Diplomarbeit

Steiner Tree Problems in the Analysis of Biological Networks

Nadja Betzler

Betreuer:	Prof. Klaus-Jörn Lange Prof. Mike Hallett Dr. Jens Gramm Prof. Rolf Niedermeier
begonnen am:	15. August 2005
beendet am:	15. Februar 2006

Arbeitsbereich für theoretische Informatik/Formale Sprachen
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen

Contents

Chapter 1

Introduction

Networks provide a useful tool to present information about molecular processes in a cell. Prominent examples are metabolic networks or networks based on protein-protein interactions, that allow for a compact and concise representation of biochemical data. The analysis of biological networks is an important field of research that can be used to gain a deeper understanding of regulatory mechanisms [ILB04] or to identify important molecules or interactions [ITR$^+$01]. As the size of the considered networks usually is too large for a complete manual analysis—even for simple organisms there are networks consisting of thousands of vertices—it is necessary to develop methods for an automatic analysis. Naturally, it therefore makes sense to use well-known graph algorithms for this purpose or to formulate the analysis goal as graph-theoretical problem and then develop solution strategies for it as it is done in this work.

In the analysis of biological networks with methods from bioinformatics there have been many recent successes. We only mention the following two examples: There have been advances in the fully automated detection of important subnetworks or paths by different graph-algorithmic approaches [IOSS02], [SIKS05]. As another example, it was possible to achieve a better understanding of the regulation of genes or so-called transcriptional regulatory processes [ILB04], [LRR$^+$02].

Our kind of analysis in this work is based on Steiner tree related problems. A *Steiner tree* spans a given set of vertices in a graph. The Steiner Tree in Graphs problem itself can be stated as follows.

> Steiner Tree in Graphs (STG)
> **Input:** An undirected graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}^{\geq 0}$, and a set of *distinguished vertices (or terminals)*

$S \subseteq V$.
Task: Find a minimum weight tree $T = (V_T, E_T)$ that spans S.
The weight of a tree T is defined as $w(T) = \sum_{e \in E_T} w(e)$.

The NP-hard Steiner Tree in Graphs problem is an immensely well-studied problem to which several books have been devoted, e.g. [DRS00] and [PS02]. Furthermore, a wide range of different kinds of algorithmic approaches has been developed ([DW72], [RZ00], and [PV02]) and there is a large amount of publications dealing with preprocessing methods such as data reduction ([DV89], [Dui00], [KM98]). This work considers some less studied variants of STG, like its vertex-weighted case, which are extremely useful for the analysis of biological networks as shown by the working group of Hallett [SPB+05] and therefore provide a fruitful link between graph theory and molecular biology. One contribution of this work was the development of a software tool, called *Steiner Package*, as part of a project in the group of Mike Hallett. More precisely, it can be described as follows:

Steiner Package The Steiner Package can be used for the computation of Steiner trees in biological networks and integrates different kinds of preprocessing and data reduction rules with algorithms. More precisely, I contributed the following features:

- Design and implementation of new data reduction rules

  Data reduction rules are an important technique for exactly solving NP-hard problems. Here, we evaluate known data reduction rules for STG and propose a set of new data reduction rules for STG and its variants. Our experimental evaluations show that for many instances the data reduction rules are essential to solve the problem exactly.

- Implementation of an approximation algorithm

  In addition to an already existing implementation of the exact Dreyfus-Wagner [DW72] algorithm, we implemented the approximation algorithm by Klein and Ravi [KR95] that can be applied to all instances.

- Biological preprocessing

  Besides graph-theoretical data reduction rules, we introduce some preprocessing that is based on the biochemical meaning of the considered graph vertices. These rules turned out to become extremely useful to obtain biological relevant solutions in case the graph-theoretical approaches alone were not sufficient.

- Structure of the program

  We worked out the design of the overall program structure, including the determination of a reasonable order for preprocessing and data reduction rules.

- Experimental results

  We carried out experimental tests for different instances and provide some promising results. In all considered cases the running time could be significantly improved by applying the data reduction rules. Furthermore, we were able to compute exact solutions for instances that could not be solved by the Dreyfus-Wagner algorithm alone.

Whereas the first part of this work is concerned with the development of a software tool, in the second part, we investigate Steiner tree related problems from a parameterized point of view. Generally, fixed-parameter algorithms can allow for efficient algorithms for some NP-hard problems as they have a running time that is only exponential in a specified part of the input, called *parameter*. If we consider the number of terminals as parameter for STG, there is a classical fixed-parameter algorithm: the Dreyfus-Wagner algorithm [DW72]. In contrast, in literature there are no fixed-parameter algorithms or negative results for the other variants of STG discussed in this work. This yields the second main contribution of this work that we can state as follows:

Parameterized complexity study We discuss the relevance of other graph-theoretical (Steiner tree related) problems for the analysis of biological networks. Then, we start a theoretical study and provide the first parameterized complexity analysis for Steiner tree related problems, that

- contains a systematic analysis of the parameterized complexity of a range of problem variants with respect to a number of parameterizations. Thereby, we gain insight into the problem structure and illustrate some facets of intractability.

- includes new fixed-parameter algorithms and hardness results.

- uses a new technique for the design of fixed-parameter algorithms that combines color coding [AYZ95], a classical parameterized approach, with enumeration.

- answers an open question posed by Hallett [Hal04] about the parameterized complexity of the so-called Generalized Vertex-Weighted

STEINER TREE IN GRAPHS (GV-STG)[1] problem; independently from this work, this question was also answered in [SIKS05]. In contrast to STG, the input of GV-STG does not contain a set of terminals. It consists of searching a subgraph of a given size such that its weight does not exceed a given threshold.

The structure of the work is as follows. In Chapter 2 we start with a summary of results regarding biological networks in combination with graph algorithms and provide an overview of their applications. Chapter 3 is concerned with a new approach, called *Steiner method*, that was first introduced by Scott et al. [SPB+05] and uses the graph-theoretical problem STEINER TREE IN GRAPHS, for a new kind of network analysis. The main contribution of this chapter is the development of a software tool for the Steiner method, which includes the design and implementation of new data reduction and preprocessing rules. Furthermore, we illustrate their usefulness by providing practical examples from molecular biology. In Chapter 4, we suggest new Steiner tree related problems and discuss their relevance for the analysis of biological networks. We provide the first parameterized complexity study for Steiner tree related problems that yields new fixed-parameter algorithms and hardness results.

1.1 Preliminaries and notation

The computational problems we will study in this work are based on graphs (or networks). A *graph* is denoted by $G = (V, E)$, where V is the set of *vertices* and E is the set of *edges*. In an *undirected graph* an edge $\{u, v\} \in E$ is an unordered pair of vertices and in a *directed graph* an edge $(u, v) \in E$ is an ordered pair of vertices. If not stated otherwise, n refers to the number of vertices in a graph, and m refers to the number of edges. To stress that the vertices V (or edges E, respectively) belong to G, we sometimes denote them as $V(G)$ (or $E(G)$, respectively).

A *subgraph* $G' = (V', E')$ of G is a graph with $V' \subseteq V$ and $E' \subseteq E \cap V' \times V'$. For a subset $V' \subseteq V$, the subgraph of G *induced* by V' is denoted by $G[V'] = (V', E')$, where $E' := E \cap (V' \times V')$.

The *(open) neighborhood* of a vertex v in graph $G = (V, E)$ is defined as $N(v) := \{u \mid \{u, v\} \in E\}$, and the *closed neighborhood* is defined as $N[v] := N(v) \cup \{v\}$. We write $deg(v)$ for the *degree* of vertex v, where $deg(v) := |N(v)|$.

[1] Also known as VERTEX-WEIGHTED k-CARDINALITY TREE problem.

In an undirected graph $G = (V, E)$, a *path* between two vertices $u, v \in V$ is a set of edges $e_1, ..., e_l \in E$ such that $u \in e_1$, $v \in e_l$, and $|e_i \cap e_{i+1}| = 1$, for $1 \leq i \leq l-1$, and $e_i \cap e_j = \emptyset$, for $1 \leq i, j \leq l$ with $|i - j| > 1$. A graph is *connected* if every pair of vertices is connected by a path. A *connected component* of a graph G is a maximal connected subgraph of G.

Two graphs $G = (V, E)$ and $G' = (V', E')$ are *isomorphic* if there exists a bijection $g : V \to V'$ such that $\{u, v\} \in G$ if and only if $\{g(u), g(v)\} \in E'$.

A *forest* is an acyclic graph and a *tree* is an acyclic connected graph. A *subtree* of G is an acyclic connected subgraph of G, and a *subforest* consists of subtrees. A *spanning tree* of a connected graph G is a tree T that is a subgraph of G and uses all vertices of G.

In some cases we need to modify a graph $G = (V, E)$. To *contract* two vertices u and $v \in V$ results in a modified graph $G' = (V', E')$ in which u and v are replaced by a new vertex v', formally, we get

$$V' = (V \backslash \{u, v\}) \cup \{v'\}$$

and

$$E' = E \backslash (\{\{v, n\} \mid n \in N[v]\} \cup \{\{u, n\} \mid n \in N[u]\}) \cup \{\{v', n\} \mid n \in (N(v) \backslash u)\} \cup \{\{v', n\} \mid n \in (N(u) \backslash v)\}$$

.

1.2 Fixed-parameter tractability

In this work, we consider NP-hard problems, i.e. problems that are not likely to be solved in polynomial time. There are several approaches like randomized algorithms, approximation algorithms or heuristic methods, that deal with NP-hard problems. None of them can guarantee to obtain optimal solutions. An exact method that can be applied to NP-hard problems with a specific problem structure, is provided by so-called "fixed-parameter algorithms". Here, we obtain algorithms that are only exponential in the size of a part of the input, called *parameter*, that means the seemingly inherent "combinatorial explosion" can be restricted to a hopefully small part of the input. Fixed-parameter algorithms turned out to be very useful for the problems considered in this work. The Steiner Tree in Graphs problem itself is a prominent representative for a problem that can be solved efficiently with this approach.

We give some basic definitions of parameterized complexity theory. For further information we refer to [DF99, Nie06].

Definition 1.1. *A* parameterized problem *is a language $L \subseteq \Sigma^* \times \Sigma^*$, where Σ is a finite alphabet. The second component is called the* parameter.

Now, we can introduce the concept of fixed-parameter tractability.

Definition 1.2. *A parameterized problem L is* fixed-parameter tractable *if the question "$(x_1, x_2) \in L$?" can be decided in running time $f(|x_2|) \cdot |x_1|^{O(1)}$, where f is an arbitrary computable function on nonnegative integers. The associated complexity class containing all parameterized problems that are fixed-parameter tractable is called* FPT.

We call an algorithm that can solve a parameterized problem in a running time as given in Definition 1.2 *fixed-parameter algorithm*. There are several general techniques to design fixed-parameter algorithms. The most common ones are search trees, dynamic programming or data reduction by preprocessing such that the size of the reduced instance depends only on the parameter. To show that a problem is not in FPT Downey and Fellows [DF99] developed a completeness program analogously to classical complexity theory as described in Section 4.3.

Chapter 2

Biochemical networks

Depending on environmental influences and their stage of development cells with exactly the same genetic information can fulfill a wide range of functions. The functionality of a single cell is therefore determined by a complicated interplay between proteins, genes, and other biochemical components. Although the fine details of cellular regulation are not well understood, recent research has significantly advanced our knowledge of how the various regulators of a cell interact and influence each other at a global level. For example, the simple eukaryote yeast has been extensively studied in this regard, and large databases with information about gene and protein expression and interaction as well as regulatory data are widely available. These databases are an important resource for the study of protein- and gene-regulatory dynamics.

In this chapter, we describe how large-scale databases can be used for achieving a deeper understanding of cellular processes and their regulation. We focus on solution strategies that involve the inference of networks from biological data. We describe some sources of biological data on basis of which networks are modeled. Furthermore, we explain for some types of networks how they are derived from the data. In particular we point out how graph-theoretic substructures in these networks can be interpreted. Discussing example studies from the literature, we show scenarios where these models have been used to obtain biologically meaningful insights into the data by a graph-theoretic analysis of their networks. In all this, we concentrate on databases, network models, and example studies that are related to topics covered in this thesis.

We start with an explanation of some basic biochemical methods and terminology that are further needed for the understanding of this chapter

(Section 2.1). As in the remainder of this work used for the computation of vertex weights, we explain briefly how large-scale information about so called "differentially expressed" genes can be obtained (Section 2.2). In Section 2.3, we give a short overview of publicly available databases relevant to this work. Next, we give some examples of network types from the literature required for this work and explain how they have been used to develop new insights about cellular regulation (Section 2.4).

2.1 Biochemical basics

In this section we very roughly explain some basic terms of molecular biology that are relevant for the understanding of this work. For further information about basic concepts of biochemistry we refer to [BST02].

Gene expression Gene (or protein) expression describes the process in which genetic information is converted into cellular processes and structures. As generally every gene encodes for one protein, gene expression basically consists of the production of proteins in the cell. Proteins influence the cell in all kinds of different ways: catalyzing reactions, working on cellular structure, or influencing gene expression. The state of a cell is determined by a number of different proteins and their corresponding cellular concentrations, called *expression levels*.

Gene expression is a multi-step process. Information is transferred from *DNA* to a transmitter molecule, called *mRNA*, and then from *mRNA* to protein. The cellular concentration of a protein often correlates to the concentration of its *mRNA* in the cell, so measurements of *mRNA* levels, often performed in a high-throughput fashion with microarrays, can give a rough indication of protein levels.

Microarrays Microarray analysis makes it possible to determine the cellular concentration of *mRNA* at a large scale. It was firstly published as Serial Analysis of Gene Expression [VZVK95] and is now widely used. A *DNA* microarray is a collection of *DNA* spots that are attached to a solid matrix to form a 2-dimensional array. A *DNA* spot consists of short single-stranded *DNA* that is characteristic of a specific gene. The *mRNA* of a cell can then be isolated, marked with fluorescent tags, and bound to the complementary *DNA* on the microarray. The resulting fluorescent-tag intensity is a measure of the relative expression of the corresponding gene.

Transcription Transcription refers to the gene-expression process whereby a DNA sequence is copied to *mRNA*. It is initiated by specific proteins known as *transcription factors*, which bind to regions of genes known as *promotors*, and prepare the DNA for information duplication.

2.2 Differentially expressed genes

As needed for the computation of weight functions for some biological networks in the remainder of this work, we discuss the measurement of gene expression over multiple conditions.

Although the genetic blueprints of different cells of an organism are exactly the same, the functionality of two cells can be quite diverse. The characteristics of a single cell depend on the expression of its genes. An important field of research with application to cancer research [DPB$^+$96] and [ZZV$^+$97] is the comparison of expression patterns of cells. If a gene is expressed in different amounts over multiple conditions we consider it as being *differentially expressed*. Microarray analysis makes it possible to observe the expression-level changes of tens of thousands of genes over multiple conditions. Hereby, data are generated from *DNA* microarrays with spots for each gene with a dye intensity that depends on the level of expression of the gene. Finding accurate models that analyze the genes that are differentially expressed based on microarray information is a critical step of analysis. The goal is to compute a value (often called p-value) for each gene that indicates the likelihood that it is differentially expressed. One then considers all genes with a p-value higher then a specific threshold as differentially expressed. Because of the error-prone nature of microarray analysis, the computation of p-values is a difficult statistical task involving error models. To give one out of many publications addressing this issue, Ideker et al [ITSH00] provide the software tools VERA and SAM for the determination of p-values. For an application example concerning the analysis of gene expression data from yeast, Ideker et al. [ITR$^+$01] used VERA and SAM in the computation of differentially expressed genes. They considered multiple conditions initiated by different perturbations of the yeast galactose-utilization pathway and provide the computed data (used for the computation of vertex weights later in this work) as part of the supplementary material at `http://science-mag.org/cgi/content/full/292/5518/929/DC1`.

2.3 Databases

In this section, we introduce some publicly available databases or data sources.

- **BIND—The Biomolecular Interaction Network Database**

  URL: `www.bind.ca`
  Maintainer: Blueprint
  Data obtained for this work: protein-protein interaction
  Reference: [AAA+05]
  General information: BIND archives biomolecular interaction, reaction, complex and pathway information. It provides details about molecular interactions that have been drawn from published experimental research. Furthermore, it makes tools available to enable data analysis. Presently (October 2005) it contains nearly 200 000 interaction records from a number of different organisms.

- **Munich Information Center for Protein Sequences (MIPS)**

  URL: `http://mips.gsf.de/`
  Maintainer: MIPS
  Data obtained for this work: protein complex information
  Reference: [GMK+05],[PKO+05]
  General information: The MIPS databases provide highly accurate information about protein-protein interaction for different plants and fungi, e.g. the Comprehensive Yeast Genome Database [GMK+05]. They also have a database containing mammalian protein-protein interactions [PKO+05].

- **TRANSFAC**

  URL: `http://www.gene-regulation.com/pub/databases.html`
  Maintainer: BIOBASE
  Data obtained for this work: protein-*DNA* interaction
  Reference: [WCF+01]
  General information: TRANSFAC is a database on eukaryotic transcription factors, their genomic binding sites and *DNA*-binding profiles.

- **SCPD: A promotor database of yeast Saccharomyces cerevisiae**

  URL: `http://rulai.cshl.edu/`
  Maintainer: Zhang Lab (Cold Spring Harbor Laboratory)
  Data obtained for this work: protein-*DNA* interaction

Reference: [ZZ99]
General information: A promotor database of yeast *Saccharomyces cerevisiae*, SCPC contains experimentally mapped transcription factor binding sites and transcriptional start sites, as well as relevant binding affinity and expression data [ZZ99].

- **ChIP-CHIP**

  URL: `http://web.wi.mit.edu/young/regulator_network/`
  Maintainer: Lee et al.
  Data obtained for this work: protein-*DNA* interaction
  Reference: [LRR+02]
  General information: Lee et al. [LRR+02] provide data derived from so-called ChIP-CHIP experiments, which combine a Chromatin Immunoprecipitation (ChIP) procedure with *DNA* microarray analysis.

Note that especially in large databases like BIND, many records are based on high-throughput projects which are error-prone. Therefore, they are likely to contain many false-positive entries. And, as there are still many interactions that have not been detected by experimental studies, none of the databases can give a complete picture of the cell.

There are different tools available for the visualization of biochemical networks, including information and annotations about molecules and built-in functions for analysis. Two examples are the Cytoscape software package, which is available to the academic community at `http://www.cytoscape.org`, or the online visualization and analysis tool for biochemical interaction data VisANT [HMWD04], freely available at `http://visant.bu.edu`.

2.4 Network types

The analysis of biological networks is a well-studied field of recent research. There are many ways of organizing biological data into networks. We describe four common biological networks—especially relevant to this work—including descriptions of how they have been used to obtain important results in the analysis of biological data. An overview is provided in Table 2.1.

2.4.1 Protein interaction network

Databases like BIND (Section 2.3) provide large amounts of protein-protein interaction data for different species and can easily be used to build large-scale protein interaction networks consisting of thousands of ten thousands

of vertices and edges. In this case, proteins are considered as vertices with an undirected edge between two proteins if they interact. Scott et al. [SIKS05] considered the protein interaction network of yeast with edge weights. The weight of each edge indicates the strength of evidence for the existence of the corresponding interaction. Using a graph-algorithmic approach to find paths in the network, they identified important substructures.

2.4.2 Metabolic network

In a metabolic network, cell substrates are interconnected through biochemical reactions. The set of vertices consists of metabolites like amino acids or carbohydrates and other molecules like enzymes. Edges correspond to biochemical reactions. In case of a reversible reaction there is an undirected edge, otherwise a directed one exists. Ihmels et al. [ILB04] integrated large-scale expression data with the structural description of the metabolic network of *Saccharomyces cerevisiae*. They systematically analyzed the expression pattern of genes associated with metabolic pathways. From a graph-theoretical point of view this can be considered as assigning a weight depending on its expression pattern to a vertex. With this approach Ihmels et al. [ILB04] were able to glean deeper insights into the principles of transcriptional control in the network. For example, they showed that coexpressed enzymes, e.g. enzymes that are expressed in similar amounts over different conditions, are often arranged in a linear order corresponding to a metabolic flow and made interesting observations about the regulation of isozymes (different enzymes that catalyse the same biochemical reaction).

2.4.3 Transcriptional regulatory network

The state of each cell is determined by specific gene expression programs involving the regulated transcription of thousands of genes. Lee et al. [LRR+02] experimentally identified most of the interactions between the transcriptional regulators and the promotor sequences of yeast genes. With this information one can build the transcriptional regulatory network for yeast, in which the vertices correspond to the genes. Furthermore, there is a directed edge from gene u to gene v if the gene product of u is a transcription factor of v. A path in such a network can be considered as a pathway that a cell can use to regulate global gene expression programs. Lee et al. [LRR+02] identified so-called *network motifs*, which can be considered as simplest units of network architecture, and provided a method that use these motifs to assemble a transcriptional regulatory network structure.

2.4.4 Interaction network

The working groups of Ideker [IOSS02] and Hallett [SPB+05] considered a network type that comprises protein-protein and protein-*DNA* interactions and denote it as interaction network. As a protein directly corresponds to a gene, an interaction network can be considered as the union of a protein interaction network (with proteins as vertices) and a transcriptional regulatory network (with genes as vertices) with vertices that can be considered either as genes or as proteins. Note that the network contains directed as well as undirected edges.

In both works ([IOSS02] and [SPB+05]) the authors chose vertex weights that are a measure for the differential expression based on data provided by a previous work of Ideker [ITR+01] as described in Section 2.2. Ideker et al. [IOSS02] provided a statistical measure for scoring subnetworks. When searching the networks for subnetworks with high score, they could identify "active subnetworks", e.g. connected set of genes with unexpectedly high level of differential expression. Applied to the yeast interaction network, they found several top-scoring subnetworks with good correspondence to known regulatory mechanisms.

Scott et al. [SPB+05] suggested another approach. They used a set of distinguished proteins as input and attempted to identify regulatory subnetworks by looking at a subgraph that connects the vertices of this set. They re-discovered known regulators of some well-studied pathways and suggested a previously unknown connection regarding the diauxic shift in yeast. Every vertex can be assigned a weight such that the weight of an active vertex is usually high and the weight of vertices corresponding to genes with very little differential expression is very low or below zero.

2.5 Summary

In addition to a description how biological networks can be defined and generated, this chapter motivates the usefulness of including graph-theoretical approaches into their analysis. This was the main motivation of this work. In the following chapters, we will firstly further investigate the approach of Scott et al. [SPB+05] that was mentioned in Section 2.4.4. Secondly, we introduce graph-theoretical problems that can be used in a way similar to [IOSS02] (explained in Section 2.4.4) for the analysis of interaction or other biological networks.

Network Type	*protein interaction*	*metabolic*	*transcriptional regulatory*	*interaction* (combination of protein interaction and transcriptional regulatory network)	
Vertices	proteins	metabolites	genes	proteins/genes	
Edges	protein interaction undirected	biochemical reactions undirected if reversible, otherwise directed	regulator-gene interaction directed	protein + regulator-gene interaction undirected + directed	
Weights	edges: indicating the strength of evidence that interaction exists	edges: correlation coefficient (expression patterns)	—	vertices: based on differential expression	
Example	Scott et al. [SIKS05] provide an method that automatically can identify known pathways.	Ihmels et al. [ILB04] gain information about possible design principles of metabolic gene regulation.	Lee et al. [LRR+02] describe eukaryotic network motifs and a method to build them into modules of function.	Ideker et al. [IOSS02] identified subnetworks with good correspondence to known regulatory mechanisms.	Scott et al. [SPB+05] provide an approach to identify regulatory subnetworks for a set of significant proteins or genes.

Table 2.1: **Overview of different types of biological networks.** *The row* Vertices *displays what kind of molecules are matched to vertices for the given network type. The row* Edges *describes the corresponding biochemical interactions that match the edges of the network. The row* Weights *tells if the network has vertex or edge weights and gives the basic idea on that the weight function is based on.* Example *cites a work that used the considered network type and very briefly summarizes its results.*

Chapter 3

Steiner Trees and biological networks

This chapter is concerned with the analysis of biological networks by means of the so-called *Steiner method*, a new approach introduced by the working group of Hallett [SPB+05]. The Steiner method that is further described in Section 3.1 can be applied whenever one is interested in determining important vertices that connect a distinguished set of vertices, usually proteins or genes, that have been obtained from biochemical experiments. Here, in line with the working group of Hallett, we deal with the development of strategies that make the Steiner approach applicable for a wider range of instances and accessible to biochemical working groups in general. A contribution of this work consists of the development and design of new preprocessing rules, including their analysis, implementation, and the experimental validation of their effectiveness. Furthermore, we provide a study about properties and structure of the regarded network that hints which preprocessing and algorithmic approaches can be applied efficiently and which can be used for the selection of methods for the conclusive software tool *Steiner Package*. A last step in the design of the Steiner package is the combining of different preprocessing rules and algorithms by determining a reasonable order in which they are applied. Finally, we provide experimental tests that show their usefulness.

In Section 3.1 we give a short overview of the Steiner method as introduced by Scott et al. [SPB+05]. In the next Sections 3.2 and 3.3 we regard algorithms and reduction rules for Steiner Tree in Graphs (STG) and Vertex-Weighted Steiner Tree in Graphs (V-STG). For each problem we start with an overview of literature and then introduce new reduction

rules. Furthermore, we investigate how known reduction rules developed for STG can be directly transferred or modified for V-STG. Next, we consider structure and properties of a typical biological network that is used for the computation of Steiner trees in this work (Section 3.4). We end this chapter with a description of the software tool Steiner Package, including its implementation and interface as well as results and examples (Section 3.5).

3.1 The Steiner method for biological networks

In this section, we present a new method for the analysis of biological networks developed by the working group of Hallett [SPB+05]. It is based on a variant of Steiner Tree in Graphs that considers vertex instead of edge weights and is defined as follows.

> Vertex-Weighted Steiner Tree in Graphs (V-STG)
> **Input:** An undirected graph $G = (V, E)$, a weight function $w : V \to \mathbb{R}^{\geq 0}$, a set of *distinguished vertices (or terminals)* $S \subseteq V$.
> **Task:** Find a connected subgraph $G' = (V', E')$ of G with $S \subseteq V'$ and where weight $w(V') = \sum_{v \in V'} w(v)$ is minimum.

We call a tree that spans a set of distinguished vertices or a connected subgraph G' as required for V-STG *Steiner tree* and the non-distinguished vertices of a Steiner tree *Steiner nodes*.

The goal of the Steiner method is to detect biological relationships between a set of distinguished proteins or genes. Many biochemical experiments present a set of genes or proteins that seems to be important in a specific scenario. A typical example is a set of genes that is differentially expressed under the same conditions. Another possibility is a list of essential genes generated by knock-out experiments. The next step of analysis is to find coherences between the proteins or genes of the distinguished set. For this, a promising approach is provided by the Steiner method. The basic idea is to consider the set of relevant proteins or genes as distinguished vertices in a biological network and compute a Steiner tree for them. Generally, Steiner nodes then correspond to proteins or genes that are candidates for the regulation of the distinguished set as the Steiner nodes connect them in the network in a compact way. As many of the regarded biological networks contain thousands of vertices a non-automated analysis seems to be elusive. To obtain more information from a computed Steiner tree, it can be regarded as a backbone and augmented by vertices of its neighborhood under some additional constraints. This vertices than can also be considered as important

candidates of proteins or genes that could explain the relationship between the proteins corresponding to the distinguished set.

In the experimental part of [SPB+05], the authors investigate the yeast interaction network composed of 5,458 proteins and 23,642 interactions from BIND version 2 [AAA+05] (restricted to yeast protein-protein interactions), TRANSFAC [WCF+01] (yeast protein-DNA interaction), SCPD (yeast protein-DNA interactions), and ChIP-Chip (yeast protein-DNA) [LRR+02] data sets. They include protein-DNA interactions from ChIP-Chip data if their associated p-value is 0.001 or less. Note that the directed edges that describe protein-DNA interactions are treated like undirected edges for this approach. (The directed Steiner tree problem which is defined in [FR99] would yield different results.) Furthermore, they employ two weight functions: The weight function w_1 that assigns one to every vertex and a weight function w_d computed from p-values based on differential expression data from [ITR+01]. More precisely, they set $w_d(u) = -\log(1 - p_u)$, where u is a vertex in the graph and p_u the corresponding p-value. They show evidence for the practical usefulness of the Steiner approach by performing different sets of experiments. The distinguished sets were obtained from microarray expression data and from substrates of known regulatory pathways (as gluconeogenesis or glycolyse pathway). Apart from re-detecting known regulatory elements for some pathways, the authors were able to detect new connections in the galactose metabolism of yeast and to support various claims from literature.

As V-STG is NP-hard [GJ79], the computation of a Steiner tree is a crucial part of the approach. Luckily, in many cases the set of distinguished vertices is small enough that a Steiner tree can be computed by the Dreyfus-Wagner algorithm, whose running time is only exponential in the size of the distinguished set and polynomial otherwise [DW72].

This work is concerned with developing further strategies to compute Steiner trees for so far unsolved instances. For this, we focus on different approaches like data reduction, preprocessing techniques based on cell molecular information, and the implementation of other algorithms.

Although the work of Hallett considers only applications of V-STG, we start by investigating the literature for the much more intensively studied STG. This is done for two reasons. First, we hope that some of the approaches for STG can be modified in a way that they are applicable for V-STG as well. A promising example for this is the Dreyfus-Wagner algorithm that was developed for STG and could be modified for V-STG in a straightforward way [SPB+05]. Another motivation to look at STG is that

there are also some biological scenarios in which this variant may be useful. For example, it could be applied to protein interaction networks with edge weights depending on the reliability of the corresponding interaction. A Steiner tree for some given products and/or reactants could identify important regulator proteins or intermediate products of the pathway. Another application could arise in interaction networks with edge weights that are based on the correlation coefficient from the p-values of the differentially expressed genes (analogously to the edge weights of the metabolic network in [ILB04]).

3.2 The Steiner Tree in Graphs problem

In this section we consider solution strategies for STG. We start by introducing some algorithms in Section 3.2.1 and go on with data reduction rules in Section 3.2.2.

3.2.1 Algorithms

Although STG is NP-hard in general, there exist some special cases which are solvable in polynomial time. If the terminal set has cardinality two, STG coincides with the Shortest Path problem, and, if the terminal set contains all vertices of the graph, it coincides with Minimum Spanning Tree. Both problems can be computed in $O(n \log n + m)$ time [CLRS01]. Furthermore, there exists a wide range of algorithms attacking the problem from different points of view. We give a brief overview of exact fixed-parameter algorithms and approximation algorithms.

Dreyfus-Wagner algorithm STG is fixed-parameter tractable with respect to the size of the terminal set k. The Dreyfus-Wagner algorithm [DW72] solves STG in $O(3^k \cdot n + 2^k \cdot n^2 + n^2 \cdot \log n + n \cdot m)$ time. As the Dreyfus-Wagner algorithm is relevant for the remainder of this work, we give its description in pseudo-code in Figure 3.1. The basic idea is to compute Steiner trees for subsets of the terminal set and combine them by dynamic programming to the solution Steiner tree. The algorithm starts with the computation for terminal sets of size two, and then uses them to compute Steiner trees for terminal sets of size three and so on. The recursion is based on the observation that one can use Steiner nodes with degree at least three in the Steiner tree to split it into subtrees. So, in every step, the algorithm computes the weight of a minimum Steiner tree for all subsets of terminals X of a specific

`Dreyfus-Wagner Algorithm`

/* **input:** A graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}_0^+$,
a terminal set $S \subseteq V$ */
/* **output:** The weight of a Steiner minimum tree T for S */

01 /* **initialization** */
02 `forall` $v, w \in V$ `do`
03 compute shortest path $p(v, w)$
04 `forall` $\{x, y\} \in S$ `do`
05 $s(\{x, y\}) := p(x, y)$

06 /* **recursion** */
07 `for` $i = 2$ `to` $k - 1$ `do`
08 `forall` $X \subseteq S$ with $|X| = i$ and all $v \in V \backslash X$ `do`
09 $s_v(X \cup \{v\} := \min_{\emptyset \neq X' \subsetneq X} \{s(X' \cup \{v\}) + s((X \backslash X') \cup \{v\})\}$
10 `forall` $X \subseteq S$ with $|X| = i$ and all $v \in V \backslash X$ `do`
11 $s(X \cup \{v\}) := \min \{ \min_{w \in X} \{p(v, w) + s(X)\}, \min_{w \in V \backslash X} \{p(v, w) + s_w(X \cup \{w\})\}\}$

Figure 3.1: **Dreyfus-Wagner algorithm** *Description in pseudo-code as given in [PS02].*

size and every vertex $v \in V \backslash X$. The value $s(X)$ denotes the value of a Steiner minimum tree for the terminal set X (*line 05, 11*). The computation for every subset starts with the computation of $s_v(X \cup \{v\})$ as given in *line 09* that simulates the possibility that v could be used to split the Steiner tree.

For a detailed proof for the correctness of the algorithm as described in Figure 3.1 we refer to [PS02].

Improved parameterized algorithm w.r.t. the number of terminals Mölle et al. [MRR05] developed a new algorithm that improves the running time of the Dreyfus-Wagner algorithm to $O((2+\epsilon)^k \cdot poly(n))$ for $0 < \epsilon < 1$. Whereas the Dreyfus-Wagner algorithm splits the tree at a single node v, they choose a subset $|X|$ of nodes to split the tree, such that $|X|$ is bounded by $1/\epsilon$ for an adjustable parameter ϵ. The improved asymptotical running time comes along with large constants hidden in the Landau notation for small values of ϵ. More precisely, small values of ϵ imply a higher exponent in the polynomial term. The authors themselves point out that their algorithm without further improvements "is very likely to be slower than the Dreyfus-Wagner algorithm". For this reason, in the remainder of this work, we use the Dreyfus-Wagner algorithm for practically solving STG. Note that the Dreyfus-Wagner algorithm could in all cases be replaced by the improved algorithm to gain fixed-parameter algorithms with a theoretically better running time.

An enumeration algorithm for bounded $|V| - k$ If the number of terminals becomes nearly as large as the number of all vertices of a graph, STG can be efficiently solved by a simple enumeration algorithm that was first developed by Hakimi [Hak72]. It can be implemented such that its running time is bounded by $O(n^2 \log n + nm + \min\{n^{k-2}, 2^{n-k}\} \cdot k^2)$ [PS02]. Basically, the algorithm generates all subsets of non-terminals and then considers the subgraphs induced by vertices of each of these subsets and of the terminal set. In a next step, it computes minimum spanning trees for these subgraphs in the corresponding distance graph, that is a complete graph whose edges have the weight of a shortest path in the original graph. An optimal Steiner tree then coincides with the minimum spanning tree over the spanning trees for all these subgraphs.

An algorithm for graphs with bounded treewidth Korach and Solel [KS90] provide a fixed-parameter algorithm with respect to the tree-

width d of a graph. The running time is given as $O(n \cdot d^d)$. As the running time grows quickly with the treewidth, the algorithm can probably only be used for instances with very small treewidth. In [KS90] no experimental results are given.

Algorithms for graphs with bounded pathwidth Polzin and Vahdati [PV02] give a practical dynamic programming algorithm whose running time is linear in the number of vertices if the pathwidth is constant. The formulation of the algorithm is based on a concept of small width that is closely related to the pathwidth of a graph. The authors showed evidence for the practical usefulness of the algorithm by solving previously unsolved benchmark instances.

Approximation algorithms Even in the case that the edge weights are restricted to $\{1, 2\}$, the Steiner Tree in Graphs problem is APX-complete [BP89]. The best known polynomial-time approximation algorithm for STG has a performance guarantee of $1 + \frac{\ln 3}{2} \approx 1.55$, (i.e. it guarantees a solutions with a size that is less or equal than $1 + \frac{\ln 3}{2}$ times the size of an optimal solution) and the aforementioned version with weights restricted to $\{1, 2\}$ is approximable within 1.28 [RZ00]. A simple and efficient approximation algorithm, based on the computation of minimum spanning trees in the distance graph, has a performance guarantee of 2 [TM80].

3.2.2 Reduction rules

Data reduction for STG is a well-studied field of research. A data reduction rule replaces, in polynomial time, a given STG instance (G, w, T) consisting of a graph G with weight function w and a terminal set T by a simpler instance (G', w', T') such that (G, w, T) has a solution iff (G', w', T') has a solution. This section gives a brief overview of some important publications. Further, we briefly introduce some basic concepts of data reduction for STG as we investigate their adaptability to V-STG. In a next step we provide new reduction rules for STG.

Literature

A fundamental work concerned with data reduction for STG is provided by Duin and Volgenant [DV89]. They give an overview of the reduction rules known at that time, generalize some of them, and introduce new concepts.

An example for an efficient way of using known reduction rules as part of a programming package is described by Koch and Martin [KM98].

There are also many publications dealing with special graph structures. Winter et al. [Win95] introduced the concept of *extension* for rectilinear STG. It assumes that an edge is part of the solution, tries to find a contradiction looking at the neighborhood of the edge, and possibly concludes that the edge cannot be part of a minimum Steiner tree. Uchoa et al. [UdAR02] show that a combination of reduction rules from Duin and Volgenant and the idea of extension can be successfully applied on "grid graphs with holes", which could not be tackled by Koch/Martin [KM98].

In a book chapter, which covers more than 50 pages, Duin gives extensive information about preprocessing the Steiner problem [Dui00]. It provides an overview of terminology, known rules, as well as even more advanced new concepts and experimental results.

Furthermore, Polzin and Vahdati [DP02] use alternative reduction rules in combination with branch-and-bound methods and thereby introduce some more sophisticated tests dealing with more general patterns, like trees, instead of vertices or edges.

Basic concepts of data reduction

As mentioned in the last paragraph, Duin and Volgenant [DV89] give an interesting overview of reduction rules and underlying concepts. We investigate them at this point as they contain many basic ideas of data reduction. We would like to test their applicability of these ideas to V-STG. Since most of the other works are either extensions of these concepts or introduce more complicated rules, we decided to consider the basic concepts as a first step. We briefly summarize the most important ideas and reduction rules. For a more detailed description of reduction rules and their correctness we refer to [DV89]. In the following presentation, we basically follow [DV89].

Depending on the reduction rule, the graph can be affected in different ways. On the one hand, there are reduction rules that determine edges and/or vertices that can be deleted, on the other hand, some rules identify *Steiner edges* (edges that have to be part of an optimum Steiner tree). If a Steiner edge $\{u, v\}$ is detected, it is incorporated into the solution and u and v are contracted and in the case of an identical neighbor w, i.e. $w \in N[u]$ and $w \in N[v]$, the new edge between the contracted vertex and w is assigned $\min\{w(\{u, w\}), w(\{v, w\})\}$. Note that this effect could decrease the size of the terminal set and therefore can improve the performance of the Dreyfus-Wagner algorithm more effectively than the removal of edges or vertices in

general.

In the following, let $d(u, v)$ denote the weight of a shortest path from u to v. We start with the description of some simple reduction rules whose correctness is obvious.

Reduction Rule 1. (Least Cost Test) *An edge $\{v, w\}$ can be removed if there is a shortest path from v to w that contains at least one intermediate vertex.*

Reduction Rule 2. (Degree Tests)

1. *A non-terminal vertex with degree one can be removed.*
2. *An edge that is incident to a terminal vertex with degree one is a Steiner edge.*
3. *A non-terminal degree-two vertex v with the incident edges $\{u, v\}$ and $\{v, z\}$ can be replaced by an edge $\{u, z\}$ with $w(\{u, z\}) = w(\{v, u\}) + w(\{v, z\})$.*

Reduction Rule 3. (Nearest Vertex Test) *For any terminal $t \in S$, let v_i be the nearest vertex, i.e., $w(\{t, v_i\}) = \min\{w(\{t, v_j\}) \mid v_j \in N(t)\}$. Then $\{t, v_i\}$ is a Steiner edge if there exists a vertex $t' \in S \backslash \{t\}$ such that*

$$w(\{t, v_i\}) + d(t', v_i) \leq \min\{w(\{t, v_j\}) \mid v_j \in V \backslash \{v_i\}\}.$$

The Nearest Vertex Test was first introduced by Beasley [Bea84] and is based on the observation that an edge e adjacent to a terminal k has to be part of an optimum Steiner tree if there is a path that connects k with another terminal such that the weight of the path is less than the weight of all other edges adjacent to k.

Another method that is described in [DV89] are **Reachability Tests**. If one has obtained an upper bound for a Steiner tree, e.g. by a heuristic or an approximation algorithm, it can be used to eliminate vertices that are "not reachable". That is, the lower bound for a Steiner tree containing a particular vertex exceeds the cost of the upper bound. Duin and Volgenant give different possibilities to obtain lower bounds.

Now, we review some so-called *bottleneck approaches* from [DV89]. The basic idea is to find an edge that cannot be part of a Steiner minimum tree as every path containing this edge cannot be part of a Steiner minimum tree or the weight of the edge exceeds the weight of an alternative solution that can connect its endpoints in a better way.

Reduction Rule 4. (Vertices Nearer to S Test). *An edge $\{u, v\}$ can be deleted if there is a vertex $k \in S$ with* $\max\{d(k, u), d(k, v)\} < w(\{u, v\})$.

Note that if k is connected to a Steiner tree that does contain $\{u, v\}$, this Steiner tree has to include a path from k to v or u without $\{u, v\}$. The correctness of Vertices Nearer to S Test then is based on the fact that any solution with edge $\{u, v\}$ is improved by replacing $\{u, v\}$ with either the shortest path from u to k or with the shortest path from v to k.

An important concept introduced by Duin and Volgenant is the *special distance* which is very useful for the formulation of further reduction rules. For an intuitive description, as given in [KM98], one can consider each terminal as a petrol station. Then, assume you like to drive from location u to location v. The special distance between u and v denotes the distance you must be able to drive without refilling if you choose among all possible routes. We give the formal definition as specified in [KM98]:

Definition 3.1. *(***Special Distance)**
Given two vertices $u, v \in V$, we consider some path $P \subseteq E$ that connects u and v. Set $T_P = (V(P) \cap S) \cup \{u, v\}$ and let

$$b(P) = \max\{w(F) \mid F \subseteq P \text{ is a path connecting two nodes from } T_P$$

$$\text{such that } |T_P \cap V(F)| = 2\}.$$

The number

$$s(u, v) = \min\{b(P) \mid P \text{ is a path connecting } u \text{ and } v\}$$

is called the special distance (between u and v).

Note that we have the following relations between special distance s, smallest distance d, and the weight of an edge between two vertices

$$s(u, v) \leq d(u, v) \leq w(\{u, v\}).$$

The special distance can be computed efficiently and leads to a very effective test for deleting edges as shown in [DV89]:

Reduction Rule 5. (Smaller Special Distance Test) *Any edge $\{u, v\}$ can be eliminated if $s(u, v) < w(\{u, v\})$.*

Furthermore, the special distance leads to more general tests having Least Cost Test and Vertices Nearer to S Test as special cases. For example, as not further described here, it can be used to improve the conditions for the so-called **Nearest Special Vertices Test (NSV)** that detects Steiner edges. Only sketching the idea, by computing minimum spanning trees in the graph NVS is able to detect some edges that can be identified as Steiner edges if their weight is higher than a value computed with the help of (special) distances in the graph.

Lastly, Duin and Volgenant consider some more general degree test and some edge cost transformation. As the degree test can only be efficiently applied for vertices with small degree and an edge cost transformation is obviously not useful for the vertex weigthed case, we omit a description of the corresponding rules.

New structure-based reduction rules

Most of the reduction rules described in the literature depend on the cost of an edge. They determine either that an edge cannot be in an optimal solution because its cost is too high or that an edge has to be part of an optimal solution because all alternative solutions would cost more. In contrast, we present in the following new reduction rules that are motivated by the application considered in this work and only look at the structure of a graph.

Components without distinguished vertices

Examining biological networks in a way as described by Scott et al. [SPB+05], we typically have to deal with large networks with few distinguished vertices. Therefore, we look for subgraphs of a network not containing distinguished vertices. In some cases such subgraphs can be found efficiently and replaced by simplified structures. A simple example is a subgraph only connected to the remaining graph by two distinguished vertices (Figure 3.2).

For the formulation of further reduction rules we need the following definitions.

Definition 3.2. *A connected subgraph $G' = (V', E')$ of G is called a* dvfree component *if all vertices from V' are non-distinguished.*

*An i-*dvfree component *C_i is a dvfree component that is separated from the remaining graph by i vertices $t_1, ..., t_i$ (i.e., deleting $t_1, ..., t_i$ there is no path from a vertex $v' \in C_i$ to a vertex $v \in V \backslash C_i$). We say that C_i* connects *the* boundary vertices *t_i.*

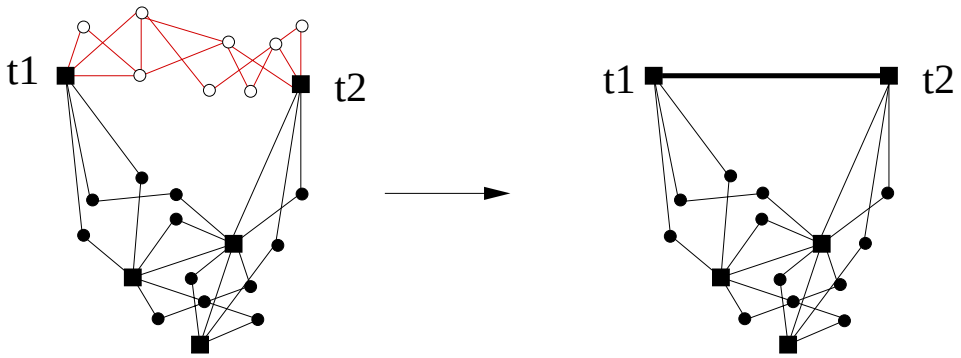

Figure 3.2: Illustration of 2-Dvfree Rule. *The network on the left-hand side contains a 2-dvfree component (white vertices). The instance can be reduced to the network given on the right-hand side. Terminals are marked as squares.*

Definition 3.3. *A* component Steiner tree (ST_C) *for a dvfree component C is a minimum Steiner tree that connects all boundary vertices $X \subseteq N[C]$ and contains only vertices from C.*

In Figure 3.2 a dvfree component is given by the white vertices. We can further classify it as a 2-dvfree component that connects the boundary vertices t_1 and t_2.

A 2-dvfree component can obviously be replaced by a shortest path or an edge with the weight of a shortest path. More precisely, we can define the following reduction rule.

Reduction Rule 6. (2-free Rule)
Let $d_C(t_1, t_2)$ denote the weight of a shortest path from t_1 to t_2 only containing vertices of C. The vertices of a 2-dvfree component connecting t_1 and t_2 can be replaced by an edge between t_1 and t_2 with edge cost $c_{new} := d_C(t_1, t_2)$.

The 2-Dvfree Rule is illustrated in Figure 3.2.

Dealing with three or more boundary vertices, the replacement of the corresponding dvfree component becomes more difficult.

If a dvfree component is connected with three boundary vertices t_1, t_2, and t_3, we have to take into account the following possibilities. Vertices of this component in a minimum Steiner Tree can connect either pairs of them ("t_1 and t_2", "t_1 and t_3", and/or "t_2 and t_3") or all of them (t_1 and t_2 and t_3). To replace the component we find a component Steiner tree connecting

i-Dvfree Rule

/* **input:** Graph $G = (V, E)$, set of distinguished vertices $S \subseteq V$, i-dvfree component C_i with boundary vertices T */
/* **output:** reduced graph $G' = (V', E')$ */

initialize $Q := \emptyset$
forall $X \subseteq T$ with $|X| \geq 2$
    compute ST_{C_i}
    add edges of ST_{C_i} to Q
forall $e \in C_i$
    if $e \notin Q$ then
      remove $e = \{u, v\}$ from E
      if u (or v) has degree 0 then
        remove u (or v) from V
return G

Figure 3.3: **Computation of i-Dvfree Rule in pseudo-code**

t_1, t_2, and t_3 and delete all vertices and edges of the component that are not in this tree or in one of the three shortest paths.

Generally, we can formulate **i-Dvfree Rule** for a given i-dvfree component as given in the pseudo-code of Figure 3.3. In the following, we consider its correctness and running time. For the running time we start by determining in which time all i-dvfree components can be obtained (Lemma 3.5) and then consider the time one needs for the computation of i-Dvfree Rule applied to an i-dvfree component (Lemma 3.6).

Lemma 3.4. *The size of a Steiner tree of a graph G with terminal set S does not change by applying i-Dvfree Rule to an i-dvfree component C_i.*

Proof. If an optimum Steiner tree of G contains edges of C_i, they must be used to connect a subset of boundary vertices, otherwise they would be redundant and the Steiner tree could not be minimum. As i-Dvfree Rule keeps one optimum local Steiner tree that connects every subset of the boundary vertices of C_i, the size of a Steiner minimum tree is not increased by its application. □

Lemma 3.5. *All i-dvfree components can be obtained in time $O(\binom{n}{i} \cdot n)$.*

Proof. An i-dvfree component must be connected to i boundary vertices out of n vertices, i.e. we have to regard $\binom{n}{i}$ candidate subsets. For every subset we can check if there is a i-dvfree component in $O(n)$ time by removing all boundary vertices from the graph and check if there are connected components which contain no distinguished vertex left. This yields the claim. □

Lemma 3.6. *The i-Dvfree Rule can be carried out in time* $(4^i \cdot n + 3^i \cdot n^2 + 2^i \cdot n^3)$

Proof. The time consuming part of i-Dvfree Rule is the computation of a Steiner tree for all subsets of T. Using the Dreyfus-Wagner algorithm this can be achieved in time $O(\sum_{j=1}^{i} \binom{i}{j} \cdot (3^i \cdot n + 2^i \cdot n^2 + n^3))$. □

As the running time of the i-Dvfree Rule grows exponentially with the number of boundary vertices i, in practice it only makes sense to apply the rule for small values of i. In graphs that are not highly connected this can still decrease their size.

3.3 Vertex-Weighted Steiner Tree in Graphs problem

In this section, we consider how to obtain solutions for V-STG. Whereas the classical Steiner Tree in Graphs problem is well-studied in the literature, there are only few publications regarding the Vertex-Weighted Steiner Tree in Graphs problem. Most of them are concerned with its approximability. We give a brief overview of algorithms for V-STG as described in the literature (Section 3.3.1) and then consider data reduction rules for V-STG. As, as far as we know, there are no publications that are concerned with data reduction for V-STG, we start by investigating the applicability from reduction rules designed for STG and then describe some new data reduction rules (Section 3.3.2).

3.3.1 Algorithms

Before presenting algorithms for V-STG in general, we consider some trivial cases for which the otherwise NP-hard V-STG can be solved in polynomial time. Analogously to STG, in case of a terminal set of size two, V-STG coincides with the Shortest Path problem which can be solved in time $O(m + n \cdot \log n)$ [CLRS01]. In case the terminal set contains all vertices, the graph itself is an optimum solution. Note, whereas STG for this case

coincides with the MINIMUM SPANNING TREE problem, there is no obvious correspondence between V-STG and MINIMUM SPANNING TREE.

Exact Algorithms

Most of the exact algorithms for STG described in Section 3.2.1 can be applied to V-STG. For the Dreyfus-Wagner algorithm and its improvement as well as for the algorithms for bounded pathwidth or treewidth one only has to adapt the computation of the weight of a tree, i.e. set $w(T) := \sum_{v \in V} w(v)$ instead of $w(T) := \sum_{e \in E} w(e)$. Obviously, this does not affect the running time of the algorithms.

In contrast, the enumeration algorithm can not be modified in a straightforward way to solve V-STG. This is due to the fact that it is based on the computation of minimum spanning trees, that cannot be translated to the vertex-weighted case. Note that the Dreyfus-Wagner algorithm, using an all-pairs shortest paths algorithm as a subroutine, can only be translated as V-STG coincides with SHORTEST PATH for two terminals.

Approximation algorithms

The V-STG problem is harder to approximate than STG. Let k denote the number of terminals. Klein and Ravi [KR95] show that there is no approximation algorithm with better ratio than $(1 - o(1)) \cdot \ln k$ unless $NP \subseteq DTIME[n^{O(polylogn)}]$ and give an approximation algorithm with performance ratio $2 \cdot \ln k$. As the algorithm is part of our software tool we give a description in pseudo-code in Figure 3.4. The algorithm works on a node-disjoint set of trees such that every terminal belongs to one of the trees. Each tree of the set is initialized by a terminal (*line 06*). The algorithm then uses a greedy strategy to merge the subtrees into larger trees. In each iteration, it selects a vertex v and at least one of the current trees so as to minimize the ratio

$$\frac{\text{weight of node } v + \text{sum of distances to the trees}}{\text{number of trees}}$$

as given in *line 17*.

As no analysis of running time is given (and we would like to include an implementation of the algorithm into our software tool), we provide the following lemma:

Lemma 3.7. *The approximation algorithm by Klein and Ravi can be carried out in time* $O(n^2 \cdot \log n + n \cdot m + n \cdot k^3 \cdot \log k)$.

```
Approximation Algorithm (Klein, Ravi)

/* input:  A graph G = (V, E) with weight function w : V → ℝ_0^+,
           a terminal set S = {s_1, ..., s_k} ⊆ V                    */
/* output: The weight of a Steiner tree T for S such that
           w(T) ≤ 2 · ln k · w(T_opt)                                */

01 /* initialization */
02 forall v, w ∈ V do
03    compute shortest path SP(v, w)
04    set d(v, w) := w(SP(v, w))
05 for i = 1 to k
06    K_i := {s_i}
07 K := {K_1, ..., K_k}
08 /* algorithm */
09 while |K| > 1
10    m_g := +∞, L := ∅
11    forall v ∈ V
12       forall K_i ∈ K
13          d(v, K_i) := min_{n∈K_i} d(v, n)
14       enumerate K in the order given by d(v, K_1) ≤ d(v, K_2) ≤ d(v, K_3)...
15       m_l := +∞, p := 0
16       for i = 1 to |K|
17          m := (w(v) + Σ_{j=1}^{i} d(v, K_j))/i
18          if m < m_l then
19             m_l := m and p := i
20       if m_l < m_g then
21          m_g := m_l and L := ⋃_{j≤p} K_j ∪ {u ∈ SP(K_j, v) | 1 ≤ j ≤ p} ∪ {v}
22    forall K_i ∈ K
23       if K_i ∩ L ≠ ∅ then delete K_i
24    K_1 := L
24 return K_1
```

Figure 3.4: **Approximation algorithm for V-STG** *Here we define $SP(v, w)$ to contain the intermediate vertices of a shortest path from v to w.*

Proof. The first step of the algorithm consists in the computation of shortest paths for all pairs of vertices and can be carried out by Johnson's algorithm in $O(n^2 \cdot \log n + n \cdot m)$ [CLRS01]. As the cardinality of S is bounded by k, the iteration over the remaining `for`-loops takes time $O(k^2 + n \cdot k^2 \cdot (k + k \cdot \log k)) = O(n \cdot k^3 \cdot \log k)$. □

As for our applications the number of terminals k is rather small, the most time consuming part of the algorithm is the computation of All-Pairs Shortest Paths (*line 02/03*).

Guha and Khuller [GK99] improved the constant factor of the performance ratio to $1.35 + \epsilon$ for any constant $\epsilon > 0$. Again, they do not provide the exact running time of the algorithm but point out that the algorithm, "although polynomial, is not very practical due to its high running time". Furthermore, they developed a simple greedy algorithm with a worst-case approximation factor of $1.6103 \ln k$ that has the same time complexity as the approximation algorithm by Klein and Ravi. Basically, it works analogously to the algorithm by Klein and Ravi, but considers more sub cases to select trees with minimum ratio in the iteration step.

3.3.2 Reduction rules

We start by investigating the adaptability of the reduction rules for STG taken from Duin and Volgenant [DV89] as given in Section 3.2.2. After that, we introduce some new reduction rules for V-STG.

(Non)adaptable rules from STG

Unfortunately, most of the reduction rules for the STG problem cannot be applied without loss of efficiency or cannot be applied at all to the vertex-weighted case.

One of the main problems is that many rules for STG inspect each edge e as follows. If it can be determined that for *every* solution containing e there is an alternative solution *not* containing e that is at least as good, e can be deleted. Usually, reduction rules make this decision based on e's endpoints only. To use the analogous way to delete a vertex we have to investigate subsets of its neighbors. This seems to be more costly in terms of running time and less promising in its effectiveness. We illustrate this problem for the **Least Cost Test** rule, one of the simplest rules for STG. Least Cost Test as defined for STG removes an edge e if there is a path that connects its endpoints that weighs less than e itself. Removing a vertex

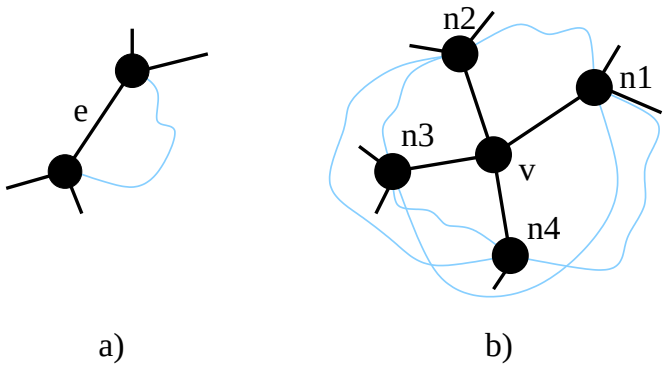

Figure 3.5: **Least Cost Test for STG and V-STG.** *a) shows an example for STG. We have to consider a shortest path (grey) between the endpoints of e. b) shows the situation for a vertex v with degree-four from a V-STG instance. All shortest paths that have to be considered are shown in grey.*

v of a V-STG instance in a similar way is only possible if there are paths between all pairs of its neighbors that do not include v and weigh less than v. Figure 3.5 demonstrates the more complex situation for V-STG. Apart from the fact that the running time grows, the rule is also less likely to apply as the weight of all considered shortest paths has to be less than the weight of v. Analogously, the rules **Vertices Nearer to S Test** and **Smaller Special Distance Test** work in this way and hence seem not to be useful for V-STG.

In the case of **Nearest Vertex Test** it is not even obvious how to transform the reduction rule for V-STG. It determines a Steiner edge incident to a terminal k if all other incident edges have a weight that is too high to be part of an optimal solution. This cannot be done for V-STG because in contrast to the incident edges of a terminal its neighbors cannot be investigated locally as they could be useful to connect other terminals.

Another problem arises from the fact that some reduction rules involve the computation of a minimum spanning tree to determine edges. As discussed in Section 3.3.1 this does not translate to V-STG. For this reason **Nearest Special Vertices Test** cannot be applied to V-STG.

Rule **Degree Test** works only for non-terminals with degree one that will be referred to as **Degree-One Rule**. In the next paragraph we give some special cases in which we can reduce vertices with degree two.

Generally, different kinds of **Reachability Tests** can be used for V-STG as well.

We can directly use *i*-**Dvfree Rule** formulated for STG (Section 3.2.2) for V-STG.

New reduction rules

The following simple reduction rules attack degree-one and degree-two vertices in a V-STG instance.

Reduction Rule 1. (Terminal Rule) *Remove a terminal with degree one and add its neighbor to the set of terminals.*

Reduction Rule 2. (Adjacent Terminals Rule) *If there are two adjacent terminals, contract them.*

Reduction Rule 3. (Path Rule) *If there are two adjacent vertices with degree two, contract them to one vertex. The weight of the new vertex is the sum of the weights of all contracted vertices.*

Reduction Rule 4. (Connected Neighbor Rule) *Remove a non-terminal vertex with degree two if there is an edge between its neighbors.*

Reduction Rule 5. (Diamond Rule) *If there are two or more non-terminal vertices with degree two vertices with the same neighbors, remove all of them except the one with the minimum weight.*

Obviously, reduction rules 7-11 do not change the size of a Steiner minimum tree. Furthermore, Terminal Rule can be carried out in time $O(k)$ and Adjacent Terminals Rule in $O(k^2)$ (if the existence of an edge can be tested in constant time). All vertices with degree two can be found in time $O(n)$. As investigating the neighborhood of a degree-two vertex can be done in constant time, Path Rule, Connected Neighbor Rule and Diamond Rule can be applied to all vertices in the graph in time $O(n)$.

Now, we define new reduction rules that investigate the local neighborhood of a vertex. Note that these rules cannot be directly used for STG.

First, we consider the neighborhood of terminals. We need the following definition.

Definition 3.8. *Regarding the neighborhood of a terminal v, we call a non-terminal vertex $i \in N(v)$* isolated *if $N(i) \subseteq N[v]$.*

Note, that we only have to consider non-terminal isolated vertices, as otherwise we could apply Adjacent Terminal Rule to contract the two terminals. We can deal with isolated vertices by the following reduction rule.

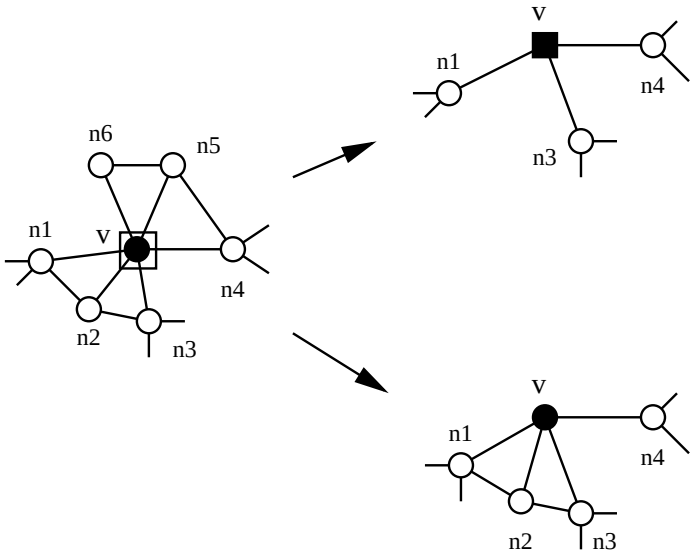

Figure 3.6: **Iso Rule 1 + 2** *We show the effect of Iso Rule 1 in contrast to the effect of Iso Rule 2. For the application of Iso Rule 1 we consider v to be a terminal and we can find three isolated vertices and delete them (on top). To apply Iso Rule 2 we consider v to be non-terminal and then can remove only two isolated vertices (on bottom).*

Reduction Rule 6. (Iso Rule 1) *Remove isolated vertices in the neighborhood of a terminal.*

Lemma 3.9. *The weight of a Steiner minimum tree is not changed by applying Iso Rule 1.*

Proof. An isolated vertex i of the neighborhood of a terminal v cannot be part of a Steiner minimum tree. This is due to the fact that all vertices that can be connected by i are already connected by v itself in a Steiner minimum tree. □

Lemma 3.10. *Iso Rule 1 can be carried out in time $O(n^3)$.*

Proof. To find the isolated vertices in the neighborhood of a vertex we have to consider the neighborhood of all its neighbors. This can be done in time $O(n^2)$. Iterating over all vertices in V yields the running time $O(n^3)$. □

We now investigate criteria to remove vertices in the neighborhood of non-terminals. Here, we need a more restricted definition of isolated vertices.

The problem is illustrated in Figure 3.6: As v has not to be part of a solution Steiner tree, the neighbor n_2 may possibly connect n_1 and n_3 in a cheaper way than v.

Definition 3.11. *We call a vertex $l \in N(v)$* linking vertex *if there is a vertex $n \in N(l)$ so that $n \notin N[v]$. The* linking set *$L[v]$ of a vertex v consists of all linking vertices of its neighborhood.*

Definition 3.12. *A vertex $p \in N[v]$ is an isolated vertex if there is exactly one linking vertex $l' \in L[v]$ so that every path from p to a vertex $l \in L[v]$ contains v or l'.*

We can now formulate a reduction rule analogously to Iso Rule 1:

Reduction Rule 7. (Iso Rule 2) *Remove non-terminal isolated vertices in the neighborhood of a non-terminal.*

Lemma 3.13. *Applying Iso Rule 2 does not affect the weight of a Steiner minimum tree.*

Proof. An isolated vertex as defined for the neighborhood of a non-terminal in Definition 3.12 can at most connect v and one of the neighbors n of v. If it is part of a Steiner tree it can be replaced by the edge between v and n. □

Lemma 3.14. *Iso Rule 2 can be carried out in time $O(n^3)$.*

Proof. In a first step, the linking set in the neighborhood of a vertex can be obtained analogously to Iso Rule 1 in time $O(n^2)$. Secondly, we have to test the connectivity for less than n non-linking vertices, which can be done in linear-time by depth-first search. Again, the iteration over all vertices leads to the running time $O(n^3)$. □

Even if these rules can be considered as a special case of i-Dvfree rule, they can still be useful because we can also apply them to dvfree components with a large number of boundary vertices and their running times are comparatively small.

3.4 Network Properties

Many biological networks seem to have similar characteristics. For example, considering protein interaction networks Jeong et al. [JMALO01] investigated networks of the two different organisms *S. cerevisiae* and *Helicobacter pyroli*. For both organisms they found "highly connected inhomogeneous scale-free networks in which a few highly connected proteins play a

central role in mediating interactions among numerous, less connected proteins" [JMALO01]. The small diameter of the yeast network increases rapidly when the most connected proteins are eliminated from the network. Jeong et al. [JMALO01] state that the detected characteristics are general network properties, which are likely to be found in protein interaction networks of other organisms as well.

Note that it is not obvious whether some characteristics observed for interaction networks arise from "natural" network structure or are a consequence of wrong or incomplete experimental results used to derive the network. The existence of a relatively small number of high-degree vertices is a typical example that could partly result from the fact that usually such vertices correspond to exceedingly well-studied proteins or genes. Therefore, we know about much more interaction corresponding to such proteins than about interactions corresponding to less well-studied proteins. Nevertheless, a hint that experimental errors are not the only reason for this property is given by the results of [JMALO01], which show evidence that the most highly connected proteins are also the most important for the survival of a cell.

As our goal is to compute Steiner trees for the interaction network of yeast, we analyze some of its network properties in the hope of finding useful starting points for data reduction or other algorithmic approaches. The next subsections examine the yeast interaction network as described in Section 3.1. We look at its largest connected component consisting of 5421 vertices and 21594 (undirected) edges.

3.4.1 Degree distribution

The degree distribution of the yeast interaction network is shown in Table 3.1 and Figure 3.7. Whereas about a fifth of all vertices have degree one, there are only very few vertices with high degree. More than 95% of all vertices have a degree lower than 30.

3.4.2 Diameter

The distribution of the lengths of all pair shortest paths is given in Figure 3.8. The diameter of the network is only 9. Furthermore, after the deletion of all vertices with degree one the diameter decreases from 9 to 7. Intuitively, this makes it even harder to tackle the remaining network by data reduction.

deg	#vertices	deg	#vertices	deg	#vertices	deg	#vertices	deg	#vertices
1	1078	25	21	49	5	78	1	116	1
2	932	26	16	50	3	79	2	120	1
3	692	27	17	52	3	80	2	124	1
4	509	28	17	54	1	81	2	125	1
5	364	29	12	55	1	82	2	126	1
6	281	30	12	56	1	83	1	127	1
7	222	31	14	57	3	86	1	138	1
8	156	32	11	58	1	88	1	144	1
9	137	33	14	59	3	89	1	145	1
10	105	34	8	61	1	90	1	155	2
11	109	35	7	62	1	91	1	156	2
12	73	36	8	63	1	92	2	158	1
13	69	37	5	64	2	96	1	161	1
14	52	38	7	65	1	97	2	163	1
15	65	39	7	66	2	100	1	199	1
16	45	40	9	67	3	101	1	205	1
17	40	41	4	68	2	104	1	292	2
18	38	42	5	69	2	105	3	458	1
19	40	43	6	70	1	106	1		
20	25	44	2	71	3	107	1		
21	24	45	8	72	2	109	1		
22	20	46	3	74	1	110	2		
23	25	47	4	75	4	112	2		
24	26	48	6	77	2	113	1		

Table 3.1: **Degree distribution** *Number of vertices (#vertices) of given degree (deg) (not counting self-loops) for the yeast interaction network.*

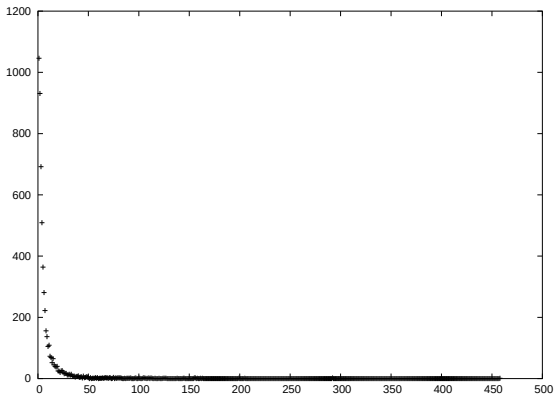

Figure 3.7: **Degree distribution** *The diagram plots the degree of vertices (x-axis) against the number of vertices with this degree (y-axis).*

length	number of paths
1	21594
2	663296
3	4168906
4	7281225
5	2213648
6	311761
7	28879
8	1538
9	63

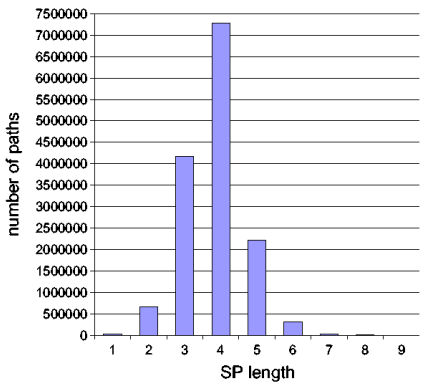

Figure 3.8: **Distribution of all pair shortest path lengths** *for the unweighted yeast interaction network.*

deg	component size	number of removed vertices
50	4488	96
40	4246	142
30	3872	232
25	3578	306
20	3171	422
15	2535	610
10	1187	978
5	116	1679

Figure 3.9: **High degree separators** *of the yeast interaction network. All vertices with degree higher than deg are removed. Component size denotes the size of the largest connected component after removing them.*

3.4.3 Separability

In this section, we investigate the "separability" of the yeast interaction network. For some algorithmic approaches like divide-and-conquer or tree decomposition-based algorithms it is necessary, in a first step, to find reasonable separators of a graph, i.e. a small subset of vertices after deletion of which the graph consists of unconnected components of limited size. The question for a separator of size k such that the remaining components are smaller than a given limit (such as half of the number of vertices of the original graph) is NP-hard [GJ79]. Therefore, to gain more information about the network structure we use a heuristic approach that attempts to find good separators by deleting vertices with high degree. Figure 3.9 shows the size of the largest remaining connected component of the yeast interaction network after removing vertices with high degree for a varying degree threshold beyond which a vertex is considered as high-degree vertex. It does not seem possible to separate the network into small subnetworks by removing only a small number of high-degree vertices. These results and the very small diameter of the network indicate that it is unlikely and, in general, difficult to find separators of a reasonable size for further algorithmic approaches. Note that, e.g. in the case of path or tree decomposition-based algorithms, the running time depends exponentially on the size of the bags and the elements of each bag have to separate the graph.

Another study that investigates the connectivity of a subnetwork of the yeast protein interaction network that consists of 778 vertices and includes only high-confidence interactions as edges is provided by Han et al. [HBH$^+$04]. They tested the connectivity of the network after removing vertices with degree higher than five which they denote as *hubs*. They make a distinction between two different kinds of hubs depending on a protein interacts with its different partners simultaneously, called *party hubs*, or at different times or locations, referred to as *date hubs*. It turns out that the removal of the date hubs (contrary to the removal of the party hubs) splits the network into small subnetworks. In the given example they are able to split the connected component with 778 vertices of the interaction network into subnetworks with about 200 or less vertices by removing all 78 date hubs. By removing the 86 party hubs or 86 randomly selected vertices the size of the largest remaining component is over 400. Although the set of all data hubs separates the network studied in [HBH$^+$04] quite well, its size is still too large for further separator-based algorithmic approaches. Moreover, as the here considered network is much smaller than our yeast interaction network from this work, it seems unlikely that applying this ap-

proach without further improvements to our network would lead to efficient separators.

Recapitulating this section, from the small diameter and the degree distribution we can follow, that the yeast interaction network is highly connected by few high-degree vertices. Furthermore, the separation of the network into small pieces seems to be difficult. In the next section, we try to use the knowledge about the network structure for the design of a software tool.

3.5 Software: The Steiner Package

In this section we describe a software tool, called *Steiner package*, that was developed in this thesis within the scope of a project conducted by the working group of Mike Hallett (McGill Center for Bioinformatics, Montreal, Canada). The goal is to make the Steiner approach available to users, e.g. from biochemistry with no computational or graph-theoretical background.

The Steiner Package integrates biological preprocessing, graph-theoretical data reduction and different algorithms to compute Steiner trees for biological networks. More precisely, it includes an implementation of two algorithms: (1) The Dreyfus-Wagner algorithm, described in Section 3.2.1, is used to determine an optimal solution unless prohibitively expensive. (2) The approximation algorithm from Klein and Ravi, described in Section 3.3.1, computes an approximative Steiner tree for any instances.

To make the Dreyfus-Wagner algorithm applicable to a wider range of instances, we include some graph-theoretical data reduction and biological preprocessing rules. In contrast to data reduction that is based on network information, the biological preprocessing depends on the meaning of the vertices in a biological network. As we are especially interested in obtaining an optimal solution if possible, we implemented data reduction and preprocessing rules that can affect the network in a way such that it can be solved by the Dreyfus-Wagner algorithm.

My contribution of this work to the Steiner Package is the development and implementation of biological preprocessing and graph-theoretical data reduction rules and an implementation of the approximation algorithm. Furthermore, I was concerned with working out the structure of the program that determines a reasonable order of preprocessing and reduction rules and combines them with the algorithms. The Steiner Package assimilates an implementation of the Dreyfus-Wagner algorithm by T. J. Perkins (McGill Center for Bioinformatics, Montreal, Canada).

We start with a short description of the input and output of the Steiner

Package and its options. We continue with the explanation of graph-theoretical data reduction (Section 3.5.2) and biological preprocessing rules (Section 3.5.3) that are part of the implementation and discuss how they can be combined in Section 3.5.4. Then, we give an overview of the structure of the program and its interfaces (Section 3.5.5). We end the section by giving examples for Steiner trees that have been computed using the Steiner Package (Section 3.5.6).

3.5.1 Input, output, and options

This subsection provides a brief overview on the input that has to be provided to the Steiner Package and the most important results that are returned by the program as this is sufficient for the understanding of the next subsections. For a detailed description of the interface we refer to Section 3.5.5.

To run the program the user must provide network data and a set of terminals. Optionally he/she can provide his/her own data for biological preprocessing as further described in Section 3.5.3. Furthermore, the user can choose an interactive mode that shows possible information for biological preprocessing provided by the MIPS database (described in Section 2.3) and then select the part of the information he wants to use.

The output contains a Steiner tree computed by the Dreyfus-Wagner algorithm if the number of terminals is smaller than a given threshold and always a Steiner tree computed by the approximation algorithm. The Steiner Package also provides information describing the results of data reduction and preprocessing.

3.5.2 Selection of graph-theoretical data reduction rules

To avoid the assimilation of futile rules that needlessly consume running time, we investigated the effectiveness of the reduction rules described in Section 3.3.2 applied to the yeast interaction network. As its network properties seem to be typical for biological networks, we use the results to select reduction rules for our implementation. We made the following observations:

Iso Rules Iso Rule 1 considers the neighborhood of terminals and therefore its overall performance depends on which vertices belong to the terminal set. To obtain a general idea about its effectiveness we treated every vertex as terminal with only non-terminal neighbors. It turned out that Iso Rule 1 applied to all vertices under that assumption still performed very badly. Most of the isolated vertices that we found had a degree of one or two

and could therefore be deleted by the simpler Deg-One Rule or Adjacent Neighbor Rule. Altogether out of 5,458 vertices we detected 13 isolated vertices of degree three and one isolated vertex with degree four that could be deleted with Iso Rule 1 (if they are in the neighborhood of a terminal). Furthermore, this also means for Iso Rule 2 that there are at most 14 vertices that are possible candidates for deletion. Therefore, we can conclude that Iso Rules 1 and 2 are not useful for data reduction in our case.

Dvfree Rules If we try to apply i-dvfree Rule for pairs or triples of all vertices, we find only few very small dvfree components spending hours of running time, e.g. we could delete less than 100 vertices by applying 2-dvfree Rule. The reason for the bad performance is that the graph is well connected as discussed in Section 3.4.2, so that it is very unlikely to find small subsets of vertices that separate the graph.

Reachability Tests We omitted an implementation of any kind of reachability tests as they are very unlikely to be effective because of the small diameter of the network.

Degree Rules Looking at the degree distribution of our network we can obviously remove nearly one fifth of the vertices because they have degree one and additionally, we can remove vertices of degree two with Path Rule, Connected Neighbor Rule and Diamond Rule.

Rules for terminals Terminal Rule and Adjacent Terminals Rule depend on the vertices that are part of the terminal set and therefore cannot be tested for the network in general. In our further experiments (Section 3.5.6) they turned out to be extremely useful.

Summarizing, many of the more "complicated rules" do not seem to apply to the structure of the biological networks, whereas the simple rules are very effective.

We do not include i-dvfree Rule, Iso Rule 1 + 2, and any kind of reachability test in the Steiner Package as they seem to consume running time without yielding considerable success in reducing biological networks.

As they significantly decrease the size of the network and cost very little running time the Steiner Package comprises Deg-1 Rule and some of the rules attacking vertices of degree two. To simplify matters we summarize them for the further description of the Steiner package and refer to them as

deg1/deg2-rules. Note, that the deletion of degree-one or degree-two vertices before applying the Dreyfus-Wagner algorithm only affects the running time of the computation of all-pairs shortest paths and not the running time of the dynamic programming step itself. This is due to the fact that in the dynamic programming one does not have to test non-terminals with degree less than three.

Furthermore, Terminal Rule and Adjacent Terminal Rule are part of the Steiner Package implementation. We denote them as *special-rules*. As shown in the experimental part of this work (Section 3.5.6), they can become extremely useful because they are able to decrease the number of terminals and therefore the value of the parameter of the Dreyfus-Wagner algorithm.

3.5.3 Biological preprocessing

In many cases the graph-theoretical data reduction is not effective enough to decrease the number of distinguished vertices to a size that is able to be computed by the Dreyfus-Wagner algorithm. Thus, we attempt to additionally use biological information to reduce the number of distinguished vertices, while still allowing for a result with reasonable biological meaning even if not necessarily optimal from a graph-theoretical point of view. Furthermore, biological preprocessing generally allows a user to bring in his biochemical knowledge to influence the solution of the Steiner tree.

We distinguish between the following two kinds of biological preprocessing.

Complex-based preprocessing. We contract a group of distinguished vertices to a single vertex if we can find a meaningful criterion that this group can be considered as entity in the cell. In the case of a protein interaction network, a typical example is the contraction of vertices that are part of some complex. Other criteria could be based upon expression profiles, such as those generated from microarray experiments. We denote this kind of preprocessing by the keyword *complex* .

Function-based preprocessing A second strategy for biological preprocessing is to compute a "local" Steiner tree for a subset of distinguished vertices. Then, the vertices of the local Steiner Tree are added to the solution and contracted for the further computation. This kind of data reduction would be applicable for a group of vertices belonging to the same pathway or participating in the same cellular process. We denote this kind of preprocessing by the keyword *function*.

```
complex
SAGA
YDR448W
YDR176W
YGR252W
YOL148C
function
vacu
YAL002W
YPL045W
YJR075W
YLR295C
...
```

Table 3.2: **User data.** *In a file provided by the user every group of genes has to start with the keyword "complex" or "function" followed by a name for the complex/function, which is then followed by the names of genes. The name of the contracted vertex will be the given name of the complex/function. For this example we would contract the vertices YDR448W, YDR176W, YGR252W, and YOL148C to a vertex called SAGA. Analogously, the vertices that follow the keyword "function" and the name "vacu" are treated as described by the function-based preprocessing.*

Data presentation

In the case of biological preprocessing, it is important to use only experimentally rigorous information and to let the user control the process. Therefore, the following two avenues are available for data presentation.

The first possibility is to provide your own data in the file format as given in Table 3.2.

The other choice is to use data from the Munich Information center for Protein Sequences (MIPS). As discussed in Section 2.3 it provides highly accurated information about protein complexes. The information is made accessible to the user by choosing an interactive mode. Then, the program displays all complexes of MIPS that contain more than one distinguished vertex as shown in Table 3.3 and the user can enter the numbers of the complexes he wishes to contract. For further information about the complexes we refer to the the MIPS home page [fPSM].

```
The following complexes containing special vertices are provided by mips:
complex number:  133.40 YCR081W YPL042C
complex number:  220 YLR447C YHR026W YOR332W YDL185W YPR036W YHR039CA
complex number:  510.190.10.20.10 YGR252W YDR176W YOL148C
...
```

Table 3.3: **MIPS data.** *Output in the interactive mode. Every complex number is followed by the names of the distinguished vertices that are part of the complex. For example, complex number 133.40 contains YCR081W and YPL042C. The complex numbers are identical to the numbers provided by MIPS.*

3.5.4 Conflicts of data reduction and preprocessing

We now consider some conflicts that arise by combining different graph-theoretical data reduction and biological preprocessing rules.

Ambiguous vertices

One protein can be part of different complexes or it can be an element of different pathways. In this case, we have to make sure that we do not contract the vertices of different complexes to one vertex. Therefore, we treat a vertex appearing in more than one complex or corresponding to more than one function in the following way. We delete it from all complexes (or function groups) and give a warning (which allows the user to change the provided information). If a vertex is part of diverse groups, the complex information provided by the user has higher priority than the function information which in turn has higher priority than the complex information based on MIPS-data. For example, if one vertex is part of a user complex and a function group, it is only removed from the function group.

Order of reduction rules and preprocessing

The biological preprocessing which we use depends on the meaning of the particular vertices. In contrast, the graph-theoretical special-rules, i.e. Terminal Rule and Adjacent Terminals Rule, only take into account the network structure. We therefore apply the biological preprocessing first. Otherwise, we would have to define in which case a vertex resulted from contraction by the special-rules is part of a complex. In our experiments, in most cases this order of data reduction decreases the number of terminals as much as possible. We describe one exception concerning a degree-one vertex which

is part of a complex. In this case, applying the biological preprocessing will yield a contracted vertex with degree higher than one. After that we cannot use the special-rules for a terminal with degree-one and put its neighbor v into the set of terminal vertices. If, additionally, the neighbor vertex v has some terminal neighbors, it could be starting point for Adjacent Terminal Rule which contracts terminals. Therefore, we can decrease the number of terminals stronger if we ignore this degree-one vertex in the biological preprocessing and attack it with the special-rules. Indeed, in some examples we could achieve a smaller number of terminals after data reduction if we remove a terminal with degree one from a complex. As it is hard to decide what works best in general, the program prints a warning for every degree one terminal which is part of a complex (function) and is used for data reduction. If the user is not absolutely sure about the complex information, he/she can try to achieve a stronger reduction by removing the vertex from the complex.

Implemented algorithms

For the computation of an optimal solution the Dreyfus-Wagner algorithm is applied if the number of terminals after data reduction is small enough. The largest size of a terminal set for which we were able to run the Dreyfus-Wagner algorithm was 13. On a standard Linux PC with one gigabyte main memory and a 2,26 GHz processor, the running time was more than 20 hours. To prevent from starting the algorithm in such a time consuming case without informing the user, the Dreyfus-Wagner algorithm is only applied automatically if the number of terminals is smaller than 12. In this case the running time is up to several minutes. Otherwise, the user can choose to run it or not.

The Steiner Package always provides a solution obtained using the approximation algorithm by Klein and Ravi as described in Section 3.3.1. As the running time of the algorithm does not depend on the size of the terminal set, it is applied to the network without any data reduction.

3.5.5 Description and usage of software

The Steiner Package is implemented in C++ and was compiled with the g++ compiler with compiler options "`-O2 -g`". The new parts of the implementation, i.e. everything except the Dreyfus-Wagner algorithm, consist of more than 2000 lines of code. We now describe the structure of the program and its user interface.

We apply the reduction rules and algorithms in the following order:

1. deg1/deg2-rules
2. biological preprocessing (user based information is processed before MIPS data)
3. special-rules (alternating, beginning with Terminal Rule)
4. deg1/deg2-rules
5. Dreyfus-Wagner, if number of terminals is less than 12 or selected by the user
6. approximation algorithm (runs independently from the data reduction.)

The user interface is described in the following. Note that as a graphical interface was not part of this work, the interface is based on command line parameters and standard input and output.

INPUT: (command line parameters in the given order)

- edge-file: each row contains names of two terminals, if and only if there is an edge between them
- file with names of terminals
- choice of MIPS-reduction ('y'/'n'): 'y' will change to an interactive mode where you can choose to use complex information provided by MIPS for biological data reduction (see Table 3.3).
- file with weights: Each row contains a vertex name followed by its weight. If you consider unit weights and do not provide complex information, the weight file is optional. (In case you wish to provide complex information for a network with unit weights, you can provide an incorrect file name. This results an error message to the standard output and the initialization of all weights with 1.0.)
- file with complex information (optional) (Table 3.2)

OUTPUT:

- Steiner tree computed by Dreyfus-Wagner algorithm, if possible (standard output)

- approximation Steiner tree (filename: `apprST`)
- special notes/warnings starting with "!!!!!" (standard output)
  - names of non-reachable terminals if there are any
  - names of terminals with degree one being part of a complex or function group (see Section 3.5.4)
  - names of terminals which are part of more than one complex or function group (for explanation see Section 3.5.4)
- local Steiner trees used for biological data reduction as described in Section 3.5.3 (filename: `localSTs`
- results of graph-theoretical special reduction as described in Section 3.5.2 (filename: `special_results`)
- files for visualization of the Steiner tree. More precisely, the undirected induced subgraph of Steiner nodes and terminals computed with data reduction in combination with the Dreyfus-Wagner algorithm (filename: `ospreyDW.txt`) and with the approximation algorithm (filename: `ospreyAppr.txt`). The osprey-format consists of the headline "GeneA GeneB" followed by the edges, that means there are two vertex names in a row iff there is an edge between the vertices.
- degree-distribution of the terminals and Steiner nodes in the unreduced network for the solution of the Dreyfus-Wagner algorithm (filename: `degDisDW`) and of the approximation algorithm (filename: `degDisAppr`); the degree distribution gives further information about the importance of the considered vertices in the whole network.

3.5.6 Results and examples

To demonstrate the application of the Steiner Package and to show its usefulness we give three examples for the computation of Steiner trees in unweighted networks. All experiments were run on a standard Linux PC with 2,26 GHz processor and one gigabyte main memory.

Example 1a: GAL80 deletion set

We considered the yeast interaction network and a distinguished set consisting of ten genes obtained by a GAL80 deletion experiment that was computed as an example for [SPB$^+$05] in the yeast interaction network. The data has

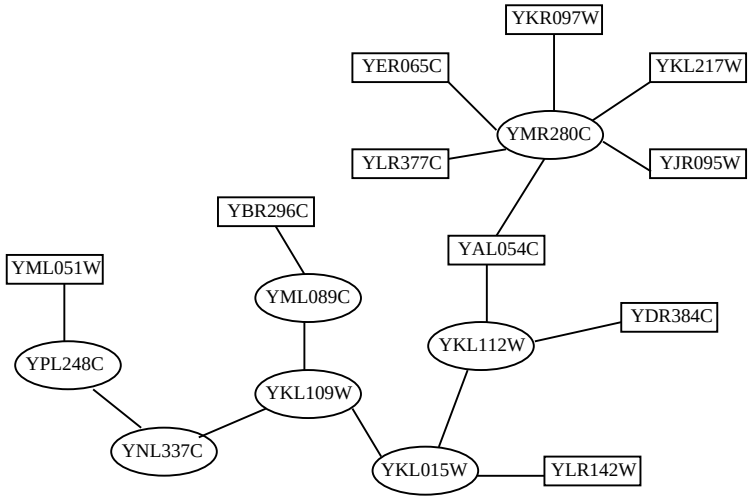

Figure 3.10: **Steiner tree for the GAL80 deletion set**. *The distinguished vertices are given in rectangular boxes.*

been obtained from the results of GAL80 deletion microarray expression experiments from [ITR+01] (as described in Section 2.2). More precisely, the deletion of the gene GAL80 in the absence of galactose leads to a large set of genes that is highly over-expressed compared to wild-type yeast grown in the presence of galactose. The distinguished set of this example contains all genes whose average log-ratio of expression is greater than two.

For this example the biological analysis of [SPB+05] based on the Steiner approach allowed for the re-discovery of known interaction networks (as the Steiner tree contains some well known interaction subnetworks). Furthermore, it suggests that GAL80 may play a role in controlling the so-called *diauxic shift*: the shift from glucose to ethanol metabolism.

For the computation of the Steiner tree we did not use any biological preprocessing, which means we chose no MIPS preprocessing and did not provide any complex information. The resulting Steiner tree is illustrated in Figure 3.10.

The connected component of the yeast interaction network that contains all distinguished of this example vertices consists of 5421 vertices and 21594 undirected edges. With deg1/deg2-rule we could remove 1085 degree-one and 82 degree-two vertices that were not distinguished. Applying the special-rules decreased the number of distinguished vertices from ten to five. Table 3.4 shows the output of the file `special_results`.

```
deg1:  terminal -> Steiner Node
YJR095W -> YMR280C

Special neighbors:
component 1:
YAL054C
YMR280C
YLR377C
YER065C
YKR097W
YKL217W
```

Table 3.4: **Output file special results.** *The terminal YJR095W has degree one and is therefore deleted and YMR280C is put into the terminal set by Terminal Rule. As you can also see in Figure 3.10, the terminals specified by* `component 1` *are neighbors and can therefore be contracted by Adjacent Terminals Rule.*

```
YJR095W : YMR280C
YAL054C : YKL112W YMR280C
YMR280C : YAL054C YLR377C YER065C YKR097W YJR095W YKL217W
YLR377C : YMR280C
YER065C : YMR280C
YKR097W : YMR280C
YKL217W : YMR280C
YKL109W : YKL015W YML089C YNL337W
YNL337W : YPL248C YKL109W
YPL248C : YML051W YNL337W
YML051W : YPL248C
YML089C : YKL109W YBR296C
YBR296C : YML089C
YKL015W : YLR142W YKL109W YKL112W
YKL112W : YAL054C YKL015W YDR384C
YDR384C : YKL112W
YLR142W : YKL015W
```

Table 3.5: **Output of a Steiner minimum tree for GAL80 deletion set from Example 1a.** *Every row contains a terminal or Steiner nodes followed by its neighbors in the Steiner tree.*

Dreyfus-Wagner and the approximation algorithm computed the same Steiner tree of size 17 that is printed to the standard output (for the DW-algorithm) and to the file `apprST` as shown by Table 3.5. For small sizes of the distinguished set, the approximation algorithm seems to perform very well. In the course of this work we computed Steiner trees for more than ten different sets of distinguished genes with size up to twelve. In these examples the size of the Steiner trees obtained by the approximation algorithm did not exceed the optimal solution by more than one or two.

The computation of the Steiner tree with both algorithms took less than a minute. The most time-consuming part of the approximation algorithm is the computation of all-pairs shortest paths independently from the size of the distinguished set. Generally, for the Dreyfus-Wagner algorithm the running time increases with the number of distinguished vertices up to hours for a distinguished set of size more than 10. That means that in this example the running time could be drastically decreased by applying the reduction rules.

Example 1b: Extended GAL80 deletion set

In another experiment we were interested in a set of distinguished genes that has been obtained analogously to Example 1a with a threshold of $+1.7$ instead of $+2.0$ for the average log-ratio expression. This leads to a distinguished set consisting of 15 genes (YML051W, YDR384C, YAL054C, YER065C, YBR296C, YKL217W, YKR097W, YHR137W, YPL061W, YLR142W, YDR256C, YLR377C, YPR001W, YJR095W, and YHR139C) for which we could not run the Dreyfus-Wagner algorithm directly. By applying the graph-theoretical data reduction rules, the size of the distinguished set was reduced to nine and we could compute an optimal solution of size 28 in less than half an hour. The size of an approximation Steiner tree obtained by the algorithm from Klein and Ravi is 30.

Example 2: Fluconazole knock-out set

In this example the distinguished set consists of 53 genes that have been experimentally detected by the group of David Y. Thomas (Department of Biochemistry, McGill Faculty of Medicine, Montreal, Canada). The absence of the obtained genes leads to the death of yeast grown in the presence of the drug fluconazole (which is only fungi-static otherwise). The computation of a Steiner tree for the 53 genes is part of the analytical process that should find out interrelations between the genes of the set. We consider the protein

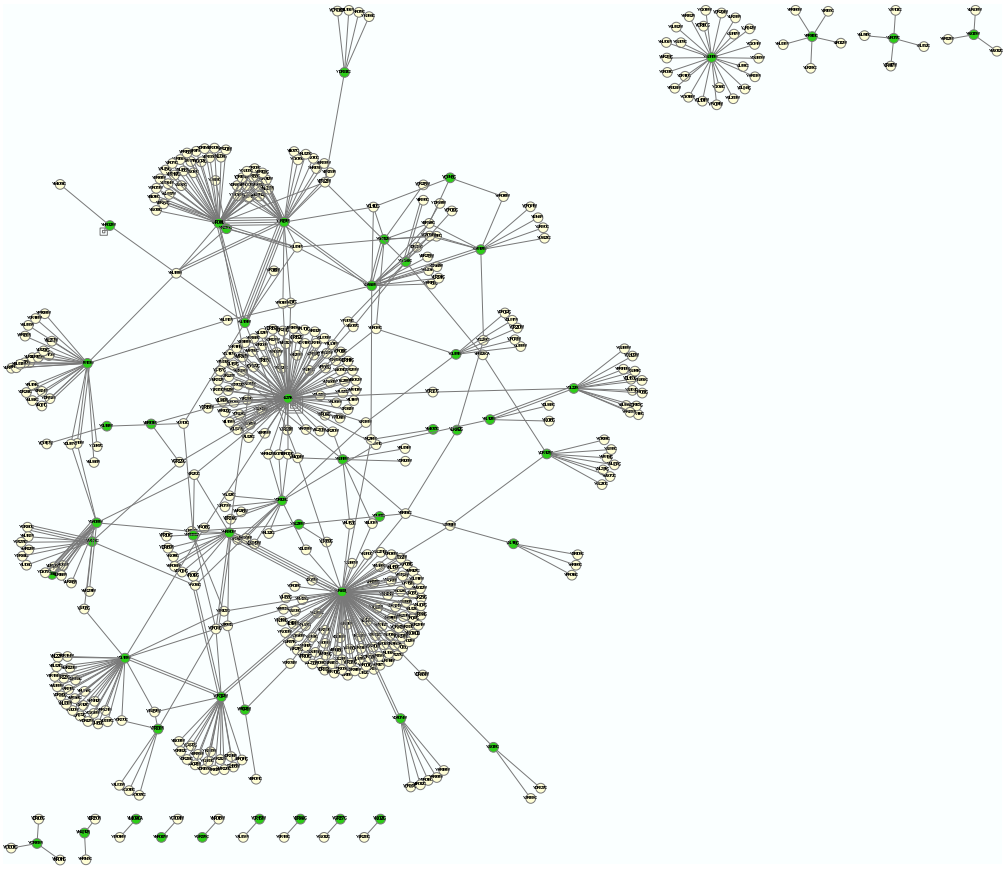

Figure 3.11: **Fluconazole knock-out set.** The figure shows the distinguished vertices (grey) and their neighbors in the protein interaction network of yeast. As the graph-theoretical Adjacent Terminals Rule applies to terminals that are neighbors, it can only attack some of the distinguished vertices that are part of the largest connected component. The picture was generated with Cytoscape (see Section 2.3) and is provided by Scott Bunnell (McGill Center of Bioinformatics, Montreal, Canada).

interaction network based data from BIND. The set of distinguished genes with all neighbors in the network is illustrated in Figure 3.11. Here, we use this example to show the potential of biological preprocessing.

In our initial examination of the data, two of the 53 proteins were not part of the considered network, which means BIND contains no information about interactions between them and other proteins. Two more proteins were not reachable from the other distinguished proteins. Therefore, we computed a Steiner tree for the remaining 49 distinguished proteins that were part of a connected component consisting of 4831 vertices and 17106 edges.

First, we tried to compute a Steiner tree without any biological preprocessing. The graph-theoretical reduction rules were able to reduce the size of the distinguished set from 49 to 32 and, the Dreyfus-Wagner algorithm could not be applied yet. The size of the Steiner tree obtained by the approximation algorithm was 85.

In the second experimental scenario we additionally applied biological preprocessing based on complex information. There are three well-known complexes containing distinguished proteins:

- **SAGA:** Four of the distinguished proteins are part of the SAGA complex, which is important for transcription in vivo and possesses histone acetylation function [SGR+99].
- **RNA pol II:** Another four distinguished proteins are associated within the *RNA polymerase II Holoenzyme Multi-protein* complex (more precisely part of Srb).
- **V-ATPase:** The complex V-ATPase, that functions in the accidification of intra-cellular compartments [SF97], includes six distinguished proteins.

Using the given complex information for biological preprocessing and applying the special-rules, the number of distinguished vertices could be decreased to 28—a size still not manageable with the Dreyfus-Wagner algorithm.

Lastly, we additionally provided information about groups of proteins that are involved in the same cellular process (treated as *function* information). More precisely, we grouped six proteins that are required for vacular protein sorting and two that are important for the biosynthesis of chorismate. Two more proteins are connected to cellular functions regarding the yeast nuclear core complex (NPC) and three proteins with the formation

```
VATPase (complex):  YLR447C YOR332W YPR036W YDL185W YHR026W YHR039CA
SRB (complex):  YGL151W YCR081W YHR041C YPL042C
SAGA (complex):  YDR448W YDR176W YGR252W YOL148C
actin (function):  YNL271C YDR129C YCR063W
chorismate (function):  YDR127W YGL148W
vacu (function):  YDR080W YPL045W YMR077C YJL029C YBR131W YHR060W
YKL119C
NPC (function):  YDL116W YLR242C
```

Table 3.6: **Data for biological preprocessing.**

of action filaments or its cytoskeletal organization. All proteins that were used for biological preprocessing are given in Table 3.6. In this scenario the preprocessing worked out extremely well. In combination with the reduction rules, the size of the distinguished set is reduced to 13. Within a bit more than 20 hours a Steiner tree of size 80 was computed by applying the Dreyfus-Wagner algorithm. Note that even though the resulting tree must not necessarily be an optimal Steiner tree from graph-theoretical point of view, it gives in a better solution than the approximation algorithm, which yielded a Steiner tree of size 85.

Altogether, we provided three biological important examples that showed the usefulness of data reduction and preprocessing rules included in the Steiner Package. The data reduction rules in Example 1a reduced the running time for the computation of an optimal Steiner tree from several hours to less than a minute. In Example 1b data reduction made it possible to compute an optimal solution for an otherwise unsolved instance. Lastly, we showed the power of biological preprocessing in Example 2: It allowed for the computation of a biological meaningful Steiner tree for a relatively large number of distinguished vertices, which size is smaller than the size of the computed approximation Steiner tree.

Chapter 4

Parameterized complexity of Steiner tree related problems

Whereas the Steiner method as described in the previous chapter allows for the identification of regulators for a given set of proteins or genes, this chapter is concerned with graph-theoretical problems which help us to screen large networks for "relevant" subnetworks without prior specification of a vertex set of interest. The identification of significant subnetworks in biological networks is an important computational task. Even the interaction network of a simple organism like yeast contains thousands of vertices, a size that makes a manual search very laborious or even impossible. A promising strategy was developed by Ideker et al. [IOSS02]. They successfully searched a vertex-weighted interaction network (as described in Section 2.4.4) for "active subnetworks", i.e. for connected sets of genes with unexpectedly high levels of differential expression. The computational task consists of identifying the highest scoring subnetworks and is defined as Maximum-Weight Connected Subgraph (MWCS) problem in [IOSS02]. The authors provided a proof for the NP-hardness of MWCS and solved it by using a simulated annealing strategy.

In this chapter, we introduce two graph-theoretical problems, Generalized Steiner Tree in Graphs (G-STG) and Vertex-Weighted Generalized Steiner Tree in Graphs (GV-STG)[1], which can be used similarly to MWCS for network analysis as described in Section 4.2. A main contribution of this chapter is to formulate fixed-parameter algorithms with respect to the size of the subgraph for both problems answering an open

[1]Also known as Edge-Weighted k-cardinality Tree problem and Vertex-Weighted k-cardinality Tree problem [FK94]

question posed by Hallett [Hal04][2]. For that, we introduce a new technique for providing fixed-parameter algorithms combining enumeration with the well-known method of color-coding[3]. In contrast, no corresponding fixed-parameter algorithm is known for MWCS.

Furthermore, we give the first systematic study of parameterized complexity of Steiner tree related problems for a range of optimization criteria.

In the following sections, we start with a formal definition of G-STG and GV-STG and give an overview of results and algorithms from literature (Section 4.1). After that, we discuss their biological relevance (Section 4.2). Then, we introduce a general scheme to classify problems from a parameterized point of view (Section 4.3). In a next step, we examine the parameterized complexity of the considered problems for a range of natural parameterizations: Our results include fixed-parameter algorithms as well as proofs of parameterized intractability. As a fundamental building block for our algorithmic results we describe a fixed-parameter algorithm which is an important subroutine for our algorithms (Section 4.4). Finally, we deal with the parameterized complexity of STG, V-STG, MWCS, G-STG and GV-STG (Section 4.5).

4.1 Problem definitions

The formal definition of the MWCS problem is given as follows.

> Maximum-Weight Connected Subgraph (MWCS):
> **Input:** An undirected graph $G = (V, E)$ and a vertex weight $w_v \in \mathbb{R}$ for each $v \in V$.
> **Task:** Find a connected subgraph $G' = (V', E')$ of G with maximum weight $w_{G'} = \sum_{v \in V'} w_v$.

The MWCS problem is substantial for the work of Ideker et al. [IOSS02] for the identification of meaningful subnetworks.

Now, we give the definition of two Steiner tree related problems. Their biological relevance will be discussed in the following section.

[2]Note that during the course of this work a similiar fixed-parameter algorithm was, independently, published by Scott et al. [SIKS05]. Their algorithm directly applies color-coding without enumeration

[3]Note that the same idea was also suggested by Narayanan and Karp [NK04] for a similar variant of the Steiner tree problem

Generalized Vertex-Weighted Steiner Tree in Graphs (GV-STG):
Input: An undirected graph $G = (V, E)$, a vertex weight $w_v \in \mathbb{R}^{\geq 0}$ for each $v \in V$, a positive integer $s \in \mathbb{N}$, and a positive real $l \in \mathbb{R}$.
Task: Find a connected subgraph $G' = (V', E')$ of G with size $|V'| \geq s$ and weight $w_{G'} = \sum_{v \in V'} w_v \leq l$.

Generalized Steiner Tree in Graphs (G-STG):
Input: An undirected graph $G = (V, E)$, an edge weight $w_e \in \mathbb{R}^{\geq 0}$ for each $e \in E$, a positive integer $s \in \mathbb{N}$, and a positive real $l \in \mathbb{R}$.
Task: Find a connected subgraph (Steiner Tree) $G' = (V', E')$ of G with size $|V'| \geq s$ and weight $w_{G'} = \sum_{e \in E'} w_e \leq l$.

Both problems seek to find a subgraph which contains at least a given number of nodes in a weighted graph such that the total weight of the subgraph is less than or equal to a specific limit. Furthermore, we only allow for non-negative weights as this is sufficient for many applications (as described in Section 4.2).

G-STG is defined on a graph with edge weights and GV-STG on a vertex-weighted graph. Note that therefore the resulting subgraph of G-STG usually has tree structure (except for edges with weight zero) and depends on the kind or strength of connections (interactions) between the nodes. In contrast, in the vertex-weighted case, we obtain a subgraph depending only on the "importance" of the vertices and the existence of a connection.

Besides from biological applications, both problem variants arise in various important applications, including oil-field leasing [HJ93], facility layout [FH92], quorum-cast routing [CK94], and telecommunications [GH97].

On general graphs G-STG and GV-STG are NP-complete [RSM+96, FK94]. The G-STG problem is still NP-hard if the edge weights are restricted to $\{1, 2, 3\}$ [RSM+96]. It is solvable in polynomial time if there are only two distinct edge weights in a complete graph [RSM+96]. Furthermore, it is NP-complete for planar graphs, whereas in the case of graphs with constant treewidth there is a polynomial algorithm with running time $O(ns^2)$ [RSM+96]. GV-STG is solvable in polynomial time if the graph G is a tree [FK94]. Furthermore, Faigle and Kern [FK94] proved that GV-STG is NP-hard for grid graphs and split graphs and in P for interval and co-graphs.

The G-STG and GV-STG problems have been extensively studied regarding their approximability. Currently, both can be approximated within

factor 3 [Gar96].

There is very little literature concerned with exact algorithms. All of it is based on integer linear programming using Branch and Bound or Branch and Cut. However, these strategies seem only applicable to very small instances. Brimberg et al. [BUM06] point out that, experimenting with the CPLEX mixed integer solver, they could only solve instances with less than 20 graph vertices.

A large body of literature is available on heuristic methods and experimental results. For example, there are local search algorithms for GV-STG [BE03] and G-STG [BUM06]. Furthermore, there are several metaheuristic approaches, including genetic algorithms and a tabu search based method [BB05].

4.2 Biological relevance of G-STG and GV-STG

An example for a graph theoretical problem with the power to gain insight into molecular processes is given by MWCS [IOSS02]. As discussed in the introduction of this chapter, MWCS is helpful for identifying active subnetworks in an interaction network at large scale.

Another example for the use of advanced graph algorithms for extracting meaningful information from biological networks is given by Scott et al. [SIKS05]. They detected important signaling pathways in an edge-weighted protein interaction network (Section 2.4.1) by searching the network for paths of a given length with maximum weight. Independently from this work, they use the same algorithmic approach as we do to tackle the problem. Whereas in their work they apply the algorithms and verify the results from a biological point of view, the focus of this work is more on the complexity of the problems from a parameterized point of view.

Furthermore, Scott et al. [SIKS05] describe (independently from this work) the possibility of searching for meaningful subtrees in a protein interaction network. This can be considered as applying G-STG (or a slightly modified version) to the network. Although, note that the experimental part of [SIKS05] is restricted to paths and we describe additional application scenarios for G-STG and GV-STG.

Whereas a possible application for G-STG is the detection of important signaling pathways in protein interaction networks, GV-STG can be used to identify relevant subnetworks in an interaction network with vertex weights based on differential expression, as described in Section 2.2. In a similar way to MWCS it could allow for finding high-scoring subnetworks of a reasonable

size chosen by the user. This seems to be an advantage as MWCS solved by the simulated annealing strategy of Ideker et al. [IOSS02] sometimes seems to return subnetworks of a size still too large to handle. Therefore, these subnetworks have to be further processed, which is done by applying the simulated annealing to the remaining subnetworks recursively.

Choosing edge weights that depend on the correlation coefficient of the differential-expression of the genes corresponding to the edges' endpoints, G-STG can be applied to an interaction network analogously.

Note that for some applications it also could make sense to look at protein interaction and transcriptional regulatory networks independently.

Altogether, G-STG and GV-STG can be a good framework for finding subsets in networks that arise from varying biological scenarios and therefore provide a new way of identifying important subsets in weighted networks for biological analysis.

4.3 Parameterized complexity

In the following sections we consider problems from a parameterized point of view: We either want to give a fixed-parameter algorithm w.r.t. a specific parameter for a problem or we would like to prove that no such algorithm exists.

Not every parameterized problem is necessarily fixed-parameter tractable. To show the intractability of parameterized problems, Downey and Fellows developed a completeness program for classifying parameterized problems analogously to classical complexity theory. However, the completeness theory of parameterized intractability involves significantly more technical effort. We very briefly sketch the basic concepts of this theory in the following (for details we refer to [DF99, Nie06]).

First, we need a concept of reducibility to compare the hardness of two parameterized problems:

Definition 4.1. *Let $L, L' \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized languages. We say that L* reduces to *L'* by a standard parameterized m-reduction *if there are functions $k \mapsto k'$ and $k \mapsto k''$ from $\mathbb{N}$ to $\mathbb{N}$ and a function $(x,k) \mapsto x'$ from $\Sigma^* \times \mathbb{N}$ such that*

1. *$(x,k) \mapsto x'$ is computable in time $k''|x|^c$ for some constant c and*
2. *$(x,k) \in L$ iff $(x',k') \in L'$.*

Now, the "lowest class of parameterized intractability" can be defined as the class of parameterized languages that are equivalent to the so-called

Short Turing Machine Acceptance problem (also known as k-Step Halting problem).

> Short Turing Machine Acceptance
> **Input:** A nondeterministic Turing machine M, an input word x, and a nonnegative integer k.
> **Question:** Does M accept x in a computation of at most k steps?

This can be considered as the parameterized analogue to the Turing machine acceptance problem—the basic generic NP-complete problem in classical complexity theory. Now, we can define the lowest class of parameterized intractability as follows.

Definition 4.2. *The class of all parameterized languages that reduce by a standard parameterized m-reduction to* Short Turing Machine Acceptance *is called W[1]. A problem to which* Short Turing Machine Acceptance *reduces by a standard parameterized m-reduction is called* W[1]-hard*; if, additionally, it is contained in W[1] then it is called* W[1]-complete.

Even if it is currently not possible to prove the inequality of FPT and W[1], this hypothesis is strongly supported by some results from classical complexity theory. For example the coincidence of FPT and W[1] would imply a $2^{o(n)}$ time algorithm for the 3-Satisfiability problem (the question whether there is a satisfying assignment for a formula in conjunctive normal form with n variables and clauses consisting of at most three literals) [DF99], which is generally believed not to hold true.

As a matter of fact, W[1] is only the lowest level of a whole hierarchy of parameterized intractability. In general, the classes W[t], $t \in \mathbb{N}$, are defined based on "logical depth" (i.e., the number of alternations between unbounded fan-in And- and Or-gates) in boolean circuits [DF99].

An important example for a W[2]-complete problem is the Dominating Set problem, where the size of the dominating set is the parameter.

> **Input:** A graph $G = (V, E)$, a positive integer k.
> **Task:** Find a subset $V' \subseteq V$ of size k such that every vertex of G either belongs to V' or has a neighbor in V'.

In the following sections we use the Dominating Set problem for the proof of the W[2]-hardness for some parameterizations of the considered problems.

4.4 Weighted Tree Coloring

Whereas we need the concept of parameterized reduction to prove the hardness of a parameterized problem, the best way to show fixed-parameter tractability is to design a concrete algorithm. There are some general methods to design fixed-parameter algorithms. The most common ones are search trees or dynamic programming in combination with data reduction. In this section we describe a fixed-parameter algorithm which is based on a dynamic programming method called *color-coding* first introduced by Alon et al. [AYZ95]. We use this algorithm as a subroutine for designing fixed-parameter algorithms in the remainder of this section.

The method of color-coding can be used to find subgraphs with specific properties (like paths, cycles, and trees or graphs with bounded treewidth) such that it is fixed-parameter with respect to the size of the subgraph k. For this, one has to avoid to look separately at all subsets of vertices of size k. Obviously, for a graph with n vertices this would lead to a running time which is $\Omega(n^k)$. Therefore, Alon et al. [AYZ95] use a dynamic programming approach in which the coloring of the vertices with k different colors provides the possibility to store tables of subsets of colors instead of all possible subsets of vertices.

More precisely, there is a randomized version of color-coding in which the first step consists of coloring the vertices of the graph randomly with k different colors. Then, one can observe that it is easier to find *colorful* subgraphs, i.e. subgraphs which contain only vertices of pairwise disjoint colors. The basic idea is that, using a dynamic programming approach, a subgraph with a particular coloring can only be enlarged by vertices with colors that do not already occur in the subgraph. Therefore, the loss of complexity is due to the fact that one can store subsets of colors instead of all subgraphs with identical coloring.

As we discuss in Section 4.4.2 the color-coding method can be derandomized. One can find a number of colorings such that we color every subgraph of size k of G with different colors at least once and such that the overall running time is still fixed-parameter tractable with respect to k.

So, with color-coding as described by Alon et al. [AYZ95] we can find a subtree of a given structure in a graph. Since for G-STG and GV-STG we do not look for any subtree of a specific structure, we have to combine the color-coding method with the enumeration of all possible tree structures of a given number of nodes. More precisely, we look for an optimal solution for every possible tree structure and pick out the optimum of all possible structures.

Furthermore, we need to incorporate the weight of the subtree into the dynamic programming routine[4]. In the following subsection we give a formulation of the color-coding algorithm that extends the algorithm as described by Alon et al. [AYZ95] in two ways:

- It additionally includes the weight of the subtrees.
- It describes the algorithm in more detail than is done in [AYZ95] or [SIKS05].

More precisely, the algorithm solves the Weighted Tree Color Coding problem, which is defined in the next subsection.

4.4.1 Algorithm for the Weighted Colorful Tree problem

We start with some definitions necessary for the formulation of the algorithm.

Definitions

We call a colored graph G *colorful* if all its vertices are colored with pairwise distinct colors.

We formulate the problem as follows.

> Weighted Colorful Tree:
> **Input:** An undirected graph $G = (V_G, E_G)$ with weight function $w : E_G \to \mathbb{R}_0^+$, a k-coloring $c : V_G \to \{1, \ldots, k\}$, and a rooted tree $T = (V_T, E_T)$ with k nodes.
> **Task:** Find the minimum weight W_T of all colorful subtrees of G which are isomorphic to T (or report that no such tree exists).

Note that we require a *rooted* tree T only for convenience when describing our algorithm since the root of T determines the processing order of dynamic programming. The root can be arbitrarily chosen in the case of an unrooted tree T.

For the formalization of the algorithm we need some further definitions regarding the tree T and the graph G:

- W.l.o.g. we assume that the nodes $V_T = \{n_1, ..., n_k\}$ are ordered in post-order. If they are not, this order can be obtained in linear time.
- Let $T(n_i)$ denote the subtree of T rooted at n_i.

[4]This was also done independently from this work by Scott et al. [SIKS05]

- Let $n \in V_T$ have $d(n)$ children $n^{(1)}, \ldots, n^{(d(n))}$. Then, for $0 \leq j \leq d(n)$ let $T(n, j)$ be the bottom-up subtree of T containing n and $T(n^{(1)}), ..., T(n^{(j)})$.
- Set $w(T) := \sum_{e \in E_T} w(e)$.
- Set $c(T) := \bigcup_{v \in V_T} c(v)$.
- In the process of the algorithm, we regard the graph G as bidirected, that means we replace each undirected edge $\{u, v\}$ by two directed edges (u, v) and (v, u).

We define the dynamic programming table to contain entries

$$W[u, n, j, C]$$

with $u \in V_G,\ n \in V_T,\ 0 \leq j \leq d(n)$, and $C \subseteq \{1, ..., k\}$ a color set which contains $|V(T(n, j))|$ pairwise disjoint colors. The value of $W[u, n, j, C]$ then denotes the minimum weight of a subtree colored with all colors of C, isomorphic to $T(n, j)$, and rooted at u.

We give some further definitions which are useful for the proof of the correctness of the algorithm.

- We call an entry $W[u, n, j, C]$ *valid* if $W[u, n, j, C] < \infty$.
- For an entry $W[u, n, j, C]$ we call a coloring C *possible* if G contains a colorful subtree T_S rooted at u and isomorphic to $T(n, j)$ with $c(T_S) = C$. We then say the subtree T_S *corresponds* to the coloring C.

Algorithm

Figure 4.1 gives a description of algorithm WTCC that solves Weighted Colorful Tree. Basically, the algorithm works as follows: Every node n of the tree is matched with every vertex v of the graph. Then it stores all possible subsets corresponding to colorful subtrees which are rooted at v and isomorphic to a subtree rooted at n. The index j denotes the number of children of n that have been processed at this iteration step. During the dynamic programming step color sets (corresponding to node-disjoint subtrees) are merged to larger color sets corresponding to subtrees in G. In the end there is an entry for every vertex $v \in G$ that returns the minimum weight of all colorful subtrees isomorphic to T and rooted at v. A solution for Weighted Colorful Tree then is obtained by iterating over these entries.

```
procedure WeightedTreeColorCoding (graph G, tree T, coloring c)

 /* input:   a graph G = (V_G, E_G), a weight function w : E_G → ℝ_0^+,
             a k-coloring c : V_G → {1,...,k},
             and a rooted tree T = (V_T, E_T) with k nodes
             and root r                                          */
 /* output:  the minimum weight W_T of a tree among all
             colorful subtrees of G which are isomorphic to T;
             in case there is no such tree, W_T = ∞              */

01  /* initialization: */
02  forall u ∈ V_G do
03    forall n ∈ V_T do
04        for j = 0..d(n) do
05          forall C ⊆ {1,...,k} do
06            if C = {c(u)} and j = 0 do
07              W[u, n, 0, {c(u)}] := 0
08            else
09              W[u, n, j, C] := ∞

10    /* dynamic programming: */
11    for i = 1..k do
12      for j = 1..d(n_i) do
13        forall (u, v) ∈ E_G do
14          forall C, C' ⊆ {1, ..., k} with C ∩ C' = ∅
15            and W[u, n_i, j − 1, C] < ∞
16            and W[v, n_i^(c_j), d(n_i^(c_j)), C'] < ∞ do
17              W[u, n_i, j, C ∪ C'] = min{W[u, n_i, j, C ∪ C'],
18                w((u, v)) + W[u, n_i, j − 1, C] + W[v, n_i^(c_j), d(n_i^(c_j)), C']}

19    /* result: */
20    return W_T = min_{v∈V_G}{W[v, r, d(r), {1, ..., k}]}
```

Figure 4.1: **Algorithm WTCC for Weighted Colorful Tree**

Lemma 4.3. *Algorithm WTCC computes an optimal solution for* Weighted Colorful Tree.

Proof. We show that after running the dynamic programming every entry $W[u, n, j, C]$ returns the weight of a colorful subtree of minimum weight rooted at u, colored by the colors of C, and isomorphic to $T(n, j)$. Then, the correctness follows directly from the choice of W_T in the last step of the algorithm. For every vertex $u \in V_G$ the entry $W[u, r, d_r, \{1, ..., k\}]$ returns the minimum weight of all colorful subgraphs of G which are isomorphic to T while mapping u to the root of T. Choosing the minimum of all these weights for all vertices $u \in V_G$ clearly yields an optimum solution.

We prove the correctness by induction on the structure of the tree T. As we traverse T in post-order, we assume that the induction hypothesis is true for all entries containing $W[*, n_i, *, *]$ where i is smaller than the index of the currently processed node of T. As the post-order numbers of the children of a node are always smaller than its own number, that means in particular that the induction hypothesis is true for the entries corresponding to children of n_i.

More precisely, looking at an arbitrary entry $W[u, n_i, j, S]$ for a node with index i, we assume the following for all $u \in V_G$, for $0 \leq j \leq d(n_{i+1})$ and all possible colorings $S \subseteq \{1, ..., k\}$: If there is a colorful subtree T_G of G with $c(T_G) = S$ that is isomorphic to $T(n_i, j)$ and rooted at u, then the entry $W[u, n_i, j, S]$ contains the weight $w(T_G)$. If there is more than one such subtree with coloring S, the entry $W[u, n_i, j, S]$ contains the weight of the tree of minimum weight among these .

Following a post-order traversal of T, we start with $i = 1$ at a leaf of T. The only tree isomorphic to a leaf of T which is rooted at an arbitrary vertex u is u itself and the only possible coloring is $C = \{c(u)\}$. Obviously, the cost of it is $w(u) = 0$. After the initialization step, for every $u \in V_G$ there is an entry $W[u, n_1, 0, \{c(u)\}]$ which is set to 0 and is not modified after this.

For the induction step we have to show that the hypothesis is true for all entries $W[u, n_{i+1}, j, S]$. That means that we compute $W[u, n_{i+1}, j, S]$ correctly for all $u \in V_G$, for $0 \leq j \leq d(n_{i+1})$, and all possible colorings $S \subseteq \{1, ..., k\}$). We have to distinguish the following three cases:

Case 1: *The node n_{i+1} is a leaf of T, that means $j = 0$.*

The correctness follows analogously to the proof of the base case for n_1.

Case 2: *The node n_{i+1} is expanded by the subtree rooted at its first child, that means $j = 1$.* The algorithm computes entry $W[u, n_{i+1}, 1, S]$ based on the valid entries $W[u, n_{i+1}, 0, C]$ and $W[v, n_{i+1}^{(1)}, d(n_{i+1}^{(1)}), C']$ with $C \cup C' = S$. Entries of the form $W[u, n_{i+1}, 0, C]$ are only changed in the initialization step. For every such entry that is valid we have $C = \{c(u)\}$. During the iteration over all edges $(u, v) \in E_G$ the algorithm examines all valid entries $W[v, n_{i+1}^{(1)}, d(n_{i+1}^{(1)}), C']$. By induction hypothesis for every such entry there must be a colorful subtree isomorphic to $T(n_1, d(n_1))$ rooted at v with $c(T(n_1, d(n_1))) = C'$. To compute the new entry $W[u, n_{i+1}, 1, C \cup C']$ the algorithm only considers entries with coloring C' such that $c(u) \notin C'$. That ensures that no subtree corresponding to the coloring C' contains u. Therefore u can be used to expand such a subtree to a tree isomorphic to $T(n_{i+1}, 1)$ with $c(T) = C \cup C'$. By induction hypothesis we can assume that, if there exists a subtree of the demanded properties, there must be a valid entry for $W[v, n_1, d(n_1), C']$, which returns a minimum weight. Therefore, and because the coloring of a tree rooted at u must contain $c(u)$, the algorithm computes an entry for every colorful subtree isomorphic to $T(n_{i+1}, 1)$. In the case that there is more than one tree corresponding to a coloring, the algorithm chooses the minimum weight of all these trees as it compares their weights by iterating over all edges.

Now, we show case 3 by induction on the index j iterating over the children of n_{i+1}. The correctness of the base case $j = 1$ is given in case 2.

Case 3: *The subtree rooted at n_{i+1} is expanded by the subtree of a non-first child, that means $2 \leq j \leq d(n_{i+1})$.*

In this case, the algorithm considers for all edges $(u, v) \in E_G$ all valid entries of $W[u, n_{i+1}, j-1, C]$ $(*_1)$ and $W[v, n_{i+1}^{(j)}, d(n_{i+1}^{(j)}), C']$ $(*_2)$ with $C' \cup C = S$.

As the algorithm increases j in every step and since we proved the correctness of the induction hypothesis for $j = 1$ in case 2, we can assume that $(*_1)$ stores the minimum weight for all colorings C of all subtrees isomorphic to $T(n_{i+1}, j-1)$ and rooted at u. Furthermore by induction hypothesis, the entry $(*_2)$ stores the minimum weight for all colorings C' of all subtrees isomorphic to $T(n_{i+1}^{(j)})$ and rooted at v.

As the algorithm looks at all colorings C and C' that are disjoint, all corresponding subtrees are also node disjoint. Therefore, they can be

connected by the edge (u, v) to a subtree isomorphic to $T(n_{i+1}, j)$ with $c(T) = C \cup C'$.

By iterating over all edges and combining all possible colorings, the algorithm ensures that there is a valid entry corresponding to every colorful subtree of G. Furthermore, this ensures that in the case that there is more than one subtree of G corresponding to a coloring the minimum weight is chosen.

□

Theorem 4.1. Weighted Colorful Tree *can be solved in time* $O(k \cdot |E_G| \cdot 3^k)$.

Proof. We use algorithm WTCC solving Weighted Tree Color Coding by Lemma 4.3.

Clearly, the initialization step of WTCC can be carried out in time $O(|V_G| \cdot k \cdot 2^k)$ as the iteration over j is upper-bounded by k as there are at most k children in a tree of size k and there are 2^k different subsets of k colors.

The time consuming part of the algorithm is the dynamic programming step. There, the algorithm iterates $O(k)$ times over $|E_G|$ edges and then has to combine the subsets of the different colorings. An upper bound for all possible combinations of all subsets C and C' during an iteration step is given by 3^k. This is due to the fact that there are 3^k possibilities to divide the set $S = \{1, ..., k\}$ into three disjoint subsets C, C' and $S \backslash (C \cup C')$. □

Lemma 4.4. *The memory requirement of Algorithm WTCC is* $O(|V_G| \cdot k \cdot 2^k)$.

Proof. An example for a dynamic programming table is given in Figure 4.2. We store an entry for every vertex $u \in V_G$ and every node $n \in V_T$ and for every child of n. As the total number of children is bounded by k, we have to store all colorings for $O(|V_G| \cdot k)$ possible assignments of vertices and nodes. The number of different colorings with $|T(n)|$ colors chosen from the possible k colors is at most the number of subsets of a set of size k which is 2^k. Therefore, the total memory requirement is $O(|V_G| \cdot k \cdot 2^k)$. □

4.4.2 Finding non-colored subtrees

Analogously to Alon et al [AYZ95], we can now describe a randomized algorithm to find a non-colored weighted subtrees. Coloring vertices of a graph uniformly at random with k colors, a subtree is colorful with probability

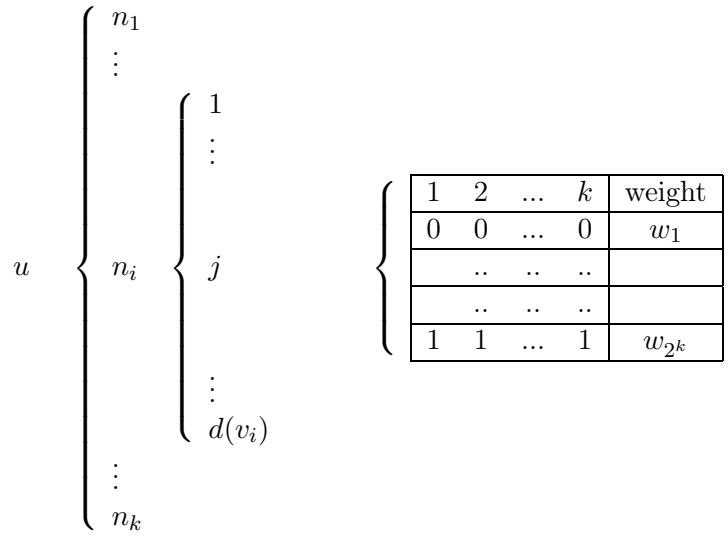

Figure 4.2: **Dynamic programming table for algorithm WTCC**

$(k!)/k^k$, which is lower-bounded by e^{-k}. Repeating WTCC for $e^k = 2^{O(k)}$ randomly chosen colorings yields an algorithm that can find a minimum weight subtree in expected running time $2^{O(k)} \cdot |E|$.

To get a deterministic algorithm, as needed to show the fixed-parameter tractability of problems, we can derandomize the algorithm analogously to Alon et al. [AYZ95]. What we need is a family of colorings such that every subgraph of G is colorful by at least one coloring of the family. Then, we can apply the `WeightedTreeColorCoding` method to every coloring of the family. Furthermore, the size of the family must be exponential only in k and there must exist a fixed-parameter algorithm w.r.t. k for its computation. Alon et al. [AYZ95] describe a possibility to construct such a family—which can be considered as a k-perfect family of hash functions from $\{1, 2, \cdots, |V_G|\}$ to $\{1, 2, \cdots, k\}$—of size $2^{O(k)} \cdot \log V_G$. Since with the described construction every coloring of the family can be evaluated in $O(1)$ time, Alon et al. conclude an extra factor of $2^{O(k)} \cdot \log V_G$ for the complexity of the original randomized algorithm. Therefore, the overall running time for finding a subtree of minimum weight in a graph is $2^{O(k)} \cdot \log |V_G| \cdot O(|E_G| \cdot k)$. We denote the derandomized Weighted Tree Color-Coding method as `dWTCC(graph G, tree T)` with the arguments graph G and tree T analogously to `WTCC(graph G, tree T, coloring C)`.

4.4.3 Extensions: Vertex weights and constructive solutions

We would also like to apply the algorithm to graphs with vertex weights, that means we define the problem analogously for a graph with weight function $w : V \to \mathbb{R}_0^+$ by defining $w(T) = \sum_{v \in V_T} w(v)$. We then have to modify the initialization step by setting $W[u, n, j, C] := w(u)$ for $j = 0$ and the computation of the new entry during the dynamic programming such that we do not add an edge weight. More concretely, in *lines 17+18*, we set

$$W[u, n_i, j, C \cup C'] = \min \left\{ \begin{array}{r} W[u, n_i, j, C \cup C'], \\ W[u, n_i, j-1, C] + W[v, n_{i_{c_j}}, d_{n_{i_{c_j}}}, C'] \end{array} \right\}.$$

We denote the modified method as `VertexWeightedTreeColorCoding` (VWTCC) and its derandomized version as `dVWTCC(graph G, tree T)` with the same arguments as `dWTCC(graph G, tree T)`.

In the next section, we use the `dWTCC` and `dVWTCC` method to describe algorithms for G-STG and GV-STG. So, we need to find a tree with minimum weight instead of the minimum weight itself. This can be easily achieved by a small modification of the algorithm. The only thing to do is to additionally store a subtree corresponding to every coloring in the table. Obviously, this does not change the running time of the algorithm.

4.5 Parameterized hardness and tractability

This section provides an overview of the parameterized complexity of STG, V-STG, G-STG and GV-STG w.r.t. a range of possible parameterizations. Furthermore, we have a brief look at the complexity of MWCS. We want to point out that G-STG and GV-STG are NP-complete. The NP-hardness can be proved by reduction from Dominating Set. We omit the proof at this point as the parameterized reductions we use to prove Theorem 4.8 and Theorem 4.15 yield this result as well.

In the next subsections, we only consider the decision variants of the problems. That is sufficient for their classification into parameterized complexity classes. Note that all of the described algorithms for the decision problems can be modified in a straight-forward way into algorithms obtaining a constructive solution within the same running times.

We start with regarding the number of nodes of the requested subgraph as parameter (Section 4.5.1). In Section 4.5.2, we use the total weight of

the requested subgraph as parameter and discuss the complexity for different weight functions. Thereby, we consider four different weight functions allowing uniform, binary, integer, and rational weights. Interestingly, the results for the different problems differ with the choice of the weight function. Then, in Section 4.5.3 we consider the MWCS problem.

4.5.1 Number of nodes of the subgraph as a parameter

First, we present some well known results for STG and V-STG, then we show the tractability for G-STG and GV-STG w.r.t. the number of nodes of the subgraph.

Known results for STG and V-STG

For STG and V-STG we can distinguish between the number of terminals and the number of Steiner nodes as parameter. This distinction leads to the following results.

Theorem 4.2. *The STG problem as well as its vertex-weighted version V-STG are fixed-parameter tractable w.r.t. the number of terminals.*

Theorem 4.2 follows directly by the Dreyfuss-Wagner algorithm, which solves the problems in a running time that is only exponential in the number of terminals [DW72] (see Section 3.2.1).

Theorem 4.3. *STG and V-STG are $W[2]-$hard with respect to the number of Steiner nodes as parameter.*

The W[2]-hardness of STG is claimed by Downey and Fellows [DF99] by reduction from Dominating Set, but they give an incorrect reference for the actual proof. In the compendium by Marco Cesati [Ces] it is claimed to be W[2]-hard by personal communication. In any case, Theorem 4.3 also follows analogously to the parameterized reductions from Dominating Set which we give in the proof of Theorem 4.8.

Furthermore, it is obvious that there is no correlation between the total number of nodes in a Steiner tree and the number of Steiner nodes. An easy example is a path of terminals, where an arbitrary large number of terminals can be connected without any Steiner Node.

New results for G-STG and GV-STG

In contrast to STG and V-STG we do not longer distinguish between the number of terminals and Steiner nodes, but consider the total number of

```
/* input:  Graph G = (E, V) with weight function w : E → ℝ₀⁺,
           positive integer s, positive real l                    */
/* output: true, if there is a subgraph G' = (V', E')
           with |V'| = s and w(E') ≤ l                            */

    forall trees T of size k do
      if dWTCC(G, T) ≤ l then
        return true
    end forall
    return false
```

Figure 4.3: **Fixed-parameter algorithm w.r.t. the number of nodes of the subtree for G-STG**

parameter	STG	V-STG	G-STG	GV-STG
# Terminals	FPT	FPT	-	-
# Steiner vertices	W[2]-hard	W[2]-hard	-	-
# Nodes of the subgraph	-	-	FPT	FPT

Table 4.1: **Tractability with respect to the number of nodes**

nodes in the tree or subgraph. This is due to the fact that no terminal set is given in the definition of G-STG and GV-STG.

Theorem 4.4. *G-STG and GV-STG are fixed-parameter tractable w.r.t. the total number $|V'|$ of nodes of the subgraph.*

Proof. In Figure 4.3 we describe a fixed-parameter algorithm for the G-STG problem that uses algorithm `dWTCC` as subroutine. As the number of pairwise non-isomorphic trees over k nodes is $O(2.96^k)$ [Ott48] and the `WTCC` algorithm has a running time only exponential in the size of the subgraph k as well, this yields a fixed-parameter algorithm with respect to k. More precisely, the running time is $2^{O(k)} \cdot \log V_G \cdot O(k \cdot |E_G|)$.

For GV-STG we can use the same algorithm as for G-STG with the `dVWTCC` method as subroutine. For every connected subgraph with k vertices there is at least on spanning tree with $k-1$ edges. Therefore, it is sufficient to iterate over all non-isomorphic trees of size k. □

Table 4.1 provides an overview of the parameterized complexity of STG, V-STG, G-STG and GV-STG with respect to the number of nodes as parameter. With this choice of the parameter the edge- and vertex-weighted problem versions have the same complexity.

4.5.2 Weight as a parameter

Now, we consider the complexity of the four problems STG, V-STG, G-STG, and GV-STG from another point of view by looking at their parameterized tractability w.r.t. the weight of the solution of the subgraph. While in Section 4.1 the problems have been defined for non-negative real edge weights, here we also consider several alternative weight functions. We distinguish between uniform, binary, integer, and rational weights.

Uniform weight function

The most restricted case of all possible weight functions is to assign the value one to every edge and every vertex, respectively. Using the total weight of the tree (subgraph) as parameter we can show the following results for this weight function.

Theorem 4.5. *STG as well as V-STG are in FPT w.r.t. the weight of the subgraph G' for a uniform weight function.*

As the number of total nodes is at least as high as the number of terminals and every node is assigned a uniform weight, Theorem 4.5 follows directly from the Dreyfus-Wagner algorithm.

Theorem 4.6. *Restricted to a uniform weight function G-STG and GV-STG are solvable in linear time.*

Proof. Both problems only have a solution if s (the minimum size of the subgraph) is smaller than l (the maximum weight of the subgraph). Therefore, we only have to look for any connected subgraph of size s. This can be done with depth-first search in linear time. □

Binary weight function

Now, we choose the binary weight functions $w : E \to \{0, 1\}$ and $w : V \to \{0, 1\}$, respectively.

Theorem 4.7. *STG is in FPT w.r.t. the weight of the Steiner tree for a binary weight function.*

```
procedure prepro_STG_bin_weight (graph G, set of terminals T)

/* input:  Graph G = (V, E), weight function w : E → {0, 1},
           set of terminals T ⊆ V                              */
/* output: reduced graph G' with weight function w : E → {1}
           reduced set of terminals T' ⊆ T                     */

    forall {u, v} ∈ E with w({u, v}) = 0 do
        insert a new vertex w with N[w] = N[v] ∪ N[u]
        if u ∈ T or v ∈ T then
            put w into the set of terminals T
        remove u and v and edges incident to u or v
    end forall
    return (G, T)
```

Figure 4.4: **Preprocessing method** *for the STG problem with binary weight function.*

Proof. We can solve the problem by applying the Dreyfus-Wagner algorithm after a simple preprocessing, contracting all weight 0 edges, with polynomial running time as given in Figure 4.4.

The running time of the Dreyfus-Wagner algorithm applied to the reduced instance is only exponential in the number of terminals z. As one needs at least $z - 1$ edges to connect the z terminals and the minimum weight of an edge is one, the weight of the Steiner Tree is at least as much as the number of terminals (minus one). That implies that the problem is fixed-parameter tractable w.r.t. the weight of the Steiner tree. □

Theorem 4.8. *V-STG and GV-STG are W[2]-hard w.r.t. the weight of the subgraph G' for a binary weight function.*

Proof. We prove the W[2]-hardness of the problems by reduction from Dominating Set.

Every vertex i of the Dominating Set instance is represented by two vertices r_i and d_i in the V-STG or GV-STG instance (Figure 4.5). One of them, r_i, represents that i has to be dominated and the other one, d_i, that i is able to dominate all vertices representing its neighborhood.

Now, we give the formal definition of the reduction.

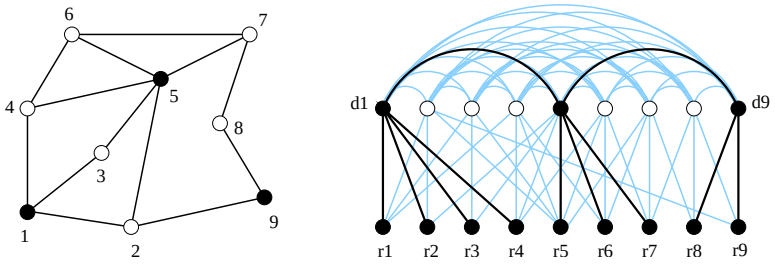

Figure 4.5: **Reduction from Dominating Set to V-STG/GV-STG** *(w.r.t. the weight of the subgraph for binary weights). On the left-hand side is an example for a* Dominating Set *instance, where the black vertices belong to an optimal dominating set. The right-hand side shows the corresponding V-STG/GV-STG instance with the subgraph of the solution colored in black.*

Given a Dominating Set instance $G = (V, E)$ with $V = \{1, ..., n\}$ and a parameter k which denotes the size of an optimal dominating set, we construct an instance $G' = (V', E')$ with weight function $w : V' \to \{0, 1\}$ for V-STG or GV-STG as follows:

- $V' = \{d_i \mid i = 1, \ldots, n\} \cup \{r_i \mid i = 1, \ldots, n\}$
- $E' = \{(d_i, d_j) \mid i, j = 1, \ldots, n, i \neq j\} \cup \{(d_i, r_j) \mid i = 1, \ldots, n, j \in N[i]\}$
- $w(d_i) = 1$, $w(r_i) = 0$ for $i = 1, \ldots, n$

Claim: G has a dominating set of size k iff

(i) for G' with terminal set $R = \{r_i \mid i = 1, ..., n\}$ V-STG has an optimal solution $S \subseteq V'$ with weight $w(S) = k$, or, equivalently

(ii) G' has a solution $S \subseteq V'$ for the GV-STG with $|S| \geq n + k$ and $w(S) = k$.

Proof of claim:
(i) "⇒": Let V_{DS} be an optimal dominating set of G of size k. Then the subset $V_{ST} = \{d_i \mid i \in V_{DS}\}$ connects all vertices r_i of the neighborhood of V_{DS}. As V_{DS} is a dominating set there is at least one edge between every vertex $r \in R$ and one vertex $d \in V_{DS}$. As, additionally, all pairs of vertices of $D := \{d_i \mid i = 1, \ldots, n\}$ are connected, $V_{ST} \cup R$ is a Steiner tree with weight k for the terminal set R.

"⇐": Let V_{ST} be the set of Steiner nodes of a Steiner tree with weight k in G' that connects the terminal set R. Because of the weight function V_{ST} consists of k vertices out of D. Following the construction of G' there is an edge between $d_i \in D$ and $r_j \in R$ only if $i \in N[j]$ in G. As every vertex $r_j \in R$ is connected to the Steiner tree, every vertex j must be in the neighborhood of a vertex i with $d_i \in V_{ST}$. Therefore, a dominating set of size k for G is given by $V_{DS} = \{i \mid d_i \in V_{ST}\}$.

(ii) "⇒": Let V_{DS} be an optimal dominating set of G. After construction the subset $V_{GST} = \{d_i \mid i \in V_{DS}\} \cup \{r_i \mid i = 1, \ldots, n\}$, $G[V_{GST}]$ is a connected subgraph of size $n + k$ and with weight k.

"⇐": Let V_{GST} be a connected subgraph of G' of size $n + k$ and with weight k. As the vertices of R have weight zero and the vertices of D have weight one, every such subgraph has to contain all vertices $r_i \in R$ and k vertices out of D. There are no edges between vertices $r \in R$, therefore, every vertex $r \in R$ must have a neighbor $d \in D$, otherwise the subgraph would not be connected. It follows directly that the subset $\{i \in V \mid d_i \in V_{GST}\}$ is a dominating set of size k. □

Theorem 4.9. *G-STG is in FPT w.r.t. the weight of the subgraph G' for a binary weight function.*

In the next subsection we give in the proof of Theorem 4.12 a fixed-parameter algorithm w.r.t. the weight of the subgraph for an integer weight function, which can be applied to the special case of binary weights as well. Therefore, the proof of Theorem 4.9 derives from the one for Theorem 4.12.

Integer weight function

Now, we consider the weight functions $w : E \to \mathbb{N}_0$ and $w : V \to \mathbb{N}_0$, respectively.

Theorem 4.10. *STG is in FPT w.r.t. the weight of the Steiner tree for an integer weight function.*

Proof. We use the same preprocessing method as in the binary weighted case (Figure 4.4), contracting the weight 0 edges. Then, we apply the Dreyfus-Wagner algorithm. After the preprocessing the minimum weight of an edge is 1, therefore the size of the Steiner tree is an upper bound for its weight. As the running time of the Dreyfus-Wagner algorithm is exponential in the size of the Steiner tree, this clearly yields an algorithm that is FPT with respect to the weight of the Steiner tree. □

```
procedure prepro_GSTG_int_weight (graph G, set of terminals T)

/* input:  Graph G = (V, E), weight function w : E → ℕ,
           set of terminals T ⊆ V                          */
/* output: reduced graph G' with weight function w : E → {1}
           reduced set of terminals T' ⊆ T, counter p : V → ℕ */

    initialize p(v) := 0 forall v ∈ V_G
    forall {u, v} ∈ E with w({u, v}) = 0 do
        insert new vertex w with N[w] = N[v] ∪ N[u]
         set p(w) = p(u) + p(v) + 1
        if u ∈ T or v ∈ T then
            add w into the set of terminals T
        remove u and v and edges incident to u or v
    end forall
    return (G, T, p)
```

Figure 4.6: **Preprocessing method** *for the G-STG problem with integer weight function.*

Theorem 4.11. *V-STG and GV-STG are W[2]-hard w.r.t. the weight of the subgraph G' for an integer weight function.*

Theorem 4.11 is an immediate consequence of the W[2]-hardness of the binary weighted case shown in Theorem 4.8.

Theorem 4.12. *G-STG is in FPT w.r.t. the weight of the subtree G' for an integer weight function.*

Proof. We prove Theorem 4.12 by describing a fixed-parameter algorithm w.r.t. the weight of the subtree for G-STG. It consists of a preprocessing step, given in Figure 4.6, that removes edges with weight equal to zero and a slightly modified version of the `WTCC` algorithm. Deleting weight-zero edges ensures that the size of the subgraph for which we seek in the dynamic programming is bounded by the weight of the subgraph. In the preprocessing step we additionally have to count for every vertex in the reduced graph how many vertices of the unreduced instance it replaces. Furthermore, we have to incorporate a counter into the dynamic programming process itself. More precisely, we use a slightly modified version of the algorithm given for the

proof Theorem 4.4. We only change the `WTCC` subroutine (Figure 4.1) as follows:

- An entry of the dynamic programming table is extended by the counter p which indicates how many nodes in the unreduced instance are part of the subgraph. We now consider entries of type $W[u, n, j, C, p]$.
- We extend the arguments by the maximum weight of the subgraph s, the minimum size of the subgraph s, and counter p as computed by the preprocessing.
- We modify the initialization step in line *07* as follows:

$$W[u, n, 0, \{c(u)\}, p(u)] := 0$$

- To take into account that the entries additionally contain counters p and p', we replace lines *17+18* by:

$$\begin{aligned} W[u, n_i, j, C \cup C', p + p'] = \min\{&W[u, n_i, j, C \cup C', p + p'], \\ &w((u, v)) + W[u, n_i, j - 1, C, p] + W[v, n_i^{(c_j)}, d(n_i^{(c_j)}), C', p']\} \end{aligned}$$

- Instead of computing the minimum weight after the dynamic programming, the algorithm returns true as soon as it computed an entry for a subtree that fulfills the required conditions. This is achieved by inserting the following code immediately after line *18*.

```
if (p + p' ≥ s and W[u, n_i, j, C ∪ C', p + p'] ≤ l)
  return true
```

Correctness of the algorithm: If the algorithm returns true, there must be connected subtree S with $|C \cup C'|$ nodes whose weight is less than l as an immediate consequence of the correctness of the non-modified algorithm. We reverse the preprocessing by replacing the nodes of $n \in S$ with $p(n) > 1$ by the nodes they replace in the original graph. In this way we obtain a connected subgraph of G with $p + p' \geq s$ nodes and weight $\leq l$.

Furthermore, the modification of the algorithm does not change its running time.

□

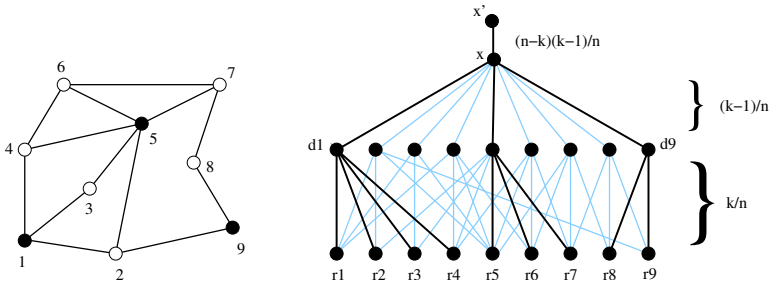

Figure 4.7: **Parameterized Reduction from Dominating Set to STG** *(w.r.t. the weight of the Steiner Tree for rational weights). On the left hand is an example for a* Dominating Set *instance such that the black vertices belong to an optimal dominating set. The right-hand side shows the corresponding STG instance with terminal set* $\{r1, ..., r9\}$*. The edges of an optimal Steiner tree are colored black.*

Rational weight function

All problems become W[2]-hard if we allow a rational weight function and parameterize with respect to the weight of the solution tree.

Theorem 4.13. *V-STG and GV-STG are W[2]-hard w.r.t. the weight of the subgraph* G' *for a rational weight function.*

The correctness of Theorem 4.13 follows directly from the W[2]-hardness for the binary weights shown in Theorem 4.8.

Theorem 4.14. *STG is W[2]-hard w.r.t. the weight of the subgraph* G' *for a rational weight function.*

We prove Theorem 4.14 by reduction from Dominating Set. Again, every vertex i of the Dominating Set instance is represented by two vertices r_i and d_i in the STG instance (Figure 4.7). The vertex r_i represents the ability of vertex i to dominate all its neighbors and d_i stands for the fact that i has to be dominated itself. The basic idea of the reduction is that in the case of a dominating set of size k in the original instance one has to choose k edges connecting x with k vertices out of $\{d_i \mid i = 1, \ldots, n\}$ to get a Steiner tree in the Steiner Tree instance. The more complex weight function is necessary to avoid that two subgraphs of the Steiner tree containing subsets of $\{r_i \mid i = 1, \ldots, n\}$ as terminal sets can be connected without using an edge with an endpoint in x.

Proof. (Theorem 4.14) Given a DOMINATING SET instance $G = (V, E)$ with $V = \{1, ..., n\}$ and parameter k which is the size of a dominating set, we construct an STG instance $G' = (V', E')$ with weight function $w : E' \to \mathbb{R}$ as follows:

- $V' = \{x, x'\} \cup \{d_i \mid i = 1, \ldots, n\} \cup \{r_i \mid i = 1, \ldots, n\}$
- $E' = \{\{x, d_i\} \mid i = 1, \ldots, n\} \cup \{\{d_i, r_j\} \mid i = j \vee \{i, j\} \in E\} \cup \{x, x'\}$
- $w(\{x, d_i\}) = \frac{k-1}{n}$ for all $i = 1, \ldots, n$
- $w(\{d_i, r_j\}) = \frac{k}{n}$ for all $i = 1, \ldots, n$
- $w(\{x, x'\}) = \frac{(n-k)\cdot(k-1)}{n}$

Claim:
G has a dominating set of size k if and only if a Steiner tree in G' with terminal set $\{r_i \mid i = 1, \ldots n\} \cup \{x'\}$ has weight $2 \cdot k - 1$.

Proof of the claim:
"⇒": Let V_{DS} be an optimal dominating set of G of size k. Define $D := \{d_i \mid i = 1, \ldots, n\}$, $R := \{r_i \mid i = 1, \ldots, n\}$. As V_{DS} is a valid dominating set, by construction there must exist a subset $C \subseteq E'$ of size n that connects every vertex $r \in R$ with a vertex $d \in D$. More precisely, there exists a subset $C = \{\{r_i, d_j\} \in E' \mid i = 1, \ldots, n, j \in N[i] \cap V_{DS}\}$. Now, we can construct a Steiner tree by the following set of edges E_{ST}: $E_{ST} = \{\{d_i, x\} \mid i \in V_{DS}\} \cup \{x, x'\} \cup C$. Obviously, the given Steiner tree is connected and its weight is $w(E') = k \cdot (k-1)/n + ((n-k) \cdot (k-1))/n + n \cdot k/n = 2 \cdot k - 1$.
"⇐:" For this direction we have to show that a Steiner tree ST with terminal set $\{r_i\} \cup \{x'\}$ and cost $2 \cdot k - 1$ contains at most k vertices of D, let them be denoted as D_{ST}. Then obviously, there must be an edge between every vertex of R and a vertex of D_{ST} and we obtain a dominating set for G by $V_{DS} = \{i \mid d_i \in D_{ST}\}$.

As ST is a valid Steiner tree, it contains at least one edge that connects every vertex $r \in R$ to a vertex $d \in D$ and it contains the edge $\{x, x'\}$. So, there must be a subforest S of ST consisting, for every $i = 1, \ldots, n$, of one edge $\{r_i, d_j\}$ for some $j \in \{1, \ldots, n\}$, and of $\{x, x'\}$. The weight of this subforest is $w(S) = n \cdot k/n + ((n-k) \cdot (k-1))/n$. The remaining weight that can be used to connect the subforest is $w_{con} = c(ST) - c(S) = (k \cdot (k-1))/n$. As the remaining edges of $E' \backslash E_S$ have either weight k/n or $(k-1)/n$, there can be at most k edges that connect the subforest S to form ST. Therefore, the subforest S can consist of at most $k + 1$ subtrees including the subtree

parameter	STG	V-STG	G-STG	GV-STG
uniform weights	FPT	FPT	P (linear)	P (linear)
binary weights	FPT	W[2]-hard	FPT	W[2]-hard
integer weights	FPT	W[2]-hard	FPT	W[2]-hard
rational weights	W[2]-hard	W[2]-hard	W[2]-hard	W[2]-hard

Table 4.2: **Tractability with respect to the total weight of the subtree depending on the weight function**

given by $\{x, x'\}$. As every subtree can contain at most one $d \in D$ there can be at most k vertices $d \in D$ that are part of ST. □

Theorem 4.15. *G-STG is W[2]-hard w.r.t. the weight of the subgraph G' for a rational weight function.*

Proof. We give a reduction from Dominating Set that is very similar to the reduction for the proof of the W[2]-hardness of STG used in the proof of Theorem 4.14. The only difference is that as we do not have a given set of terminals we have to simulate this by adding new vertices that are connected by edges with weight $w = 0$. More precisely, we construct G' as in the proof of Theorem 4.14 and add $n + 1$ vertices $\{t_i \mid i = 1, \ldots, n + 1\}$ and connect them by the edges $T = \{\{t_i, r_i\} \mid i = 1, \ldots, n\} \cup \{t_{n+1}, x'\}$ with edge weight $w(t) = 0$ forall $t \in T$.

We then can claim the following. A graph G has a Dominating Set of size k if and only if there is a Generalized Steiner Tree in G' of size $2n + k + 3$ and weight $2 \cdot k - 1$. The proof is analogous to the proof of the W[2]-hardness for STG. □

Table 4.2 provides on overview of of parameterized complexity w.r.t. the weight of the subgraph for all four problems. Allowing only for uniform weights simplifies G-STG and GV-STG such that they are solvable in linear time. In contrast, STG and V-STG remain NP-hard and can be computed with efficient fixed-parameter algorithms. Going from binary to integer weight functions, every problem remains in the same complexity class. Interestingly, the decisive factor for their complexity seems to be the type of the weights (vertex or edge weights). The complexity class does not change with the step from the original problem versions to the generalized ones. Though the algorithm for G-STG is more complex and practically there is a big gap between the performance of it and the algorithm for STG, both are

still fixed-parameter tractable. As to the vertex-weighted problems, V-STG is already W[2]-hard for a binary weight function. Stepping further form an integer to a rational weight function STG and G-STG become W[2]-hard as well.

4.5.3 Parameterized complexity of MWCS

In the supplementary materials of [IOSS02] Karp showed that MWCS is NP-hard by reduction from Set Cover.

Whereas, as can be seen in Table 4.1 in the case of the other four problems it is possible to find a fixed-parameter algorithm w.r.t. the number of nodes of the subgraph, this seems to be difficult for MWCS. This is due to the weight function that allows for negative weights as well. Therefore, for a subgraph S of size k, the information that there is no subgraph of size $k+1$ with higher weight than S is not sufficient to conclude that there is no such subgraph with more than $k+1$ vertices. Moreover, it is not obvious whether MWCS is W[2]-hard w.r.t the number of vertices of the subgraph.

Regarding the total weight of the maximum connected component as parameter we can show the following.

Theorem 4.16. *MWCS is W[2]-hard w.r.t. the total weight of the maximum connected component.*

Proof. We prove Theorem 4.16 by reduction from Dominating Set.

Given a DS instance $G = (V, E)$ with $V = \{1, ..., n\}$ and parameter k which is the size of a dominating set, we construct an MWCS instance $G' = (V', E')$ with weight function $w : V' \to \mathbb{R}$ as follows:

- $V' = \{d_i \mid i = 1, \ldots, n\} \cup \{r_i \mid i = 1, \ldots, n\}$
- $E' = \{(d_i, d_j) \mid i, j = 1, \ldots, n \wedge i \neq j\} \cup \{(d_i, r_j) \mid i = 1, \ldots, n \wedge j \in N[i]\}$
- $w(r_i) = \frac{2 \cdot k}{2 \cdot n - k}$ forall $i = 1, \ldots, n$
- $w(d_i) = -\frac{k}{2 \cdot n - k}$ forall $i = 1, \ldots, n$

Claim: G has a dominating set of size k iff the weight of a maximum-weight connected subgraph is k.

As the proof of the Claim is similar to the proof of Theorem 4.14 we only sketch the basic observations needed for it. Denote $R := \{r_i \mid i = 1, ..., n\}$ and $D := \{d_j \mid j = 1, ..., n\}$.

- Vertices $r_i \in R$ can only be connected by vertices $d_j \in D$.
- It is always worth connecting a node $r_i \in R$ to the subgraph as its weight is higher than the negative weight one has "to pay" for its connector vertex.
- A maximum-weight connected subgraph contains all $r_i \in R$ and as few $d_j \in D$ as possible.
- The weight of a subgraph containing k vertices out of D (which correspond to the dominating set) and all vertices $r_i \in R$ is $w(S) = n \cdot \frac{2 \cdot k}{2 \cdot n - k} - k \cdot \frac{k}{2 \cdot n - k} = k$.

□

4.5.4 Discussion

In this chapter, we introduced G-STG and GV-STG, which are graph theoretical problems that can be applied in the analysis of biological networks. Therefore, they provide another fruitful link between graph theory and biochemistry. As an example for this link, we describe how algorithms for the problems can be used to screen protein interaction or transcriptional networks for important subnetworks. Out of lack of time, we omitted an implementation and experiments. However, the work of Scott et a. [SIKS05], that uses a similar color-coding based approach in the analysis of protein interaction networks, shows evidence that our approach might be effective in practice. To give a more complete picture of this field of research we also present some basic ideas of the work of Ideker et al. [IOSS02] based on the MWCS problem and contrasting our new problems to MWCS.

The main part of this chapter is concerned with the parameterized complexity of G-STG, GV-STG, and MWCS as well as of the STEINER TREE IN GRAPHS problem and its vertex-weighted variant. We were able to prove the W[2]-hardness w.r.t. a range of parameters and to give fixed-parameter algorithms w.r.t. other practically relevant parameterizations.

One of the main results is the formulation of fixed-parameter algorithms for G-STG and GV-STG w.r.t. the number of nodes. No such algorithm is known for MWCS. Although the running time of the algorithm for G-STG (and GV-STG) is not practical yet, there are some simple modifications which have the potential to make the algorithms more efficient. First, one can use the randomized version to call the `WTCC` algorithm, which after a reasonable number of random colorings finds with high probability a specific

subgraph. Then, a further improvement of the running time for practical application can be achieved by iterating only over some selected tree structures instead of iterating over all possible tree structures. What kind of tree structures seem to occur more likely than others in metabolic or other pathways therefore seems to be an interesting field of future research. One example for such a tree structure is mentioned by Scott et al. [SIKS05]. It consists of two separate paths that merge into a single path. Although they, independently from our work, also describe the color-coding method for trees, the experimental part of their work is focused on paths only and gives no further hint for common tree structures.

Looking at the W[2]-hardness results w.r.t. the weight of the subgraph, an interesting observation is that it shows the rise of complexity while going from the edge to the vertex-weighted variants, e.g. going from STG to V-STG. The observation that a problem becomes more complex if one considers vertices instead of edges seems to be quite common for graph-theoretical problems. Another well-known example is the domination of all edges in a network which is called Vertex Cover and is in FPT whereas the domination of all vertices, the Dominating Set problem, is W[2]-hard [DF99, Nie06]. Furthermore, it indicates that V-STG is more difficult than STG. This fits perfectly to the observation that a lot of reduction rules for the STG can not be modified to apply them to V-STG and to the fact that V-STG is harder to approximate than STG as described in Section 3.3.

Chapter 5

Conclusion

In the following we summarize this work including a brief overview of the most important contributions. Finally, we pose open questions in the context of this work.

5.1 Summary

We considered various algorithmic techniques in the analysis of biological networks. The work has two main parts, a practical and a theoretical one. Firstly, we presented the software tool Steiner Package and substantiated its benefit for biological analysis by providing examples with biological importance. An important part of the Steiner Package are the data reduction and preprocessing rules that have been developed in this work. Secondly, we concentrated on the parameterized complexity of Steiner tree related problems. We gave new hardness results and developed fixed-parameter algorithms for problems with biological significance. Furthermore, the systematic analysis of the problems gave interesting insights into their relations and problem structures.

5.2 Open problems and challenges

Subsequent to this work there is still a wide range of open questions and approaches for further research.

Improved solution strategies for V-STG. As V-STG is important for biological analysis it would be important to develop further strategies to obtain optimal or biological meaningful solutions. Although in this work

we greatly profited from using simple reduction rules, we know about no work that presents other concepts, like more advanced reduction rules or alternative solution techniques.

Problem kernel for STG (and V-STG) Although data reduction by preprocessing is immensely well-studied for STG, the question of how to obtain a problem kernel as defined in [DF99] is still open.

Enumeration of Steiner trees. A nice extension of the Steiner approach would be the enumeration of all optimal Steiner trees. As the number of optimal Steiner trees could be exponential in the size of the network, the enumeration itself and the extraction of information of all optimal solutions yields new problems. An interesting question in this context is if there is a compact representation of all optimal solutions analogously to the work of Damaschke [Dam04]. Regarding the analysis of biological networks, further information about vertices could be obtained by considering if it is part of more than one optimal solution, or if there is an optimal solution without it.

Further applications for the Steiner approach Until now the Steiner approach was mainly used for the analysis of interaction or protein interaction networks. Therefore, it could be interesting to investigate further applications, i.e. on other types of biological networks.

Parameterized complexity of Maximum-Weight Connected Subgraph. An interesting open question is the parameterized complexity of MWCS w.r.t. the number of nodes as parameter. As discussed in Section 4.5.3, the color-coding based approach that helped to a breakthrough for G-STG and GV-STG seems not to be applicable to MWCS.

Implementation of new algorithms for G-STG and GV-STG From a biological point of view it seems to be important to implement the introduced algorithms and investigate how useful they are for real life applications. Furthermore, it could be useful to study possible combination of G-STG and GV-STG for applications to networks with edge and vertex weights, i.e. protein interaction networks with edge weights based on reliability of an interaction and vertex weights depending on differential expression data.

Bibliography

[AAA+05] C. Alfarano, C. E. Andrade, K. Anthony, N. Bahroos, et al. The Biomolecular Interaction Network Database and related tools 2005 update. *Nucleic Acids Research*, 33(Database issue):D418-24, Jan 2005. PubMed. 14, 21

[AYZ95] N. Alon, R. Yuster, and U. Zwick. Color-coding. *J.ACM*, 42(4):844–856, 1995. 7, 65, 66, 71, 72

[BB05] C. Blum and M. J. Blesa. New metaheuristic approaches for the edge-weighted k-cardinality tree problem. *Computers and Operations Research*, 32:1355–1377, 2005. 62

[BE03] C. Blum and M. Ehrgott. Local search algorithms for the k-cardinality tree problem. *Discrete Applied Mathematics*, 128:511–540, 2003. 62

[Bea84] J. E. Beasley. An algorithm for the Steiner problem in graphs. *Networks*, 14:147–159, 1984. 27

[BP89] M. W. Bern and P. E. Plassmann. The Steiner problem with edge lengths 1 and 2. *Information Processing Letters*, 32(4):171–176, 1989. 25

[BST02] J. M. Berg, L. Stryer, and J. L. Tymoczko. *Biochemistry*. W. H. Freeman, 5 edition, 2002. 12

[BUM06] J. Brimberg, D. Urošević, and Mladenović. Variable neighborhood search for the vertex weighted k-cardinality tree problem. *European Journal of Operational Research*, 171:74–84, 2006. 62

[Ces] Marco Cesati. Compendium of parameterized complexity. http://bravo.ce.uniroma2.it/home/cesati/research/compendium/. 74

[CK94] S. Y. Cheung and A. Kumar. Efficient quorumcast routing algorithms. In *Proceedings of IEEE INFOCOM 94*, 1994. 61

[CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts, 2 edition, 2001. 22, 32, 35

[Dam04] P. Damaschke. Parameterized enumeration, transversals, and imperfect phylogeny reconstruction. In *Proceedings of 1st IWPEC*, volume 3162 of *LNCS*, pages 1–12. Springer, 2004. Long version to appear in *Theoretical Computer Science*. 90

[DF99] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999. 9, 10, 63, 64, 74, 87, 90

[DP02] S. V. Daneshmand and T. Polzin. Extending reduction techniques for the Steiner tree problem. In *Algorithms - ESA 2002: 10th Annual European Symposium*, volume 2461 of *LNCS*, pages 795–807. Springer, 2002. 26

[DPB+96] J. DeRisi, L. Penland, P. O. Brown, M. L. Bittner, et al. Use of a cDNA microarray to analyse gene expression patterns in human cancer. *Nature Genetics*, 14(4):457–460, 1996. 13

[DRS00] D-Z. Du, J. H. Rubenstein, and J. M. Smith, editors. *Advances in Steiner Trees*. Kluwer Academic Publishers, 2000. 6

[Dui00] C. Duin. *Advances in Steiner Trees*, chapter "Preprocessing the Steiner Problem in Graphs", pages 175–233. Kluwer Academic Publishers, 2000. 6, 26

[DV89] C.W. Duin and A. Volgenant. Reduction tests for the Steiner problem in graphs. *Networks*, 19:549–567, 1989. 6, 25, 26, 27, 28, 35

[DW72] S. E. Dreyfus and R. A. Wagner. The Steiner problem in graphs. *Networks*, 1:195–207, 1972. 6, 7, 21, 22, 74

[FH92] L. R. Foulds and H. W. Hamacher. A new integer programming approach to (restricted) facilities layout problems allowing flexible shapes. Technical Report 1992-3, University of Waikato, Department of Management Science, 1992. 61

[FK94] U. Faigle and W. Kern. Computational complexity of some maximum average weight problems with precedence constraints. *Operations Research*, 42(4):688–693, 1994. 59, 61

[fPSM] Munich Information Center for Protein Sequences (MIPS). http://mips.gsf.de/. 48

[FR99] J. Feldman and M. Ruhl. The Directed Steiner Network problem is tractable for a constant number of terminals. In *40th IEEE Symposium on Foundations of Computer Science*, pages 299–308, 1999. 21

[Gar96] N. Garg. A 3-approximation for the minimum tree spanning k vertices. In *Proceedings of the 37th IEEE Symposium on Foundations of Computer Science*, pages 302–309, 1996. 62

[GH97] N. Garg and D. Hochbaum. An $O(\log k)$ approximation algorithm for the k minimum spanning tree in the plane. *Algorithmica*, 18:111–121, 1997. 61

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability.* W. H. Freeman and Company, 1979. 21, 43

[GK99] S. Guha and S. Khuller. Improved methods for approximating node weighted Steiner trees and connected dominating sets. *Information and Computation*, 150:57–74, 1999. 35

[GMK+05] U. Güldener, M. Münsterkötter, G. Kastenmüller, N. Strack, et al. CYGD: the Comprehensive Yeast Genome Database. *Nucleic Acids Research*, 33 Database Issue:D364-8, Jan 2005. 14

[Hak72] S. L. Hakimi. Steiner's problem in graphs and its applications. *Networks*, 1:113–133, 1972. 24

[Hal04] M. Hallett. personal communication, April 2004. 7, 60

[HBH+04] J-D. J. Han, N. Bertin, T. Hao, D. S. Goldberg, et al. Evidence for dynamically organized modularity in the yeast protein-protein interaction network. *Nature*, 430:88–93, 2004. 43

[HJ93] H. W. Hamacher and K. Jörnsten. Optimal relinquishment according to the norwegian petroleum law: A combinatorial optimization approach. Technical Report 7/93, Norwegian School of Economics and Business Administration, Bergen, 1993. 61

[HMWD04] Z. Hu, J. Mellor, J. Wu, and C. DeLisi. VisANT: an online visualization and analysis tool for biological interaction data. *BMC Bioinformatics*, 5:17, 2004. 15

[ILB04] J. Ihmels, R. Levy, and N. Barkai. Principles of transcriptional control in the metabolic network of Saccharomyces cerevisiae. *Nature Biotechnology*, 22(1):86–92, Jan 2004. 5, 16, 18, 22

[IOSS02] T. Ideker, O. Ozier, B. Schikowski, and A. F. Siegel. Discovering regulatory and signalling circuits in molecular interaction networks. *Bioinformatics*, 18:233–240, 2002. 5, 17, 18, 59, 60, 62, 63, 85, 86

[ITR+01] T. Ideker, V. Thorsson, J. A. Ranish, R. Christmas, et al. Integrated genomic and proteomic analyses of a systematically perturbed metabolic network. *Science*, 292(5518):929–34, May 2001. 5, 13, 17, 21, 53

[ITSH00] T. Ideker, V. Thorsson, A. F. Siegel, and L. E. Hood. Testing for differentially-expressed genes by maximum-likelihood analysis of micrarray data. *Journal of Computational Biology*, 7(6):805–17, 2000. 13

[JMALO01] H. Jeong, S. P. Mason, Barabási A.-L., and Z. N. Oltvai. Lethality and centrality in protein networks. *Nature*, 411:41–42, May 2001. 39, 40

[KM98] T. Koch and A. Martin. Solving Steiner tree problems in graphs to optimality. *Networks*, 32:207–232, 1998. 6, 26, 28

[KR95] P. Klein and R. Ravi. A nearly best-possible approximation algorithm for node-weighted Steiner trees. *Journal of Algorithms*, 19:104–115, 1995. 6, 33

[KS90] E. Korach and N. Solel. Linear time algorithm for minimum weight Steiner tree in graphs with bounded tree-width. Technical Report 632, Technion - Israel Institute of Technology, Computer Science Department, Haifa, Israel, 1990. 24, 25

[LRR+02] T. I. Lee, N. J. Rinaldi, F. Robert, D. T. Odom, et al. Transcriptional regulatory networks in Saccharomyces cerevisiae. *Science*, 298(5594):799–804, Oct 2002. 5, 15, 16, 18, 21

[MRR05] D. Mölle, S. Richter, and P. Rossmanith. A faster algorithm for the Steiner tree problem. Technical Report AIB-2005-04, Department of Computer Science, RWTH Aachen, 2005. To appear in STACS2006, Marseille, France, Feb 2006, Springer LNCS. 24

[Nie06] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006. 9, 63, 87

[NK04] M. Narayanan and R. Karp. personal communication, July 2004. 60

[Ott48] R. Otter. The number of trees. *Annals of Mathematics, 2nd Ser.*, 49(3):583–599, July 1948. 75

[PKO+05] P. Pagel, S. Kovac, M. Oesterheld, B. Brauner, et al. The MIPS mammalian protein-protein interaction database. *Bioinformatics*, 21(6):832–834, 2005. 14

[PS02] H. J. Prömel and A. Steger. *The Steiner Tree Problem*. Vieweg, 2002. 6, 23, 24

[PV02] T. Polzin and S. Vahdati. Using (sub)graphs of small width for solving the Steiner problem. Technical report, MPI-I-2002-1-001, 2002. 6, 25

[RSM+96] R. Ravi, R. Sundaram, M. V. Marathe, D. J. Rosenkrantz, and S. S. Ravi. Spanning trees short or small. *SIAM Journal on Discrete Mathematics*, 9(2):178–200, 1996. 61

[RZ00] G. Robins and A. Zelikovsky. Improved Steiner tree approximation in graphs. *Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms*, pages 770–779, 2000. 6, 25

[SF97] T. H. Stevens and M. Forgac. Structure, function, and regulation of the vacuolar (H^+)-ATPase. *Annual Review of Cell and Developmental Biology*, 13:779–808, Nov 1997. 57

[SGR+99] D. E. Sterner, P. A. Grant, S. M. Roberts, et al. Functional organization of the yeast SAGA complex: distinct components involved in structural integrity, nucleosome acetylation, and TATA-binding protein interaction. *Molecular and Cellular Biology*, 19(1):86–98, Jan 1999. 57

[SIKS05] J. Scott, T. Ideker, R. M. Karp, and R. Sharan. Efficient algorithms for detecting signaling pathways in protein interaction networks. In *Research in Computational Molecular Biology: 9th International Conference, RECOMB 2005*, volume 3500 of *LNCS*. Springer, 2005. 5, 8, 16, 18, 60, 62, 66, 86, 87

[SPB+05] M. S. Scott, T. Perkins, S. Bunnell, F. Pepin, D. Y. Thomas, and M. Hallett. Identifying regulatory subnetworks for a set of genes. *Molecular and Cellular Proteomics*, 4:683–692, 2005. 6, 8, 17, 18, 19, 20, 21, 29, 52, 53

[TM80] H. Takahashi and A. Matsuyama. An approximate solution for the Steiner problem in graphs. *Math. Japonica*, 9:463–470, 1980. 25

[UdAR02] E. Uchoa, M. P. de Aragao, and C. C. Ribeiro. Preprocessing Steiner problems from VLSI layout. *Networks*, 40(1):38–50, 2002. 26

[VZVK95] V. E. Velculescu, L. Zhang, B. Vogelstein, and K. W. Kinzler. Serial analysis of gene expression. *Science*, 270(5235):484–7, Oct 20 1995. 12

[WCF+01] E. Wingender, X. Chen, E. Fricke, R. Geffers, et al. The TRANSFAC system on gene expression regulation. *Nucleid Acids Research*, 29:281–283, 2001. 14, 21

[Win95] P. Winter. Reductions for the rectilinear Steiner tree problem. *Networks*, 26:187–198, 1995. 26

[ZZ99] J. Zhu and M. Q. Zhang. SCPD: A promotor database of yeast Saccharomyces cerevisiae. *Bioinformatics*, 15:607–611, 1999. 15

[ZZV+97] L. Zhang, W. Zhou, V. E. Velculescu, S. E. Kern, et al. Gene expression profiles in normal and cancer cells. *Science*, 276(5316):1268–72, May 1997. 13