpreted similarly to the constraints above. We only count those 2-edges that leave the set S without immediately returning. In our tests, we always use (4.23) independently of the subtour size $|S|$.

These constraints are facet defining if $n \geq 6$ and $2 \leq |S| \leq n-3$ (see FISCHER and HELMBERG [21]), but do not speed up the solution process, as it can be seen in Tables 4.3 and 4.4. The methods F(V)/I(V) tend to perform worse for all instance groups (at least for larger n) and they are significantly worse for the Angle-instances.

Finally, let us take a short overview of the computational results in Tables 4.3 and 4.4. We can observe that subtour elimination constraints which are stronger from a theoretical point of view tend to slow down the algorithm. This fact is surprising since one would expect that stronger models will lead to better bounds during the solution process. One possible explanation might be the number of non-zero entries in the constraint matrix (see DEY et al. [15]): The methods F(I)/I(I) and F(II)/I(II) perform similarly to the *elementary approaches* F/I, whereas variants F(III)/I(III), F(IV)/I(IV) and F(V)/I(V) all require larger running times and all have a larger number of non-zero entries in their constraint matrix. However, since we use the ILP solver as a "black box", this is only a guess.

4.4.2 A geometry-based MILP linearization for AngleTSP

The standard linearization described in Section 4.2 requires a cubic number of additional integer variables $y_{e^{\langle 3 \rangle}} = y_{ijk}$ for all $e^{\langle 3 \rangle} = \langle i,j,k \rangle \in V^{\langle 3 \rangle}$. Exploiting the geometry of the AngleTSP we can derive a different linearization adding only a linear number of real-valued variables. A related construction was used by MEIER and CLAUSEN [36] for a single allocation hub location problem. Clearly, this approach can be applied immediately also to the Angle-Distance-instances by splitting the objective function into a linear distance component and the turning angle:

$$\min \quad w_1 \sum_{e \in V^{\{2\}}} d_e x_e + w_2 \sum_{\substack{e^{\langle 3 \rangle} = \langle i,j,k \rangle \in V^{\langle 3 \rangle} \\ e=(i,j),\ f=(j,k)}} \alpha_{e^{\langle 3 \rangle}} x_e x_f \tag{4.24}$$

Euclidean distances between vertices i and j are denoted by $d_e = d_{ij}$ and $\alpha_{e^{\langle 3 \rangle}} = \alpha_{ijk}$ gives the *turning angle* as defined in (4.2). The parameters w_1

and w_2 can be used to weight the two components of the objective function. Our Angle-instances and Angle-Distance-instances defined in Section 4.3.1 correspond to the settings $w_1 = 0$, $w_2 = 1000$ and $w_1 = 100$, $w_2 = 4000$, respectively.

The quadratic terms in the second part of (4.24) can be easily moved into the set of constraints by introducing a new variable $y_j \in \mathbb{R}_0^+$ for every vertex $j \in V$ corresponding to the turning angle of a tour in vertex j. Thus, we replace (4.24) by

$$\min \quad w_1 \sum_{e \in V^{\{2\}}} d_e x_e + w_2 \sum_{j \in V} y_j \tag{4.25}$$

with

$$y_j \geq \sum_{\substack{i,k \in V \setminus \{j\} \\ i<k}} \alpha_{ijk} x_{ij} x_{jk} \quad \forall j \in V \tag{4.26}$$

as new constraints. We will now prove that the set of quadratic inequalities of type (4.26) is—assuming that the degree constraints (4.6) and the integrality conditions (4.8) are satisfied—equivalent to the following linear inequalities:

$$y_j \geq \sum_{k \in V \setminus \{j\}} \alpha_{ijk} x_{jk} - \pi \quad \forall i, j \in V,\ i \neq j. \tag{4.27}$$

For the proof we need the following geometric lemma.

Lemma 4.2. *For each $i, j, k, l \in V$ with $i \neq j$, $k \neq j$ and $l \neq j$ we have*

$$\alpha_{ijk} + \pi \geq \alpha_{ljk} + \alpha_{lji}. \tag{4.28}$$

Proof. Recall from (4.3) that $\widehat{\alpha}_{ijk} = \pi - \alpha_{ijk}$ denotes the *inner angle*. For any $l \in V$ we have $\widehat{\alpha}_{ijl} + \widehat{\alpha}_{ljk} \geq \widehat{\alpha}_{ijk}$ which shows the claim (using also the symmetry of the first and last index in $\widehat{\alpha}$). □

Theorem 4.3. *The set of constraints* (4.6) *and* (4.26) *is equivalent to the set of constraints* (4.6) *and* (4.27) *for binary variables $x_{ij} \in \{0, 1\}$, $i, j \in V^{\{2\}}$, and variables $y_j \in \mathbb{R}_0^+$, $j \in V$.*

Proof. Assume first that (4.6) and (4.26) are satisfied. Let $y \in V$ be fixed. Then we can do the following calculation by applying Lemma 4.2 for a vertex

$l \in V, l \neq j$:

$$y_j \geq \sum_{\substack{i,k \in V\setminus\{j\} \\ i<k}} \alpha_{ijk} x_{ij} x_{jk} \geq \sum_{\substack{i,k \in V\setminus\{j\} \\ i<k}} \big(\alpha_{ljk} + \alpha_{lji} - \pi\big) x_{ij} x_{jk} \tag{4.29}$$

$$= \sum_{\substack{i,k \in V\setminus\{j\} \\ i<k}} \alpha_{ljk} x_{ij} x_{jk} + \sum_{\substack{i,k \in V\setminus\{j\} \\ i<k}} \alpha_{lji} x_{ij} x_{jk} - \pi \sum_{\substack{i,k \in V\setminus\{j\} \\ i<k}} x_{ij} x_{jk} \tag{4.30}$$

$$\overset{i \Leftrightarrow k}{=} \sum_{\substack{i,k \in V\setminus\{j\} \\ i<k}} \alpha_{ljk} x_{ij} x_{jk} + \sum_{\substack{i,k \in V\setminus\{j\} \\ k<i}} \alpha_{ljk} x_{kj} x_{ji} - \pi \sum_{\substack{i,k \in V\setminus\{j\} \\ i<k}} x_{ij} x_{jk} \tag{4.31}$$

$$= \sum_{\substack{i,k \in V\setminus\{j\} \\ i\neq k}} \alpha_{ljk} x_{ij} x_{jk} - \tfrac{\pi}{2} \sum_{\substack{i,k \in V\setminus\{j\} \\ i\neq k}} x_{ij} x_{jk}. \tag{4.32}$$

Then we eliminate the condition $i \neq k$ by subtracting the sum for $i = k$ and exploit the binarity of the x-variables. This results in:

$$y_j \geq \sum_{i,k \in V\setminus\{j\}} \alpha_{ljk} x_{ij} x_{jk} - \sum_{k \in V\setminus\{j\}} \alpha_{ljk} x_{kj} x_{kj} - \frac{\pi}{2} \left(\sum_{i,k \in V\setminus\{j\}} x_{ij} x_{jk} - \sum_{k \in V\setminus\{j\}} x_{jk} x_{jk} \right) \tag{4.33}$$

$$= \sum_{k \in V\setminus\{j\}} \alpha_{ljk} x_{jk} \sum_{i \in V\setminus\{j\}} x_{ij} - \sum_{k \in V\setminus\{j\}} \alpha_{ljk} x_{kj} - \frac{\pi}{2} \left(\sum_{i \in V\setminus\{j\}} x_{ij} \sum_{k \in V\setminus\{j\}} x_{jk} - \sum_{k \in V\setminus\{j\}} x_{jk} \right) \tag{4.34}$$

$$\overset{(4.6)}{=} \sum_{k \in V\setminus\{j\}} \alpha_{ljk} x_{jk} \cdot 2 - \sum_{k \in V\setminus\{j\}} \alpha_{ljk} x_{kj} - \pi \tag{4.35}$$

$$= \sum_{k \in V\setminus\{j\}} \alpha_{ljk} x_{jk} - \pi. \tag{4.36}$$

This shows that (4.6) and (4.26) together with the integrality of the x-variables imply (4.27).

To prove the other direction, let $\widehat{x}, \widehat{y}$ fulfill (4.6) and (4.27). For every $j \in V$, (4.6) implies the existence of i_j and k_j with $\widehat{x}_{i_j j} = 1$, $\widehat{x}_{k_j j} = 1$ and $\widehat{x}_{ij} = 0$ for all $i \in V, i \neq i_j, j \neq k_j$. Now we evaluate (4.27) for $i = i_j$ and get

$$\begin{aligned}\widehat{y}_j &\geq \sum_{k \in V \setminus \{j\}} \alpha_{ijk} \widehat{x}_{jk} - \pi = \alpha_{ijk_j} + \alpha_{iji_j} - \pi = \alpha_{ijk_j} \\ &= \sum_{\substack{i,k \in V \setminus \{j\} \\ i<k}} \alpha_{ijk} \widehat{x}_{ij} \widehat{x}_{jk}.\end{aligned} \tag{4.37}$$

Therefore, $\widehat{x}, \widehat{y}$ also satisfy (4.26). □

Computational experiments for the linearization given by (4.27) are reported in Table 4.2. Two variants were tested, the former using the standard separation process done on fractional solutions (column F^{L}) and the latter based on the integer subtour approach (column I^{L}). It turns out that the linearization works quite well for medium-sized Angle-Distance-instances with up to 55 vertices where it is superior to the respective elementary versions. However, for larger n it is clearly outperformed by the *elementary fractional* and by the *elementary integral approach*. Moreover, the performance is significantly worse for all Angle-instances. Recall that this approach can be used neither for the Random-instances nor for the maximization problems.

From a theoretical perspective it is also interesting to look at the root node gap, i. e. the difference between objective function values of the LP relaxation in the root node of the ILP solver (without any subtour constraints) for both linearizations and the optimal solution values. Clearly, these are the same for the fractional and integral approaches since the differences in the subtour elimination have effects only later. In Table 4.2 we report the corresponding ratios in columns *ratio* and $ratio^{\mathrm{L}}$. It can be seen that the linearization with (4.27) yields larger root node gaps for all instance classes. Yet they do not differ for the Angle-Distance-instances as much as for the Angle-instances which may explain the differences in performance. However, it should also be pointed out that in the Angle-Distance-instances the linearizations of the y variables is less significant than for the Angle-instances due to the weighted sum in the objective function given by the parameters w_1 and w_2. It can be assumed that the behavior of both linearizations would converge for both models as $\rho \to 0$, i. e. $w_2 \to 0$ in (4.25).

The close relationship to the TSP may also be the reason for the extremely small variance between the root node ratios for different Angle-Distance-instances. Beardwood et al. [8] proved that the expected length of an optimal TSP tour is asymptotically equal to $\beta\sqrt{n}$, where β is a constant, if uniformly random points in the Euclidean planes are considered. Moreover, in Chapter 3 we empirically observed that this convergence property leads

to very small variances of TSP solution values even for small values of n. A similar behavior can be observed in Table 4.2 in the Appendix where the average root node ratios for the Angle-Distance-instances do not vary at all.

Summarizing, this linearization does not yield competitive results for large instances with at least 60 vertices and thus we did not consider it with the other variants of subtour elimination constraints given in Section 4.4.1.

4.5 Maximization problem

In this section we deal with the maximization variant of SQTSP. First, we focus on the MaxAngleTSP from a theoretical point of view and then we provide computational tests for all instance classes of MaxSQTSP as in the minimization case.

4.5.1 Theoretical Analysis of MaxAngleTSP

Recall that for MaxAngleTSP the vertices of the graph correspond to points in the Euclidean plane and the weights d_{ijk} represent the turning angles α_{ijk} for all $i, j, k \in V,\ i, k \neq j$. Moreover, in (4.3) and (4.4) we defined inner angles $\widehat{\alpha}_{ijk} := \pi - \alpha_{ijk}$ and the corresponding reverse objective value $\widehat{OV}(G,T) := n \cdot \pi - OV(G,T)$. It will be convenient to address MaxAngleTSP as a *minimization* problem w. r. t. $\widehat{OV}(G,T)$.

While for SQTSP there was no fundamental difference between Angle-instances and other instance groups, there is a surprising and highly interesting dichotomous behavior to be observed for MaxAngleTSP. We will show below that the optimal reverse objective value equals π for any instance if n is odd. This means that for odd n the optimal solution value of MaxAngleTSP does not require any computation at all. Note that this remarkable result implies that the solution of MaxAngleTSP by our integral approach will never produce any subtour elimination constraint if the vertices are in general position (i.e. if no three vertices are on a line) and so we need exactly one iteration step without adding subtour elimination constraints at all. This follows from the observation that any partition of the odd vertex set into subtours will contain at least one subset of odd cardinality which again would have an optimal reverse objective value equal to π on its own. As every subset of even cardinality implies a solution having a positive reverse objective value (at least for vertices in general position), the overall reverse objective value of such solution is strictly greater than π and thus this solution cannot be optimal. Nevertheless, even without subtours the solution of the ILP model for odd instances of MaxAngleTSP by a standard

solver requires extensive running times as described in Section 4.5.2. As an alternative solution variant we will present a constructive algorithm which calculates an optimal tour in polynomial time.

For n even no equivalent statement holds. In fact, the optimal value of the reverse objective function can attain values between 0 and 2π.

MaxAngleTSP: n is odd

Let n be odd. First, we prove that $\widehat{OV}(G,T) \geq \pi$ for every tour T and thereafter we provide a constructive algorithm which yields a tour T^* reaching this lower bound, i. e., with $\widehat{OV}(G,T^*) = \pi$.

Let ijk be three vertices traversed in succession in a tour T. If the vertex k lies left to the ray $\overrightarrow{ij}$, we say that the tour T has a *left orientation* in the vertex j. Otherwise, we say that the tour T has a *right orientation* in the vertex $j \in V$ (if the vertex k lies on the line ij, we say that the tour has a right orientation in the vertex j as well).

Lemma 4.4. *Let $G = (V,E)$ be an instance of MaxAngleTSP. If n is odd, then the following holds for any tour T:*

$$\widehat{OV}(G,T) \geq \pi. \tag{4.38}$$

Proof. Consider an arbitrary tour T in G and let $L \subseteq V$ be the set of vertices in which the tour T has a left orientation and $R \subseteq V$ the set of vertices in which the tour T has a right orientation. Furthermore, let us define the turning angles of T in j as $\alpha_j^T := \alpha_{ijk}$ and the inner angles in j as $\widehat{\alpha}_j^T := \widehat{\alpha}_{ijk}$, where ijk are three vertices traversed in succession in the tour T. Since the tour T has to be closed, there is

$$\sum_{j \in L} \alpha_j^T - \sum_{j \in R} \alpha_j^T = 2\pi k$$

for some $k \in \mathbb{Z}$. The identity can be reformulated by using (4.3):

$$\begin{aligned}
\sum_{j \in L} (\pi - \widehat{\alpha}_j^T) - \sum_{j \in R} (\pi - \widehat{\alpha}_j^T) &= 2\pi k \\
\big(|L| \cdot \pi - \sum_{j \in L} \widehat{\alpha}_j^T\big) - \big(|R| \cdot \pi - \sum_{j \in R} \widehat{\alpha}_j^T\big) &= 2\pi k \\
\big(|L| - |R|\big)\pi - \sum_{j \in L} \widehat{\alpha}_j^T + \sum_{j \in R} \widehat{\alpha}_j^T &= 2\pi k.
\end{aligned}$$

Since either $|L|$ or $|R|$ is odd, we get

$$\sum_{j\in R} \widehat{\alpha}_j^T - \sum_{j\in L} \widehat{\alpha}_j^T = \pi + 2\pi k'$$

for some $k' \in \mathbb{Z}$. Moreover,

$$\sum_{j\in V} \widehat{\alpha}_j^T = \sum_{j\in R} \widehat{\alpha}_j^T + \sum_{j\in L} \widehat{\alpha}_j^T \geq \Big|\sum_{j\in R} \widehat{\alpha}_j^T - \sum_{j\in L} \widehat{\alpha}_j^T\Big| \geq \sum_{j\in R} \widehat{\alpha}_j^T - \sum_{j\in L} \widehat{\alpha}_j^T,$$

and thus

$$\widehat{OV}(G,T) \geq \pi + 2\pi k'.$$

$\widehat{OV}(G,T) \geq 0$ trivially implies

$$\widehat{OV}(G,T) \geq \pi. \qquad \square$$

Observe that the proof also reveals an interesting property of the structure of T in case $\widehat{OV}(G,T) = \pi$. This extremal situation can only happen if $\sum_{j\in R} \widehat{\alpha}_j^T + \sum_{j\in L} \widehat{\alpha}_j^T$ equals $\big|\sum_{j\in R} \widehat{\alpha}_j^T - \sum_{j\in L} \widehat{\alpha}_j^T\big|$, which only happens if the orientation of the tour is the same at all vertices.

Theorem 4.5. *Let P be a set of n points in the Euclidean plane, with n being odd. Then there exists a solution to the MaxAngleTSP with reverse objective value π. The tour can be constructed in $O(n \log n)$ time.*

Proof. Assume first that P is in general position, i. e., no three points are on a line. Let $p \in P$ be an extreme point of P, i. e., a vertex of the convex hull $\mathrm{CH}(P)$ of P, and let v be a line through p that separates $P \setminus \{p\}$ into two equal halves. For ease of presentation, we rotate the plane such that v becomes vertical, in a way that the bottommost point of v inside the convex hull of P be p. Let $L \subset P$ be the points to the left of v and $R \subset P$ be the points to the right of it. We have $|L| = |R| = \frac{n-1}{2}$. Let b be a line that (i) contains a point of $l \in L$ and a point of $r \in R$, that (ii) is directed from r to l, and that (iii) has $\mathrm{CH}(L)$ in its left closed half-plane and $\mathrm{CH}(R)$ in its right closed half-plane. See Figure 4.5 for an illustration. We call b the *counterclockwise bitangent* of L and R. Note that p is to the left of b. We set $l_1 := l$ and let l_1 be the successor of p in our tour. Next, we consider the counterclockwise bitangent of $L \setminus \{l_1\}$ and R, which is defined by two points $r' \in R$ and $l' \in L \setminus \{l_1\}$. Note that r' may be different from r. We set $r_1 := r'$ and let r_1 be the successor of l_1 in our tour. In general, we connect l_i to the point r_i that is on the counterclockwise bitangent of $L_i := L \setminus \{l_1, \ldots, l_i\}$ and $R_{i-1} := R \setminus \{r_1, \ldots, r_{i-1}\}$, and connect r_i to the point l_{i+1} that is on the counterclockwise bitangent of L_i and R_i. Finally, we connect $r_{(n-1)/2}$ to p, closing the tour.

We show that the tour makes a clockwise turn at r_i. Observe that l_i is to the right of the counterclockwise bitangent β of L_i and R_{i-1}, as it has been on the counterclockwise bitangent of L_{i-1} and R_{i-1} together with some point $r \in R$; see Figure 4.6. By definition, r_i is on β, and it is also by definition that r_i is to the right of v. Thus v and β divide the plane into four quadrants, where l_i is on the upper-left quadrant, r_i is on β between the lower-right and the upper-right quadrant, and l_{i+1} can only be in the lower-left quadrant or on β between the upper-left and the lower-left quadrant. Hence, the tour makes a clockwise turn at r_i. Note that this also holds when L is empty and we connect $r_{(n-1)/2}$ to p. With the analogous arguments, we see that we always make a clockwise turn at each point in L. Observe that the slope of $l_i r_i$ is less than the slope of β (r_i is on β and l_i is in the upper-right quadrant), while the slope of $r_i l_{i+1}$ is equal to or greater than the slope of β.

Thus, our edges always intersect v, and their slopes are increasing. We can think of an upward-directed copy v' of v that rotates counterclockwise around p until it contains l_1, at which time it continues rotating around l_1 until it contains r_1 and so on. The slope of v' is increasing until it reaches p again, where we can rotate it counterclockwise around p until it matches v again. By this, v' is in total rotated by π, which is thus our reverse objective value. For an illustration of an optimal tour we refer to Figure 4.7.

Finally, observe that our argument also holds if there are three points on a line. In fact, if we are to choose the next point of, say, L and there are two points of L on the current counterclockwise bitangent, then we can choose either of them. The next counterclockwise bitangent will be identical to the current one, and we will end up with an angle of 0. Also, if our halving line v has to contain additional points, we can slightly perturb it to a halving line of $P \setminus \{p\}$ that does not contain any point by a counterclockwise rotation.

It remains to show that our construction can be done in $O(n \log n)$ time. Hershberger and Suri [27] give an $O(n \log n)$ time algorithm to construct a data structure that maintains the convex hull while deleting a sequence of up to n points. The overall cost of deleting these points is $O(n \log n)$. We apply this independently to our sets L and R. Since the data structure of Hershberger and Suri supports binary search [27, p. 254], one can apply an algorithm by Guibas et al. [24, Lemma A.1] to find the counterclockwise bitangent in $O(\log n)$ time in each iteration. □

We remark that the proof is inspired by the technique used by Abellanas [1, Theorem 3.1], where bitangents with both subsets on the same side are used for creating a plane path that alternates between the two subsets.

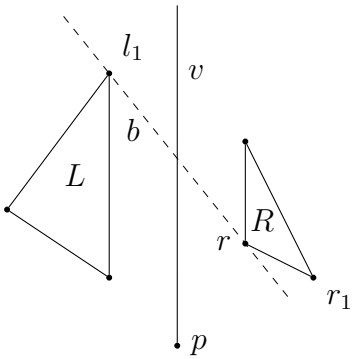

Figure 4.5: Angle-instance with $n = 7$: Construction of an optimal tour: First bitangent.

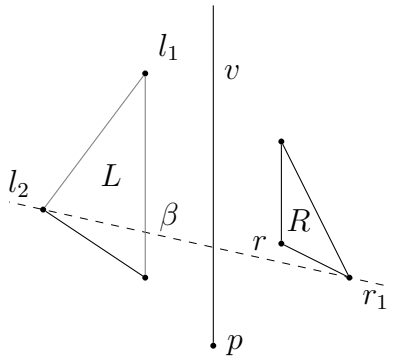

Figure 4.6: Angle-instance with $n = 7$: Construction of an optimal tour: Second bitangent.

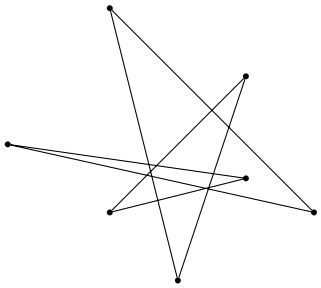

Figure 4.7: Angle-instance with $n = 7$: Optimal MaxAngleTSP tour.

Even case

Lemma 4.6. *Let $G = (V, E)$ be an instance of MaxAngleTSP. If n is even, then for every optimal tour T^* there is:*

$$0 \leq \widehat{OV}(G, T^*) \leq 2\pi. \tag{4.39}$$

Proof. The lower bound is trivial. To show the upper bound we apply the algorithm described in Section 4.5.1 after removing an arbitrary vertex q to obtain a tour $\widetilde{T}$ on $V \setminus \{q\}$. Pick an arbitrary edge ij of $\widetilde{T}$ and replace it by the edges iq and qj, obtaining a tour T on V. The reverse objective value for $\widetilde{T}$ was π, and $\widetilde{T}$ and T only differ in the inner angles at i, j and q. The sum of the changes in these three inner angles is the angle sum of the triangle ijq,

which is π. Hence, the reverse objective value for T can increase by at most π, hence the upper bound of 2π. (Observe that this increase may be less than π if, say, the edge jk of $\widetilde{T}$, $k \neq i$, intersects the interior of the triangle ijq.) □

Note that the lower bound can be reached e.g. if all vertices lie on a line.

4.5.2 Computational results for MaxSQTSP

Let us now focus on the both considered maximization variants of our problem MaxSQTSP and MaxAngleTSP from a computational point of view. The results are summarized in Tables 4.5 and 4.6. They both use the *elementary fractional approach* (column F) as the reference method and thus the ratios in the remaining columns are comparable. In particular, Table 4.5 contains the results for all methods based on the fractional and Table 4.6 for the integral separation process, respectively. As we have seen in Subsection 4.5.1, MaxAngleTSP behaves differently in the odd and in the even case from a theoretical point of view. Thus, we evaluated the Angle- and Angle-Distance-instances separately for n even and odd in our tables.

Angle-instances, n even: Similar to the minimization problems, all introduced integral approaches beat their fractional counterparts. However, different from the minimization case, the stronger subtour elimination constraints improve the performance in the case of maximization. All the methods F(III)/I(III), F(IV)/I(IV) and F(V)/I(V) beat the *elementary integral approach*. Moreover, the methods F(IV)/I(IV) lie between the *elementary approaches* and the variants F(III)/I(III) as one could expect from a theoretical point of view (for details see Section 4.4.1). The best approach among all is I(III) for this instance type.

Angle-instances, n odd: As already proved in Section 4.5.1, the inclusion of subtour elimination constraints is not necessary to solve these instances (at least for points in non-collinear position). However, some subtour elimination constraints can be separated inside the branch and bound tree during the solution process. Consequently, we can observe different solution times caused by the particular methods. In fact, the *elementary fractional approach* performs as well as the *elementary integral approach* and moreover, we cannot observe a clear split between the fractional and integral methods. Among all tested approaches, the method F(III) outperforms the other ones significantly. Contrary to the even case, the variants (II) and (IV) seem to slow down the solution process.

Instance	**F**	**F(I)**	**F(II)**	**F(III)**	**F(IV)**	**F(V)**
Angle_10	0.1	0.99	1.01	0.98	0.95	0.85
Angle_12	1.2	0.96	1.04	0.80	0.88	0.80
Angle_14	6.2	1.04	0.95	0.75	0.95	0.72
Angle_16	263.2	0.89	0.94	0.49	1.03	0.59
Angle_18	4270.8	0.97	1.16	0.38	0.98	0.57
mean		*0.97*	*1.02*	*0.65*	*0.96*	*0.70*
Angle_11	0.3	1.02	0.99	0.91	0.98	0.83
Angle_13	1.1	1.17	1.25	0.93	0.98	0.92
Angle_15	7.3	0.95	1.02	0.69	0.92	0.85
Angle_17	22.9	0.79	1.20	0.82	1.34	1.10
Angle_19	254.1	1.05	1.24	0.41	1.08	0.90
mean		*0.99*	*1.13*	*0.72*	*1.05*	*0.92*
Angle-Distance_16	0.8	0.98	1.00	0.94	1.01	0.97
Angle-Distance_18	1.3	0.99	1.07	0.81	1.04	0.95
Angle-Distance_20	2.8	0.98	1.02	0.56	1.09	0.83
Angle-Distance_22	24.4	0.86	0.89	0.41	0.86	0.55
Angle-Distance_24	50.0	0.97	0.92	0.44	1.18	0.63
Angle-Distance_26	304.8	0.85	1.10	0.42	1.19	0.62
mean		*0.94*	*1.00*	*0.56*	*1.06*	*0.74*
Angle-Distance_41	4.8	0.99	0.98	0.78	1.04	1.02
Angle-Distance_43	5.3	0.94	0.93	0.70	0.91	0.91
Angle-Distance_45	17.3	0.95	0.86	0.51	0.96	0.71
Angle-Distance_47	6.3	1.02	1.06	0.68	0.93	0.86
Angle-Distance_49	16.1	1.21	1.17	0.77	1.18	1.02
Angle-Distance_51	17.7	0.86	0.91	0.45	0.97	0.98
Angle-Distance_53	39.3	1.03	1.05	0.60	0.99	1.00
Angle-Distance_55	32.1	0.91	0.96	0.60	0.81	0.85
Angle-Distance_57	74.2	1.04	0.99	0.42	0.96	1.14
Angle-Distance_59	325.7	0.86	0.74	0.33	0.86	0.78
Angle-Distance_61	63.4	1.11	0.94	0.59	1.00	0.97
Angle-Distance_63	107.7	1.00	0.90	0.48	0.84	0.78
Angle-Distance_65	257.6	0.84	0.99	0.32	0.86	0.77
mean		*0.98*	*0.95*	*0.54*	*0.94*	*0.90*
Random_20	4.1	0.95	0.94	1.02	0.93	1.04
Random_25	39.8	0.90	1.02	1.01	0.75	0.92
Random_30	425.0	0.77	0.87	0.89	0.70	0.97
Random_35	3911.1	0.88	1.15	1.11	0.75	0.91
mean		*0.87*	*0.99*	*1.00*	*0.78*	*0.96*

Table 4.5: Maximization case: Comparing of the running times. We compare the elementary fractional approach I with approaches using different variants of subtour elimination constraints.

Angle-Distance-instances, n even: The variants F(III)/I(III) and F(V)/ I(V) yield the best running times in this case, similarly as for the Angle-instances with n even. We can also observe that the integral approaches outperform their fractional counterparts with the notable exceptions of methods I(II) and I(IV) which surprisingly perform extremely poor. Although the entries greater than 20 are always caused by one single outlier instance, also the other ratios are quite high. The overall best running times for this instance type are derived with the methods I(III) and I(V), where I(III) slightly outperforms I(V).

Instance	F	I	I(I)	I(II)	I(III)	I(IV)	I(V)
Angle_10	0.1	0.95	0.95	0.97	0.96	0.96	0.83
Angle_12	1.2	0.95	0.97	1.05	0.77	0.86	0.75
Angle_14	6.2	0.86	0.84	1.12	0.69	0.96	0.71
Angle_16	263.2	0.62	0.58	1.04	0.44	1.04	0.49
Angle_18	4270.8	0.40	0.32	0.62	0.33	0.72	0.42
mean		*0.72*	*0.68*	*0.94*	*0.59*	*0.90*	*0.62*
Angle_11	0.3	0.97	0.94	0.93	0.92	0.90	0.81
Angle_13	1.1	1.00	0.96	1.16	1.01	1.11	0.84
Angle_15	7.3	0.92	0.77	1.02	0.83	1.04	0.92
Angle_17	22.9	1.34	1.08	1.34	0.92	1.23	0.97
Angle_19	254.1	0.83	0.79	0.88	0.69	0.89	0.72
mean		*1.00*	*0.90*	*1.05*	*0.87*	*1.03*	*0.85*
Angle-Distance_16	0.8	1.05	0.98	1.64	0.93	1.69	0.99
Angle-Distance_18	1.3	1.03	1.11	2.28	0.79	2.20	0.73
Angle-Distance_20	2.8	1.07	1.02	5.91	0.61	4.73	0.62
Angle-Distance_22	24.4	0.69	0.65	22.31	0.33	21.24	0.44
Angle-Distance_24	50.0	0.84	0.69	9.04	0.46	8.57	0.48
Angle-Distance_26	304.8	0.73	0.59	20.80	0.37	12.65	0.45
mean		*0.89*	*0.81*	*6.73*	*0.54*	*5.86*	*0.59*
Angle-Distance_41	4.8	1.16	1.20	1.30	0.91	1.35	1.11
Angle-Distance_43	5.3	1.03	1.01	0.88	0.99	0.96	1.10
Angle-Distance_45	17.3	0.98	1.04	0.96	0.74	1.27	0.99
Angle-Distance_47	6.3	1.16	1.19	1.11	0.86	1.11	0.91
Angle-Distance_49	16.1	1.13	1.22	1.27	1.11	1.24	1.06
Angle-Distance_51	17.7	1.03	1.08	0.96	0.62	1.10	0.81
Angle-Distance_53	39.3	1.26	0.97	1.29	0.89	1.18	0.89
Angle-Distance_55	32.1	0.82	0.96	0.83	0.80	1.02	0.82
Angle-Distance_57	74.2	1.32	1.27	1.19	0.71	1.45	0.83
Angle-Distance_59	325.7	0.90	0.98	1.05	0.60	0.77	0.69
Angle-Distance_61	63.4	0.94	0.89	0.87	0.71	1.01	0.82
Angle-Distance_63	107.7	1.16	1.13	1.24	0.75	1.45	0.93
Angle-Distance_65	257.6	0.80	0.85	0.77	0.51	0.69	0.60
mean		*1.04*	*1.05*	*1.04*	*0.77*	*1.10*	*0.88*
Random_20	4.1	0.81	0.93	0.85	0.86	0.83	0.98
Random_25	39.8	0.80	0.90	0.64	0.89	0.73	0.88
Random_30	425.0	0.86	0.72	0.73	0.72	0.74	0.97
Random_35	3911.1	1.02	0.80	0.77	0.87	0.74	0.93
mean		*0.87*	*0.83*	*0.74*	*0.83*	*0.76*	*0.94*

Table 4.6: Maximization case: Comparing of the running times. We compare the elementary integral approach I with the elementary fractional approach F and with approaches using different variants of subtour elimination constraints.

Angle-Distance-instances, n odd: For these instances the trends are less clear. We can see that even the absolute running times in the first column do not increase monotonically and similar variances can be observed in other columns. Looking at all the table entries and not only the mean values, the method F(III) provides the best performance. Moreover, apart from the methods F(V)/I(V) the fractional approaches outperform their integral counterparts.

A rather surprising observation shows that many instances yield the same optimal tour as it would be obtained by solving the corresponding Angle-instances on the same point sets. Of course, the optimization goals of maximizing the turning angles and the distances are not contradictory, but we could observe that even for $\rho = 0$ we often got an optimal MaxAngleTSP tour. For these instances, the optimal tour does not change by increasing ρ, however, a significant increase of the running time occurs for larger values of ρ.

Random-instances: Finally, the integral methods mostly beat their fractional counterparts for Random-instances. The methods I(II) and I(IV) yield the best running times, although I(I) and I(III) do not lag far behind.

Summing up the results for the maximization problem we cannot identify a clear winner for all instance classes. Different from the minimization case, the variants of stronger subtour elimination constraints improve the performance in many cases. In particular, the stronger subtour elimination constraints (III) improve the performance for all angle-based test instances. However,

- the integral approach outperforms the fractional one if n is even and
- the fractional approach outperforms the integral one if n is odd

for both the Angle- and Angle-Distance-instance classes.

5. Conclusions and outlook

This PhD thesis deals with three different problems: the *data arrangement problem on regular trees* (*DAPT*), the *traveling salesman problem* (*TSP*) and the *symmetric quadratic traveling salesman problem* in its minimization (*SQTSP*) and maximization (*MaxSQTSP*) version. Our contribution related to the DAPT focuses on some approximable special cases of the problem and on its complexity. Our contributions concerning the TSP, SQTSP and MaxSQTSP address computational tests and a special case of the MaxSQTSP.

Summary of the contribution concerning the DAPT. The DAPT is an $\mathcal{NP}$-hard special case of the *generic graph embedding problem* (*GEP*). In Chapter 2 we deal with the case where both the guest and the host graph are d-regular trees of heights h_G and $h = h_G + 1$, respectively. After identifying some basic properties, we focus on binary regular trees first (i.e. on the case $d = 2$). In particular, an approximation algorithm with approximation ratio $\frac{203}{200}$ is proposed. Moreover, we provide a closed formula for the objective function value of the arrangement generated by that approximation algorithm.

The analysis of the approximation algorithm and the estimation of the approximation ratio involve a special case of the *k-balanced partitioning problem* (*k-BPP*) as an auxiliary problem. More precisely, we give a formula and a lower bound for the optimal objective function value of the k-BPP, where the input graph is a binary regular tree and k is a power of 2.

Further, we investigate the relationship between the considered particular cases of the DAPT and the k-BPP and derive a lower bound for the value of the objective function of the $DAPT(G, 2)$. This is done by solving h_G instances of the $2^{k'}$-*BPP*, where h_G is the height of the guest graph G and $k' = 1, 2, \ldots, h_G$. This lower bound then leads to the above mentioned approximation ratio.

After that we generalize the introduced algorithm for all d-regular trees, where $d \geq 3$ is arbitrary but fixed. A weaker general lower bound based on a

similar partitioning problem is provided. In particular, we use the property that the objective function value $c(G, \mathscr{V})$ of every partition $\mathscr{V}$ equals the overall number of the connected components of the subgraphs induced in $G = (V, E)$ by the partition sets of V minus 1. The following analysis leads to a proof of a $\frac{585}{392}$-approximation ratio for the generalized algorithm.

Finally we consider the complexity of the DAPT and settle an open question from the literature. We prove that the DAPT with a host graph being a d-regular tree, $d \geq 2$, $d \in \mathbb{N}$, is $\mathcal{NP}$-hard, even if the guest graph is a tree. The proof is done by means of a reduction from the *numerical matching with target sums* (*NMTS*) problem.[1]

However, the complexity of the DAPT in the case where both the guest and the host graph are d-regular trees, where $d \geq 2$ is fixed, remains an open question. Other open questions concern the more general cases of the DAPT where the guest graph G is a d_G-regular tree and the host graph T is a d-regular tree with $d_G \neq d$.

It would be also interesting to investigate whether some alternative analysis of the proposed algorithm for the $DAPT(G, d)$ could lead to a better approximation ratio. Numerical results provide some hints that the answer to this question might be "yes" and suggest an empirical approximation ratio of 1.0098 in the case of binary regular trees (i.e. for $d = 2$) and 1.2038 in the general case (i.e. for every fixed $d \geq 2$).

Summary of the contribution concerning the TSP. In Chapter 3 we provide a "test of concept" of a very simple approach to solve TSP instances of medium size to optimality by exploiting the power of state-of-the-art ILP solvers. The approach consists of iteratively solving incomplete ILP models with relaxed subtour elimination constraints to integer optimality. Then it is easy to identify integral subtours in the current optimal solution and add the corresponding subtour elimination constraints to the ILP model. Iterating this process until no more subtours are contained in the solution obviously solves the TSP to optimality.

Further, we focus on the structure of subtour elimination constraints and address the question of how to find a "good" set of subtour elimination constraints in reasonable time. Therefore we aim to identify the local structure of the vertices of a given TSP instance by running a clustering algorithm. Based on empirical observations and results from random graph theory we extend this clustering-based approach and develop a hierarchical clustering method with a mechanism to identify subtour elimination constraints as "relevant"

[1] This reduction could also be used to prove the $\mathcal{NP}$-hardness of the DAPT if the guest graph is limited to be a union of more stars.

if they appear in consecutive iterations of the algorithm.

Additionally, we provide some theoretical results for random Euclidean graphs. In particular, we apply proof techniques from literature to show that the expected length of an optimal 2-matching (i.e. the objective function value of the first iteration of our algorithm) is asymptotically equal to $\beta\sqrt{n}$, where β is a constant, if uniformly distributed random points in the Euclidean plane are considered. This might be the first step towards a more exhaustive theoretical analysis of our approach.

We mostly refrained from adding features which are highly likely to improve the performance considerably. These include starting heuristics (cf. Section 3.3.4), lower bounds or adding additional cuts. In the future it would be interesting to explore the limits of the performance one can reach with a pure integer linear programming approach by adding these improvements. Clearly, we cannot expect such a basic approach to compete with the performance of *Concorde* (see APPLEGATE et al [6]), which has been developed over many years and basically includes all theoretical and technical developments known so far. However, it turns out that most of the standard benchmark instances with up to 400 vertices can be solved in a few minutes by this pure integer strategy.

Summary of the contribution concerning the SQTSP and MaxSQTSP. In Chapter 4 we deal with the *symmetric quadratic traveling salesman problem* (*SQTSP*) and its geometric variant on the Euclidean plane, namely the *Angular-Metric Traveling Salesman Problem* (*AngleTSP*) where the costs correspond to the *turning angles* of the tour. Moreover, we introduce the maximization variants *MaxSQTSP* and *MaxAngleTSP* which have not been treated in the literature before.

Contributing to ILP-based solution approaches, we consider a standard linearization and test a purely integral subtour elimination process. This basic approach turns out to significantly outperform the standard fractional separation procedure known from the literature for all types of test instance in all problem variants. After that we implant other kinds of subtour elimination constraints introduced by FISCHER and HELMBERG [21] into the separation process. Although stronger from a theoretical point of view, the usage of these constraints consistently leads to larger computational times in the minimization variants of the problem as more non-zero entries are contained in the constraint matrix.

We also introduce a completely new, geometrically based MILP linearization for the AngleTSP which involves only a linear number of additional variables while the standard linearization requires a cubic number. The downside

of this approach are larger root node gaps which may be the reason for worse running times of this approach.

In the second part of the chapter we deal with the MaxSQTSP. From a computational point of view, in the maximization case – different from the minimization case – some of the subtour elimination constraints suggested by FISCHER and HELMBERG [21] do speed up the solution process in many cases. The comparison between purely integral and fractional subtour elimination is less clear and depends on the particular type of test instances. It also turns out that for geometrically based test instances a dramatic difference between the cases of even and odd number of vertices can be observed which led us to interesting theoretical results.

If n is odd, we can show a surprising result for MaxAngleTSP by a geometric argument: In this case the optimal value of the reverse objective function (sum of inner turning angles) is always equal to π. This also implies that the solution of the standard ILP model will never produce subtours and no integral separation occurs at all. Nevertheless, the solution times for this simple ILP increase dramatically in the maximization case. Fortunately, this result can be complemented by a characterization of the optimal solution structure and we can also provide a simple constructive polynomial time algorithm to find such an optimal solution. On the other hand, if n is even, it can be shown that the optimal value of the reverse objective function can be any value between 0 and 2π.

Note that the complexity status of MaxAngleTSP for n even remains an open question. It is also unknown whether the upper bound 2π on the optimal value of the reverse objective function can be reached.

From a computational point of view, it could be also interesting to investigate whether the pure integer linear programming approach outperforms the standard one for TSP problems of higher than quadratic order.

Bibliography

[1] M. Abellanas, J. García, G. Hernández, M. Noy, and P. Ramos, "Bipartite embeddings of trees in the plane," *Discrete Applied Mathematics*, vol. 93, no. 2–3, pp. 141–148, 1999.

[2] T. Achterberg, "SCIP: solving constraint integer programs," *Mathematical Programming Computation*, vol. 1, no. 1, pp. 1–41, 2009. Available at: `http://mpc.zib.de/index.php/MPC/article/view/4`.

[3] A. Aggarwal, D. Coppersmith, S. Khanna, R. Motwani, and B. Schieber, "The angular-metric traveling salesman problem," *SIAM Journal on Computing*, vol. 29, no. 3, pp. 697–711, 2000.

[4] E. Amaldi, G. Galbiati, and F. Maffioli, "On minimum reload cost paths, tours, and flows," *Networks*, vol. 57, no. 3, pp. 254–260, 2011.

[5] K. Andreev and H. Räcke, "Balanced graph partitioning," *Theory of Computing Systems*, vol. 39, no. 6, pp. 929–939, 2006.

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

[7] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi, *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer-Verlag Berlin Heidelberg, 1999.

[8] J. Beardwood, J. H. Halton, and J. M. Hammersley, "The shortest path through many points," *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 55, no. 4, pp. 299–327, 1959.

[9] R. Bosch, "Connecting the dots: The ins and outs of TSP art," in *Proceedings of Bridges Leeuwarden: Mathematics, Music, Art, Architecture, Culture* (R. Sarhangi and C. H. Séquin, eds.), pp. 235–242, Southwestern College, 2008.

[10] E. Çela and R. Staněk, "Heuristics for the data arrangement problem on regular trees," *Journal of Combinatorial Optimization*, vol. 30, no. 3, pp. 768–802, 2015.

[11] E. Çela, J. Schauer, and R. Staněk, "The data arrangement problem on binary trees," 2015. Submitted for publication to *Algorithmica*, available as: `arXiv:1512.08404`.

[12] F. R. K. Chung, "On optimal linear arrangements of trees," *Computers & Mathematics with Applications*, vol. 10, no. 1, pp. 43–60, 1984.

[13] H. Crowder and M. W. Padberg, "Solving large-scale symmetric travelling salesman problems to optimality," *Management Science*, vol. 26, no. 5, pp. 495–509, 1980.

[14] G. Dantzig, R. Fulkerson, and S. Johnson, "Solution of a large-scale traveling-salesman problem," *Journal of the Operations Research Society of America*, vol. 2, no. 4, pp. 393–410, 1954.

[15] S. S. Dey, M. Molinaro, and Q. Wang, "How good are sparse cutting-planes?," in *Integer Programming and Combinatorial Optimization: 17th International Conference, IPCO 2014, Bonn, Germany, June 23–25, 2014, Proceedings* (J. Lee and J. Vygen, eds.), pp. 261–272, Springer International Publishing, 2014.

[16] A. Dumitrescu, J. Pach, and G. Tóth, "Drawing hamiltonian cycles with no large angles," *The Electronic Journal of Combinatorics*, vol. 19, no. 2, 2012.

[17] S. P. Fekete and G. J. Woeginger, "Angle-restricted tours in the plane," *Computational Geometry: Theory and Applications*, vol. 8, no. 4, pp. 195–218, 1997.

[18] A. E. Feldmann and L. Foschini, "Balanced partitions of trees and applications," *Algorithmica*, vol. 71, no. 2, pp. 354–376, 2015.

[19] A. Fischer, F. Fischer, G. Jäger, J. Keilwagen, P. Molitor, and I. Grosse, "Exact algorithms and heuristics for the quadratic traveling salesman problem with an application in bioinformatics," *Discrete Applied Mathematics*, vol. 166, pp. 97–114, 2014.

[20] A. Fischer, "A polyhedral study of quadratic traveling salesman problems." PhD Thesis, Chemnitz University of Technology, Department of Mathematics, 2013. Available at: `http://nbn-resolving.de/urn:nbn:de:bsz:ch1-qucosa-118345`.

[21] A. Fischer and C. Helmberg, "The symmetric quadratic traveling salesman problem," *Mathematical Programming*, vol. 142, no. 1, pp. 205–254, 2013.

[22] M. R. Garey and D. S. Johnson, "Computers and intractability: A guide to the theory of NP-completeness," in *A Series of Books in the Mathematical Sciences* (V. Klee, ed.), W. H. Freeman and Company, San Francisco, 1979.

[23] M. Grötschel and O. Holland, "Solution of large-scale symmetric travelling salesman problems," *Mathematical Programming*, vol. 51, no. 1, pp. 141–202, 1991.

[24] L. Guibas, J. Hershberger, and J. Snoeyink, "Compact interval trees: a data structure for convex hulls," *International Journal of Computational Geometry & Applications*, vol. 1, no. 1, pp. 1–22, 1991.

[25] G. Gutin and A. P. Punnen, eds., *The Traveling Salesman Problem and Its Variations*, vol. 12 of *Combinatorial Optimization*. Springer US, 2007.

[26] K. Helsgaun, "LKH – version 2.0.2." Website, 2008. Available at: `www.akira.ruc.dk/~keld/research/LKH/LKH-2.0.2.tgz`.

[27] J. Hershberger and S. Suri, "Applications of a semi-dynamic convex hull algorithm," *BIT Numerical Mathematics*, vol. 32, no. 2, pp. 249–267, 1992.

[28] G. Jäger and P. Molitor, "Algorithms and experimental study for the traveling salesman problem of second order," *Lecture Notes in Computer Science*, vol. 5165, pp. 211–224, 2008.

[29] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*. Prentice Hall PTR, 1988.

[30] M. Juvan and B. Mohar, "Optimal linear labelings and eigenvalues of graphs," *Discrete Applied Mathematics*, vol. 36, no. 2, pp. 153–168, 1992.

[31] M. Koivisto, "Partitioning into sets of bounded cardinality," in *Parameterized and Exact Computation* (J. Chen and F. V. Fomin, eds.), pp. 258–263, Springer Berlin Heidelberg, 2009.

[32] B. Korte and J. Vygen, *Combinatorial Optimization: Theory and Algorithms*. Springer-Verlag Berlin Heidelberg, 5th ed., 2012.

[33] R. Krauthgamer, J. S. Naor, and R. Schwartz, "Partitioning graphs into balanced components," in *SODA '09 Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms* (C. Mathieu, ed.), pp. 942–949, Society for Industrial and Applied Mathematics, 2009.

[34] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, eds., *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Wiley-Interscience Series in Discrete Mathematics and Optimization, 1985.

[35] M. J. Luczak and S. D. Noble, "Optimal arrangement of data in a tree directory," *Discrete Applied Mathematics*, vol. 121, no. 1–3, pp. 307–315, 2002.

[36] J. F. Meier and U. Clausen, "Solving classical and new single allocation hub location problems on euclidean data," 2015. Available as: Optimization Online 2015-03-4816.

[37] P. Miliotis, "Integer programming approaches to the travelling salesman problem," *Mathematical Programming*, vol. 10, no. 1, pp. 367–378, 1976.

[38] D. Naddef and S. Thienel, "Efficient separation routines for the symmetric traveling salesman problem II: separating multi handle inequalities," *Mathematical Programming*, vol. 92, no. 2, pp. 257–283, 2002.

[39] T. Öncan, İ. Kuban Altınel, and G. Laporte, "A comparative analysis of several asymmetric traveling salesman problem formulations," *Computers & Operations Research*, vol. 36, no. 3, pp. 637–654, 2009.

[40] M. Padberg and G. Rinaldi, "An efficient algorithm for the minimum capacity cut problem," *Mathematical Programming*, vol. 47, no. 1, pp. 19–36, 1990.

[41] M. Penrose, *Random Geometric Graphs*. Oxford University Press, 2003.

[42] U. Pferschy and R. Staněk, "Using pure integer solutions to solve the traveling salesman problem," in *Proceedings of the 16th International Multiconference Information Society* (M. Gams *et al.*, eds.), vol. A, pp. 565–568, Institut »Jožef Stefan«, 2013.

[43] U. Pferschy and R. Staněk, "Generating subtour elimination constraints for the TSP from pure integer solutions," 2015. Available as: `arXiv:1511.03533`.

[44] U. Pferschy and R. Staněk, "Generating subtour elimination constraints for the TSP from pure integer solutions," 2016. To appear in *Central European Journal of Operations Research*.

[45] G. Reinelt, "TSPLIB95." Website, 1995. Available at: `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/`.

[46] G. Reinelt, *The Traveling Salesman: Computational Solutions for TSP Applications*. Springer-Verlag Berlin Heidelberg, 1994.

[47] W. T. Rhee, "A matching problem and subadditive Euclidean functionals," *The Annals of Applied Probability*, vol. 3, no. 3, pp. 794–801, 1993.

[48] K. Savla, E. Frazzoli, and F. Bullo, "Traveling salesperson problems for the Dubins vehicle," *IEEE Transactions on Automatic Control*, vol. 53, no. 6, pp. 1378–1391, 2008.

[49] A. Schrijver, "Combinatorial optimization: Polyhedra and efficiency," in *Algorithms and Combinatorics* (R. L. Graham *et al.*, eds.), Springer-Verlag Berlin Heidelberg, 2003.

[50] Y. Shiloach, "A minimum linear arrangement algorithm for undirected trees," *SIAM Journal on Computing*, vol. 8, no. 1, pp. 15–32, 1979.

[51] R. Staněk, "Heuristiken für das optimale data-arrangement-problem in einem baum." Master Thesis, Graz University of Technology, 2012. In German, available at: `http://www.rostislavstanek.at/daten/Masterarbeit.pdf`.

[52] R. Tošić and O. Bodroža, "On the number of hamiltonian cycles of $P_4 \times P_n$," *Indian Journal of Pure and Applied Mathematics*, vol. 21, no. 5, pp. 403–409, 1990.

[53] J. E. Yukich, *Probability Theory of Classical Euclidean Optimization Problems*. Springer-Verlag Berlin Heidelberg, 1998.