Diss. ETH No. 13364

Resolving Conflicts in Problems from Computational Biology

A dissertation submitted to the
SWISS FEDERAL INSTITUTE OF TECHNOLOGY (ETH)
ZÜRICH

for the degree of
Doctor of Technical Sciences

presented by
ULRIKE STEGE
Dipl. Math.
born May 1, 1969
Mannheim, Germany
Citizen of Germany

accepted on the recommendation of
Prof. Dr. Gaston H. Gonnet, examiner
Prof. Dr. Emo Welzl, co-examiner

1999

Contents

Zusammenfassung

Evolutionsbäume stellen ein zentrales Thema im Gebiet der Biologie dar. Mit der Verfügbarkeit von grossen Mengen molekularer Sequenzdaten werden neue und verbesserte Methoden entwickelt, um Evolutionsbäume zu bestimmen. Die Dissertation untersucht mathematische Modelle aus dem Gebiet der Konfliktresolution in Sequenzdaten.

Die vorliegende Arbeit konzentriert sich auf zwei spezifische Konfliktresolutions-Probleme: das Problem, Inkonsistenzen zwischen Genbäumen und Speziesbäumen zu erklären; und das Problem, Konfliktgraphen, die man findet, wenn man Multiple Sequence Alignments (MSAs) bestimmen möchte zu lösen. Beide Probleme sind $\mathcal{NP}$-hart, aber effiziente praktische Lösungen sind gefragt. Wir untersuchen die parameterisierte Komplexität von diesen Problemen, um effiziente Parameterisierungen zu finden, die zu praktischen fixed-parameter-tractable Algorithmen führen. Damit wenden wir die neueste Ergebnisse aus dem Informatikgebiet der parameterisierten Komplexität auf Probleme aus der Computational Biology an.

Diese Dissertation besteht aus drei Hauptteilen. Der erste Teil motiviert die vorliegende Forschungarbeit und führt Definitionen und Terme aus der Graphentheorie, der klassischen Komplexitätstheorie und der parameterisierten Komplexitätstheorie ein, die dann in nachfolgenden Kapiteln verwendet werden. Im zweiten Teil studieren wir das Problem der Identifikation von Speziesbäumen, d.h., korrekte Evolutionsbäume für eine Menge Spezies, wenn eine Menge von (i.a. unterschiedlichen) Genbäumen gegeben ist.

Wir beginnen mit einer Übersicht von mathematischen Modellen für unterschiedliche Bäume und präsentieren die bekanntesten Methoden, um die Evolutionsbäume zu berechnen. Die vorliegende Arbeit fasst Modelle zusammen, um Evolutionsereignisse zu bewerten und, vom Duplication-und-Loss Modell ausgehend, entwickelt neue Modelle. Zwei

Probleme, die daraus resultieren sind GENE DUPLICATION and MULTIPLE GENE DUPLICATION. Der kleinste gemeinsame Superbaum (smallest common supertree) einer Menge von Genbäumen impliziert eine untere Schranke für die Anzahl von Genduplikationen, die nötig sind, um einen Genbaum mit einem Speziesbaum zu erklären. Wir zeigen, dass das SMALLEST-COMMON-SUPERTREE Problem $\mathcal{NP}$-vollständig und $\mathcal{W}[1]$-hart ist, wenn es nach der Anzahl von Inputbäumen parameterisiert wird. Danach untersuchen wir Eigenschaften des GENE DUPLICATION Problems, die zu einem fixed-parameter-tractable Algorithmus führen. Um die Komplexität des MULTIPLE GENE DUPLICATION Problems zu analysieren, haben wir das kombinatorische Spiel BALL AND TRAP erfunden, das mit einen Baum der mit Bällen und Fallen bestückt ist, gespielt wird. Das BALL-AND-TRAP Spiel wird dann verwendet um zu zeigen, dass das MULTIPLE-GENE-DUPLICATION Problem $\mathcal{NP}$-vollständig und $\mathcal{W}[1]$-hart ist.

Das Konstruieren von MSAs ist ein fundamentales Problem in Computational Biology. Die bekanntesten Algorithmen, um MSAs zu berechnen, produzieren gewöhnlich nicht eine exakte Lösung für bezüglich des zugrunde liegenden Modells, weil das Problem $\mathcal{NP}$-hart ist. Das Hauptproblem ist die falsch Plazierung von Gaps. Im dritten Teil von dieser Dissertation modellieren wir dieses Problem anhand eines Konfliktgraphen dessen Knoten bzw. Kanten Gaps bwz. Konflikte, repräsentieren. Das Ziel ist, die minimale Zahl von Gaps zu identifizieren, die die Konstruktion eines eindeutigen Evolutionsbaums verhindert. Damit haben wir das Problem in das VERTEX-COVER Problem transformiert. Für das k-VERTEX-COVER Problem fassen wir bekannte fixed-parameter-tractable Algorithmen zusammen und entwickeln einen neuen fixed-parameter-tractable Algorithmus, um Konfliktgraphen zu lösen. Die Hauptidee dieses Algorithmus ist eine verbesserte Kernelization, welche durch neue Reduktionsregeln und eine verbesserte Struktur des Suchbaumes erreicht wurde. Die Zeitkomplexität dieses Algorithmus ist $O(kn+r^k k)$, $r \approx 1.2906$, was den bisher besten Algorithmus von Niedermeier and Rossmanith, mit einer Laufzeit von $O(kn+r^k \cdot k^2)$, $r \approx 1.2917$, verbessert.

Abstract

Evolutionary trees, trees that reflect the ancestral relationships among species, have been a central topic in biology for many years. With the availability of large amounts of molecular sequence data, new and improved methods for estimating evolutionary trees are being developed. This dissertation investigates mathematical models in the area of conflict resolution in sequence data. This thesis concentrates on two specific conflict resolution problems: the problem of resolving inconsistencies between gene trees and species trees; and the problem of resolving conflict graphs encountered when computing Multiple Sequence Alignments (MSAs). Both problems are $\mathcal{NP}$-hard, but require efficient solutions in practice. We investigate the parameterized computational complexity of these problems to find effective parameterizations, which lead to practical fixed-parameter-tractable algorithms. Thus, we apply recent results of the computer science field parameterized complexity to problems of computational biology.

The thesis consists of three major parts. Part I provides motivation for this research and introduces definitions and terms from graph theory, classical computational complexity, and parameterized computational complexity used in subsequent chapters. In Part II we study the problem of identifying the species tree, that is, the evolutionary tree, for a set of species, when a set of (usually contradictory) gene trees is given. We begin with a survey of mathematical models for contradictory trees and present the best known methods for computing evolutionary trees. The thesis then surveys and develops models for counting evolutionary events based on the duplication-and-loss model. Two resulting problems are GENE DUPLICATION and MULTIPLE GENE DUPLICATION. The smallest common supertree of a set of gene trees implies a lower bound for the number of gene-duplication events necessary to rectify the gene tree with respect to a species tree. We show that the SMALLEST-COMMON-SUPERTREE problem is $\mathcal{NP}$-complete and $\mathcal{W}[1]$-hard when parameterized by the number of input trees. We then investigate properties of the GENE DUPLICATION problem, which lead to a fixed-parameter-tractable algorithm. To analyze the complexity of the MULTIPLE GENE DUPLICATION problem, we invented a combinatorial game called BALL AND TRAP which is played on a tree decorated with balls and traps. Using the BALL-AND-TRAP GAME, we show that the MULTIPLE-GENE-DUPLICATION problem is $\mathcal{NP}$-complete and $\mathcal{W}[1]$-hard.

Constructing MSAs is a fundamental problem in computational biology. The best known algorithms for computing MSAs usually fail to produce an exact solution corresponding to the underlying model due to the $\mathcal{NP}$-hardness of this problem. The main problem is the misplacement of gaps. In Part III of this dissertation, we model this problem by means of a conflict graph where the vertices and edges represent gaps and conflicts, respectively. The goal is to identify a minimum number of gaps which prevents the construction of a unique evolutionary tree. Thus, we have transformed the problem into the VERTEX-COVER problem. We present a survey of known fixed-parameter-tractable algorithms for the k-VERTEX-COVER problem and develop a new fixed-parameter-tractable algorithm to resolve conflict graphs. The main idea of this algorithm is an improved kernelization accomplished by new reduction rules and an improved structure of the search tree. The time complexity of this algorithm is $O(kn + r^k k)$, $r \approx 1.2906$, improving on the previous best algorithm by Niedermeier and Rossmanith, which runs in $O(kn + r^k \cdot k^2)$, $r \approx 1.2917$.

Part I

Introductory Part

Chapter 1

Introduction

1.1 Motivation

Evolutionary trees, trees that reflect the ancestral relationships among species, have been a central topic in biology for many years. With the availability of large amounts of sequence data (nowadays DNA and amino acid sequence data), which provide a rich source of information, new and improved methods for estimating evolutionary trees are being developed. As a result, many interdisciplinary research programs have emerged to store, manipulate, analyze, and visualize sequence data effectively.

This dissertation investigates selected mathematical models in the general area of *conflict resolution in molecular sequence data.* Conflicts in molecular sequence data arise, for example, due to random events amplified by the evolution of species, the wrong interpretation of experimental data, or the incorrect manipulation and storage of data.

In this thesis, we concentrate, in particular, on mathematical models for two specific conflict resolution problems:

1. the problem of resolving inconsistencies between gene trees and species trees; and

2. the problem of resolving conflict graphs encountered when computing multiple sequence alignments.

As many problems in this area, both investigated problems are $\mathcal{NP}$-hard but require efficient solutions in practise.

The most famous approaches to deal with $\mathcal{NP}$-hard problems are heuristics [32] and approximation algorithms [43]. Another way to deal with $\mathcal{NP}$-hard problems is to study the parameterized complexity for reasonable parameterizations of the problems [19]. Thus, this interdisciplinary thesis applies recent results of the computer science field *parameterized complexity* to problems in computational biology.

1.2 Problems

We first study the problem of identifying the correct *species tree,* that is, the correct evolutionary tree for a set of species, when a set of (usually contradictory) *gene trees* is given (cf. Figure 1.1). A gene tree is an evolutionary tree built over families of homologous genes. Two genes are said to be *homologous* if they evolved from a common ancestor. The inconsistencies among the different gene trees are caused by gene divergence and are the result of either a *speciation event* or a *duplication event.* A speciation event takes place in the genome of the least common ancestor taxa of the two corresponding genes whereas a duplication event occurs during evolution [22, 42]. We focus on *mathematical models* explaining the contradictions in the topologies of the gene trees via gene-duplication events and subsequent losses, that occur during the evolution of a gene family [36, 59, 73].

The second problem we consider in this dissertation concerns the *resolution of conflict graphs.* This problem has important practical applications in other areas of computer science, including fault-tolerant LCD digit design and traffic-light design. In computational biology, conflict resolution occurs when, for example, when constructing *Multiple Sequence Alignments* (MSA). MSAs can be used for building evolutionary trees and for predicting the secondary structure of proteins; both are fundamental problems in computational biology.

The problem of computing MSAs for different biological models is $\mathcal{NP}$-hard [13, 34, 39, 44, 71]. The known methods for computing MSAs usually fail to produce an exact solution. Often, the computed MSAs do not allow building a unique corresponding evolutionary tree (assuming the existence of an evolutionary tree corresponding to an MSA). One way to deal with this problem is to detect conflicts among sequences and then to transform the problem into a conflict graph where the sequences correspond to the vertices and the conflicts to the edges in the

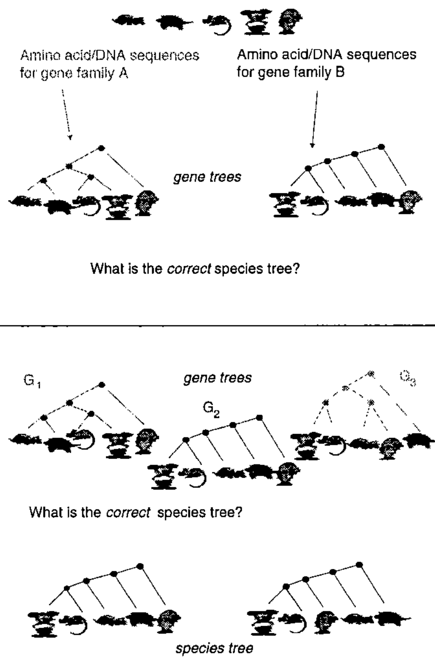

Amino acid/DNA sequences for gene family A
Amino acid/DNA sequences for gene family B
gene trees
What is the *correct* species tree?
G_1
gene trees
G_2
G_3
What is the *correct* species tree?
species tree

Figure 1.1: Gene trees are evolutionary trees built over families of homologous genes (upper figure). Given contradictory gene trees, the question is how to resolve the species tree. The species tree is not necessarily one of the given gene trees (lower figure).

graph. The goal then is to eliminate the minimum number of sequences (vertices) such that there is no conflict in the multiple sequence alignment of the remaining sequences. The graph problem to solve here is the $\mathcal{NP}$-complete problem VERTEX COVER [19, 21, 31, 46].

1.3 Approach and Major Results

The dissertation consists of four major parts. The first part collects the theoretical foundations necessary for the thesis. Part II investigates the problem of resolving inconsistencies between gene trees with respect to a species tree. Motivated by the MSA problem, Part III studies the resolution of conflict graphs on the example of VERTEX COVER. Part IV concludes the dissertation and poses open problems.

Part I: In Chapter 2, we provide a short introduction to the necessary graph theory, and we sketch the basics in classical and parameterized complexity theory. In parameterized complexity analysis [19], the goal is to identify useful ranges of a parameter k, e.g., for an $\mathcal{NP}$-hard problem and determine if the problem (for instances of size n) can be solved in time $f(k)n^{\alpha}$ for some constant α independent of the parameter. This behavior (*fixed-parameter tractability*) can be viewed as a generalization of $\mathcal{P}$-time. The analog of $\mathcal{NP}$ in parameterized terms is the complexity class $\mathcal{W}[1]$.

Part II: Assuming the evolution of a set of organisms is explainable by means of an evolutionary tree, we study the problem of resolving the correct species tree for a given set of (possibly contradictory) gene trees. Chapter 3 gives the biological background for this part. Related work to this problem is presented in Chapter 4. Chapter 5 introduces models that count evolutionary events to measure the inconsistencies between a gene tree and its corresponding species tree.

Besides a general concept for these kinds of models (Section 5.1), we describe the DUPLICATION-AND-LOSS MODEL [36, 38, 73]. The GENE-DUPLICATION MODEL (Section 5.3) is a restriction of the DUPLICATION-AND-LOSS MODEL to gene-duplication events only. Both the DUPLICATION-AND-LOSS MODEL and the GENE-DUPLICATION MODEL treat gene duplications as independent events and compute the minimum number of events (duplication and/or losses) necessary to rectify a gene tree with respect to a species tree. In Section 5.4 we introduce the MULTIPLE-GENE-DUPLICATION MODEL. Here gene duplications are not necessarily independent events; the model takes into account the evidence that genomes (e.g., Eukaryotic organisms) have been entirely duplicated one or more times or individual chromosomes (or parts of it) have been duplicated multiple times [29, 37, 38, 63].

Resulting from these models, we discuss the problems GENE DUPLICATION (Chapters 6 and 7) and MULTIPLE GENE DUPLICATION (Chapter 8). GENE DUPLICATION asks for the species tree which implies the smallest number of gene duplications necessary to rectify a set of gene trees with respect to the species tree; MULTIPLE GENE DUPLICATION asks for the species tree which implies the smallest number of multiple gene duplications necessary to rectify a set of gene trees with respect to the species tree.

SMALLEST COMMON SUPERTREE, the problem discussed in Chapter 6, has an interesting relation to GENE DUPLICATION since it implies a lower bound of the number of gene duplications necessary to rectify a set of gene trees with an optimal species tree. Given a set of binary trees, SMALLEST COMMON SUPERTREE asks for a smallest binary tree that is a supertree of the input trees. Though we show that the problem is $\mathcal{W}[1]$-hard when parameterized by the number of input trees (Section 6.2), the problem becomes fixed-parameter tractable when a small number of duplicated leaves is permitted additionally in the output tree (Section 6.2.2,[25]).

In Chapter 7, we present a fixed-parameter-tractable algorithm for the $\mathcal{NP}$-complete problem GENE DUPLICATION when parameterized by the number of gene duplications. In contrast to GENE DUPLICATION, MULTIPLE GENE DUPLICATION is $\mathcal{NP}$-complete even when the species tree is given and restricted to only two input trees (Chapter 8). Here, we also prove $\mathcal{W}[1]$-hardness of MULTIPLE GENE DUPLICATION for a reasonable parameterization.

Part III: We first describe the basic ideas of the known fixed-parameter-tractable algorithms of k-VERTEX COVER (Chapter 9). While the best algorithm in the literature runs in time $O(kn + r^k \cdot k^2)$, $r \approx 1.29175$, [58], we present an improved fixed-parameter tractable algorithm with a complexity of $O(kn + r^k k)$ and $r \approx 1.2906$ (Chapter 10).

In Chapter 11 we compare an implementation solving VERTEX COVER, which uses a fixed-parameter-tractable algorithm for k-VERTEX COVER, with two heuristics for VERTEX COVER.

Part IV concludes this thesis with Conclusions and Open Questions.

Chapter 2

Preliminaries

This chapter begins with a presentation of the graph theoretical notations used in subsequent chapters. Section 2.2 is a brief introduction to classical complexity theory; parameterized complexity theory is introduced in Section 2.3. By means of two examples, namely the famous CLIQUE and VERTEX COVER problems, we point out the likely differences between $\mathcal{W}[1]$-hardness and fixed-parameter tractability (Section 2.3.2).

2.1 Notation

A *graph* $G = (V, E)$ consists of a set of *vertices* V and a set of *edges* E, where $E \subseteq \binom{V}{2}$ is a set of unordered pairs. Usually, we denote the number of edges $|E|$ by m and the number of vertices $|V|$ by n. The graph $G^* = (V^*, E^*)$ is called the *complementary graph* of graph $G = (V, E)$ if $V^* = V$ and $E^* = \binom{V}{2} - E$.

A *path* $p(u, v)$ in $G = (V, E)$ from vertex $u \in V$ to vertex $v \in V$ is an ordered set $p(u, v) = [u, v_1, v_2, \ldots, v_k, v]$ of vertices of V such that $(u, v_1), (v_k, v) \in E$, and $(v_i, v_{i+1}) \in E$ for $i = 1, \ldots, k-1 (k \in \mathbb{N})$. Furthermore, in our context for a path $p(u, v)$ all the edges $(u, v_1), (v_k, v) \in E$ and $(v_i, v_{i+1}) \in E$ $(i = 1, \ldots, k-1)$ are pairwise distinct. The *length* $|p(u, v)|$ of a path $p(u, v) = [u, v_1, v_2, \ldots, v_k, v]$ is $k+1$, namely the number of edges between u and v in $p(u, v)$. The v_i, $i = 1, \ldots, k$, are called the *elements* of path $p(u, v)$ (in short, $v_i \in p(u, v)$, $i = 1, \ldots, k$). A path $p(u, v)$ in G of length at least 3 with $u = v$, $u \neq v_1, \ldots, v_k$, and $v_i \neq v_j$

for $i \neq j$ is called a *cycle* in G.

If $(v, w) \in E$, then we call w and v *neighbors* (or *adjacent vertices*). The *neighborhood* of a vertex v is $N(v) = \{w|(v, w) \in E\}$. For $u, v \in V$, we abbreviate $N(u) \cup N(v)$ by $N(u, v)$. $N[v] = N(v) \cup \{v\}$ denotes the *closed neighborhood* of $v \in V$.

The *degree* $\deg(v)$ of a vertex $v \in V$ is defined to be $\deg(v) = |N(v)|$. Let $x \in \mathbb{N}$. A graph $G = (V, E)$ is called *x-regular* if $\deg(v) = x$ for all $v \in V$.

Let $G = (V, E)$ be a graph and let $V' \subseteq V$. $G|_{V'} = (V', E')$, $E' = \{(u, v)|u, v \in V'$ and $(u, v) \in E\}$, is called the *restriction* of G to V'. A graph $G' = (V', E')$ is called a *subgraph* of $G = (V, E)$, if $V' \subseteq V$ and $E' \subseteq E$. We denote the subgraph property with $G' \subseteq G$.

For a graph $G = (V, E)$ and a vertex $u \in V$ we write $G - u$ to denote $G|_{V-\{u\}}$, the graph G without the vertex u. For a set V' we write $G - V'$ for $G|_{V-V'}$.

A *tree* $T = (V, E)$ is a connected, acyclic graph. A *rooted tree* $T = (V, E)$ is a tree with a distinguished vertex *root*(T), called the *root* of T. For each vertex v there is exactly one vertex v', $(v, v') \in E$, such that v' is on the (unique) path from v to the root in T. v' is called the *parent* of v. We denote v' with $\mathit{parent}_T(v)$. All the other vertices w, $w \neq v'$ and $(v, w) \in E$, are called the *children* of v in T. Vertices having no children are called *leaves* of T. Every vertex having one or more children is called an *internal vertex* of T. For each vertex $v \in V$, all the vertices belonging to the path $p(v, \mathit{root}(T))$ are *ancestors* of v. All trees occurring in this thesis are rooted trees. We simply call them *trees*. The *size* $|T|$ of a tree $T = (V, E)$ we define to be $|T| = |V|$.

A tree is *binary* if each internal vertex has at most two children. A tree is *complete binary* if each internal vertex has exactly two children. For each $v \in V$ in a complete binary tree T we call the two children v_l and v_r of v, the *left child* and the *right child* of v. Due to algorithmic reasons we use the ordered version of a complete binary tree, although in most cases we make no use of the ordering.

A *leaf-labeled tree* $T = (V, E, L)$ is a tree with vertex set V, edge set E, and a set of labels L, where each leaf $v \in V$ is labeled by an element $l \in L$, denoted by $\ell(v) = l$. We say $T = (V, E, L)$ is a leaf-labeled tree over L, if for each $l \in L$ there is a leaf $v \in V$ such that $\ell(v) = l$.Then L is called the *leafset* of T.

Let $T = (V, E, L)$, $T' = (V', E', L')$ be leaf-labeled trees. T' is a *subtree* of T, $T' \leq_{top} T$, if T' is contained in T by topological containment that respects ancestry with label isomorphism at the leaves. For a leaf-labeled tree $T = (V, E, L)$ and a vertex $u \in V$, $T(u) = (V', E', L')$, $V' = \{v \in V | u$ is on the path $p(v, root(T))\}$ and $E' = \{(v, w) | v, w \in V'$ and $(v, w) \in E\}$, is the *subtree* of T *induced by* u. The leafset of $T(u)$ is denoted by $LS(u) = L'$.

A leaf-labeled tree $T = (V, E, L)$ is called a *p-tree* (for *phylogenetic tree*) if there are no two leaves labeled by the same leaf label. Otherwise, we call T an *rl-tree* (or *repeated-leaf tree*). That is, in rl-trees leaf labels may be repeated. Using p-trees, we simplify the notation of a leaf-labeled tree $T = (V, E, L)$ over L by identifying the leaf labels and the leaves: we define $\ell(v) = v$ for all leaves v in T. Thus, $L \subseteq V$ denotes the set of leaves in a p-tree T.

We call two leaf-labeled p-trees $T_1 = (V_1, E_1, L_1)$ and $T_2 = (V_2, E_2, L_2)$ to be *leaf-labeled isomorphic*, $T_1 \cong T_2$, if $T_1 \leq_{top} T_2$ and $L_1 = L_2$.

Suppose we are given a leaf-labeled tree $T = (V, E, L)$ over L and a set $L' \subseteq L$. A *least common ancestor* $lca(L')$ in T is a vertex u, $u \in V$, s.t.

1. $L' \subseteq LS(u)$, and
2. there is no vertex x in $T(u)$, $x \neq u$, with $L' \subseteq LS(x)$.

Note, that the least common ancestor of a set of leaves is unique when a p-tree is given.

2.2 Classical Computational Complexity

In this section, we give the necessary definitions of classical computational complexity theory. For further background we recommend the book *Computers and Intractability* by Garey and Johnson [31]. We assume basic knowledge in formal language theory.

A *decision problem* Π consists of a set D_Π of *instances* and a subset $Y_\Pi \subseteq D_\Pi$ of *yes-instances*. For any finite set Σ of symbols, Σ^* denotes the set of all finite strings of symbols over Σ. If $\mathcal{L}$ is a subset $\mathcal{L} \subseteq \Sigma^*$ we say that $\mathcal{L}$ is a *language* over the *alphabet* Σ.

Given an encoding scheme e for a problem Π, the connection between decision problems and languages is represented by $\mathcal{L}[\Pi, e] = \{x \in \Sigma^* | \Sigma$ is the alphabet used by e, and x is the encoding under e of an instance $I \in Y_\Pi\}$. Usually we abbreviate $\mathcal{L}[\Pi, e]$ by $\mathcal{L}$.

A deterministic Turing-machine program M with input alphabet Σ *accepts* $x \in \Sigma^*$ if and only if M halts in the distinguished accepting state when applied to input x. The language $\mathcal{L}_M$ recognized by the machine M is given by $\mathcal{L}_M = \{x \in \Sigma^* | M \text{ accepts } x\}$. We define the time-complexity function $T_M : \mathbb{Z}^+ \to \mathbb{Z}^+$, $T_M(n) = \max\{m|$ there is an $x \in \Sigma^*$ with $|x| = n$, such that the computation of M on input x takes m steps$\}$. A deterministic Turing-machine program M is called a *polynomial-time deterministic Turing-machine program* if there exist a polynomial p such that, for all $n \in \mathbb{Z}^+$, $T_M(n) \leq p(n)$.

We define the class $\mathcal{P}$, the *class of polynomial-time algorithms.* $\mathcal{P} = \{\mathcal{L}|$ there is a polynomial-time deterministic Turing-machine program M for which $\mathcal{L} = \mathcal{L}_M\}$. To introduce the class $\mathcal{NP}$, the *class of all languages recognizable nondeterministically in polynomial time*, we define for a nondeterministic Turing-machine program M, the time-complexity function $T_M : \mathbb{Z}^+ \to \mathbb{Z}^+$: $T_M(n) = \max(\{1\} \cup \{m|$ there is an $x \in \mathcal{L}_M, |x| = n$, such that the time to accept x by M is $m\})$. A nondeterministic Turing-machine program M is called a *polynomial-time nondeterministic Turing-machine program* if there exist a polynomial p such that $T_M(n) \leq p(n)$ for all $n \geq 1$. Finally, $\mathcal{NP} = \{\mathcal{L}|$ there is a polynomial-time nondeterministic Turing-machine program M for which $\mathcal{L} = \mathcal{L}_M\}$.

A *polynomial transformation* from a language $\mathcal{L}_1 \subseteq \Sigma_1^*$ to a language $\mathcal{L}_2 \subseteq \Sigma_2^*$ is a function $f : \Sigma_1^* \to \Sigma_2^*$ that satisfies the following two conditions:

1. There is a polynomial-time deterministic Turing-machine program that computes f.

2. For all $x \in \Sigma_1^*$, $x \in \mathcal{L}_1$ if and only if $f(x) \in \mathcal{L}_2$.

A language $\mathcal{L}$ is *$\mathcal{NP}$-complete*, if

1. $\mathcal{L} \in \mathcal{NP}$, and

2. for all languages $\mathcal{L}' \in \mathcal{NP}$ there is a polynomial transformation from $\mathcal{L}'$ to $\mathcal{L}$.

We call $\mathcal{L}$ *$\mathcal{NP}$-hard*, if it satisfies condition 2.

2.3 Parameterized Computational Complexity

The theory of parameterized complexity was introduced by Downey and Fellows [15, 16, 17, 18, 19]; for a detailed introduction in the area we recommend [19]; our surveys about coping with intractability in terms of parameterized-complexity theory give a brief introduction [20, 21] (joined work with Downey and Fellows).

2.3.1 Definitions

A *parameterized language* $\mathcal{L}$ is a subset $\mathcal{L} \subseteq \Sigma^* \times \mathbb{N}$. If $\mathcal{L}$ is a parameterized language and $\langle x, k \rangle \in \mathcal{L}$, then we will refer to x as the *main part*, and to k as the *parameter*. A parameterized language $\mathcal{L}$ is *fixed-parameter tractable* if it can be determined in time $f(k)n^\alpha$ whether $\langle x, k \rangle \in \mathcal{L}$, where $|x| = n$, α is a constant independent of both n and k, and f is an arbitrary function. The class of fixed-parameter-tractable parameterized languages is denoted $\mathcal{FPT}$. Note that the class $\mathcal{FPT}$ is unchanged if the definition above is modified by replacing $f(k)n^\alpha$ by $f(k) + n^\alpha$.[1] About half of the naturally parameterized problems cataloged as $\mathcal{NP}$-complete by Garey and Johnson [31] are in $\mathcal{FPT}$ [19].

Let $\mathcal{L}$ and $\mathcal{L}'$ be parameterized languages. We say that $\mathcal{L}$ *reduces* to $\mathcal{L}'$ by a *parameterized reduction* if there is an algorithm which transfers $\langle x, k \rangle$ into $\langle x', g(k) \rangle$ in time $f(k)|x|^\alpha$, where $f, g : \mathbb{N} \to \mathbb{N}$ are arbitrary functions and α is a constant independent of k, so that $\langle x, k \rangle \in \mathcal{L}$ if and only if $\langle x', g(k) \rangle \in \mathcal{L}'$.

The parameterized analog of the complexity class $\mathcal{NP}$ is $\mathcal{W}[1]$. Moreover, $\mathcal{W}[1]$-hardness is basic evidence that a parameterized problem is likely not to be fixed-parameter tractable. The analogy is very strong, since the k-STEP HALTING PROBLEM for nondeterministic Turing-machines is complete for $\mathcal{W}[1]$ [12].

Before we define the class $\mathcal{W}[1]$ we need some definitions. Let us consider Boolean expressions as Boolean circuits (e.g., a 3-conjunctive normal

[1]To see this useful property we consider a running time of $f(k)n^\alpha$. Then there exists a function $g(k)$ such that $g(k)+n^{\alpha+1}$ is also a time bound. Let $g(k) = f(k)^{\alpha+1}$. We show $f(k)n^\alpha \leq g(k) + n^{\alpha+1}$ for all n and k. If $n \leq f(k)$ then $g(k) \geq f(k)n^\alpha$. If $n \geq f(k)$ then $n^{\alpha+1} \geq f(k)n^\alpha$.

formula is a Boolean circuit consisting of one input (of unbounded fan-out) for each variable, a possibly inversion gate for each variable, and if built of a large *and* of small *or*s with a single output line). More general, a *decision circuit* C is a Boolean circuit with exactly one output line. Given $x \in \{0,1\}^n$, an input vector to the n input lines of C, $C(x)$ is said to be true if the output line has value 1. Otherwise, it is said to be false.

For a decision circuit we define *small gates* to be *not* gates, *and* gates, and *or* gates with some predetermined bound on fan-in, and *large gates* to be *and* gates and *or* gates with unrestricted fan-in. A decision circuit is of *mixed type* if it consists of circuits having small gates and large gates.

A Boolean circuit has fan-out one. The *depth* of a circuit C is defined to be the maximum number of gates (small or large) on any path in C from the input variables to the output line. The *weft* of a circuit C is defined to be the maximum number of large gates on any path from the input variables to the output line. A family of decision circuits F has *bounded depth* if there is a constant h such that every circuit in the family F has depth at most h. F has *bounded weft* if there is a constant t such that every circuit in the family F has weft at most t. A decision circuit has a satisfying assignment of *weight* k if the assignment is true and it has exactly k variables set to be true.

Let $F = \{C_1, C_2, \ldots, C_i, \ldots\}$ be a family of decision circuits. Associated with F is a parameterized language $\mathcal{L}_F = \{\langle C_i, k\rangle | C_i$ has a weight k satisfying assignment$\}$.

A parameterized language $\mathcal{L}$ belongs to the class $\mathcal{W}[1]$ if $\mathcal{L}$ reduces to the parameterized problem $\mathcal{L}_{F(1,h)}$ for the family $F(1,h)$ of mixed type circuits of weft at most 1 and depth at most h, for some constant h.

A parameterized language $\mathcal{L}$ is *$\mathcal{W}$[1]-complete*, if

1. $\mathcal{L} \in \mathcal{W}[1]$, and
2. for all parameterized languages $\mathcal{L}'$, $\mathcal{L}' \in \mathcal{W}[1]$, $\mathcal{L}'$ reduces to $\mathcal{L}$ by a parameterized reduction.

We call $\mathcal{L}$ *$\mathcal{W}$[1]-hard*, if it satisfies condition 2.

What about the practicality of a fixed-parameter-tractable algorithm? Of course, considering a running time of $O(f(k)n^\alpha)$ of an algorithm for a fixed parameter k, the range of k depends strongly on the function f. Downey and Fellows introduce the ***klam value*** which is the largest k such that $f(k) \leq 10^{20}$ to characterize the practicality of a fixed-parameter-tractable algorithm [19]. In the example of k-VERTEX COVER, (cf. Problem 2.1 and Problem 2.3) we present our fixed-parameter-tractable algorithm which has a klam value of 157 (cf. Chapter 10).

2.3.2 Two Examples: Clique and Vertex Cover

In 1972, Karp proved, in one of his seminal papers, that both CLIQUE and VERTEX COVER are $\mathcal{NP}$-complete [46].

The classical versions

Problem 2.1. VERTEX COVER
Instance: A graph $G = (V, E)$, a positive integer k.
Question: Does G have a k-vertex cover (i.e., does there exist a subset $V' \subseteq V$, $|V'| \leq k$, such that for each $(x, y) \in E$ either x or y belongs to V')?

Problem 2.2. CLIQUE
Instance: A graph $G = (V, E)$, a positive integer k.
Question: Does G have a clique of size $\geq k$ (i.e., does there exist a subset $V' \subseteq V$, $|V'| \geq k$, such that $\binom{V'}{2} \subseteq V$?)?

In fact, the $\mathcal{NP}$-completeness of CLIQUE is shown via a reduction from VERTEX COVER. It turns out that the complexity of the naturally parameterized versions of these problems is likely to be different.

The parameterized versions

Problem 2.3. k-VERTEX COVER
Instance: A graph $G = (V, E)$, a positive integer k.
Parameter: k.
Question: Does G have a k-vertex cover (i.e., does there exist a subset $V' \subseteq V$, $|V'| \leq k$, such that for each $(x, y) \in E$ either x or y belongs to V')?

Problem 2.4. k-CLIQUE
Instance: A graph $G = (V, E)$, a positive integer k.
Parameter: k.
Question: Does G have a clique of size $\geq k$ (i.e., does there exist a subset $V' \subseteq V$, $|V'| \geq k$, such that $\binom{V'}{2} \subseteq V$?)?

While k-CLIQUE is shown to be $\mathcal{W}[1]$-complete [18], k-VERTEX COVER is fixed-parameter tractable. This follows directly from this observation.

Observation 2.1. *Given a graph $G = (V, E)$. Then for each $v \in V$ and each vertex cover VC of G*

$$v \in VC \text{ or } N(v) \subseteq VC.$$

Thus, given an input $\langle G, k\rangle$, the original input graph G has a k-vertex cover if $\langle G - v, k - 1\rangle$ or $\langle G - N(v), k - |N(v)|\rangle$ has a solution. Since the fixed parameter k reduces in each such step by at least one, we can decide in time $O(2^k|V|)$ whether G has a vertex cover of size k [16, 24]. The technique used, the method of bounded search trees, is described below. For further historical information and an improved fixed-parameter-tractable algorithm for k-VERTEX COVER running in time $O(k|V| + r^k k)$, $r \approx 1.2906$, we refer to Part III.

2.3.3 $\mathcal{FPT}$ Techniques

We describe two important techniques for constructing fixed-parameter-tractable algorithms. The first technique, the method of reduction to a problem kernel, can be applied not only as a preprocessing step but also at each node of a bounded search tree.

The method of reduction to a problem kernel

The main idea of this technique is to reduce in polynomial time the given problem $\langle x, k\rangle$ to an equivalent problem where the problem size is bounded by a function of k. More precisely, this method reduces a problem instance I to an equivalent instance I', where the size of I' is bounded by some function of the parameter k. The instance I' is exhaustively analyzed, and a solution for I' can be lifted to a solution for I in the case where a solution exists. Often, this technique leads to an additive rather than a multiplicative $f(k)$ exponential factor. We will illustrate this technique in Chapter 10 using the k-VERTEX COVER example.

The method of bounded search trees

This method was first introduced by Downey and Fellows in [17]. The idea here is to maintain a bounded search tree of the different possibilities for finding a solution of the problem $\langle x, k \rangle$. A search tree is a rooted tree bounded in the size by a function $f(k)$. To avoid confusion we call the vertices of a search tree *nodes*. The *nodes* of the search tree are labeled by *k-solution candidate sets.* The search tree does not have to be *small* (i.e., an exponential size is allowed).

In general, a fixed-parameter-tractable algorithm using the method of bounded search trees is described as follows.

1. Compute a search tree.
2. Run an efficient algorithm on each *branch* of the tree.

If the size of the search tree depends only upon the parameter, then, for a fixed k, the search tree becomes constant size and the algorithm is then efficient for each fixed k. This method is illustrated with the examples of GENE DUPLICATION (Problem 7.1, Chapter 7) and k-VERTEX COVER (Section 10.3).

Part II

Resolving Inconsistencies between Gene Trees and Species Trees

Seite Leer /
Blank leaf

Chapter 3

Biological Background

In the past, the main source of information for the reconstruction of evolutionary relationships among species was studying the history of a *character*. Characters are independent variables where values are collections of mutually exclusive character states (e.g., in the binary case: "Does a species have wings?") [42]. It is of prime importance to study homologous characters. These are characters that are based on evolutionary comparable structures [22].

As DNA sequences become easier to obtain, the evolution of a gene receives more importance. We consider homologous genes of different taxa that have an analogous function in their organisms. A set of homologous genes is called a *gene family*.

To compare gene trees of different gene families for one set of taxa, one usually exposes inconsistencies among the different gene trees [8, 36, 59]. We consider inconsistencies caused by gene divergence like *speciation* events or *duplication* events (cf. Figure 3.1). If the common ancestry of two homologous genes can be tracked back to a speciation event, then they are said to be related by *orthology*; two such genes are called *orthologous*. If the ancestry is traced back to a gene-duplication event, then they are related by *paralogy* [29, 42] and the two genes are called *paralogous* (cf. Figure 3.2). Once a gene has been duplicated, each copy can evolve independently. Thus, a single species may contain several copies of what was a single gene in an ancestor.

When we talk about gene duplications we only mean the duplicated genes which are accepted from their organisms. There is ample evidence that gene duplication is the most important mechanism for generat-

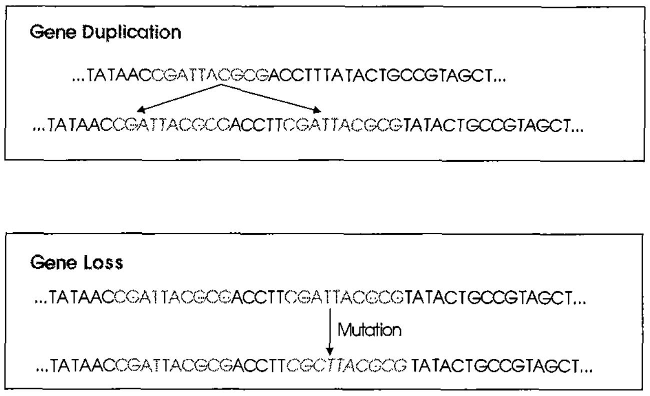

Gene Duplication
...TATAACCGATTACGCGACCTTTATACTGCCGTAGCT...
...TATAACCGATTACGCCACCTTCGATTACGCGTATACTGCCGTAGCT...
Gene Loss
...TATAACCGATTACGCGACCTTCGATTACGCGTATACTGCCGTAGCT...
Mutation
...TATAACCGATTACGCGACCTTCGCTTACGCG TATACTGCCGTAGCT...

Figure 3.1: An example of a gene duplication on a DNA sequence is shown in the upper figure. The gene (red) is replicated and inserted in the DNA sequence. In the lower figure an example of a gene loss, caused by a mutation is shown. In one of the duplicates of a gene, a mutation changed the base A into base C. Thus, the gene can loose its functionality. Gene losses can be caused by other evolutionary events like insertions or deletions.

ing new genes and new biochemical processes that have facilitated the evolution of complex organisms from primitive ones [51]. A duplicate gene may accumulate harmful mutations and become nonfunctional, as long as the other duplicate gene is functioning normally. These nonfunctional duplicate genes are called *pseudogenes* [51]. The existence of pseudogenes is much more likely than the possibility that a gene evolves into a new gene. In the model considered in this dissertation we do not distinguish between gene losses, pseudogenes, and newly evolved genes. We simply consider them all as *gene losses* (cf. Figure 3.1).

We assume the existence of an evolutionary tree for all the taxa we talk about. Furthermore, we make the basic assumption that we have exactly one gene from each contemporary species present in a gene tree [36, 38, 59]. We postulate gene losses for all the other possible homologous.

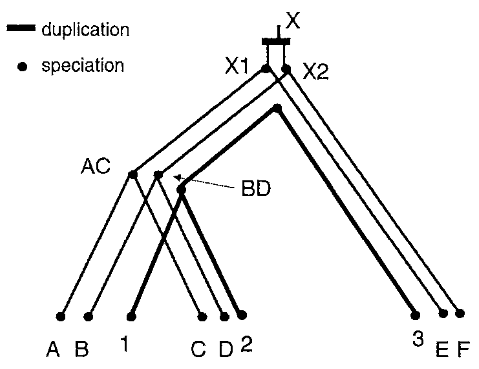


Figure 3.2: The evolution of a gene family (red). The phylogeny for the taxa 1,2, and 3 is the black tree. Orthologous are: A and C, B and D, AC and E, and BD and F. Paralogous in 1,2, and 3 are: A and B, C and D, and E and F, all caused by the duplication of the ancestor gene X).

Where does a duplication event happen? A duplication event involves a stretch of DNA in the genome of an organism. Thus, a duplication event can effect one or more genes at once (cf. Figure 3.3). We talk about a *multiple gene duplication* if a set of genes was duplicated in one event creating a set of paralogous genes.

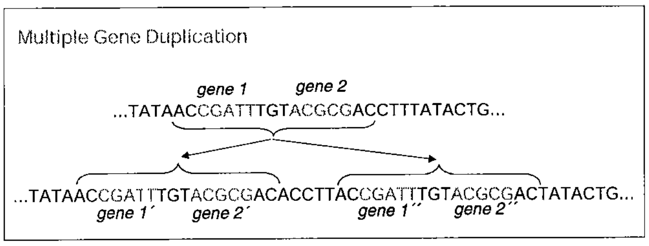

Multiple Gene Duplication
gene 1
gene 2
...TATAACCGATTTGTACGCGACCTTTATACTG...
...TATAACCGATTTGTACGCGACACCTTACCGATTTGTACGCGACTATACTG...
gene 1′
gene 2′
gene 1″
gene 2″

Figure 3.3: An example of a multiple gene duplication. Note that more than one gene is duplicated at once.

Chapter 4

Mathematical Models of Contradictory Trees

In this chapter, we give a short overview of the most famous approaches in the literature for computing a tree describing common properties of a given set of contradictory trees over a leafset L. Three different types of methods can be found in the literature: consensus methods, agreement methods, and the duplication-loss approach. Consensus trees (Section 4.1) as well as agreement trees (Section 4.2) preserve mathematical properties of trees but have little intuitive justification in a biological sense. In mathematical terms, a *consensus tree* is a leaf-labeled tree, possibly an rl-tree. Since this thesis focuses on rooted trees only, we ignore the unrooted versions of agreement and consensus trees and refer to the cited articles. As an example Figure 4.1 depicts different consensus and agreement trees for a given set of input trees. The duplication-loss approach in detail is discussed in Chapters 5, 6, 7, and 8.

4.1 Consensus Trees

A *consensus tree* $T = (V, E, L)$ is a leaf-labeled tree over leafset L, built from a set of binary p-trees (all over leafset L). In this section, we give a short overview of the main concepts of consensus trees in the literature. The *strict consensus tree* [54, 70, 72] is a consensus tree containing all the internal vertices inducing subtrees whose leafsets agree with a leafset of an induced subtree of each of the input trees. Obviously, the strict

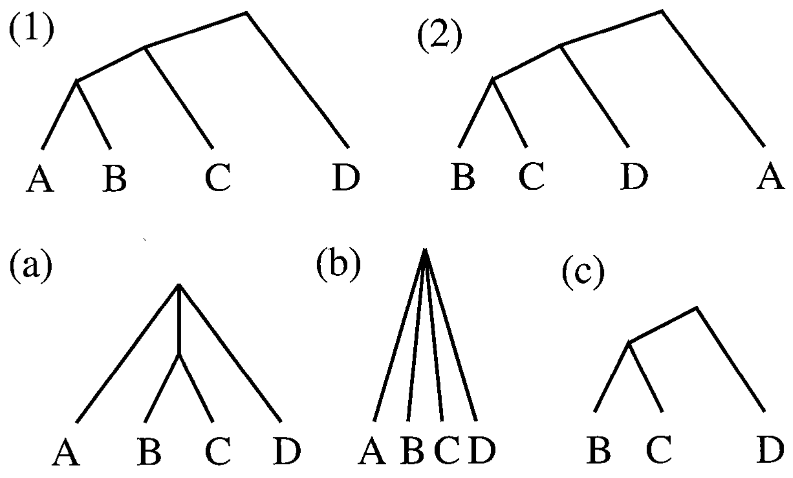


Figure 4.1: (1) and (2) are two gene trees. (a) is the Adams consensus tree (1) and (2); (b) is the strict consensus tree of (1) and (2), and (c) the maximum agreement subtree (1) and (2).

consensus tree is not very informative and therefore various extensions exist (cf. Figure 4.1 (b)).

The most popular consensus tree is the *Adams consensus tree* [1, 2, 53, 54, 70, 72]. It can be viewed as a refinement of the strict consensus tree. The Adams consensus tree T, for a given set of input trees, is built starting at *root*(T). Each child of *root*(T) is labeled with a nonempty set I. I is the intersection of the leafsets of the subtrees induced by one child each of the roots of the input trees. The procedure is applied for every newly created vertex v for T labeled by a set consisting of at least two elements; the input trees are restricted to the leafset of v (cf. Figure 4.1 (a)).

The *majority-rule tree* [7, 53, 54, 70] and the *median consensus tree* [7, 61] are related concepts. While the latter is based on a distance measure between the trees, the majority-rule tree contains exactly the leafsets of the by internal vertices induced subtrees that have the property that these leafsets are contained in more than half the input trees. Various generalizations of the majority-rule tree exist [5, 6]. In general, determining binary median trees is $\mathcal{NP}$-hard [55], but for two trees the problem is solvable in polynomial time [10].

In contrast to the strict consensus methods, in the *cluster-height consensus tree* there is a height assigned to each vertex of the set of input trees assigned. The height increases or decreases monotonically with respect to set inclusion [57, 68, 69].

The *clique consensus tree* was first suggested by Swofford [70]. The idea is to apply a weighting on frequency or edge weightings on all of the clusters of the input trees and find a maximum clique when defining the consensus trees. Bryant introduced the *maximum edge-weight consensus tree* [10]. The tree is defined to be the maximum weighted clique of the set of all leafsets of the input trees, each weighted by the sum of the edge weights of the corresponding edges in the weighted input tree. It is interesting to note that several properties of the maximum weight clique methods lead to consensus trees mentioned above, specifically the strict consensus tree, the majority-rule tree, or the loose consensus tree [10].

4.2 Agreement Trees

An agreement tree of a set of trees is built over a subset of the maximum leafset containing information common to the given trees. These trees do not necessarily contain the full leafset.

The most famous agreement tree is the *maximum agreement subtree* (MAST). This was introduced by Swofford in 1991 (cf. Figure 4.1(c))[70]. The MAXIMUM AGREEMENT SUBTREE problem is stated as follows.

Problem 4.1. MAXIMUM AGREEMENT SUBTREE
Input: p-trees $T_1, ..., T_k$ over leafset L and a positive integer m.
Question: Is there a tree T of size $|T| \geq m$ leaves, with $T \leq_{top} T_i$ for $i = 1, ..., k$?

When the input trees are bounded by degree d, the problem is solvable in polynomial time. In 1995, Farach, Przytycka, and Thorup have shown an $O(k^3|L| + |L|^d)$ time algorithm [23]. In 1997, Bryant presented a simpler version with the same time complexity [10]. The algorithm by Przytycka has a running time of $O(k^3|L|+k^d)$ [62]. A generalized version of the MAST problem where leaf labels are allowed to be repeated is $\mathcal{NP}$-complete for binary trees [25]; the MAST problem for unbounded degree is shown to be $\mathcal{NP}$-complete for $k \geq 3$ [3]. For two trees polynomial time algorithms were developed independently by Steel and Warnow [65], and Goddard *et al.* [33]. Furthermore, Hein *et al.* have shown that the MAST problem for three trees with unbounded degree cannot be approximated in polynomial time within a ratio of $2^{\log^{\delta} n}$ for any $\delta < 1$ (unless $\mathcal{P} = \mathcal{NP}$) [41].

Chapter 5

Models for Counting Evolutionary Events

Instead of looking for a common consensus or agreement for a set of (possibly contradictory) gene trees, Goodman *et al.* suggested in 1979 the DUPLICATION-AND-LOSS MODEL [36]. For a given gene tree and species tree, the model explains the differences in the topology with the minimum number of gene-duplication events and gene losses necessary to rectify the gene tree with respect to the species tree [36]. The model has been discussed by Page [59], Guigo *et al.* [38], Mirkin *et al.* [56], Zhang [73], and Ma *et al.* [52].

Before stating the DUPLICATION-AND-LOSS MODEL, we introduce a general concept for models that rectify a gene tree with respect to a species tree via speciation events and gene-duplication events (Section 5.1). We then describe the DUPLICATION-AND-LOSS MODEL (Section 5.2). Furthermore, we introduce two variants of this model: the GENE-DUPLICATION MODEL (Section 5.3) and the MULTIPLE-GENE-DUPLICATION MODEL (Section 5.4). The GENE-DUPLICATION MODEL, a restriction of the DUPLICATION-AND-LOSS MODEL, is the basis for Chapter 6 (published in [25], which is joint work with Fellows, Hallett, and Korostensky) and Chapter 7 (published in [66]). The MULTIPLE-GENE-DUPLICATION MODEL, which was suggested in [38] and formalized in [26] (joint work with Fellows and Hallett), is the basis for Chapter 8 and [26].

If not noted differently, all trees in the rest of this part are leaf labeled; gene trees and species trees are binary leaf-labeled trees.

5.1 Modeling the History of a Gene Tree

Let a gene tree G and a species tree S be denoted by $G = (V_G, E_G, L_G)$ and $S = (V_S, E_S, L_S)$.

Every model we describe in this chapter (cf. Section 5.2–5.4) uses three functions: a *location function* $loc_{G,S} : V_G \to V_S$, the *event function* $event_{G,S} : V_G \to \{dup, spec\}$, and a cost function *cost*. To a given gene tree and a given species tree, the cost function assigns a value reflecting the number of evolutionary events happening in the history of the gene tree with respect to the species tree. $loc_{G,S}$ is a *tree-stable* function associating each vertex in G with a vertex in S.

Definition 5.1. Let $G = (V_G, E_G, L_G)$ be a gene tree, $S = (V_S, E_S, L_S)$ be a species tree, $L_G \subseteq L_S$, and let $l : V_G \to V_S$ be a function. We call l *tree stable*, if for all $u, v \in V_G$ the following holds. $l(v) = v$ for all $v \in L_G$ and if v is an ancestor of u in G then $l(v)$ is an ancestor of $l(u)$ in S.

Note that, because $loc_{G,S}$ is tree stable, for $L_G = L_S$ $loc_{G,S}$ always maps the root of the gene tree to the root of the species tree: $loc_{G,S}(root(G)) = root(S)$. Furthermore, if $loc_{G,S}(w) = root(S)$ for a vertex $w \in V_G$, then $loc_{G,S} = root(S)$ for all ancestors of w in G.

Finally, $event_{G,S}$ indicates whether the event in G corresponds to a gene-duplication event or a speciation event. More precisely, we define $event_{G,S}$ as follows.

For each $u \in V_G - L_G$,

$$event_{G,S}(u) = \begin{cases} spec & \text{if } loc_{G,S}(u') \neq loc_{G,S}(u), \text{for all } u' \text{ where } u' \text{ is a} \\ & \text{child of } u \text{ in } G. \\ dup & \text{otherwise} \end{cases}$$

Thus, each vertex in the gene tree G is associated with either a gene-duplication event or with a speciation event; the event is located at a vertex in the species tree S.

As a property of such a model we remark that for a vertex $w \in V_G$ with $loc_{G,S}(w) = u$ ($u \in V_S$), there is no ancestor v of w in G such that $loc_{G,S}(v) = u$ and $event_{G,S}(v) = spec$.

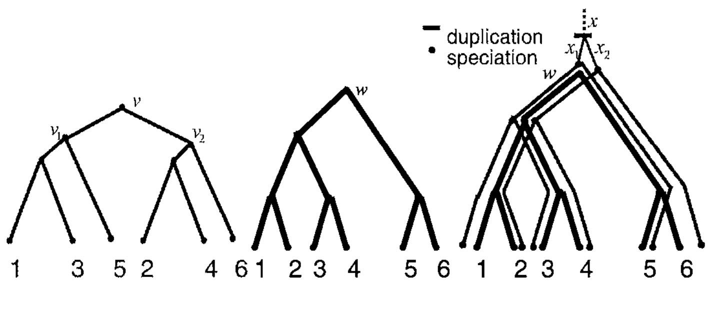
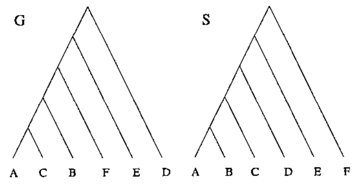


Figure 5.1: A gene tree (red tree, left) is depicted representing the evolutionary tree of the successors of a gene x. The leaf labels indicate the species the gene-family members are located in. The species tree is the black tree, shown in the middle figure. The right figure shows the gene tree mapped into the species tree. Thus, the evolution of gene x we interpret as follows. The gene x was contained in the species w which is the ancestor of 1, 2, 3, 4, 5, and 6. w's latest moment before speciating is represented by the root of the species tree. Before the mentioned speciation event takes place, gene x is copied inside of w and therefore, when speciating, the children of the root of the species tree contain 2 copies (i.e., x_1 and x_2) each. The evolution of the rest of the gene family coincides with the speciation events.

Intuitively, an internal vertex v of the gene tree indicates a gene-duplication event happening before vertex w in the species tree, if v is located at w (i.e., $loc_{G,S}(v) = w$) and at least one of v's children, say v', is also located at w (i.e., there exist a vertex $v' \in V_G$, $parent_G(v') = v$, such that $loc_{G,S}(v') = v$). Thus, the gene existing in the parent of w, that represents an ancestor of gene v, has at least two copies in the species that is represented by w (cf. Figure 5.1).

In general, the whole history of the gene tree $G = (V_G, E_G, L)$ for a given species tree $S = (V_S, E_S, L)$ (corresponding to speciation events and duplication events during the evolution of a gene family) can be viewed by the *explanation tree* $\mathcal{E} = (V_{\mathcal{E}}, E_{\mathcal{E}}, L)$ which we define next. The explanation tree for a given gene tree and a species tree is a tree which is leaf-labeled-isomorphic to the gene tree (cf. definition on page 11 in Chapter 2), representing the evolutionary events of the gene family that happened along the evolution of the species tree. An example of an explanation tree for a given gene tree and species tree is shown in Figure 5.2.



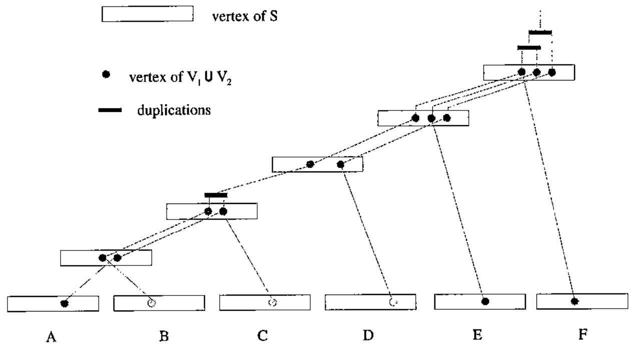


Figure 5.2: The lower figure shows an explanation tree for gene tree G with respect to species tree S depicted in the upper figure. The duplications (red boxes) are vertices in V_3, the red vertices of degree 2 that do not have duplications as parents are vertices in V_2, all the other red vertices are elements of V_1.

In order to define the explanation tree $\mathcal{E} = (V_{\mathcal{E}}, E_{\mathcal{E}}, L)$ we first introduce the set V^*. Due to technical reasons we introduce two bijective functions $f_1 : V_S \times V_G \to \Sigma_1$ and $f_2 : V_S \times V_G \to \Sigma_2$, with $\Sigma_1 \cap \Sigma_2 = \emptyset$, and $|\Sigma_1| = |\Sigma_2| = |V_S \times V_G|$. Then $V^* = \{x | \exists u, w \text{ with } (u,w) \in V_S \times V_G \text{ and } f_1(u,w) = x\} \cup \{y | \exists u, w \text{ with } (u,w) \in V_S \times V_G \text{ and } f_2(u,w) = y\}$.

We define the vertex set $V_\mathcal{E} \subseteq V^* \cup L$ of the explanation tree $\mathcal{E} = (V_\mathcal{E}, E_\mathcal{E}, L)$. $V_\mathcal{E}$ is composed of the given leafset L and the following sets $V_1, V_2, V_3 \subseteq V^*$.

- Each internal vertex $w \in V_G$ with $loc_{G,S}(w) = u$ is represented as the element $f_1(u,w) \in V^*$ in the set V_1, if not both of the children of w imply a duplication event in S and if w_l and w_r are also located at u:

  $V_1 = \{f_1(u,w) \in V^* | u = loc_{G,S}(w), w \in (V_G - L), u \in V_S, \text{ and not } (loc_{G,S}(w_l) = loc_{G,S}(w_r) = u \text{ and } event_{G,S}(w_l) = event_{G,S}(w_r) = dup)\}$.

- For each vertex $v \in V_G - \{root(G)\}$ and its parent $w = parent_G(v)$, each internal vertex $u \in V_S$ in the path $p(loc_{G,S}(v), loc_{G,S}(w))$ in the species tree, excluding the vertices $loc_{G,S}(w)$ and $loc_{G,S}(v)$, is represented as the element $f_1(u,w)$ in V_2:

  $V_2 = \bigcup_{v \in V_G - \{root(G)\}} \text{int}(v)$, where

$$\begin{aligned} \text{int}(v) = \{ & f_1(u,w) \in V^* | \exists u, w \text{ with } (u,w) \in V_S \times V_G, \\ & u \neq loc_{G,S}(v), u \neq loc_{G,S}(w), w = parent_G(v), \text{and} \\ & u \in p(loc_{G,S}(v), loc_{G,S}(w)\}. \end{aligned}$$

- Furthermore, each vertex $w \in V_G$, $loc_{G,S}(w) = u$, which is a duplication event in S (i.e., $event(w) = dup$), is represented as the vertex $f_2(u,w)$ in V_3:

$$V_3 = \{f_2(u,w) \in V^* | loc_{G,S}(w) = u \text{ and } event(w) = dup\}.$$

Note, that for a vertex $w \in V_G$ with $event(w) = dup$ and $loc(w) = u$ it is possible, that both $f_1(u,w)$ and $f_2(u,w)$ are elements of $V_\mathcal{E}$, namely $f_1(u,w) \in V_1$ and $f_2(u,w) \in V_3$.

We remark, that each vertex $w \in V_G$ has at least one corresponding vertex in $V_\mathcal{E}$ (i.e., there is a vertex $u \in V_S$ with $f_1(u,w) \in V_\mathcal{E}$ or $f_2(u,w) \in V_\mathcal{E}$). Furthermore, V_1, V_2, and V_3 are pairwise disjoint. (Obviously, $V_1 \cap V_3 = \emptyset$ and $V_2 \cap V_3 = \emptyset$. Assume $V_1 \cap V_2 \neq \emptyset$. That is, there is an element $x \in V_1 \cap V_2$, and there are elements $u \in V_S$ and $w \in V_G$ with $x = f_1(u,w)$. Because $x \in V_1$ we know $loc_{G,S}(w) = u$. But because $x \in V_2$ we know $loc_{G,S}(w) \neq u$. Contradiction.)

The edges $E_\mathcal{E}$ of the explanation tree are introduced corresponding to the gene tree G. f_1^{-1} and f_2^{-1} denote the inverse functions of f_1 and f_2.

- Let $v \in L$, $a \in V_1 \cup V_2$, and $f_1^{-1}(a) = (u,w)$. Then $a = parent_\mathcal{E}(v)$ if and only if $w = parent_G(v)$ and $u = parent_S(v)$.
- Let $a,b \in V_1$ and $f_1^{-1}(a) = (u,w)$, $f_1^{-1}(b) = (x,y)$. Then $b = parent_\mathcal{E}(a)$ if and only if $y = parent_G(w)$ and $x = parent_S(u)$.
- Let $a,b \in V_2$ and $f_1^{-1}(a) = (u,w)$, $f_1^{-1}(b) = (x,y)$. Then $b = parent_\mathcal{E}(a)$ if and only if $w = y$ and $x = parent_S(u)$.
- Let $a,b \in V_3$ and $f_2^{-1}(a) = (u,w)$, $f_2^{-1}(b) = (x,y)$. Then $b = parent_\mathcal{E}(a)$ if and only if $y = parent(w)$ and $u = x$.
- Let $a \in V_1$, $b \in V_2$, and $f_1^{-1}(a) = (u,w)$, $f_1^{-1}(b) = (x,y)$. Then $b = parent_\mathcal{E}(a)$ if and only if $y = parent_G(w)$ and $x = parent_S(u)$.
- Let $a \in V_2$, $b \in V_1$, and $f_1^{-1}(a) = (u,w)$, $f_1^{-1}(b) = (x,y)$. Then $b = parent_\mathcal{E}(a)$ if and only if $w = y$ and $x = parent_S(u)$.
- Let $a \in V_1$, $b \in V_3$, and $f_1^{-1}(a) = (u,w)$, $f_2^{-1}(b) = (x,y)$. Then $b = parent_\mathcal{E}(a)$ if and only if either ($y = parent_G(w)$ and $u = x$) or ($y = parent_G(w)$ and $x = parent_S(u)$).
- Let $a \in V_3$, $b \in V_1$ and $f_2^{-1}(a) = (u,w)$, $f_1^{-1}(b) = (x,y)$. Then $b = parent_\mathcal{E}(a)$ if and only if $y = parent_G(w)$ and $x = parent_S(u)$.
- Let $a \in V_2$, $b \in V_3$, and $f_1^{-1}(a) = (u,w)$, $f_2^{-1}(b) = (x,y)$. Then $b = parent_\mathcal{E}(a)$ if and only if $w = y$ and $x = parent_S(u)$.
- Let $a \in V_3$, $b \in V_2$, and $f_2^{-1}(a) = (u,w)$, $f_1^{-1}(b) = (x,y)$. Then $b = parent_\mathcal{E}(a)$ if and only if $y = parent_G(w)$ and $x = parent_S(u)$.

Lemma 5.1. *Let $\mathcal{E} = (V_\mathcal{E}, E_\mathcal{E}, L)$ be the explanation tree for a gene tree G, a species tree S, a tree-stable location function $loc_{G,S}$, and the event function $event_{G,S}$. Then*

1. *$\mathcal{E} = (V_\mathcal{E}, E_\mathcal{E}, L)$ is a leaf-labeled rooted tree.*
2. *$G \cong \mathcal{E}$ (i.e., G is leaf-labeled isomorphic to $\mathcal{E}$).*

Proof. 1. We show

(a) every vertex in $V_\mathcal{E}$ except one has exactly one parent in $\mathcal{E}$.
(b) the elements in L are leaves (i.e., no element in L has a child).
(c) there is exactly one element $a \in V_\mathcal{E}$ s.t. either

$$f_1^{-1}(a) = (root(S), root(G)) \text{ or}$$

$$f_2^{-1}(a) = (root(S), root(G)).$$

Furthermore a has no parent in $\mathcal{E}$.

These properties are sufficient to show that $\mathcal{E}$ is a tree. Because of (a) and (b), $\mathcal{E}$ is connected, contains no cycles, and has leaves L. Property (c) shows that $\mathcal{E}$ is rooted.

The verifications of the claims above are as follows.

(a) We first show, that every leaf in $V_\mathcal{E}$ has exactly one parent in $\mathcal{E}$. Let $v \in L$, $w = parent_G(v)$, and $loc_{G,S}(w) = u$, then $event_{G,S}(w) = spec$. (This follows directly from the definition of the event function, since leaves of the gene tree are always located at leaves of the species trees and internal vertices of the gene tree are always located at internal vertices of the species tree, and therefore a leaf of the gene tree is never located at the same vertex in the species tree as its parent.) Therefore, there exists an element $a \in V_1 \cup V_2$ with $f_1^{-1}(a) = (u, w)$, and, due to the definition of $E_\mathcal{E}$, $a = parent_\mathcal{E}(v)$.

Since both the species tree and the gene tree are well-defined trees, v has exactly one parent in G and in S. Therefore, the existence of $a \in V_1 \cup V_2$ (with $f_1^{-1}(a) = (u, w)$, $w = parent_G(v)$, and $u = parent_S(v)$) is unique.

Let a be an internal vertex of $V_\mathcal{E}$ (i.e., $a \notin L$). Furthermore, we assume that $f_1^{-1}(a) \neq (root(S), root(G))$ and $f_2^{-1}(a) \neq (root(S), root(G))$.

Let $u \in V_S$, $w \in V_G$ such that $a = f_1(u,w)$ or $f_2(u,w)$. Then we can assume that $w \neq \mathit{root}(G)$. (Assume $w = \mathit{root}(G)$. Then $u \neq \mathit{root}(S)$. But this is a contradiction to the definition of G and S, since G and S have the same leafset. Therefore the root of the gene tree is located at the root of the species tree.)

We now show that a has a parent b in $\mathcal{E}$ and that b is uniquely determined.

- Let $a \in V_1$ with $f_1^{-1}(a) = (u,w)$. Then $\mathit{loc}_{G,S}(w) = u$. We distinguish the following cases:
  i. $u \neq \mathit{root}(S)$ and $w \neq \mathit{root}(G)$.
     W.l.o.g. let $u^* = \mathit{parent}_S(u)$ and $w^* = \mathit{parent}_G(w)$. Then there is an element $b \in V_{\mathcal{E}}$ such that either $b = f_1(u^*,w^*)$, $b = f_2(u^*,w^*)$, or $b = f_2(u,w^*)$. (Assume there is no such element b. That is, w^* is neither located at u nor at u^*: $\mathit{loc}_{G,S}(w^*) \neq u$ (i.e., $\mathit{event}_{G,S}(w^*) \neq \mathit{dup}$) and $\mathit{loc}_{G,S}(w^*) \neq u^*$. Furthermore $u^* \notin p(u, \mathit{loc}_{G,S}(w^*))$. But this is a contradiction to the property, that the location function is tree stable.)
  ii. $u = \mathit{root}(S)$. Then $w \neq \mathit{root}(G)$.
     W.l.o.g. $w^* = \mathit{parent}_G(w)$. Then there is an element $b \in V_{\mathcal{E}}$ such that $b = f_2(u,w^*)$. Element b exists, because all the ancestors of w in G are all located at the root in the species trees and therefore all of them are duplication events. Thus, $b \in V_3$ and $b = \mathit{parent}_{\mathcal{E}}(a)$.

  If $b = f_1(u^*,w^*)$, then $u^* = \mathit{loc}_{G,S}(w^*)$ (i.e., $b \in V_1$. Or $u^* \in p(u, \mathit{loc}_{G,S}(w^*))$). But then $b \in V_2$. Therefore in this case $b = \mathit{parent}_{\mathcal{E}}(v)$.

  If $b = f_2(u,w^*)$ or $b = f_2(u^*,w^*)$, then $b \in V_3$ and therefore $b = \mathit{parent}_{\mathcal{E}}(v)$.

  We now show that b is uniquely determined. Assume there exist a vertex $b' \in V_{\mathcal{E}}$, $b' \neq b$, and $b' = \mathit{parent}_{\mathcal{E}}(v)$.
  - If $b' \in V_1$ then there exist $x \in V_S$, $y \in V_G$ with $b' = f_1(x,y)$ and $x = \mathit{parent}_S(u)$, $y = \mathit{parent}_G(w)$. But then $b' = b$. Contradiction.
  - If $b' \in V_2$ then there exist $x \in V_S$, $y \in V_G$ with $b' = f_1(x,y)$ and $x = \mathit{parent}_S(u)$, $y \in p(u, \mathit{loc}_{G,S}(w^*))$.

But then $b' = b$. Contradiction.

- If $b' \in V_3$ then there exist $x \in V_S$, $y \in V_G$ with $b' = f_2(x,y)$, $y = parent_G(w)$ and either $x = u$ or $x = parent_S(u)$. But then $b' = b$. Contradiction.

- Let $v \in V_2$ with $v = f_1(u,w)$. Again, we distinguish the following cases:
  i. $u \neq root(S)$ and $w \neq root(G)$.
  Let $u^* = parent_S(u)$, $w^* = parent_G(w)$, and $u' = loc_{G,S}(w)$. Then $u^* \in p(u,u')$ and there is an element $b \in V_{\mathcal{E}}$ such that $b = f_1(u^*,w^*)$. If $u^* = u'$ then $b \in V_1$, else $b \in V_2$.
  We show b is uniquely determined.
  Assume there exist a vertex $b' \in V_{\mathcal{E}}$, $b' \neq b$, and $b' = parent_{\mathcal{E}}(v)$. But this is a contradiction to the definition of species tree and gene tree.
  ii. $u = root(S)$. Then $w \neq root(G)$. But then $a \notin V_2$. Contradiction.

- Let be $v \in V_3$ and $v = f_2(u,w)$.
  As before, we distinguish the following cases:
  i. $u \neq root(S)$ and $w \neq root(G)$.
  W.l.o.g. $w^* = parent_G(w)$. Then there is an element $b \in V_{\mathcal{E}}$ such that $b = f_2(u,w^*)$. Element b exists, because all the ancestors of w in G are all located at the root in the species trees and therefore all of them are duplication events. Thus, $b \in V_3$ and $b = parent_{\mathcal{E}}(a)$.
  Let $u^* = parent_S(u)$ and $w^* = parent_G(w)$. Then $loc_{G,S}(w) = u$ and there is an element $b \in V_{\mathcal{E}}$ such that either $b = f_1(u^*,w^*)$ or $b = f_2(u,w^*)$.
  ii. $u = root(S)$. Then $w \neq root(G)$. Since w is located at the root of the species tree all the ancestors of w in G are located at $root(S)$ as well. Then there exist $b \in V_3$ with $b = f_2(u,w^*)$.

  If $b = f_1(u^*,w^*)$ then $b \in V_1 \cup V_2$, else if $b = f_2(u,w^*)$ then $b \in V_3$.
  We show b is uniquely determined.
  Assume there exist a vertex $b' \in V_{\mathcal{E}}$, $b' \neq b$, and $b' = parent_{\mathcal{E}}(v)$.

   - If $b' \in V_1$ then there exist $x \in V_S$, $y \in V_G$ with $b' = f_1(x,y)$ and $x = \mathit{parent}_S(u)$, $y = \mathit{parent}_G(w)$. But then $b' = b$. Contradiction.
   - If $b' \in V_2$ then there exist $x \in V_S$, $y \in V_G$ with $b' = f_1(x,y)$ and $x = \mathit{parent}_S(u)$, $y \in p(u, \mathit{loc}_{G,S}(w^*))$. But then $b' = b$. Contradiction.
   - If $b' \in V_3$ then there exist $x \in V_S$, $y \in V_G$ with $b' = f_2(x,y)$, $y = \mathit{parent}_G(w)$ and either $x = u$ or $x = \mathit{parent}_S(u)$. But then $b' = b$. Contradiction.

   (b) The claim follows directly from the definition of the set of edges $E_{\mathcal{E}}$.

   (c) We consider the cases where $f_1^{-1}(a) = (\mathit{root}(S), \mathit{root}(G))$ or $f_2^{-1}(a) = (\mathit{root}(S), \mathit{root}(G))$.
   Because $\mathit{loc}_{G,S}(\mathit{root}(G)) = \mathit{root}(S)$, such an element a always exists in $V_{\mathcal{E}}$. Furthermore, because then neither u nor w has an ancestor in its corresponding tree, no parent for a in $\mathcal{E}$ is defined. To see that it is impossible that $f_1(\mathit{root}(S), \mathit{root}(G)) \in V_{\mathcal{E}}$ and $f_2(\mathit{root}(S), \mathit{root}(G)) \in V_{\mathcal{E}}$, we assume $f_1(\mathit{root}(S), \mathit{root}(G)) \in V_{\mathcal{E}}$. Then $f_1(\mathit{root}(S), \mathit{root}(G)) \in V_1$. But this means, there cannot be a child of $\mathit{root}(G)$ which is located at the root of S, the necessary condition for the existence of $f_2(\mathit{root}(G), \mathit{root}(S))$ in $V_{\mathcal{E}}$.

2. Because for each vertex $v \in V_G$ $(\mathit{loc}_{G,S}(v), v)$ is ancestor in $\mathcal{E}$ of each element in $LS(v)$. That is, $LS(v) \subseteq LS((\mathit{loc}_{G,S}(v), v))$. Therefore $G \leq_{top} S$, which proves the claim.

□

This section introduced the explanation tree, which is a general concept for viewing the history of a gene tree G with respect to a species tree S. A cost-model taking into account the event function $\mathit{event}_{G,S}$ and a tree-stable location function determines the explanation tree. The following sections define three specific models to rectify a set of (contradictory) gene trees with respect to a species tree.

5.2 The Duplication-and-Loss Model

We describe the DUPLICATION-AND-LOSS MODEL introduced in [36]. Let $G = (V_G, E_G, L)$ be a gene tree and let $S = (V_S, E_S, L)$ be a species tree. The location function $loc_{G,S} : V_G \to V_S$, is defined by the least-common-ancestor mapping, i.e., $loc_{G,S}(u) = lca_S(LS(u))$ for all $u \in V_G$. Obviously, $loc_{G,S}$ is tree stable. The cost function is defined by

$$\begin{aligned} cost_{DL}(G,S) &= |\{u|u \in V_G - L, event_{G,S}(u) = dup\}| \\ &+ \sum_{u \in V_G - L} (|\mathrm{p}(loc_{G,S}(u_l), loc_{G,S}(u))| \\ &+ |\mathrm{p}(loc_{G,S}(u_r), loc_{G,S}(u))| - 2)\,, \end{aligned}$$

where

$$\sum_{u \in V_G - L} (|\mathrm{p}(loc_{G,S}(u_l), loc_{G,S}(u))| + |\mathrm{p}(loc_{G,S}(u_r), loc_{G,S}(u))| - 2).$$

defines the number of losses.

The location function, the least-common-ancestor mapping, implies that $cost_{DL}(G,S)$ is the minimum number of gene-duplication events and gene losses necessary to rectify the gene tree G with the species tree S.

It is not very difficult to verify that for a gene tree G and species tree S $cost_{DL}(G,S)$ can be computed in linear time [73], since computing the least common ancestor is possible in linear time [40, 64].

Figure 5.3 (a) and (b), page 42, show the explanation trees for two gene trees and a species tree under the DUPLICATION-AND-LOSS MODEL. The question which is implied by the DUPLICATION-AND-LOSS MODEL is stated as follows.

Problem 5.1. DUPLICATION AND LOSS
Input: Gene trees $G_1, \ldots, G_k$ over leafset L.
Output: A species tree S with minimal cost (i.e., $\sum_{i=1}^{k} cost_{DL}(G_i, S)$ is minimized).

Recently, Ma, Li and Zhang have shown that DUPLICATION AND LOSS is $\mathcal{NP}$-complete [52].

5.3 The Gene-Duplication Model

The GENE-DUPLICATION MODEL [26, 66, 73] is the same as the DUPLICATION-AND-LOSS MODEL, but restricted to gene-duplication events only. Gene losses are not considered in this model. Using the same location function as in Section 5.2, the minimum number of gene-duplication events necessary to rectify a gene tree G with a species tree S is defined by $cost_{GD}(G, S) = |Dups_{GD}|$,

$$Dups_{GD} = \{u | u \in V_G - L, event_{G,S}(u) = dup\}.$$

As in the case of the cost function in the DUPLICATION-AND-LOSS MODEL, for a gene tree G and a species tree S, $cost_{GD}(G, S)$ can be computed in linear time. Similar to DUPLICATION AND LOSS we state the GENE DUPLICATION problem as follows:

Problem 5.2. GENE DUPLICATION
Input: Gene trees $G_1, \dots, G_k$ over leafset L.
Output: A species tree S with minimal cost, i.e., $\sum_{i=1}^{k} cost_{GD}(G_i, S)$ is minimized.

As for the DUPLICATION AND LOSS problem, GENE DUPLICATION was also shown to be $\mathcal{NP}$-complete [52]. In Section 6 and Section 7 we investigate GENE DUPLICATION in more details.

5.4 The Multiple-Gene-Duplication Model

In Section 5.2 and Section 5.3, gene-duplication events are considered to be independent events. The MULTIPLE-GENE-DUPLICATION MODEL, which we formalized in [26], takes into account that a duplication event happening on the nucleotide level, can involve more than one gene at once and motivates the definition of a *multiple gene duplication.* The idea of clustering gene-duplication events was suggested by Guigo *et al.* [38].

Suppose we are given the gene trees $G_1, \dots, G_k$ and the species tree S. Consider a vertex u in S. In any model following the concept suggested in Section 5.1, each gene tree G_i has some number of vertices d (possibly zero) with $loc_{G_i,S}(d) = u$ and $event_{G_i,S}(d) = dup$ $(1 \leq i \leq k)$. Let $Dup(u) = \{d_1, d_2, \dots, d_c\}$ denote this set. We can partition $Dup(u)$ into classes with the property that each class has at most one vertex from each gene tree G_i and so that these sets are maximal. One such set is termed a *multiple gene duplication* and it counts exactly one to the overall number of multiple gene duplications required to rectify the gene trees with respect to the species tree. The multiple-gene-duplication score for the vertex u is the total number of such partitions. By "moving" gene duplication events in G_i towards the root of S according to a set of rules, we can decrease the total number of multiple gene duplications required (as illustrated on page 40). We define the cost function to be $cost_{MG}(G_1, \dots, G_k, S) =$

$$\sum_{u \in V_S} \max_{i=1}^{k} \left|\{v | v = loc^{-1}_{G_i,S}(u) \text{ and } event(v) = dup\}\right|.$$

Let $G = (V_G, E_G, L)$ be a gene tree and let $S = (V_S, E_S, L)$ be a species tree. What does "moving a gene duplication towards the root of S" mean? For a vertex $u \in V_G$ with $event_{G,S}(u) = dup$ we simply change the location $loc_{G,S}(u)$ of a vertex u to the location of its parent $v = parent_G(u)$ (i.e., after each move the location function is redefined with $loc_{G,S}(u) := loc_{G,S}(v)$). Now $event_{G,S}(v) = dup$.

Consider a vertex $u \in V_G$ such that $event_{G,S}(u) = dup$ and $u \neq root(G)$, and for all $x \in V_G$, where $x \neq u$ and x is ancestor of u in G, $loc_{G,S}(x) \neq loc_{G,S}(u)$. Let $v = parent_G(u)$. The rules for moving duplication events towards the root of a species tree are specified as follows:

Move 1: If $event_{G,S}(v) = dup$, we may move the duplication associated with u from $loc_{G,S}(u)$ to $loc_{G,S}(v)$. Now $loc_{G,S}(u) = loc_{G,S}(v)$.

Move 2: If $event_{G,S}(v) = spec$, when moving the duplication associated with u from $loc_{G,S}(u)$ to $loc_{G,S}(v)$, we must change $event_{G,S}(v)$ to be dup. Now $loc_{G,S}(u) = loc_{G,S}(v)$.

Note, that after applying Move 2 the number of duplication events is increased by one. Note that both Move 1 and Move 2 preserve the tree stability of the new location function.

Let us reconsider the GENE-DUPLICATION MODEL. For a gene tree G and a species tree S, $Dups_{GD}(G, S)$ is the set of vertices in G having gene-duplication events in S under the GENE-DUPLICATION MODEL; as shown earlier, the number of gene duplications is minimized over all possible tree-stable location functions. Furthermore, the locations $loc_{G,S}(d)$ for the vertices $d \in Dups_{GD}(G, S)$ are the "latest" possibilities where the events can take place (i.e., the vertices in $Dups_{GD}(G, S)$

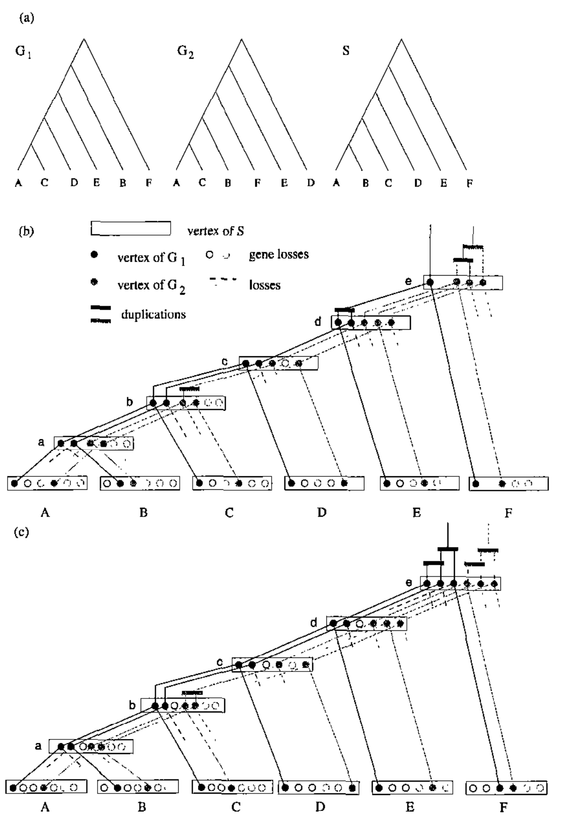


Figure 5.3: (a) depicts two gene trees, G_1 and G_2, and a proposed species tree S. (b) shows that the species tree S has the explanation trees for G_1 and G_2 embedded inside of it according to the standard DUPLICATION AND LOSS model. Note that G_1 causes one duplication (vertex d) whilst G_2 causes 3 gene duplications (vertex b and twice vertex e). The score according to the DUPLICATION AND LOSS model is 4 duplications and 15 losses; the score according to the GENE DUPLICATION model and to the MULTIPLE-GENE-DUPLICATION model is 4. (c) After moving the gene duplication of G_1 located at vertex d to the root e of the species tree S, two additional gene duplications for G_1 need to be postulated. Nevertheless, the score according to the MULTIPLE GENE DUPLICATION model is now 3 (even though we count 5 duplications). Note that it is not beneficial to move the gene duplication from G_2 located at c towards the root. The two gene duplications located initially at the root from G_2 cannot move upwards.

cannot be located closer to the leaves of S). Certainly, if we do not care about the minimum in the number of gene-duplication events, gene-duplication events can happen "earlier" in the history of a genc family than postulated in the GENE-DUPLICATION MODEL (i.e., a duplication event can be located closer to the root of the species tree), and also more gene-duplication events may be necessary.

Observation 5.2. *Given the gene trees $G_1, G_2, \ldots, G_k$ and a species tree S, $\sum_{i=1}^{k} cost_{GD}(G_i, S)$ provides an upper bound for the number of multiple gene duplications.*

Definition 5.2. Given a gene tree G, a species tree S, and the functions $loc_{G,S}$ and $event_{G,S}$ mapping G into S, we say that S *receives* G, if the configuration given by $loc_{G,S}$ and $event_{G,S}$ can be reached by a series of moves (Move 1 and 2 on page 41 starting from the initial configuration obtained by applying the location function $loc_{G,S}$ and the event function $event_{G,S}$ defined in the GENE-DUPLICATION MODEL (cf. Section 5.2).

Figure 5.3 on page 42 gives a concrete example.

Finally, we state the MULTIPLE GENE DUPLICATION problem as follows:

Problem 5.3. MULTIPLE GENE DUPLICATION
Input: Gene trees $G_1, \ldots, G_k$ over leafset L, integer c.
Question: Does there exist a species tree S and location functions $loc_{G_i,S}$, $1 \leq i \leq k$, such that S receives $G_1, \ldots, G_k$ with at most c multiple gene duplications, i.e., $cost_{MG}(G_1, \ldots, G_k, S) \leq c$?

Chapter 6

The Smallest-Common-Supertree Problem

This chapter, as well as Chapter 7, focuses on the GENE DUPLICATION problem. We present the problem SMALLEST COMMON SUPERTREE which has an interesting relation to GENE DUPLICATION. The smallest common supertree of a set of gene trees gives us a lower bound for the number of gene-duplication events necessary to rectify the gene trees with respect to a species tree. Section 6.1 motivates and introduces the problem; Section 6.2 analyzes the parameterized complexity of SMALLEST COMMON SUPERTREE.

6.1 Smallest Common Supertree – Problem Statement and Motivation

Before introducing *Smallest Common Supertree* we begin with the following definition.

Definition 6.1. A *common supertree* $T = (V, E)$ of a given set of gene trees $G_1, \ldots, G_k$ is a binary rl-tree with $G_i \leq_{top} T$ $(i = 1, .., k)$. A vertex $v \in V$ is a *duplication vertex* of T if $LS(v_l) \cap LS(v_r) \neq \emptyset$.

We formalize the base problem of this chapter.

Problem 6.1. SMALLEST COMMON SUPERTREE
Input: Gene trees $G_1, \dots, G_k$ over leafset L and a positive integer m.
Question: Does there exist a common supertree T such that $|T| \leq m$?

The SMALLEST COMMON SUPERTREE problem is motivated as follows. Given gene trees $G_1, \dots, G_k$ and a species tree S, all over leafset L, then there is an rl-tree S^* which is a common supertree of $G_1, \dots, G_k$, and S^*'s duplication vertices are a lower bound for the number of gene-duplication events necessary to rectify $G_1, \dots, G_k$ with respect to S.

First, we show how to compute S^* for a single gene tree G (Algorithm `Build`S^*) . Then we generalize the algorithm and compute a common supertree for a given set of k gene trees (Algorithm `BuildAll`S^*).

```
Algorithm Build$S^*$(gene tree $G$, species tree $S$): rl-tree $S^*$
  create a copy $S^*$ of $S$
  $S^*$ := CreateRl($G, S, S^*$)
  return($S^*$)
end Bild$S^*$.

procedure CreateRl(gene tree $G$, species tree $S$, rl-tree $S^*$): rl-tree $S^*$
  if $loc_{G,S}(root(G) = root(S)$ and $event_{G,S}(root(G)) = dup$ then
    create a copy $T$ of $S$
    create a new vertex $w$
    $w_l := root(S^*)$; $w_r := root(T)$
    $root(S^*) := w$
    (* $S^*$ is now extended by a new root $w$, which is the parent of  *)
    (* the former root of $S^*$; the other child of $w$ is $T$.          *)
    $S^*(root(S^*)_l)$ := CreateRl($G(root(G)_l), S, S^*(root(S^*)_l)$)
    $S^*(root(S^*)_r)$ := CreateRl($G(root(G)_r), S, S^*(root(S^*)_r)$)
  else
    if $LS(root(G)_r) \subseteq LS(root(S^*)_l)$ then
      swap left and right subtree of $G$
    endif
    $S^*(root(S^*)_l)$ := CreateRl($G(root(G)_l), S(root(S)_l), S^*(root(S^*)_l)$)
    $S^*(root(S^*)_r)$ := CreateRl($G(root(G)_r), S(root(S)_r), S^*(root(S^*)_r)$)
  endif
  (* $S^*(root(S^*)_l)$ and $S^*(root(S^*)_r)$ have been replaced in $S^*$      *)
  (* by the results computed by the CreateRl-procedure calls.        *)
  return($S^*$)
```

end CreateRl.

Lemma 6.1. *Suppose we are given a gene tree G and a species tree S. Let S^* be the rl-tree built with the Algorithm* BildS^*. *Then*

1. $G \leq_{top} S^*$.
2. *the number of duplication vertices of S^* corresponds exactly to the number of gene-duplication events $cost_{GD}(G, S)$.*

Proof.

1. We prove by induction over $cost_{GD}(G, S)$.

   Let $cost_{GD}(G, S) = 0$. Then $G = S$ and therefore $G \leq_{top} S$.

   Assuming the claim holds for $cost_{GD}(G, S) \leq i$, we show, that it also holds for $cost_{GD}(G, S) = i + 1$. Let G be a gene tree and S a species tree s.t. $cost_{GD}(G, S) = i + 1$.

   If $event_{G,S}(root(G)) = dup$, then both $cost_{GD}(G(root(G)_l), S) \leq i$ and $cost_{GD}(G(root(G)_r), S) \leq i$. Furthermore, $G(root(G)_l) \leq_{top} S^*(root(S^*)_l)$ and $G(root(G)_r) \leq_{top} S^*(root(S^*)_r)$ and therefore $G \leq_{top} S^*)$.

   If $event_{G,S}(root(G)) = spec$, then either all vertices $u \in V_G$ with $event_{G,S}(u) = dup$ are in one subtree of the root of G (i.e, either in $G(root(G)_l)$ or in $G(root(G)_r)$) or $cost_{GD}(G(root(G)_l), S) \leq i$ and $cost_{GD}(G(root(G)_r), S) \leq i$. In the latter case, $G(root(G)_l) \leq_{top} S^*(root(S^*)_l)$ and $G(root(G)_r) \leq_{top} S^*(root(S^*)_r)$ and therefore $G \leq_{top} S^*$. If all the vertices $u \in V_G$ with $event_{G,S}(u) = dup$ are in one subtree of the root of G, let $v \in V_G$ be the least common ancestor of all those vertices u. Then $cost_{GD}(G_v) = i + 1$ and either $G(v) \leq_{top} G(root(G)_l)$ or $G(v) \leq_{top} G(root(G)_r)$. W.l.o.g. let $G(v) \leq_{top} G(root(G)_l)$. It is sufficient to show that the tree $T^* :=$ Build$S^*(G(v), S(loc_{G,S}(v)))$ contains $G(v)$ as a subtree, i.e., $G(v) \leq_{top} T^*$. If $event_{G(v),S}(v) = spec$, then neither the left nor the right subtree of $root(G(v))$ has more then i vertices w with $event_{G(v),S}(w) = dup$ (otherwise we have a contraction to the property that v is the least common ancestor in G with $cost_{GD}(G(v), S) = i + 1$). If $event_{G(v),S}(v) = dup$ then neither the left nor the right subtree of $root(G(v))$ has more then i vertices w with $event_{G(v),S}(w) = dup$.

2. Since S has no duplication vertex (S is a p-tree), the only candidates for duplication vertices are those vertices w in S^*, which are

created when calling BuildS^* for each duplication event. Clearly, the number of newly created vertices is exactly $cost_{GD}(G, S)$, and they are all duplication vertices.

□

We generalize Algorithm BuildS^* to Algorithm BuildAllS^*, which computes an rl-tree S^* for k gene trees $G_1, \ldots, G_k$ and a species tree S such that S^* is a common supertree of $G_1, \ldots, G_k$ and the number of duplication vertices of S^* is a lower bound for $\sum_{i=1}^{k} cost_{GD}(G_i, S)$. In the recursive call, we extend the species by a new root vertex and let a copy of the species tree be the other subtree of the root, if at least one of the given gene trees has a gene duplication at the root of S.

```
Algorithm BuildAll$S^*$(gene trees $G_1, \ldots G_k$, species tree $S$):
    rl-tree $S^*$
  create a copy $S^*$ of $S$
  $S^*$ := CreateAllRl($G_1, \ldots, G_k, S, S^*$)
  return($S^*$)
end BildAll$S^*$.

procedure CreateAllRl(gene trees $G_1, \ldots, G_k$, species tree $S$, rl-tree $S^*$):
    rl-tree $S^*$
  if there is at least one gene tree $G \in \{G_1, \ldots, G_k\}$ with
      $loc_{G,S}(root(G)) = root(S)$ and $event_{G,S}(root(G)) = dup$ then
    create a copy $T$ of $S$
    create a new vertex $w$
    $w_l := root(S^*)$; $w_r := root(T)$
    $root(S^*) := w$
    (* $S^*$ is now extended by a new root $w$, which is the parent of  *)
    (* the former root of $S^*$; the other child of $w$ is $T$.          *)
    for each $G \in \{G_1, \ldots, G_k\}$ with $event_{G,S}(root(G)) = spec$ do
      if $LS(root(G)_r) \subseteq LS(root(S)_l)$ then
        swap left and right subtree of $G$
      endif
    endfor
    $S^*(root(S^*)_l)$ := CreateAllRl($G_1(root(G_1)_l), G_2(root(G_2)_l), \ldots,$
      $G_k(root(G_k)_l), S, S^*(root(S^*)_l)$)
    $S^*(root(S^*)_r)$ := CreateAllRl($G_1(root(G_1)_r), G_2(root(G_2)_r), \ldots,$
      $G_k(root(G_k)_r), S, S^*(root(S^*)_r)$)
  else
    for each $G \in \{G_1, \ldots, G_k\}$ with $LS(root(G)_r) \subseteq LS(root(S)_l)$ do
      swap left and right subtree of $G$
    endfor
    $S^*(root(S^*)_l)$ := CreateRl($G_1(root(G_1)_l), G_2(root(G_2)_l), \ldots,$
      $G_k(root(G_k)_l), S(root(S)_l), S^*(root(S^*)_l)$)
    $S^*(root(S^*)_r)$ := CreateRl($G_1(root(G_1)_r), G_2(root(G_2)_r), \ldots,$
      $G_k(root(G_k)_r), S(root(S)_r), S^*(root(S^*)_r)$)
  endif
  (* $S^*(root(S^*)_l)$ and $S^*(root(S^*)_r)$ are replaced in $S^*$ by the     *)
  (* results computed by the CreateAllRl-procedure calls.          *)
  return($S^*$)
end CreateAllRl.
```

Corollary 6.2. *Suppose we are given k gene trees $G_1, \ldots, G_k$ and a species tree S. Let S^* be the rl-tree built with the Algorithm* BuildAllS^*. *Then*

1. *$G_i \leq_{top} S^*$ for $i = 1, \ldots, k$.*
2. *the number of duplication vertices of S^* is a lower bound for the number of gene-duplication events $\sum_{i=1}^{k} cost_{GD}(G_i, S)$.*

A solution of the minimization version of SMALLEST COMMON SUPERTREE is called a *smallest common supertree.* Because of Corollary 6.2, a smallest common supertree gives a lower bound for the number of duplication vertices resulting from the optimal species tree, the solution of the minimization version of the problem GENE DUPLICATION. But this is also a lower bound for the number of gene-duplication events necessary to rectify gene trees with respect to a species tree.

6.2 The Complexity of Smallest Common Supertree

6.2.1 Intractability of Smallest Common Supertree

We show $\mathcal{NP}$-completeness of SMALLEST COMMON SUPERTREE (Problem 6.1) and $\mathcal{W}[1]$-hardness of SMALLEST COMMON SUPERTREE for parameter k (Problem 6.2). The hardness is proven by a reduction from

Shortest Common Supersequence restricted to *p-sequences*[1] (Problem 6.3 and 6.4). p-Sequence Shortest Common Supersequence is known to be $\mathcal{NP}$-complete and $\mathcal{W}[1]$-hard when parameterized by the number of input sequences (cf. Theorem 6.3 and Theorem 6.4, [25, 27]).

Problem 6.2. Smallest Common Supertree (parameterized version)
Input: Gene trees $G_1, \dots, G_k$ over leafset L and a positive integer m.
Parameter: k.
Question: Does there exist a common supertree T such that $|T| \leq m$?

Problem 6.3. p-Sequence Shortest Common Supersequence
Input: p-sequences $x_1, ..., x_k$ over alphabet Σ and a positive integer M
Question: Does there exist an rl-sequence x, with $|x| \leq M$ and x_i is subsequence of x for $i = 1, ..., k$?

Problem 6.4. p-Sequence Shortest Common Supersequence (parameterized version of Problem 6.3)
Input: p-sequences $x_1, ..., x_k$ and a positive integer M
Parameter: k.
Question: Does there exist an rl-sequence x, with $|x| \leq M$ and x_i is subsequence of x for $i = 1, ..., k$?

Theorem 6.3. *[25, 27]* p-Sequence Shortest Common Supersequence *is hard for $\mathcal{W}[1]$ parameterized by k and $\mathcal{NP}$-complete.*

Theorem 6.4. Shortest Common Supertree *is hard for $\mathcal{W}[1]$ parameterized by k and $\mathcal{NP}$-complete [25, 27].*

Proof. We reduce from p-Sequence Shortest Common Supersequence (parameterized version). As an instance I of p-Sequence Shortest Common Supersequence (parameterized version) we are given k p-sequences $x_1, x_2, \dots, x_k$ over an alphabet Σ. Let M be a positive integer. W.l.o.g. we assume that each symbol of Σ occurs in at least one of the input sequences. Let a sequence x_i let be of length t_i and consist of the symbols $x_i = x_i[1]x_i[2] \dots x_i[t_i]$ $(1 \leq i \leq t_i)$.

[1]Analogous to a p-tree, for an alphabet Σ a *p-sequence* is a sequence where each symbol of Σ occurs at most once. In an *rl-sequence* symbols may be repeated.

We transform the instance I of p-SEQUENCE SHORTEST COMMON SUPERSEQUENCE (parameterized version) into an instance I' of SMALLEST COMMON SUPERTREE (parameterized version). Let $L = \Sigma \cup \{\$_1, \$_2\} \cup \{\sigma_1, \sigma_2, \ldots, \sigma_M\}$ with $\$_l \notin \Sigma, \sigma_j \notin \Sigma, l \in \{1,2\}, 1 \leq j \leq M$. For each x_i we construct a binary leaf-labeled tree T_i represented by the expression $(((\ldots((F, x_i[1]), x_i[2]), \ldots), x_i[t_i]), \$_l)$, $1 \leq i \leq k$ and $l = 1$ if i is odd and $l = 2$ if i is even (cf. Figure 6.1); here F is a binary leaf-labeled caterpillar tree over the leafset $\{\sigma_1, \sigma_2, \ldots, \sigma_M, \$_{l'}\}$ with $l' = 1$ if i is even and $l' = 2$ if i is odd. Furthermore, for simplicity we denote the caterpillar tree built over the leafset $\{\sigma_1, \sigma_2, \ldots, \sigma_M\}$ with F_σ. Let $m = 2M+4$.

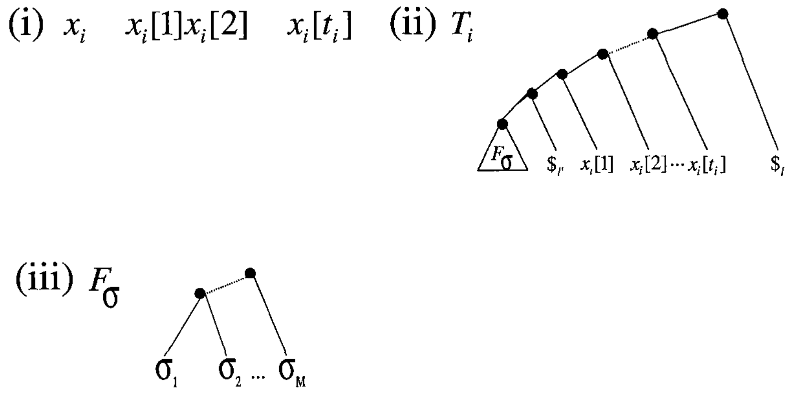


Figure 6.1: Gadget for proof of Theorem 6.4. In the p-SHORTEST COMMON SUPERSEQUENCE instance each sequence x_i (i) is transformed into a p-tree T_i (ii).

Because each x_i is a p-sequence, F is a p-tree, and for each i $\$_l \neq \$_{l'}$, each T_i is a p-tree $(1 \leq i \leq k)$ and therefore I' is a valid instance for SMALLEST COMMON SUPERTREE (parameterized version).

We show I is a *yes*-instance if and only if I' is a *yes*-instance.

$\Rightarrow$ Let $x = x[1]x[2]\ldots x[M']$, $M' \leq M$, be a solution for I. Then $T = ((\ldots(((F_\sigma, (\$_1, \$_2)), x[1]), x[2]), \ldots, x[M']), (\$_1, \$_2))$ is a solution for I', $|T| = M + 2 + M' + 2 \leq 2M + 4 = m$.

$\Leftarrow$ Let the rl-tree T^* be a smallest solution for I', $|T^*| = m' \leq m$. It is sufficient to show that there exist a common supertree T, $|T| \leq m'$, of the form $((\ldots(((F_\sigma, (\$_1, \$_2)), x[1]), x[2]), \ldots, x[M']), (\$_1, \$_2))$, $M' \leq \frac{m'-4}{2} \leq M$. Then from T we can construct the sequence $x[1]x[2]\ldots x[M']$, which is a common supersequence of all x_i, $1 \leq i \leq k$.

Because F_σ is a subtree of each T_i, $1 \leq i \leq k$, and none of F_σ's leaf symbols $LS(root(F_\sigma))$ is element of Σ, we can assume that T^* contains F_σ as a subtree, and $LS(v) = \{\sigma_1, \sigma_2, \ldots, \sigma_M\}$ where v is the least common ancestor of $\{\sigma_1, \sigma_2, \ldots, \sigma_M\}$ in T^*. Furthermore, w.l.o.g. T^* contains twice the sibling pair $(\$_1, \$_2)$ and the least common ancestor of the one pair of $\$_1$ and $\$_2$ is a child of the root of T^*, while the least common ancestor of the other pair is a child of the parent of v.

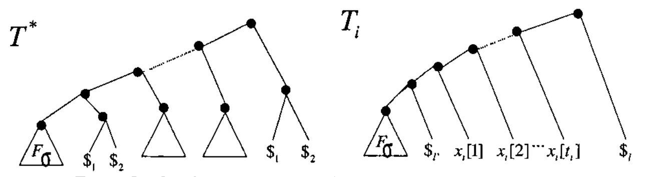


Figure 6.2: Proof of Theorem 6.4, "$\Leftarrow$": Rearranging of T^* is possible because of the caterpillar structure of the T_i.

We show that we can rearrange T^* to T. We refer to the subtrees $((\ldots(x[1], x[2]), \ldots), x[M'])$ in T and $((\ldots(x_i[1], x_i[2]), \ldots), x_i[t_i])$ in T_i as the caterpillar parts of T and T_i $(1 \leq i \leq k)$. Consider the tree X^*, which is the tree T^* restricted to the subtree containing all the leaves $x_i[1], x_i[2], \ldots, x_i[t_i]$ $(1 \leq i \leq k)$. W.l.o.g., we consider the case that X^* is not a caterpillar tree (i.e., T^* differs from T). Then there is an internal vertex v in T^*, $v \neq root(T^*)$, s.t. none of v's children v_1 and v_2 is a leaf and neither v_1 nor v_2 is the root of F_σ. Let v_2 be the child of v with $T^*(v_2)$ not containing any leaves in $\{\sigma_1, \ldots, \sigma_M\}$. Then for each i, $1 \leq i \leq k$, because $\{x_i[1], x_i[2], \ldots, x_i[t_i]\}$ build the caterpillar part of T_i there is at most one leaf of $\{x_i[1], x_i[2], \ldots, x_i[t_i]\}$ contained in the subtree of T^* induced by v_2. Therefore, we can reorganize the subtree induced by v_2 in T^* such that each leaf in $T^*(v_2)$ is a child from the path between v_1 and v (cf. Figure 6.2).

□

6.2.2 A Tractable Parameterization

Because of the interesting result we want to mention the following parameterization of SMALLEST COMMON SUPERTREE which is fixed-parameter tractable. Although SMALLEST COMMON SUPERTREE (parameterized version) is W[1]-hard, the problem becomes fixed-parameter tractable if we allow a bounded number of leaves r additionally to the size of the common supertree; the parameter is (k, r).

Problem 6.5. BOUNDED SMALLEST COMMON SUPERTREE
Input: k gene trees G_i $(1 \leq i \leq k)$ over leafset L, $|L| = n$, and a positive integer r
Parameter: (k, r)
Question: Does there exist a common supertree T of the G_i of size at most $n + r$?

Here we can interpret r as representing the number of gene-duplication events.

Theorem 6.5. *[25, 27]* BOUNDED SMALLEST COMMON SUPERTREE *is fixed-parameter tractable.*

Because of the analogy between supertree and supersequence problems we state the following two parameterizations of SHORTEST COMMON SUPERSEQUENCE which are also both fixed-parameter tractable [25, 27].

Problem 6.6. BOUNDED DUPLICATION FOR P-SEQUENCES
Input: k p-sequences $x_i \in \Sigma^*$ for $i = 1, ..., k$ and a positive integer r representing the number of duplication events. We assume that $|\Sigma| = n$ and that each symbol of Σ occurs in at least one of the input sequences.
Parameter: (k, r)
Question: Does there exist a common supersequence x of length at most $n + r$?

Problem 6.7. BOUNDED DUPLICATION FOR COMPLETE P-SEQUENCES
Input: Complete [2] p-sequences x_i $(1 \leq i \leq k)$ over an alphabet Σ of size n, a positive integer r, and a cost function $c : \Sigma \to Z^+$.
Parameter: r
Question: Does there exist a common supersequence x of duplication cost $||x||_c \leq r$ where the duplication cost is defined:

$$||x||_c = \sum_{a \in \Sigma} (n_a(x) - 1)c(a)$$

$n_a(x)$, $a \in \Sigma$, denotes the number of occurrences of symbol a in x.

[2] A *complete* p-sequence over an alphabet Σ contains every symbols of Σ exactly once.

Chapter 7

A Fixed-Parameter-Tractable Algorithm for Gene Duplication

The main result of this chapter is our fixed-parameter-tractable algorithm for the problem GENE DUPLICATION parameterized by the number of gene duplications c (Problem 7.1). The underlying model for this problem is the GENE-DUPLICATION MODEL (cf. Section 5.3). A preliminary version of this result was published by Stege in [66].

Problem 7.1. GENE DUPLICATION (parameterized version)
Input: Gene trees $G_1, \ldots, G_k$ over leafset L, a positive integer c.
Parameter: c.
Question: Does there exist a species tree S such that $G_1, \ldots, G_k$ can be rectified with respect to S with at most c gene-duplication events, i.e., $\sum\limits_{i=1}^{k} cost_{GD}(G_i, S) \leq c$?

Before we describe a fixed-parameter-tractable algorithm solving Problem 7.1, we point out a restriction of the non-parameterized version of the $\mathcal{NP}$-complete problem GENE DUPLICATION (Problem 5.2, page 40), where the problem becomes solvable in linear time. We restate GENE DUPLICATION in its minimization version.

Problem 7.2. GENE DUPLICATION (minimization version)
Input: Gene trees $G_1, \ldots, G_k$ over leafset L.
Output: A species tree S such that the number of gene-duplication events necessary to rectify $G_1, \ldots, G_k$ with respect to S is minimized (i.e., a species tree s.t. $\sum_{i=1}^{k} cost_{GD}(G_i, S)$ is minimized).

We make the following observation.

Observation 7.1. *Suppose we are given gene trees $G_1, \ldots G_k$. Let $c^* = \min_{j=1}^{k}(\sum_{i=1}^{k} cost_{GD}(G_i, G_j))$. Then c^* is an upper bound for the number of gene-duplication events required to rectify the gene trees $G_1, \ldots, G_k$ with an optimal species tree.*

Thus, we can determine a gene tree which is a best possible solution for a species tree when we consider the gene trees $G_1, \ldots, G_k$ as possible solutions only. Furthermore, if a species tree S, which solves Problem 7.2, does not agree with one of the gene trees $G_1, \ldots, G_k$, then $\sum_{i=1}^{k} cost_{GD}(G_i, S) \geq k$, because for each gene tree we need at least one gene-duplication event for the rectification of the differences with the species tree. Therefore, if $c^* \leq k$, we know that there is an optimal species tree $S \in \{G_1, \ldots, G_k\}$ and therefore in this special case S is computable in linear time depending on $|L|$ and k.

Observation 7.2. *Suppose we are given gene trees $G_1, \ldots G_k$. Let $c^* = \min_{j=1}^{k}(\sum_{i=1}^{k} cost_{GD}(G_i, G_j))$. If $c^* \geq k$ then k is a lower bound for the number of gene-duplication events necessary to rectify the gene trees with respect to a species tree.*

We now prepare for the algorithm solving Problem 7.1 by introducing a generalized version of the GENE-DUPLICATION MODEL, because it provides useful properties for solving GENE DUPLICATION.

7.1 A Generalization of the Gene-Duplication Model

In this section we introduce the GENE-DUPLICATION MODEL FOR SPLITS. The notion of a *split* is one of the cornerstones of our fixed-parameter-tractable algorithm.

Definition 7.1. Let L be a leafset, $|L| > 1$. We call $\mathcal{D} = (\mathcal{D}_l|\mathcal{D}_r)$ a *split* of L if

1. $\mathcal{D}_l, \mathcal{D}_r \subseteq L$,
2. $D_l \neq L$
3. $\mathcal{D}_l \neq \emptyset$, and
4. $\mathcal{D}_l \cap \mathcal{D}_r = \emptyset$.

We call $\mathcal{D}$ *complete* if $\mathcal{D}_l \cup \mathcal{D}_r = L$ and $\mathcal{D}_r \neq \emptyset$. Otherwise we call $\mathcal{D}$ *incomplete.*

Definition 7.2. Let L be a leafset and let $\mathcal{D} = (\mathcal{D}_l|\mathcal{D}_r)$, $\mathcal{D}' = (\mathcal{D}'_l|\mathcal{D}'_r)$ be splits of L with $\mathcal{D}_l \cup \mathcal{D}_r = \mathcal{D}'_l \cup \mathcal{D}'_r$. Then $\mathcal{D} = \mathcal{D}'$ if and only if $\mathcal{D}_l = \mathcal{D}'_l$ or $\mathcal{D}_l = \mathcal{D}'_r$.

Definition 7.3. Suppose we are given a leafset L and the gene trees $g_1 = (V_{g_1}, E_{g_1}, L_1), \ldots, g_m = (V_{g_m}, E_{g_m}, L_m)$ $(m \geq 1)$ with $L_i \subseteq L$ $(i = 1, \ldots, m)$ and $\bigcup L_i = L$. If $L_i \cap L_j = \emptyset$ for all $i, j \in \{1, \ldots, m\}$, $i \neq j$, then we call the $g_1, \ldots, g_m$ a *gene forest* over L. Furthermore, for a gene forest G consisting of the trees $g_1, \ldots, g_m$ we write $V_G = \bigcup_{i=1}^{m} V_{g_i}$

To simplify the description and the implementation of the algorithm we extend the definition of the cost function $cost_{GD}$ in the GENE-DUPLICATION MODEL and allow a gene forest as an input instance instead of a gene tree.

Definition 7.4. Suppose we are given a gene forest G and a species tree S. Then $cost_{GD}(G, S) = |Dups_{GD}(G, S)|$, where

$$Dups_{GD}(G, S) = \{u | u \in V_G - L, event_{G,S}(u) = dup\}.$$

We next define the cost for a gene forest and a split similarly to the cost for a gene forest and a species tree.

Definition 7.5. (GENE-DUPLICATION MODEL FOR SPLITS)
Let $G = (V_G, E_G, L)$ be a gene forest consisting of gene trees $g_1, \dots, g_m$ over L. Let further $\mathcal{D} = (\mathcal{D}_l|\mathcal{D}_r)$ be a split of L. If $\mathcal{D}_r \neq \emptyset$ then $cost(G, \mathcal{D}) = |Dups(G, \mathcal{D})|$, where

$$Dups(G,\mathcal{D}) = \{v \in V_G | LS(v) \cap \mathcal{D}_l \neq \emptyset \text{ and } LS(v) \cap \mathcal{D}_r \neq \emptyset \text{ and}$$
$$((LS(v_l) \cap \mathcal{D}_l \neq \emptyset \text{ and } LS(v_l) \cap \mathcal{D}_r \neq \emptyset) \text{ or } (LS(v_r) \cap \mathcal{D}_l \neq \emptyset \text{ and}$$
$$LS(v_r) \cap \mathcal{D}_r \neq \emptyset))\}.$$

Otherwise,

$$cost(G,\mathcal{D}) = \min_{b \in L - \mathcal{D}_l} cost(G, (\mathcal{D}_l, \{b\})).$$

Definition 7.6. For given gene forests $G_1, G_2, \dots, G_k$ over a leafset L and a split $\mathcal{D}$ over L, we call a species tree S_0 *optimal depending on* $\mathcal{D}$ if $\sum_{i=1}^{k} cost_{GD}(G_i, S_0)$ is minimized over all possible species trees S with $\mathcal{D}_l \subseteq LS(root(S)_l)$ and $\mathcal{D}_r \subseteq LS(root(S)_r)$.

The following observations and Lemma 7.6 provide the main ingredients for the fixed-parameter-tractable algorithm described in Section 7.2.

Observation 7.3. *Suppose we are given gene forests $G_1, \dots, G_k$ over a leafset L. Let $\mathcal{D} = (\mathcal{D}_l|\mathcal{D}_r)$ be a complete split of L. Furthermore, let S be an optimal species tree for $G_1, \dots, G_k$ depending on $\mathcal{D}$. Then for each $G \in \{G_1, \dots, G_k\}$*

$$Dups(G, \mathcal{D}) \subseteq Dups_{GD}(G, S)$$

and $\sum_{i=1}^{k} cost(G_i, \mathcal{D})$ is exactly the number of gene-duplication events located at the root of S (cf. Figure 7.1).

Observation 7.4. *Suppose we are given gene forests $G_1, \dots, G_k$ over L. Let $\mathcal{D} = (\mathcal{D}_l|\mathcal{D}_r)$, $\mathcal{D}' = (\mathcal{D}'_l|\mathcal{D}'_r)$ be complete splits of L. Let S be an optimal species tree for $G_1, \dots, G_k$ depending on $\mathcal{D}$ and let S' be the optimal species tree for $G_1, \dots, G_k$ depending on $\mathcal{D}'$. If $Dups(G_i, \mathcal{D}) \subseteq Dups(G_i, \mathcal{D}')$ for $i \in \{1, \dots k\}$, then*

$$\sum_{i=1}^{k} cost_{GD}(G_i, S) \leq \sum_{i=1}^{k} cost_{GD}(G_i, S').$$

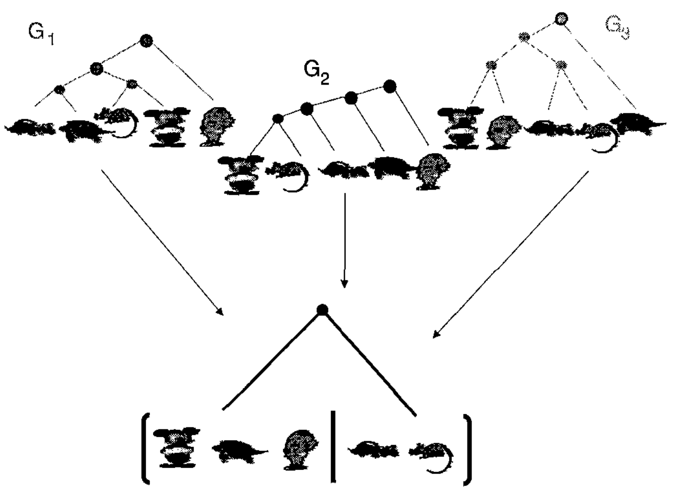


Figure 7.1: Gene forests G_1, G_2, and G_3 are mapped into a split. The marked vertices are the vertices which create a duplication event at the root of the tree induced by the split. Since every internal vertex in a gene forests corresponds to exactly one event (duplication or speciation), every species tree depending on the split has the same number of duplication events located at the root.

Definition 7.7. Let $G_1, \ldots, G_k$ be gene forests over a leafset L, let $\mathcal{D} = (\mathcal{D}_l|\mathcal{D}_r)$ be a split of L with $\sum_{i=1}^{k} cost(G_i, \mathcal{D}) = 0$. Let $a, b \in L - (\mathcal{D}_l \cup \mathcal{D}_r)$, $a \neq b$. Furthermore let $\mathcal{D}_1 = (\mathcal{D}_l \cup \{a,b\}|\mathcal{D}_r)$, $\mathcal{D}_2 = (\mathcal{D}_l \cup \{a\}|\mathcal{D}_r \cup \{b\})$, $\mathcal{D}_3 = (\mathcal{D}_l \cup \{b\}|\mathcal{D}_r \cup \{a\})$, and $\mathcal{D}_4 = (\mathcal{D}_l|\mathcal{D}_r \cup \{a,b\})$. We call (a,b) a *conflict pair* if either

- $\sum_{i=1}^{k} cost(G_i, \mathcal{D}_j) > 0$ for $j \in \{1,2,3,4\}$ or
- if the following properties are all fulfilled.

  1. $\mathcal{D}_r = \emptyset$.

2. $\sum_{i=1}^{k} cost(G_i, \mathcal{D}_j) > 0$ for $j \in \{2, 3, 4\}$.
3. For each gene forest G_i $cost(G_i, \mathcal{D}_1) = 0$.
4. $\bigcap_{i=1}^{k} L_i = \emptyset$ where
$L_i = \{c \in L - \mathcal{D}_l | cost(G_i, (\mathcal{D}_l \cup \{a, b\} | \{c\}) = 0\}, i \in \{1, \dots, k\}$.

Definition 7.8. Let $G_1, \dots, G_k$ be gene forests over a leafset L, let $\mathcal{D} = (\mathcal{D}_l | \mathcal{D}_r)$ be a split of L with $\sum_{i=1}^{k} cost(G_i, \mathcal{D}) = 0$. If we can supplement the split $\mathcal{D}$ to a complete split $\mathcal{D}^* = (\mathcal{D}_l^* | \mathcal{D}_r^*)$ of L, $\mathcal{D}_l \subseteq \mathcal{D}_l^*$, $\mathcal{D}_r \subseteq \mathcal{D}_r^*$, such that $\sum_{i=1}^{k} cost(G_i, \mathcal{D}^*) = 0$, then we call $\mathcal{D}^*$ a *completion* of $\mathcal{D}$.

Observation 7.5. *Suppose we are given k gene forests $G_1, \dots, G_k$ over a leafset L. Let $\mathcal{D} = (\mathcal{D}_l | \mathcal{D}_r)$ be a split of L with $\mathcal{D}_l \cup \mathcal{D}_r \neq L$. Let $\sum_{i=1}^{k} cost(G_i, \mathcal{D}) = 0$. Furthermore, let the gene forest G_i consist of the gene trees $g_1^i, \dots, g_{p_i}^i$ ($i \in \{1, \dots, k\}$). Then for any vertex $v = root(g_j^i)_s$ ($i \in \{1, \dots, k\}$, $j \in \{1, \dots, p_i\}$, and $s \in \{l, r\}$)*

$$LS(v) \cap (\mathcal{D}_l \cup \mathcal{D}_r) \subseteq \mathcal{D}_l \text{ or } LS(v) \cap (\mathcal{D}_l \cup \mathcal{D}_r) \subseteq \mathcal{D}_r.$$

Lemma 7.6. *Suppose we are given two gene forests G and H over a leafset L. Let $\mathcal{D} = (\mathcal{D}_l | \mathcal{D}_r)$ be a split of L with $\mathcal{D}_l \cup \mathcal{D}_r \neq L$. Suppose, furthermore, that $cost(G, \mathcal{D}) = cost(H, \mathcal{D}) = 0$. Then either there is a conflict pair (a, b), $a, b \in L - (\mathcal{D}_l \cup \mathcal{D}_r)$, or there is a completion $\mathcal{D}^* = (\mathcal{D}_l^* | \mathcal{D}_r^*)$ of $\mathcal{D}$.*

Proof. Let the gene forest G consist of the trees $g_1, \dots, g_p$ and let the gene forest H consist of the trees $h_1, \dots, h_q$. Furthermore, let $v_l^{g_i} = root(g_i)_l$, $v_r^{g_i} = root(g_i)_r$ ($i \in \{1, \dots, p\}$) and $v_l^{h_j} = root(h_j)_l$, $v_r^{h_j} = root(h_j)_r$ ($j \in \{1, \dots, q\}$).

We show that if there is no conflict pair, then $\mathcal{D}$ has a completion $\mathcal{D}^*$. Let $a, b \in L - (\mathcal{D}_l \cup \mathcal{D}_r)$. Then for G we distinguish the following cases.

- There is a tree $g_i \in \{g_1, \dots, g_p\}$ with $a, b \in LS(v_s^{g_i})$ ($s \in \{l, r\}$).
- There are g_{i_1}, g_{i_2} with $a \in LS(root(g_{i_1})), b \in LS(root(g_{i_2}))$ ($i_1 = i_2$ is possible).

Furthermore, because there is no conflict pair in $L - (\mathcal{D}_l \cup \mathcal{D}_r)$ we know

1. $cost(G, \mathcal{D}_1) = cost(H, \mathcal{D}_1) = 0$, $\mathcal{D}_1 = (\mathcal{D}_l \cup \{a,b\} | \mathcal{D}_r)$,
2. $cost(G, \mathcal{D}_2) = cost(H, \mathcal{D}_2) = 0$, $\mathcal{D}_2 = (\mathcal{D}_l \cup \{a\} | \mathcal{D}_r \cup \{b\})$,
3. $cost(G, \mathcal{D}_3) = cost(H, \mathcal{D}_3) = 0$, $\mathcal{D}_3 = (\mathcal{D}_l \cup \{b\} | \mathcal{D}_r \cup \{a\})$, or
4. $cost(G, \mathcal{D}_4) = cost(H, \mathcal{D}_4) = 0$, $\mathcal{D}_4 = (\mathcal{D}_l | \mathcal{D}_r \cup \{a,b\})$.

We consider all possible cases for any two leaves $a, b \notin L - (\mathcal{D}_l \cup \mathcal{D}_r)$, $a \neq b$, (symmetric cases are excluded).

Case 1 If $a, b \in LS(v_s^{g_i})$ $(s \in \{l, r\})$ we know that $cost(G, \mathcal{D}_1) = cost(H, \mathcal{D}_1) = 0$ or $cost(G, \mathcal{D}_4) = cost(H, \mathcal{D}_4) = 0$. Because of Observation 7.5 and because there is no conflict pair for $\mathcal{D}$ in $L - (\mathcal{D}_l \cup \mathcal{D}_r)$, there do not exist $t_1, t_2, t_3 \in \{v_s^{g_i} | i \in \{1, \dots p\}, s \in \{l, r\}\} \cup \{v_{s'}^{h_j} | j \in \{1, \dots, q\}, s' \in \{l, r\}\}$ such that $LS(t_1) \cap LS(t_2) \neq \emptyset$, $LS(t_1) \cap LS(t_3) \neq \emptyset$, $LS(t_2) \cap (\mathcal{D}_l \cup \mathcal{D}_r) \subseteq \mathcal{D}_l$, and $LS(t_r) \cap (\mathcal{D}_l \cup \mathcal{D}_r) \subseteq \mathcal{D}_r$.

Therefore, in both cases there is a completion $\mathcal{D}^*$ of $\mathcal{D}$, namely $\mathcal{D}^* = \big(\mathcal{D}_l \cup \{a,b\} \cup E_1^* \mid \mathcal{D}_r \cup \{l \in L - (\mathcal{D}_l \cup \mathcal{D}_r \cup \{a,b\}) \mid l \notin E_1^*\}\big)$, where $E_1^* = \Big\{l \in L - (\mathcal{D}_l \cup \mathcal{D}_r \cup \{a,b\}) \mid$ if $l \in LS(v_s^{g_i}) \cap LS(v_{s'}^{h_j})$ then $\neg\exists x \in \big(LS(v_s^{g_i}) \cup LS(v_{s'}^{h_j})\big)$ with $x \in \mathcal{D}_r\Big\}$ if $cost(G, \mathcal{D}_1) = cost(H, \mathcal{D}_1) = 0$ and otherwise, $\mathcal{D}^* = \big(\mathcal{D}_l \cup \{l \in L - (\mathcal{D}_l \cup \mathcal{D}_r \cup \{a,b\}) \mid l \notin E_4^*\} \mid \mathcal{D}_r \cup \{a,b\} \cup E_4^*\big)$ where $E_4^* = \Big\{l \in L - (\mathcal{D}_l \cup \mathcal{D}_r \cup \{a,b\}) \mid$ if $l \in LS(v_s^{g_i}) \cap LS(v_{s'}^{h_j})$ then $\neg\exists x \in LS(v_s^{g_i}) \cup LS(v_{s'}^{h_j})$ with $x \in \mathcal{D}_l\Big\}$.

Case 2 If $a \in LS(v_s^{g_i})$ $(s \in \{l, r\})$ and $b \in LS(v_{s'}^{g_{i'}})$ $(s' \in \{l, r\})$ then similarly

1. if $cost(G, \mathcal{D}_1) = cost(H, \mathcal{D}_1) = 0$ then $\mathcal{D}^* = \big(\mathcal{D}_l \cup \{a,b\} \cup E_1^* \mid \mathcal{D}_r \cup \{l \in L - (\mathcal{D}_l \cup \mathcal{D}_r \cup \{a,b\}) \mid l \notin E_1^*\}\big)$, where $E_1^* = \Big\{l \in L - (\mathcal{D}_l \cup \mathcal{D}_r \cup \{a,b\}) \mid$ if $l \in LS(v_s^{g_i}) \cap LS(v_{s'}^{h_j})$ then $\neg\exists x \in \big(LS(v_s^{g_i}) \cup LS(v_{s'}^{h_j})\big)$ with $x \in \mathcal{D}_r\Big\}$, is a completion of $\mathcal{D}$.

2. if $cost(G, \mathcal{D}_2) = cost(H, \mathcal{D}_2) = 0$ then $\mathcal{D}^* = \big(\mathcal{D}_l \cup \{a\} \cup E_2^* \mid \mathcal{D}_r \cup \{l \in L - (\mathcal{D}_l \cup \mathcal{D}_r \cup \{a,b\}) \mid l \notin E_2^*\}\big)$, where $E_2^* = \Big\{l \in L - (\mathcal{D}_l \cup \mathcal{D}_r \cup \{a,b\}) \mid$ if $l \in LS(v_s^{g_i}) \cap LS(v_{s'}^{h_j})$ then $\neg\exists x \in \Big(LS(v_s^{g_i}) \cup LS(v_{s'}^{h_j})\Big)$ with $x \in \mathcal{D}_r\Big\}$, is a completion of $\mathcal{D}$.
3. $cost(G, \mathcal{D}_3) = cost(H, \mathcal{D}_3) = 0$ then $\mathcal{D}^* = \big(\mathcal{D}_l \cup \{a,b\} \cup E_3^* \mid \mathcal{D}_r \cup \{l \in L - (\mathcal{D}_l \cup \mathcal{D}_r \cup \{a,b\}) \mid l \notin E_3^*\}\big)$, where $E_3^* = \Big\{l \in L - (\mathcal{D}_l \cup \mathcal{D}_r \cup \{a,b\}) \mid$ if $l \in LS(v_s^{g_i}) \cap LS(v_{s'}^{h_j})$ then $\neg\exists x \in \Big(LS(v_s^{g_i}) \cup LS(v_{s'}^{h_j})\Big)$ with $x \in \mathcal{D}_r\Big\}$, is a completion of $\mathcal{D}$.
4. if $cost(G, \mathcal{D}_4) = cost(H, \mathcal{D}_4) = 0$ then $\mathcal{D}^* = \big(\mathcal{D}_l \cup \{l \in L - (\mathcal{D}_l \cup \mathcal{D}_r \cup \{a,b\}) \mid l \notin E_4^*\} \mid \mathcal{D}_r \cup \{a,b\} \cup E_4^*\big)$, where $E_4^* = \Big\{l \in L - (\mathcal{D}_l \cup \mathcal{D}_r \cup \{a,b\}) \mid$ if $l \in LS(v_s^{g_i}) \cap LS(v_{s'}^{h_j})$ then $\neg\exists x \in LS(v_s^{g_i}) \cup LS(v_{s'}^{h_j})$ with $x \in \mathcal{D}_l\Big\}$, is a completion of $\mathcal{D}$.

□

Corollary 7.7. *Suppose we are given a leafset L. Let $G_1, \ldots, G_k$ be gene forests with $LS(root(G_i)) \subseteq L$ ($i \in \{1, \ldots, k\}$). Let $\mathcal{D}$ be an incomplete split of L with $\sum_{i=1}^{k} (G_i, \mathcal{D}) = 0$. Then $G_1, \ldots G_k$ have a completion of $\mathcal{D}$ if and only if each pair of $\{G_1, \ldots, G_k\}$ has a completion of $\mathcal{D}$.*

Lemma 7.6 and Corollary 7.7 lead to the following theorem.

Theorem 7.8. *Suppose we are given a leafset L and gene forests $G_1, \ldots, G_k$, where $LS(root(G_i)) \subseteq L$ ($i \in \{1, \ldots, k\}$). Let $\mathcal{D} = (\mathcal{D}_l | \mathcal{D}_r)$ be an incomplete split of L where $\mathcal{D}_l$, $\mathcal{D}_r$ are leafsets and $\mathcal{D}_l, \mathcal{D}_r \neq \emptyset$. Then either there is a completion of $\mathcal{D}$ or there is a conflict pair (a, b), $a, b \in L - (\mathcal{D}_l \cup \mathcal{D}_r)$.*

7.2 A Fixed-Parameter-Tractable Algorithm

The outline of the algorithm is described as a recursive bounded search-tree algorithm based on Observation 7.3. We compute the leafsets of the left and the right subtree of the root of the species tree first and then refine this split to a binary tree recursively.

Before describing the algorithm in detail, we make the following observations concerning the number of input trees/input forests.

Definition 7.9. Suppose we are given a gene tree G over a leafset L. Let $\mathcal{D}$ be a complete split of L.

1. We say that G is of *type* $\mathcal{D}$ if and only if $(LS(root(G)_l)|LS(root(G)_r)) = \mathcal{D}$.
2. For the given gene trees $G_1, \dots, G_k$ we define the set $\mathcal{T} := \{\mathcal{D}|\mathcal{D}$ is a complete split of L and $\exists G_i$ where G_i is tree of type $\mathcal{D}\}$ to be the *set of different types* of $G_1, \dots, G_k$.

Observation 7.9. *Let $\mathcal{D}$ be a complete split of a leafset L and let G be a gene tree over L with $(LS(root(G)_l)|LS(root(G)_r)) \neq \mathcal{D}$. Then $cost(G, \mathcal{D}) > 0$.*

Observation 7.10. *Suppose we are given gene trees $G_1, \dots, G_k$ over a leafset L and an integer $c > 0$. If $|\mathcal{T}| > c$ then the question whether there exists a complete split $\mathcal{D}$ of L with $\sum_{i=1}^{k} cost(G_i, \mathcal{D}) \leq c$ has the answer* no.

Observation 7.11. *Let $G_1, \dots, G_k$ be gene trees over L and let c be a positive integer. If there exists a complete split $\mathcal{D}^*$ of L where $|\{G_i|G_i$ is a tree of type $\mathcal{D}^*\}| > c$ and the answer to the question whether there exists a complete split $\mathcal{D}$ of L with $\sum_{i=1}^{k} cost(G_i, \mathcal{D}) \leq c$ is* yes, *then $\sum_{i=1}^{k}(G_i, \mathcal{D}^*) \leq c$ and $\sum_{i=1}^{k}(G_i, \mathcal{D}) > c$ for all complete splits $\mathcal{D} \neq \mathcal{D}^*$. Furthermore, $|\{G_i|cost(G_i, \mathcal{D}^*) > 0\}| \leq c$.*

Corollary 7.12. *Let $G_1, \dots, G_k$ be gene trees over a leafset L and let c be a positive integer. If the question whether there exists a complete split $\mathcal{D}$ with $\sum_{i=1}^{k}(G_i, \mathcal{D}) \leq c$ has the answer* yes, *then the following properties are satisfied.*

1. *$|\mathcal{T}| \leq c$*
2. *If $|\{G_i|G_i$ is a tree of type $\mathcal{D}\}| \leq c$ for all $\mathcal{D} \in \mathcal{T}$, then $k \leq c^2$.*
3. *$|\{G_i|cost(G_i, \mathcal{D}) > 0\}| \leq c$ for all $\mathcal{D}$ $\sum_{i=1}^{k}(G_i, \mathcal{D}) \leq c$*
4. *If $k > c$ and $\neg\exists\mathcal{D}$ with $|\{G_i|G_i$ is a tree of type $\mathcal{D}\}| > c$ then there is a split $\mathcal{D}^* \in \mathcal{T}$ with $\sum_{i=1}^{k}(G_i, \mathcal{D}^*) \leq c$.*

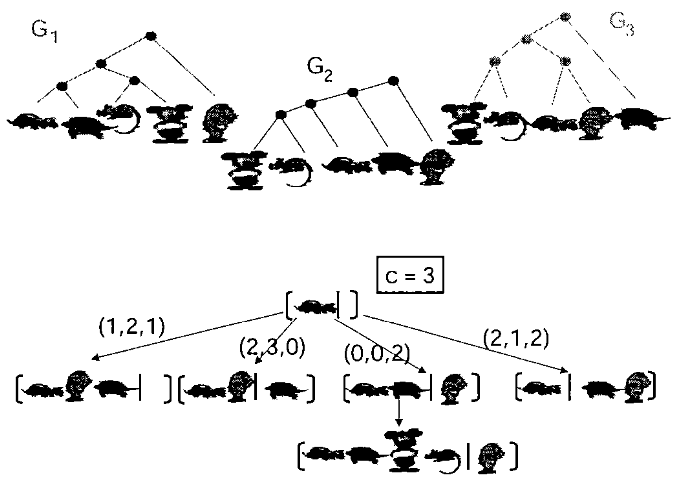


Figure 7.2: An example illustrating the fixed-parameter-tractable algorithm solving Gene Duplication. A species tree for three gene trees is computed. Here after the first step only one SearchTreeNode has to be kept. A triple (k_1, k_2, k_3) at a branch of the search tree means that gene tree G_1, which is mapped into the split at the child of the branch, can be rectified with k_1 duplications, gene tree G_2 with k_2 duplications, and finally G_3 with k_3 duplications. This figure is continued on page 76.

We now describe the fixed-parameter-tractable algorithm solving Gene Duplication. The algorithm is presented in pseudo code. We start with the definitions of the data structures.

```
**type**
nodeRef = **pointer** to SearchTreeNode;
SearchTreeNode = **record**
  leaf: boolean;
  complete: boolean;
  $k$: integer;
  $G_1, \ldots, G_k$: GeneForest;
  $\mathcal{D}_l, \mathcal{D}_r$: Leafset;
  $L$: Leafset;
  $c$: integer;
  $\mathcal{M}$: set of Leafset;
  child$_1, \ldots$, child$_k$: nodeRef;
end;

type
PairOfLeaves = record
  conflict: boolean;
  $p_1, p_2$: Leaf;
end;

type
Split = record
  $\mathcal{D}_l, \mathcal{D}_r$: Leafset;
end;
```

The main function, **geneDuplication**, answers for an input of k gene trees over a leafset L and an integer c the question, whether there exists a species tree S over L s.t. $\sum_{i=1}^{k} cost(G_i, S) \leq c$. First, the root of the search tree is created. Then, following Corollary 7.12, we consider several cases depending on the number of gene-tree types and the number of gene trees of a given type. The search tree is produced by the two functions **blueTree** and **redTree**. The function **blueTree** is responsible for all possible splits costing no more than $c \geq k$ for a certain instance. The function **redTree** recurses on an instance associated with a complete split $\mathcal{D} = (\mathcal{D}_l|\mathcal{D}_r)$ processing $\mathcal{D}_l$ before $\mathcal{D}_r$.

```
function geneDuplication($G_1, \ldots, G_k$: GeneForest; $L$: Leafset, $c, k$:
integer): boolean;
var
  $\mathcal{T}$: set of Split;
  $\mathcal{D}$, $\mathcal{D}^*$: Split;
  $\mathcal{D}_l$, $\mathcal{D}_r$: Leafset;
  $a$: Leaf;
  $j$, count: integer;
  root, left, right, out: SearchTreeNode;
```

```
begin
  $root$.complete := false;
  $root.G_1, \ldots, root.G_k := G_1, \ldots, G_k$;
  $root.L := L$;
  $root.c := c$;
  $root.k := k$;

/* consider the cases depending on the number of gene-tree types */
/* and the number of gene trees of a given type (cf. Corollary 7.12) */
  $\mathcal{T} := \{\mathcal{D} | \exists G_i$ with $G_i$ is tree of type $\mathcal{D}\}$;

  if $|\mathcal{T}| = 1$ then
  /* all trees agree on their type; no branching is necessary to */
  /* compute a complete split of $L$, all the $k$ input gene trees */
  /* are unchanged (cf. Observation 7.4) */
    let $\mathcal{D}^* \in \mathcal{T}$;
    $root$.complete := true;
    $root.\mathcal{D}_l := \mathcal{D}^*.\mathcal{D}_l$;
    $root.\mathcal{D}_r := \mathcal{D}^*.\mathcal{D}_r$;
    $root.\mathcal{M} := \emptyset$;
    $root$ := compute($root$);

  else if $|\mathcal{T}| > c$ then
  /* any split would cause costs higher than $c$ */
    return false;

  else
  /* $|\mathcal{T}| \leq c$; kernelize in the number of forests */
    $count := 0$;
    for each $\mathcal{D} \in \mathcal{T}$ do
      if $|\{G_i | G_i$ is tree of type $\mathcal{D}\}| > c$ then
        $count$++;
        $\mathcal{D}^* := \mathcal{D}$;
      end
    end;

    if $count = 1$ then
    /* if this split is not chosen, all the trees ($> c$) of this type */
    /* would cause costs */
      $root$.complete := true;
      $root.\mathcal{D}_l := \mathcal{D}^*.\mathcal{D}_l$;
      $root.\mathcal{D}_r := \mathcal{D}^*.\mathcal{D}_r$;
      $root :=$ compute($root$);
    /* the number of forests which are not trees is bounded by $c$ */

    else if $count > 1$ then
      return false;

    else if $k > c$ then
    /* $count = 0$ */
    /* the split of the root of a possible solution is in $\mathcal{T}$ */
      for $i = 1$ to $|\mathcal{T}|$ do
        $root.\text{child}_i :=$ copy($root$);
      end;

      $j := 0$;
      for each $\mathcal{D} \in \mathcal{T}$ do
        $j{+}{+}$;
        $root.\text{child}_j$.complete := true;
        $root.\text{child}_j.\mathcal{D}_l := \mathcal{D}.\mathcal{D}_l$;
        $root.\text{child}_j.\mathcal{D}_r := \mathcal{D}.\mathcal{D}_r$;
        $root.\text{child}_j :=$ compute($root.child$);
      end;

    else
    /* $k \leq c$ */
      let $a \in L$;
      $root.\mathcal{D}_l := \{a\}$;
      $root :=$ blueTree($root$);
    end
  end;

  for each search-tree leaf $left$ in $root$ with $left.c \geq 0$ do
    $\mathcal{D}_l := left.\mathcal{D}_l$;
    $\mathcal{D}_r := left.\mathcal{D}_r$;
    $left :=$ redTree($left, \mathcal{D}_l$);

    if $left =$ nil then return false;

    for each search-tree leaf $right$ in $left$ with $right.c \geq 0$ do
```

```
      right := redTree(right, $\mathcal{D}_r$);

      if right = nil then return false;

      if $\exists$ a complete search-tree leaf out in right with $out.c \geq 0$
        and not($out.\mathcal{M} \neq \emptyset$ and $out.\mathcal{D}_r \cap (\bigcap out.\mathcal{M}) = \emptyset$) then
          return true;
      end
    end
  end;
  return false;
end geneDuplication;
```

Function **redTree** is called for input instances associated with complete splits and proceeds in a manner similar to the function **geneDuplication**.

```
function redTree(node: SearchTreeNode, L: Leafset): SearchTreeNode;
var
  cnode: SearchTreeNode;
  $\mathcal{T}$: set of Split;
  $\mathcal{D}$, $\mathcal{D}^*$: Split;
  $\mathcal{D}_l$, $\mathcal{D}_r$: Leafset;
  j, count: integer;

begin
  if $|L| = 1$ then
    return node;
  end;

  cnode := copy(node);
  cnode.complete := false;
  cnode.L := L;
  cnode.k := the number $cnode.G_i$ that are trees when restricted to $L$;

  $\mathcal{T} := \{\mathcal{D} | \exists cnode.G_i$ with $cnode.G_i$ is tree and tree of type $\mathcal{D}\}$;

  if $|\mathcal{T}| = 1$ then
```

```
/* all trees agree on their type */
  let $\mathcal{D}^* \in \mathcal{T}$;
  $cnode$.complete := true;
  $cnode.\mathcal{D}_l := \mathcal{D}^*.\mathcal{D}_l$;
  $cnode.\mathcal{D}_r := \mathcal{D}^*.\mathcal{D}_r$;
  $cnode$ := compute($cnode$);

else if $|\mathcal{T}| > cnode.c$ then
  $cnode.c := -1$;

else
/* $|\mathcal{T}| \leq c$; kernelize in the number of forests */
  $count := 0$;

  for each $\mathcal{D} \in \mathcal{T}$ do
    if $|\{cnode.G_i | cnode.G_i$ is tree of type $\mathcal{D}\}| > c$ then
      $count$++;
      $\mathcal{D}^* := \mathcal{D}$;
    end
  end;

  if $count = 1$ then
  /* if this split is not chosen, all the trees ($> c$) of this type */
  /* would cause costs */
    $cnode$.complete := true;
    $cnode.\mathcal{D}_l := \mathcal{D}^*.\mathcal{D}_l$;
    $cnode.\mathcal{D}_r := \mathcal{D}^*.\mathcal{D}_r$;
    $cnode$ := compute($cnode$);
  /* the number of forests which are not trees is bounded by $c$ */

  else if $count > 1$ then
    return false;

  else if $cnode.k > cnode.c$ then
    $j := 0$;
    for each $\mathcal{D} \in \mathcal{T}$ do
      $j$++;
      $child_j$ := copy($cnode$);
      $child_j$.complete := true;
      $child_j.\mathcal{D}_l := \mathcal{D}.\mathcal{D}_l$;
```

```
        $child_j.\mathcal{D}_r$ := $\mathcal{D}.\mathcal{D}_r$;
        $child_j$ := **compute**($child_j$);
        $cnode$.child$_j$ := $child_j$;
      end;

    else
    /* $cnode.k \leq cnode.c$ */
      $cnode$ := **blueTree**($cnode$);
    end;
  end;

  for each search-tree leaf $left$ in $cnode$ with $left.c \geq 0$ do
    $\mathcal{D}_l$ := $left.\mathcal{D}_l$;
    $\mathcal{D}_r$ := $left.\mathcal{D}_r$;
    $left$ := **redTree**($left$,$\mathcal{D}_l$);
    if $left$ = nil then return nil;

    for each search-tree leaf $right$ in $left$ with $right.c \geq 0$ do
      $right$ := **redTree**($right$, $\mathcal{D}_r$);
      if $right$ = nil then return nil;
    end
  end;
  return $cnode$;
end redTree;
```

When function **blueTree** is called, the number of trees in the set of input forests is bounded by c and the number of input forests is bounded by $\frac{1}{2}(c^2 + c)$.

```
function blueTree($node$: SearchTreeNode): SearchTreeNode;
var
  $pair$: PairOfLeaves;
  $rec$, $child_1$, $child_2$, $child_3$, $child_4$: SearchTreeNode;

begin
  $pair$ := **conflictPair**($node$);

  if $pair$.concflict then
    $child_1$ := **copy**($node$);
    $child_2$ := copy($node$);
    $child_3$ := copy($node$);
    $child_4$ := copy($node$);

    $child_1.\mathcal{D}_l$ := $node.\mathcal{D}_l \cup \{pair.p_1, pair.p_2\}$;
    $node$.child$_1$ := compute($child_1$);

    $child_2.\mathcal{D}_l$ := $node.\mathcal{D}_l \cup \{pair.p_1\}$;
    $child_2.\mathcal{D}_r$ := $node.\mathcal{D}_r \cup \{pair.p_2\}$;
    $node$.child$_2$ := compute($child_2$);

    $child_3.\mathcal{D}_l$ := $node.\mathcal{D}_l \cup \{pair.p_2\}$;
    $child_3.\mathcal{D}_r$ := $node.\mathcal{D}_r \cup \{pair.p_1\}$;
    $node$.child$_3$ := compute($child_3$);

    $child_4.\mathcal{D}_r$ := $node.\mathcal{D}_r \cup \{pair.p_1, pair.p_2\}$;
    $node$.child$_4$ := compute($child_4$);

    for each search-tree leaf $rec$ in $node$ with $rec.c \geq 0$ do
      $rec$ := blueTree($rec$);
    end;
    return node;

  else
    return Complete($node$) ;
  end;
end blueTree;

function conflictPair($node$: SearchTreeNode): PairOfLeaves;
var
  $done$: boolean;
  $L_1, \ldots, L_k$: Leafset;
  $cost_1, cost_2, cost_3, cost_4$: integer;
  $pair$: PairOfLeaves;
  $child_1, child_2, child_3, child_4$: SearchTreeNode;

begin
  $done$ := false;
  $child_1$ := copy(node);
  $child_2$ := copy(node);
```

```
$child_3$ := copy(node);
$child_4$ := copy(node);

for each $pair$ in $node.L$ do
  if not $done$ then
    $child_1.\mathcal{D}_l := node.\mathcal{D}_l \cup \{pair.p_1, pair.p_2\}$;
    $child_2.\mathcal{D}_l := node.\mathcal{D}_l \cup \{pair.p_1\}$;
    $child_2.\mathcal{D}_r := node.\mathcal{D}_r \cup \{pair.p_2\}$;
    $child_3.\mathcal{D}_l := node.\mathcal{D}_l \cup \{pair.p_2\}$;
    $child_3.\mathcal{D}_r := node.\mathcal{D}_r \cup \{pair.p_1\}$;
    $child_4.\mathcal{D}_r := node.\mathcal{D}_r \cup \{pair.p_1, pair.p_2\}$;
    $cost_1 := \sum_{i=1}^{node.k} cost(node.G_i, (child_1.\mathcal{D}_l | child_1.\mathcal{D}_r))$;
    $cost_2 := \sum_{i=1}^{node.k} cost(node.G_i, (child_2.\mathcal{D}_l | child_2.\mathcal{D}_r))$;
    $cost_3 := \sum_{i=1}^{node.k} cost(node.G_i, (child_3.\mathcal{D}_l | child_3.\mathcal{D}_r))$;
    $cost_4 := \sum_{i=1}^{node.k} cost(node.G_i, (child_4.\mathcal{D}_l | child_4.\mathcal{D}_r))$;
    if $\neg\exists\ cost_i = 0\ (i = 1, \dots, 4)$ then
      $done$ := true;
      $pair$.conflict := true;

    else if $\neg\exists\ cost_i = 0\ (i = 2, \dots, 4)$ and $child_1.\mathcal{D}_r = \emptyset$ then
      for $i := 1, \dots, node.k$ do
        $L_i := \{l \in L | cost(nodeG_i, (child_1.\mathcal{D}_l | child_1.\mathcal{D}_r)) = 0\}$;
      end;

      if $\bigcap_{i=1}^{node.k} L_i = \emptyset$ then
        $pair$.conflict := true;

      else
        $pair$.conflict := false;
      end;

    else
      $pair$.conflict := false;
    end
  end;
  return pair;
end conflictPair;
```

Note, that in the case where $\mathcal{D}_r = \emptyset$ for a split $\mathcal{D} = (\mathcal{D}_l|\mathcal{D}_r)$ the set of vertices in a gene forest is not uniquely determined when the forest is mapped into the split. Therefore, we keep building possible solutions under the assumption that a valid result can be found, that is, $\mathcal{D}_r$ contains at least one element of each element in $\mathcal{M}$ (if $\mathcal{M} \neq \emptyset$).

```
function compute(node: SearchTreeNode): SearchTreeNode;
var
  del, $\mathcal{D}_l$, $\mathcal{D}_r$: Leafset;
  $\mathcal{D}$: Split;

begin
  $\mathcal{D}_l := node.\mathcal{D}_l$;
  $\mathcal{D}_r := node.\mathcal{D}_r$;
  $\mathcal{D}.\mathcal{D}_l := \mathcal{D}_l$; $\mathcal{D}.\mathcal{D}_r := \mathcal{D}_r$;

  if $\mathcal{D}_r \neq \emptyset$ then
    $del := Dups(node.G_i, \mathcal{D}\}$;
    $node.G_i := node.G_i - del$;
    $node.c := node.c - |del|$;

  else /* $\mathcal{D}_r = \emptyset$; this is only the case when called from function */
  /* blueTree; i.e., $node.k \leq node.c$; */
    for $i := 1, \dots, node.k$ do
      $L_i := \{l \in L | cost(node.G_i, \mathcal{D}) = 0\}$;
    end;

    if $\sum_{i=1}^{node.k} cost(node.G_i, \mathcal{D}) > 0$ then
      let $node.G$ be an element from $node.G_1, \dots, node.G_k$
        with $cost(node.G, \mathcal{D}) > 0$;

      if $node.G$ is a tree then
        $del := \{v \in V_{node.G} | LS(v) \cap \mathcal{D}_l \neq \emptyset$ and $(LS(v_l) \cap \mathcal{D}_l \neq \emptyset$ or
          $LS(v_r) \cap \mathcal{D}_l \neq \emptyset)$ and $\forall l \in Ll \in LS(v)\}$;
        $node.G := node.G - del$;
        $node.c := node.c--;$

        if $\sum_{i=1}^{node.k} cost(node.G_i, \mathcal{D}) > 0$ and $node.c \geq 0$ then
          $node := \mathbf{compute}(node);$
        end;

      else
      /* $node.G$ is a forest consisting of the trees $g_1$ and $g_2$ */
      /* (otherwise $cost(node.G, \mathcal{D}) = 0$) */
        $node.\mathrm{child}_1 := \mathrm{copy}(node);$
        $node.\mathrm{child}_2 := \mathrm{copy}(node);$
        $M_1$ := the leafset of $node.G$ such that $cost(g_1, \mathcal{D}) > 0;$
        $del := \{v \in V_{node.G} | LS(v) \cap \mathcal{D}_l \neq \emptyset$ and $(LS(v_l) \cap \mathcal{D}_l \neq \emptyset$ or
          $LS(v_r) \cap \mathcal{D}_l \neq \emptyset)$ and $\forall l \in M_1 l \in LS(v)\};$
        $node.\mathrm{child}_1.G := node.G - del;$
        $node.\mathrm{child}_1.c := node.c - |del|;$
        $node.\mathrm{child}_1.\mathcal{M} := node.\mathcal{M} \cup \{M_1\};$

        if $\sum_{i=1}^{node.k} cost(node.G_i, \mathcal{D}) > 0$ and $node.c \geq 0$ then
          $node.\mathrm{child}_1 := \mathbf{compute}(node.\mathrm{child}_1);$
        end;
        $M_2$ := the leafset of $node.G$ such that $cost(g_2, \mathcal{D}) > 0;$
        $del := \{v \in V_g | LS(v) \cap \mathcal{D}_l \neq \emptyset$ and $(LS(v_l) \cap \mathcal{D}_l \neq \emptyset$ or
          $LS(v_r) \cap \mathcal{D}_l \neq \emptyset) and \forall l \in M_2 l \in LS(v)\};$
        $node.\mathrm{child}_2.G := node.G - del;$
        $node.\mathrm{child}_2.c := node.c - |del|;$
        $node.\mathrm{child}_2.\mathcal{M} := node.\mathcal{M} \cup \{M_2\};$

        if $\sum_{i=1}^{node.k} cost(node.G_i, \mathcal{D}) > 0$ and $node.c \geq 0$ then
          $node.\mathrm{child}_2 := \mathbf{compute}(node.\mathrm{child}_2);$
        end
      end;

    else
    /* $\bigcap_{i=1}^{node.k} L_i = \emptyset$ */
      for $i := 1, \ldots, node.k$ do /* $k \leq \frac{1}{2}(c^2 + c)$ */
```

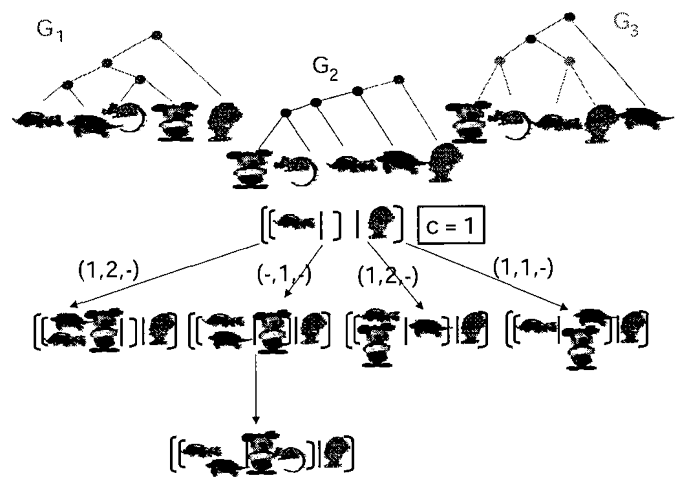
```
if there is exactly one tree $g$ in $node.G_i$ with $cost(g, \mathcal{D}) > 0$ then
  $del := \{v \in V_{node.G_i} | LS(v) \cap \mathcal{D}_l \neq \emptyset$ and $(LS(v_l) \cap \mathcal{D}_l \neq \emptyset$ or
    $LS(v_r) \cap \mathcal{D}_l \neq \emptyset)$ and $\forall l \in (L - L_i) l \in LS(v)\}$;
  $node.\text{child}_i.\mathcal{M} := node.\mathcal{M} \cup \{L - L_i\}$;
  $node.\text{child}_i.G_i := node.G_i - del$;
  $node.\text{child}_i.c := node.c - |del|$ ;

  if $\sum_{i=1}^{k} cost(node.\text{child}_i.G_i, \mathcal{D}) > 0$ and $node.c \geq 0$ then
    $node.\text{child}_i :=$ compute($node.\text{child}_i$);
  end;

else
  let $g_1, g_2$ be the trees in $node.G_i$ with
    $cost(g_1, \mathcal{D}) > 0$ and $cost(g_2, \mathcal{D}) > 0$;
  $node.\text{child}_i := \text{copy}(node)$;
  $node.\text{child}_i.\text{child}_1 := \text{copy}(node)$;
  $node.\text{child}_i.\text{child}_2 := \text{copy}(node)$;
  $M_1 :=$ the leafset of $node.G$ such that $cost(g_1, \mathcal{D}) > 0$;
  $del := \{v \in V_{node.G} | LS(v) \cap \mathcal{D}_l \neq \emptyset$ and $(LS(v_l) \cap \mathcal{D}_l \neq \emptyset$ or
    $LS(v_r) \cap \mathcal{D}_l \neq \emptyset)$ and $\forall l \in M_1 l \in LS(v)\}$;
  $node.\text{child}_i.\text{child}_1.G := node.G - del$;
  $node.\text{child}_i.\text{child}_1.c := node.c - |del|$;
  $node.\text{child}_i.\text{child}_1.\mathcal{M} := node.\mathcal{M} \cup \{M_1\}$;

  if $\sum_{i=1}^{k} cost(node.\text{child}_i.\text{child}_1.G_i, \mathcal{D}) > 0$ and
  $node.\text{child}_i.\text{child}_1.c \geq 0$ then
    $node.\text{child}_i.\text{child}_1 :=$ compute($node.\text{child}_i.\text{child}_1$);
  end
  $M_2 :=$ the leafset of $node.G$ such that $cost(g_2, \mathcal{D}) > 0$;
  $del := \{v \in V_{node.G} | LS(v) \cap \mathcal{D}_l \neq \emptyset$ and $(LS(v_l) \cap \mathcal{D}_l \neq \emptyset$ or
    $LS(v_r) \cap \mathcal{D}_l \neq \emptyset)$ and $\forall l \in M_2 l \in LS(v)\}$;
  $node.\text{child}_i.\text{child}_2.G := node.G - del$;
  $node.\text{child}_i.\text{child}_2.c := node.c - |del|$;
  $node.\text{child}_i.\text{child}_2.\mathcal{M} := node.\mathcal{M} \cup \{M_2\}$;

  if $\sum_{i=1}^{k} cost(node.\text{child}_i.\text{child}_2.G_i, \mathcal{D}) > 0$ and
  $node.\text{child}_i.\text{child}_2.c \geq 0$ then
```

```
            node.child_i.child_2 := compute(node.child_i.child_2);
        end
    end
  end;
  return node;
end compute;

function Complete(node: SearchTreeNode): SearchTreeNode;

Follows Lemma 7.6.

end Complete;
```

To see that the algorithm described above has a fixed-parameter-tractable running time for parameter c, we consider the size of the search tree built. In the function **geneDuplication** the root of the search tree is created. Then the instance at the root either is completed without any branching, calls the function **blueTree**, or generates at most c branches, where on each branch a complete instance is computed and c is decreased.

In **blueTree**, either a conflict pair is computed, or the split is completed (cf. Lemma 7.6). In case of the existence of a conflict pair, there are at most $\frac{1}{2}(c^2 + c)$ branches. At each branch, c is decreased.

The work to do at each node of the search tree, i.e., to decide whether to complete the split or to branch, can clearly be accomplished in polynomial time.



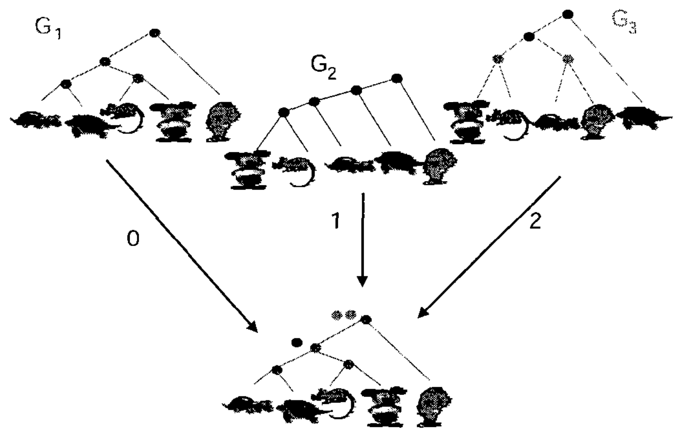


Figure 7.2: continued

Chapter 8

On the Multiple-Gene-Duplication Problem

In this chapter we study the complexity of MULTIPLE GENE DUPLICATION (Problem 5.3). All the results we published in [26] (joint work with Fellows and Hallett). The underlying model is the MULTIPLE-GENE-DUPLICATION MODEL introduced in Section 5.4. A restricted version of MULTIPLE GENE DUPLICATION has the species tree S known. We state it as follows:

Problem 8.1. MULTIPLE GENE DUPLICATION II
Input: Set of gene trees $G_1, \ldots, G_k$, a species tree S, integer c.
Question: Does there exist functions $loc_{G_i,S}$, $event_{G_i,S}$ $(1 \leq i \leq k)$ such that S receives $G_1, \ldots G_k$ with at most c multiple gene duplications (i.e., $cost_{MG}(G_1, \ldots, G_k, S) \leq c$)?

By reduction to and from a combinatorial problem called the BALL-AND-TRAP GAME, we show $\mathcal{W}[1]$-hardness and $\mathcal{NP}$-completeness for MULTIPLE GENE DUPLICATION II. We introduce the BALL-AND-TRAP GAME in Section 8.1. In Section 8.3 we show $\mathcal{W}[1]$-hardness and $\mathcal{NP}$-completeness of two different parameterizations of the BALL-AND-TRAP GAME. The intractability of MULTIPLE GENE DUPLICATION II is presented in Section 8.4.

8.1 The Ball-and-Trap Game

The BALL-AND-TRAP GAME is played on a rooted tree $T = (V, E)$ decorated with a set of *traps* D and a set of *balls* B. Let $L \subseteq V$ be the set of leaves of T. Every ball and trap has a color associated with it; this is given by the functions $c_B : B \to [1 : k]$ and $c_D : D \to [1 : k]$, respectively. The balls and traps are initially associated with internal vertices of T by means of the *attaching functions* $l_B : B \to V - L$ and $l_D : D \to V - L$. Each ball $b \in B$ of color $c_B(b)$ is labeled with a (possible empty) subset $R_b \subseteq D$ of traps. For each ball b every trap $d \in R_b$ is of the color $c_D(d) = c_B(b)$. A ball with a given set of traps may occur many times in the tree (i.e., for $b, b' \in B$ $R_b = R'_b$ and $c_B(b) = c_B(b')$ but $l_B(b) \neq l_B(b')$ is possible). Also, a vertex in the tree can be decorated with many different balls and traps.

A *game* consists of some number of moves, after which the score is calculated. The rules of the game are as follows:

1. Balls and traps are initially placed at internal vertices of T according to l_B and l_D.
2. Balls may not move down the tree (i.e., towards the leaves). They may either stay in the same place or move upwards (i.e., towards the root) according to the topology of T. In each turn, a ball b on a vertex v can be moved to the *parent*(v).
3. We say that a trap $d \in D$ is *dangerous* for a ball $b \in B$ if $d \in R_b$. A ball b *sets off* a trap if the ball is placed at the vertex of a trap dangerous for b.
4. When a trap d is *set off* by a ball b, it is removed from the game and replaced by two new balls b_{new}, $b_{new'}$ such that
   (a) $c_B(b_{new}) = c_B(b_{new'}) = c_B(b)$,
   (b) $l_B(b_{new}) = l_B(b_{new'}) = l_D(d)$,
   (c) $R_{b_{new}} = R_{b_{new'}} = R_b - d$ and $R_b = R_b - d$, and
   (d) $R_{b'} = R_{b'} - d$, for all $b' \in B$.

The goal of the game is to minimize the score of tree T which is defined by

$$s_{max}(v) = \sum_{v \in V} \max\{s(1, v), \dots, s(k, v)\}$$

where $s(c, v)$ denotes the number of balls of color c at vertex v in T.

Problem 8.2. BALL AND TRAP—OPTIMIZATION
Input: A rooted tree $T = (V, E)$ ($L \subseteq V$ is the leafset of T), a set of balls B, a set of traps D, two coloring functions $c_B : B \to [1 : k]$ and $c_D : D \to [1 : k]$, two initial attaching functions $l_B : B \to V_T - L$, $l_D : D \to V_T - L$, and for each ball $b \in B$ a set $R_b \subseteq D$ where for each $d \in R_b$ $c_D(d) = c_B(b)$.
A Round: Each round of the game consists of the player moving any number of same colored balls up the tree or deciding not to move any balls (halting move).
Output: The location function $l'(B)$ generated according to the above rules which minimizes $\sum\limits_{v \in V} s_{max}(v)$.

The input is measured as follows: n denotes the size of T, k denotes the number of colors, r denotes the number of traps, and there are at most m balls on any vertex of T in the initial configuration. The above defined game leads to the following decision variant of the problem:

Problem 8.3. BALL AND TRAP—DECISION
Input: A rooted tree $T = (V, E)$ with leafset $L \subseteq V$, a set of balls B, a set of traps D, two coloring functions $c_B : B \to [1 : k]$ and $c_D : D \to [1 : k]$, two initial attaching functions $l_B : B \to V_T - L$, $l_D : D \to V_T - L$, for each ball $b \in B$ a set $R_b \subseteq D$ where for each $d \in R_b$ $c_D(d) = c_B(b)$, and a positive integer t.
Question: Can the BALL-AND-TRAP GAME be played on T to achieve a score of at most t?

Theorem 8.1. MULTIPLE GENE DUPLICATION II *reduces to* BALL AND TRAP—DECISION.

Proof. We construct an instance I' of BALL AND TRAP—DECISION from an instance of I of MULTIPLE GENE DUPLICATION II. Let S be the species tree $S = (V_S, E_S, L)$. Let $T = (V_S, E_S)$ with leafset $L \subseteq V_S$. Furthermore, let $t = c$ and the number of colors k' of I' be equal to the number of gene trees k from I. Thus, each color corresponds to one of the input gene trees. Apply $M(G_i, S)$ (cf. Section 5.2) for the least-common-ancestor mapping $loc_{G_i,S}(u) = lca_S(LS(u))$ for all $u \in V_{G_i}$, $1 \leq i \leq k$.

We create a ball b with $c_B(b) = i$ for every vertex $u \in V_G$ such that $event_{G_i,S}(u) = dup$. Let $l_B(b) = loc_{G_i,S}(u)$. If $loc_{G_i,S}(u) \neq root(S)$, then let

$$Traps(u) = \{v | v \in V_{G_i}, v \text{ is an ancestor of } u, event_{G_i,S}(v) = spec\}.$$

For each $v \in Traps(u)$ we create a trap d and let $l_D(d) = loc_{G_i,S}(v)$ in T and $c_D(d) = i$. Place v in R_b.
We prove that I' is a *yes*-instance of BALL AND TRAP—DECISION if and only if I is a *yes*-instance of MULTIPLE GENE DUPLICATION II. One need only to verify that the legal moves for a ball in the BALL-AND-TRAP GAME correspond to the legal moves for a duplication event for MULTIPLE GENE DUPLICATION.

Clearly, each legal move of a duplication corresponds to a move in the BALL-AND-TRAP GAME. But, what if there is more than one ball attached to a vertex x in the species tree and the balls correspond to duplication events such that there is a series $u_1, u_2, \dots, u_q \in V_{G_i}$ and $event_{G_i,S}(u_p) = dup$, $loc_{G_i,S}(u) = x$ and u_p is a direct descendant of u_{p+1} in G_i, for all $1 \leq p < q$? But then the traps dangerous for these balls are all equivalent. □

Figure 8.1 shows two BALL AND TRAP instances. Figure 8.1(a) is the BALL AND TRAP version for the instance shown in Figure 5.3(b) on page 42. Figure 8.1(b) corresponds to the situation in Figure 5.3(c).

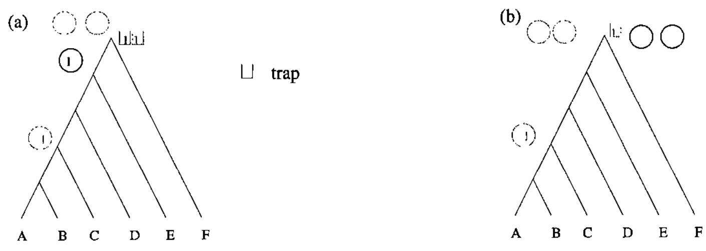


Figure 8.1: Two instances of the BALL-AND-TRAP GAME. Notice in (a) that the red ball located at the vertex which is the least common ancestor of the leaves A, B and C contains the red Trap 1. If it is moved upwards to the root, which is decorated by the red Trap 1, an additional ball will be added to the game (i.e., to the root of the tree) (b).

8.2 Parameterizations of the Ball-and-Trap Game

We consider the following parameterizations of BALL AND TRAP.

Problem 8.4. BALL AND TRAP I
Input: A rooted tree $T = (V, E)$ with leafset $L \subseteq V$, a set of balls B, a set of traps D, two coloring functions $c_B : B \rightarrow [1 : k]$ and $c_D : D \rightarrow [1 : k]$, two initial attaching functions $l_B : B \rightarrow V_T - L$, $l_D : D \rightarrow V_T - L$, for each ball $b \in B$ a set $R_b \subseteq D$ where for each $d \in R_b$ $c_D(d) = c_B(b)$, and a positive integer t.
Parameters: $k = 2$, for each $b \in B$ let $|R_b| \leq 2$, the number of traps r.
Question: Can the BALL-AND-TRAP GAME be played on T to achieve a score of at most t?

Problem 8.5. BALL AND TRAP II
Input: A rooted tree $T = (V, E)$ with leafset $L \subseteq V$, a set of balls B, a set of traps D, two coloring functions $c_B : B \rightarrow [1 : k]$ and $c_D : D \rightarrow [1 : k]$, two initial attaching functions $l_B : B \rightarrow V_T - L$, $l_D : D \rightarrow V_T - L$, for each ball $b \in B$ a set $R_b \subseteq D$ where for each $d \in R_b$ $c_D(d) = c_B(b)$, and a positive integer t.
Parameters: k, r, m, t.
Question: Can the BALL-AND-TRAP GAME be played on T to achieve a score of at most t?

Problem 8.6. BALL AND TRAP III
Input: A rooted tree $T = (V, E)$ with leafset $L \subseteq V$, a set of balls B, a set of traps D, two coloring functions $c_B : B \rightarrow [1 : k]$ and $c_D : D \rightarrow [1 : k]$, two initial attaching functions $l_B : B \rightarrow V_T - L$, $l_D : D \rightarrow V_T - L$, for each ball $b \in B$ a set $R_b \subseteq D$ where for each $d \in R_b$ $c_D(d) = c_B(b)$, and a positive integer t.
Parameters: k, r.
Question: Can the BALL-AND-TRAP GAME be played on T to achieve a score of at most t?

Problem 8.7. BALL AND TRAP IV:
Input: A binary tree $T = (V_T, E_T)$ with leafset $L \subseteq V$, a set of balls B, a set of traps D, two coloring functions $c_B : B \to [1 : k]$ and $c_D : D \to [1 : k]$, two initial attaching functions $l_B : B \to V_T - L$, $l_D : D \to V_T - L$, for each ball $b \in B$ a set $R_b \subseteq D$ where for each $d \in R_b$ $c_D(d) = c_B(b)$, and a positive integer t.
Parameters: $k = 2$ and the number of traps r.
Conditions:

1. $|R_b| \leq 2$, for all $b \in B$.
2. For a vertex $v \in V_T$, let $ps(c, v)$ denote the maximal possible number of balls of color c attached to vertex v. Then for all $v \in V_T$ and each color c, $s(c, v) \leq ps(c, v)$ and

$$\begin{aligned} ps(c,v) \;\leq\; & ps(c,v_l) - s(c,v_l) + ps(c,v_r) - s(c,v_r) \\ & + \begin{cases} 2 & \text{if } s(c,v_l) = s(c,v_r) = 0 \text{ and } v_l, v_r \notin L \\ 1 & \text{if either } s(c,v_l) = 0 \text{ or } s(c,v_r) = 0 \\ 1 & \text{if } s(c,v_l) = s(c,v_r) = 0 \text{ and} \\ & \text{either } v_l \in L \text{ or } v_r \in L \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

3. $R_b = R'_b$ if $l_B(b) = l_B(b')$ and $c_B(b) = c_B(b')$.
4. $R_b \subseteq R_{b'}$ if $l_B(b)$ is an ancestor of $l_B(b')$ in T.
5. No *useless* traps are allowed (a trap d is useless if no ball b in the subtree, where the trap is located, has $d \in R_b$).
6. If $b, b' \in B$ where the vertex $l_B(b)$ is a descendant of the vertex $l_B(b')$, then all the traps $d \in R_b - R_{b'}$ are placed at vertices on the path from b to b' (inclusive).

Question: Can the BALL-AND-TRAP GAME be played on T to achieve a score of at most t?

Before we analyze the complexity of BALL AND TRAP I (Problem 8.4) and BALL AND TRAP IV (Problem 8.7) we want to mention the following two theorems which we published in [26]. These theorems give us hope that a reasonable parameterization for the BALL-AND-TRAP GAME can be found. This would lead to a fixed-parameter tractable algorithm for a parameterization of MULTIPLE GENE DUPLICATION II.

Theorem 8.2. *[26] For every fixed set of parameter values (k, r, m, t), the problem* Ball and Trap II *can be solved in time linear in the size of the tree.*

Theorem 8.3. *[26]* Ball and Trap III *can be solved in time n^c where $c = O((k2^r)^m)$.*

8.3 Intractability of Ball and Trap

We prove the $\mathcal{W}[1]$-hardness of Ball and Trap I and $\mathcal{NP}$-completeness of Ball and Trap by means of a polynomial-time parameterized reduction from the $\mathcal{W}[1]$-complete problem k-Clique (cf. Problem 2.4, Section 2.3.2). As an intermediate step we prove that the following parameterized problem is hard for $\mathcal{W}[1]$.

Problem 8.8. (r, k)-Small Union
Input: A family $\mathcal{F}$ of distinct subsets of $\{1, \ldots, n\}$, positive integers r and k.
Parameter: (r, k)
Question: Is there a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ with $|\mathcal{F}'| \geq r$ such that the union of the sets in $\mathcal{F}'$ has cardinality at most k?

Lemma 8.4. Small Union *is $\mathcal{NP}$-complete and hard for $\mathcal{W}[1]$ parameterized by k and r.*

Proof. Let $\langle G, k\rangle$ be an instance of k-Clique. We transform $\langle G, k\rangle$ into an instance $\left\langle \mathcal{F}, \left(\binom{k}{2}, k\right)\right\rangle$ of (r, k)-Small Union, where $\mathcal{F}$ is the family of 2-element sets corresponding to the edges E of G, with the vertices of G identified with $V = \{1, \ldots, n\}$.

We show that $G = (V, E)$ has a k-clique if and only if $\mathcal{F}$ has a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ with $|\mathcal{F}'| \geq \binom{k}{2}$ such that $|\mathcal{F}'| \leq k$.

$\Rightarrow$ Let V' be a k-clique in G. Then each pair $(a, b) \in V' \times V'$ is an edge in G and therefore $\{a, b\} \in \mathcal{F}$. We define $\mathcal{F}' = \{\{a, b\} | (a, b) \in V' \times V'$. Then $|\mathcal{F}'| = \binom{k}{2}$ and $|\bigcup \mathcal{F}'| = k$.

$\Leftarrow$ Let $\mathcal{F}' \subseteq \mathcal{F}$ be a set of $|\mathcal{F}'| \geq \binom{k}{2}$ elements with $|\bigcup \mathcal{F}'| = k$. We define $V' = \bigcup \mathcal{F}'$. V' is a k-clique in G, since for any $(a, b) \in V' \times V'$ $(a \neq b)$. $\{a, b\} \in \mathcal{F}'$ (Otherwise, there exist x, y with $\{a, x\}, \{b, y\} \in \mathcal{F}'$, $x, y \in V'$. Therefore $|\bigcup \mathcal{F}'| \geq k + 2$. Contradiction.) But then $\{a, b\} \in \mathcal{F}$, that is $(a, b) \in E$.

□

Theorem 8.5.

1. Ball and Trap *is* $\mathcal{NP}$*-complete.*

2. Ball and Trap I *is* $\mathcal{W}[1]$*-hard.*

Proof. Ball and Trap is well-defined for non-binary trees, and, hence, we describe how Small Union can be reduced to Ball and Trap I. Let $\langle \mathcal{F}, (r,k) \rangle$ be an instance of Small Union. We can assume, by the reduction from k-Clique to (r,k)-Small Union (described above), that $\mathcal{F}$ consists of 2-element sets. In order to describe the reduction, we must describe a tree T with decorations, and the target value t for the game on T.

The tree T is a star of degree $n = |\bigcup \mathcal{F}|$ (with the root being the central vertex). Each leaf of T is associated with an element of $\mathcal{F}$. The two colors are *red* and *blue*. There are n red traps $\tau_1, \ldots, \tau_n$. Each leaf is decorated with a single red ball labeled with the set of traps $\{\tau_u, \tau_v\}$ for the associated ("edge") set $\{u,v\}$. The root is decorated with $k+r$ blue balls, each labeled with the empty set of traps. The root is also decorated with all the n red traps. We set $t = (n-r) + (k+r) = n+k$. We show if $\langle \mathcal{F}, (r,k) \rangle$ is a *yes*-instance of Small Union, then the tree described above is a *yes*-instance of Ball and Trap I. Initially, the score is $n+k+r$. The only possible move is to move a red ball from a leaf to the root. If r balls can be moved up to the root from the leaves, with the r balls chosen so that the union of their trap-label sets has cardinality k, then the result is a total of $k+r$ red balls at the root (where there are $k+r$ blue balls, so the cost of the root in the final score remains $k+r$). Thus the score at the end is t.

Conversely, if a score of t is achieved by a game g, then necessarily at least r red balls must be moved up from the leaves. Let g' denote the truncated game consisting of the first r moves. There are two cases to consider:

Case 1 g' also achieves a score of at most t, and

Case 2 the score for the game g' is greater than t.

In Case 1, exactly r red balls are moved to the root and consequently the score for the root vertex is at most $k+r$, which implies that the union of the trap label sets for the balls moved up has cardinality at

most k. This implies that $\langle \mathcal{F}, (r,k) \rangle$ is a *yes*-instance for the SMALL UNION problem.
In Case 2, there are more than $k+r$ red balls at the root after the moves of g'. Since the number of red balls now exceeds the number of blue balls at the root, each further move of g is of no advantage in decreasing the total score, contradicting that g is a game that achieves a score of at most t. □

The next theorem shows that BALL AND TRAP remains $\mathcal{W}[1]$-hard even if restricted to BALL AND TRAP IV.

Theorem 8.6. BALL AND TRAP IV *is* $\mathcal{W}[1]$*-hard.*

Proof. We use the following modified construction from the proof of Theorem 8.5. Again, we reduce from SMALL UNION. Let $\langle \mathcal{F}, (r,k) \rangle$ be an instance of SMALL UNION. As in the proof of Theorem 8.5 we assume, by the reduction described above, that $\mathcal{F}$ consists of 2-element sets. We build a tree T', a preliminary stage of T. The tree T' is a tree isomorphic to a binary caterpillar tree of $|\mathcal{F}| + n$ leaves, where $n = |\bigcup \mathcal{F}|$ and one child of *root*(T') is a leaf. Let u be the internal vertex of T' where $T'(u)$ has exactly $|\mathcal{F}|$ leaves. Each leaf of $T'(u)$ corresponds to an element of $\mathcal{F}$. The colors are *red* and *blue*. There are n red traps $\tau_1, \ldots, \tau_n$. Each leaf of $T'(u)$ is decorated with a single red ball labeled with the set of traps $\{\tau_u, \tau_v\}$ for the associated edge (u,v) (i.e., the corresponding element $\{u,v\} \in \mathcal{F}$). The internal vertices of $T'(u)$ receive neither balls nor traps. The n red traps are placed on the n internal nodes of T', who are not elements of the vertex set of $T'(u)$. To construct T we extend T' such that each leaf of T' becomes an internal vertex of T with two children, each of these children having two leaves. That is, T is a binary tree over $4(\mathcal{F} + n)$ leaves. Finally, the root of T is decorated with $k+r$ blue balls. Figure 8.2 illustrates the construction of T by means of an example. Let $t = n + k$.
We show that $\langle \mathcal{F}, (r,k) \rangle$ is a *yes*-instance of SMALL UNION if and only if the tree described above is a *yes*-instance of BALL AND TRAP IV.

⇒ Initially, the score is $n + k + r$. The red balls are the only possible balls to move. Moving up red balls towards u, which is the vertex inducing the subtree of T having exactly $4|\mathcal{F}|$ balls, does not change the score. If r red balls can be moved up to the root of T, with the r balls chosen so that the union of their trap-labeled sets has cardinality k, then the result is a total of $k + r$ blue balls at

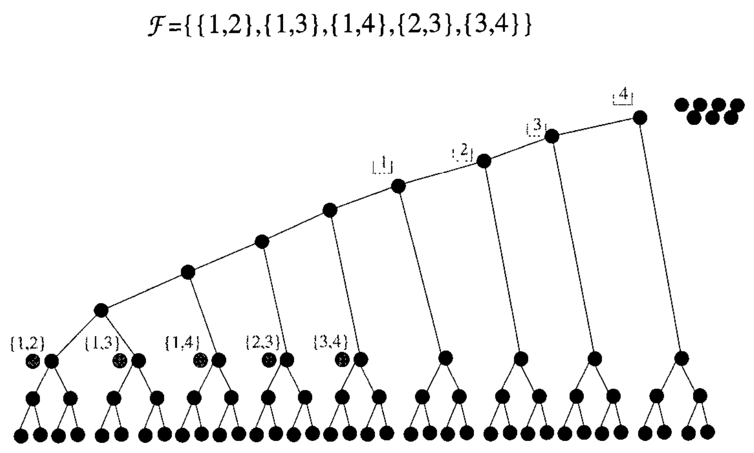


Figure 8.2: An example for the construction of a Ball and Trap IV instance for a (r, k)-Small Union instance. Here $n = 4$, $k = 3$, and $r = 3$.

the root, the score at *root*(T) is $k + r$, and therefore the score in total is t.

$\Leftarrow$ analogous to this direction in the proof of Theorem 8.5 on page 84.

□

8.4 Intractability of Multiple Gene Duplication

The following theorem proves $\mathcal{W}[1]$-hardness and $\mathcal{NP}$-completeness of Multiple Gene Duplication II.

Theorem 8.7. Ball and Trap IV *reduces to* Multiple Gene Duplication II *[26].*

Proof. We construct an instance I' of Multiple Gene Duplication II from an instance I of the $\mathcal{W}[1]$-hard problem Ball and Trap IV (cf. Theorem 8.6). Our reduction builds a species tree $S = (V_S, E_S, L)$ and gene trees G_1 and G_2. I is restricted to two colors; we associate color

1 with G_1 and color 2 with G_2. We give S the topology of $T = (V, E)$, the tree of instance I. That is, $V_S = V$, $E_S = E$, and $L \subseteq V$ is the leafset of T identified with a set of leaf labels.
W.l.o.g. we restrict our attention to balls and traps of only one color c. We describe the construction of gene tree G_c in two steps. In Step I we build the *contradictory topology* τ of G_c. That is, $\tau = (V_\tau, E_\tau, L_\tau)$ is a leaf-labeled tree with $\tau \subseteq G_c$. Attached to the vertices of τ are sets of vertices which help us embed the leaves without influencing the differences between G_c and S (Step II).

Step I: We build the *contradictory topology* $\tau = (V_\tau, E_\tau, L_\tau)$ such that $\tau \subseteq G_c$. For each vertex $v \in V$ one of the following three cases:

Case 1. $v \in L$. We set $\mathit{free}(v) := \{v\}$.

Case 2. $v \in V - L$ and v is decorated by a ball but not by a trap.

1. Pick a leaf $l_1 \in L$ from $\mathit{free}(v_l)$.
2. Pick a leaf $l_2 \in L$ from $\mathit{free}(v_r)$.
3. Create a new vertex w in τ.
4. $w := \mathit{parent}_\tau(v_l)$, $w := \mathit{parent}_\tau(v_r)$
5. $f_1 := \mathit{free}(v_l) - \{l_1\}$, $f_2 := \mathit{free}(v_l) - \{l_2\}$
6. For each ball b at v:
   - If there is an element in $f_1 \cup f_2$, which is an internal vertex in τ, then pick such an internal vertex $e \in f_1 \cup f_2$. Otherwise pick a leaf $e \in f_1 \cup f_2$, $e \in L$.
   - If $e \in f_1$ then $f_1 := f_1 - e$, otherwise $f_2 := f_2 - e$.
   - Create a new vertex w' in τ.
   - $w' := \mathit{parent}_\tau(w), w' := \mathit{parent}_\tau(w)$.
   - Rename w' by w.
7. $\mathit{free}(v) := f_1 \cup f_2 \cup \{w\}$.

Case 3. $v \in V - L$ and v is decorated by a trap.

1. If there exists an internal vertex of τ in $\mathit{free}(v_l)$ and $T(v_l)$ has a ball b with $d \in R_b$, then pick such an internal vertex e_1, $e_1 \in free(v_l)$.
2. If there exists an internal vertex of τ in $\mathit{free}(v_r)$ and $T(v_r)$ has a ball b with $d \in R_b$, then pick such an internal vertex e_1, $e_1 \in free(v_r)$.

Note that at least one element of e_1 and e_2 is an internal vertex of τ (cf. Condition 5, Problem 8.7).

```
3. Create a new vertex $w$ in $\tau$.
4. $w := parent_\tau(e_1)$, $w := parent_\tau(e_2)$.
5. For each ball $b$ at $v$ do:
    • If there is an element in $f_1 \cup f_2$ which is an internal
      vertex in $\tau$ then pick such an internal vertex $e \in$
      $f_1 \cup f_2$, otherwise pick a leaf $e \in f_1 \cup f_2$, $e \in L$.
    • If $e \in f_1$ then $f_1 := f_1 - e$, otherwise $f_2 := f_2 - e$.
    • Create a new vertex $w'$ in $\tau$.
    • $w' = parent_\tau(w), w' = parent_\tau(w)$.
    • Rename $w'$ by $w$.
6. $free(v) := f_1 \cup f_2 \cup \{w\}$.
```

Case 4. $v \in V - L$ and v is not decorated. We set $free(v) :=$ $free(v_l) \cup free(v_r)$.

It is easy to verify that now $free(root(T)) = L - L_\tau$.

Step II: For embedding the leaves $L - L_\tau$ in G_c, we complete G_c from τ by embedding the remaining leaves $free(root(T))$ in accordance to the topology of T. We build the maximal subtrees $T_v = (V_{T_v}, E_{T_v}, L_{T_v})$ of T over the elements of *free*. For each such tree T_v, we compute the sibling w of v in S and specify p, the least common ancestor of the leaves of L_{T_v} in τ. Then we subdivide edge $(p, parent_\tau(p))$ in (p, p') and $(p', parent_\tau(p))$ and add T_v as the sibling of p in τ.

The proof that I' is a *yes*-instance from MULTIPLE GENE DUPLICATION II if and only if I is a *yes*-instance from BALL AND TRAP IV follows directly from the fact that the moves for a ball correspond to the legal moves of a duplication. □

Part III

Resolving Conflicting Sequences Using Vertex Cover

In this part we study the resolution of conflict graphs using the problem VERTEX COVER (cf. Problems 2.1 and 2.3). We motivate the problem with an application in computational biology, namely the fundamental problem of constructing *Multiple Sequence Alignments* (MSAs). The problem is, for a given set of biological sequences (e.g., DNA or protein sequences) to determine whether the sequences display sufficient similarity to justify the inference of homology (i.e., the existence of a common ancestor of the sequences). Therefore the goal is to find a good alignment of the sequences. Given a perfect MSA, one can predict the evolution of the sequences.

The known algorithms computing MSAs usually fail to produce an exact solution corresponding to the underlying model due to the $\mathcal{NP}$-hardness of the problem [13, 34, 39, 44, 71]. The main problem, from a biological point of view, is the misplacement of *gaps* (i.e., insertion and deletion events during evolution). Therefore we can view the problem of computing MSAs as the problem of inserting gaps at the correct places [48, 49]. Assuming that all the input sequences to the multiple-sequence-alignment problem are homologous, there exists an evolutionary tree corresponding to the MSA. We further assume that we can construct the tree directly from a (biologically) perfect MSA of the input sequences (cf. Figure 9.0).

```
1. KETAAAKFQRQHMDSSTSSASSSN_YCNQMMKSRNM_SDRC
2. KESAAAKFERQHIDSSTSSVSSSN_YCNQMMTSRNL_QDRC
3. KESAAAKFERQHMDPSPSSASSSN_YCNQMMQSDRLTQDRC
4. __QDWSSFQNKHIDYPETSASNPNAYCDLMMQRRNLNPTKC
5. __TRYEKFLRQHVDYPKSSAPDSRTYCNQMMQRRGMTSPVC
```

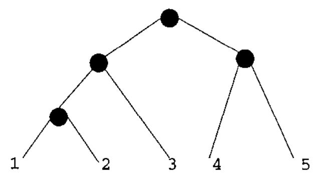


Figure 9.0: The upper part of this figure depicts an MSA for five amino-acid sequences whereas the lower figure depicts its corresponding evolutionary tree. The sequences correspond to the leaves of the tree, the internal nodes represent the ancestry. The gaps are represented as '_'.

Once we are given an MSA with misplaced gaps, it is possible that conflicts prevent the tree construction. A gap in a sequence of a computed MSA M is defined by its location in the sequence. Two sequences in M share a common gap if both sequences have a gap at the same location.

```
1. KETAAAKFQRQHMDSSTSSASSSN_YCNQMMKSRNM_SDRC
2. KESAAAKFERQHIDSSTSSVSSSN_YCNQMMTSRNL_QDRC
3. KESAAAKFERQHMDPSPSSASSSN_YCNQMMQSDRLTQDRC
4. __QDWSSFQNKHIDYPETSASNPNAYCDLMMQRRNLNPTKC
5. __TRYEKFLRQHVDYPKSSAPDSRTYCNQMMQRRGMTSPVC

   g1                      g2          g3
```

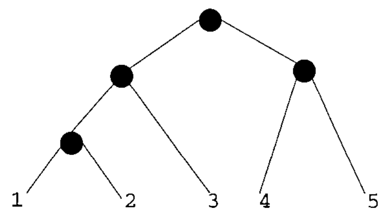
```
1.KETAAAKFQRQHMDSSTSSASSSN_YCNQMMKSRNM_SDRC
2.KESAAAKFERQHID_STSSVSSSN_YCNQMMTSRNL_QDRC
3.KESAAAKFERQHMD_SPSSASSSN_YCNQMMQSDRLTQDRC
4.__QDWSSFQNKHIDYPETSASNPNAYCDLMMQRRNLNPTKC
5.__TRYEKFLRQHVDYPKSSAPDSRTYCNQMMQRRGMTSPVC

  g1            g2        g3          g4
```

Figure 9.1: The MSA from Figure 9.0 contains three different gaps as shown in the upper figure: g_1 is shared by sequences 4 and 5, g_2 by sequences 1,2, and 3 and g_3 by sequences 1 and 2. An MSA with four different gaps is depicted in the lower figure: g_1 is shared by sequences 4 and 5, g_2 by sequences 2, and 3, g_3 by sequences 1,2, and 3, and g_4 by sequences 1 and 2.

Let g_1 and g_2 be two gaps in a given MSA M. We assume gap g_1 appears in the set of sequences A, and gap g_2 appears in the set of sequences B. We say g_1 and g_2 have a conflict with respect to MSA M if $A \not\subseteq B$, $(M - A) \not\subseteq B$, $A \not\subseteq (M - B)$, and $(M - A) \not\subseteq (M - B)$. In other words the two gaps overlap in the MSA.

One way to resolve this problem is to compute the minimum number of gaps such that the MSA ignoring these gaps is conflict free and therefore a tree construction corresponding to the MSA is possible.

Representing the gaps as vertices and the conflicts as edges between the vertices corresponding to the conflicting gaps, we model the pro-



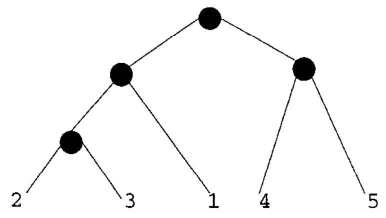


Figure 9.2: Two possible evolutionary trees corresponding to the MSA in Figure 9.1, lower figure. In the one tree (upper figure) sequences 1 and 2 are closer related to each other (corresponding to gap g_2) than sequence 3 to one of them; in the other tree sequences 2 and 3 are siblings (corresponding to gap g_4.)

blem by means of a conflict graph. We compute the minimum number of vertices covering all edges (cf. Figure 9.3). Thus, we have the problem transformed into VERTEX COVER (Problem 2.1) [49]. Note that the size of the input graph in this application of vertex cover is bounded by approximately 20 vertices due to the small number of gaps appearing in a realistic MSA.

VERTEX COVER is also useful when the problem is to compute the minimum number of sequences in a given database representing all domains appearing in the sequences [50]. We represent the sequences as vertices and two vertices are adjacent if and only if they have a common domain. The corresponding graph problem to solve is the $\mathcal{NP}$-complete problem DOMINATING SET [31]. We introduce the parameterized version

Problem 9.0. k-DOMINATING SET
Input: A graph $G = (V, E)$, a positive integer k.
Parameter: k
Question: Does G have a *dominating set* of size $\leq k$ (i.e., does there

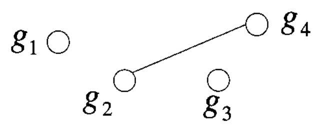


Figure 9.3: The conflict graph corresponding to the MSA in Figure 9.1, lower figure. The conflict between gaps g_2 and g_4 is represented by edge (g_2, g_4).

exist a subset $V' \subseteq V$, $|V'| \leq k$, such that for each vertex $v \in V$: $v \in V'$ or $N(v) \cap V' \neq \emptyset$)?

Since k-Dominating Set is known to be $\mathcal{W}[2]$-complete [16] and every vertex cover in a graph is also a dominating set, it might be a useful method to compute a minimum vertex cover as a first step of a heuristic for solving Dominating Set. (Note that every graph has a dominating set of size $\lfloor \frac{n}{2} \rfloor$.)

In Part III we develop a new fixed-parameter-tractable algorithm for k-Vertex Cover, the parameterized version of Vertex Cover (cf. Problem 2.3, 15). We first describe the basic ideas of the known fixed-parameter-tractable algorithms of k-Vertex Cover (Chapter 9). The best algorithm so far by Niedermeier and Rossmanith runs in time $O(k|V|+(1.29175)^k \cdot k^2)$ [58]. We present a new improved fixed-parameter tractable algorithm with a time complexity of $O(k|V|+r^k k)$, $r \approx 1.2906$ (Chapter 10). We improve the ***klam value*** of 143 by 16 to 157. Besides a further improvement of the search tree and a better analysis of the running time, we also developed a better kernelization. In Chapter 11, we compare an implementation of our fixed-parameter-tractable algorithm with two algorithms based on heuristics for Vertex Cover.

Chapter 9

Known $\mathcal{FPT}$ Algorithms for k-Vertex Cover

In Section 2.3.2, we introduced the k-VERTEX COVER problem (Problem 2.3) which was one of the first problems shown to be fixed-parameter tractable [19, 28]. In this section, we describe the basic ideas of the known fixed-parameter-tractable algorithms for k-VERTEX COVER. The first of those algorithms was presented by Fellows with a bounded search-tree algorithm and a running time of $O(2^k n)$ (cf. Section 2.3.2, page 16 and [24]). The algorithm due to S. Buss has an $O(kn + k^{2k+2})$ time complexity [11]. Papadimitriou and Yannakakis, while proving that k-VERTEX COVER is in $\mathcal{P}$ when k is restricted to $O(\log n)$, provided an $O(3^k n)$ algorithm [60]. Though this algorithm does not have a better complexity, by using an observation due to Buss, Downey and Fellows improved the running time of this algorithm to $O(kn + 3^k k^2)$ [19]. Downey and Fellows present a different algorithm that runs in time $O(kn + 2^k k^2)$ [18]. Balasubramanian, Fellows, and Raman improved the running-time bound to $O(kn + r^k k^2)$, $r \approx 1.3247$, with an improved search-tree algorithm [4]. This algorithm has been improved by Downey, Fellows, and Stege by using a better kernelization of the input graph to obtain a running time of $O(kn + r^k \cdot k^2)$, $r \approx 1.3195$ [21]. The algorithm by Niedermeier and Rossmanith runs in time $O(kn + r^k \cdot k^2)$, $r \approx 1.2917$, using a further improved search tree [58].

9.1 Buss' Algorithm

This section describes the algorithm by Buss which runs in $O(kn+k^{2k+2})$ [11]; the klam value of this algorithm is 9. Given a graph $G = (V, E)$ and a positive integer k, it checks whether it has a vertex cover of size k. The algorithm is based on the method of reduction to a problem kernel (cf. 2.3.3, page 17). The following algorithm assumes that the instance graph $G = (V, E)$ is given in the adjacency-list representation.

Step 1 Find the set H of all vertices of degree more than k in G. Let $|H| = b$. If $b > k$ then answer *no*. Otherwise include H in the vertex cover, remove H and the edges incident to H from G. Let $k' = k - b$. Remove any resulting isolated vertices.

Step 2 If the resulting graph has more than $k \cdot k'$ edges, then answer *no*.

Step 3 Find by brute-force whether the resulting graph has a vertex cover of size k'. If so then answer *yes*. Otherwise answer *no*.

9.2 The Algorithm by Papadimitriou and Yannakakis

This algorithm has a running time of $O(3^k n)$ [60]. Papadimitriou and Yannakakis investigated the complexity of some $\mathcal{NP}$-hard problems when the parameter k is restricted to be logarithmic in the input size and designed the following algorithm to show that the k-VERTEX COVER problem is in $\mathcal{P}$ when k is restricted to $O(\log n)$.

Step 1 Find a maximal matching in the graph (A *matching* is a subset of the edges of the graph such that no two elements have a common vertex). Let the size of the matching be m. If $m > k$ answer *no*. If $2m \leq k$, then answer *yes*. The $2m$ vertices form a vertex cover.

Step 2 Let U be the set of the endpoints of the m edges of the maximal matching. For every edge of the matching, either one of the end points or both are in any vertex cover of G. Furthermore, once a subset of U is picked in a vertex cover, the rest of the vertex cover is uniquely determined: a vertex in $V - U$ is included in the vertex cover if and only if there is an edge incident with it whose other end point (which is in U) has not been picked in the vertex cover. Thus, cycle through the 3^m subsets of U (by picking either one or both of the endpoints of each edge in the matching) and check, for each subset whether it, along with its unique extension to V, is of size at most k. If it is so for any subset, answer *yes*, otherwise answer *no*.

By preprocessing the entire graph by applying Steps 1 and 2 of Buss' algorithm (cf. Section 9.1), we can assume that the resulting graph G has $O(k^2)$ vertices and edges after spending $O(kn)$ time, ignoring the singletons in G since none of them has to be included in a minimum vertex cover.[1] Thus, the bound for the algorithm reduces to $O(kn+3^k k^2)$ [19].

9.3 The Algorithm by Balasubramanian, Fellows, and Raman

This algorithm has a running time $O(kn+(1.324718)^k k^2)$ [4] and consists of two steps: a preprocessing step, based on the method of reduction to a problem kernel (cf. Section 2.3.3, page 16) and a step based on the method of bounded search trees (cf. Section 2.3.3, page 17). The klam value of this algorithm is 129.

Step 1 Additionally to Steps 1 and 2 of Buss' algorithm (cf. Section 9.1) included in the vertex cover are

- all neighbors of vertices with degree 1
- Let v be a vertex of degree 2 and y and z be the neighbors of v. Then we include y and z in the vertex cover if $(y, z) \in E$.
- Let x, y, z be vertices of degree 2, and let $N(x) = \{y, z\}$, $N(y, z) = \{x, a\}$. Then we include $\{x, a\}$ in the vertex cover.

The number of vertices and edges of the resulting graph is $O(k^2)$.

[1]To verify the size of $O(k^2)$ of the kernel, assume that G has more than $k^2 + k$ vertices. Then there is a vertex v in each k-vertex cover of G and v must be incident to more than k vertices; but then $\deg(v) > k$. Therefore, $|E| \leq k^2$ and $|V| \leq k(1+k)$. The running time of the preprocessing comes from the fact, that we only need include at most k vertices of degree more than k in a k-vertex cover (otherwise we can answer *no* and thus we can delete each of the at most k vertices in time $O(n)$ from G, assuming an adjacency-list representation of the graph.

Step 2 The search tree is built as follows. The root consists of the kernelized graph problem $\langle G, k\rangle$. Note that we treat the last case after the others, that is, the graph left is 4-regular.

1. If $\deg(v) = 2$, let $N(v) = \{y, z\}$. Since $(y, z) \notin E$, w.l.o.g. either v and $N(y, z)$ or y and z are in a minimum vertex cover. Therefore we branch in $\langle G', k - |N(y, z)|\rangle$ and $\langle G'', k - 2\rangle$. G' is the graph resulting from G by including $N(y, z)$ in the vertex cover, for G'' y and z are included in the vertex cover.
2. If $\deg(v) \geq 5$ then either v or all it's neighbors are included in the vertex cover. The branches are accordingly.
3. If $\deg(v) = 3$, let $N(v) = \{x, y, z\}$.
   (a) $(x, y) \in E$. Include either $\{x, y, z\}$ or $N(z)$ in the vertex cover.
   (b) There is a common neighbor w, $v \neq w$, between a pair of the vertices x, y, and z. Either x, y, and z or v and w are included in the vertex cover.
   (c) There are no edges among x, y, and z, and x has at least three neighbors other than v. We include either $\{x, y, z\}$, $N(x)$ or $N(y, z) \cup \{x\}$.
   (d) There are no edges among x, y, and z and each of x, y, and z has, apart from v, exactly two private neighbors. Let $N(x) = \{v, a, b\}$. Either $\{x, y, z\}$, $\{v, a, b\}$, or $N(y, z, a, b)$ are included in the vertex cover.
4. Let $N(v) = \{a, b, c, d\}$.
   (a) If $(a, b) \in E$, then we include in the vertex cover either $\{a, b, c, d\}$, $N(c)$, or $\{c\} \cup N(d)$.
   (b) If there is no edge among a, b, c, and d, and a, b, and c share a common neighbor $w \neq v$ then we include in the vertex cover either $\{a, b, c, d\}$ or $\{v, w\}$.
   (c) If there is no edge among a, b, c, and d, each one of them has three neighbors other than v, and no three of them have a common neighbor other than v, then either $\{a, b, c, d\}$, $N(b)$, $\{b\} \cup N(d)$ or $\{b, d\} \cup N(a, c)$ are included in the vertex cover.

The running time of $O(kn + (1.324718)^k k^2)$ is obtained as follows. As a preprocessing, we can do Step 1 (the kernelization and reduction) in time $O(kn)$. After applying Step 1 we are left with a graph of size $|G| = O(k^2)$. The complexity determining a k-vertex cover of the kernelized graph consists of the size of the search tree times $O(k^2)$, because $O(k^2)$ is the work to do for finding a vertex to branch at and the branching itself. The size $f(k)$ of the search tree we obtain from the most expensive branching rule 3 (b), which implies that $f(k) \leq f(k-3)+f(k-2)+1$ (i.e., 3 or 2 vertices are included in the vertex cover via the branching). The polynomial corresponding to this recurrence equation is $x^3 - x - 1 = 0$ with the solution $x \approx 1.3247$.[2]

9.4 The Algorithm by Downey, Fellows and Stege

The main idea here is an improved kernelization, which is not only applied as a preprocessing step, but also reapplied after each branching in the search tree. The graph is reduced to a graph having no vertices of degree less than four and no vertices having a degree of more than k. These rules are described more explicitly (but also generalized) in Section 10. Thus, we build the search tree from branching at vertices of degree at least four as described below. We published his algorithm in [21].

Following Observation 10.1, we perform this branching if there is a vertex of degree at least 6. By repeating this branching procedure, at each step reapplying the reduction rules, we can assume, that at each leaf of the resulting search tree we are left with considering a graph where every vertex has degree 4 or 5. If there is a vertex v of degree 4, then the following branching rules are applied. Suppose that the neighbors of a vertex v are $N(v) = \{a, b, c, d\}$. We consider various cases according to the number of edges present among the vertices a, b, c, d.

Note that, if not all of the vertices in $\{a, b, c, d\}$ are in a vertex cover, then we can assume that at most two of them are.

Case 1. $G|_{N(v)}$ has an edge, say (a, b). Then it is not possible that c and d are in a minimum vertex cover at once, unless all four vertices of a, b, c, d are there. We can conclude that one of the following cases is necessarily a subset of the vertex cover C and we branch accordingly:

[2]For further information about recurrence equations cf. [47].

1. $\{a,b,c,d\} \subseteq C$
2. $N(c) \subseteq C$
3. $\{c\} \cup N(d) \subseteq C$.

Case 2. The subgraph $G|_{N(v)}$ is empty. We consider three subcases.

Subcase 2.1 Three of the vertices (say a, b, and c) have a common neighbor y other than v.

If not all elements of $\{a,b,c,d\}$ are in a k-vertex cover, v and y must be. We can conclude that one of the following is a subset of vertex cover C and branch accordingly:

1. $\{a,b,c,d\} \subseteq C$
2. $\{v,y\} \subseteq C$.

Subcase 2.2 If Subcase 2.1 does not hold, then there may be a pair of vertices who has a total of six neighbors other than v, suppose a and b. If all of a,b,c,d are not in the vertex cover C then $c \notin C$, or $c \in C$ and $d \notin C$, or both $c,d \in C$ (in which case $a,b \notin C$). We can conclude that one of the following sets is a subset of the vertex cover C and branch accordingly:

1. $\{a,b,c,d\} \subseteq C$
2. $N(c) \subseteq C$
3. $\{c\} \cup N(d) \subseteq C$
4. $\{c,d\} \cup N(a,b) \subseteq C$.

Subcase 2.3 If Subcases 2.1 and 2.2 do not hold, then the graph must have the following structure in the vicinity of v:

(1) v has four neighbors a,b,c,d and each of these has degree four.

(2) There is a set E of six vertices such that each vertex in E is adjacent to exactly two vertices in $\{a,b,c,d\}$, and the subgraph induced by $E \cup \{a,b,c,d\}$ is a subdivided K_4 with each edge subdivided once.[3]

In this case we can branch according to:

[3]i.e., w.l.o.g. $G|_{E\cup\{a,b,c,d\}}$ has the edge set $\{(a,v_1), (v_1,b), (b,v_2), (v_2,c), (c,v_3), (v_3,d), (d,v_4), (v_4,a), (a,v_5), (v_5,c), (b,v_6), (v_6,d)\}$

1. $\{a, b, c, d\} \subseteq C$
2. $(E \cup \{v\}) \subseteq C$.

If the graph G is regular of degree 5 (that is, there are no vertices of degree 4 to apply one of the above branching rules to) and none of the reduction rules can be applied, then we choose a vertex v of degree 5 and do the following. First, we branch from $\langle G, k\rangle$ to $\langle G - v, k - 1\rangle$ and $\langle G - N[v], k - 5\rangle$. Then we choose a vertex u of degree 4 in $G - v$ and branch according to one of the above cases. The net result of these two combined steps is that from $\langle G, k\rangle$ we have created a subtree where one of the following cases holds:

1. There are four children with parameter values $k - 5$, from Case 1.
2. There are three children with parameter values $k_1 = k - 5$, $k_2 = k - 5$ and $k_3 = k - 3$, from Subcase 2.1.
3. There are five children with parameter values $k_1 = k - 5$, $k_2 = k - 5$, $k_3 = k - 5$, $k_4 = k - 6$ and $k_5 = k - 9$, from Subcase 2.2.

The bottleneck recurrence comes from the degree 5 situation which produces four children with parameter values $k-5$. The total running time of the algorithm is therefore $O(r^k k^2 + kn)$, where $r = 4^{1/5}$, or $r \approx 1.3195$ approximately. The klam value is 130.

9.5 The Algorithm by Niedermeier and Rossmanith

This algorithm [58] is an extension of the algorithm by Balasubramanian, Fellows, and Raman (cf. Section 9.3, [4]). As done in [4] (cf. Section 9.3), the reduction to a problem kernel is used as a preprocessing step only. The search tree is improved; here the main idea is that in each branch of the search tree there is at most one graph being c-regular for each c, and therefore the most expensive branching leads to a running time of $O(kn + r^k \cdot k^2)$, $r \approx 1.2917$, and a klam value of 141.

Since in Section 10 we make only use of branching rules using vertices of degree 4, 5 and 6, we skip the exact description of the more complicated branching rules in the search tree for the vertices of degree 3 end refer to [58].

Case 1 If there is a vertex v with degree 1, then branch according to $N(v)$ (and nothing else).

Case 2 If there is a vertex v with degree 6 or more, then branch according to v and $N(v)$.

Case 3 If there are no vertices with degree 1 or at least 6, but there is a vertex with degree 2 then proceed as follows.

1. If the graph is 2-regular, an optimal vertex cover is easy to construct in linear time.

Otherwise let v be a vertex with degree 2 and a, b its neighbors, where a has degree ≥ 3. The algorithm chooses the first case that applies.

1. If there is edge $(a, b) \in E$ or if there is a path of length two from a to b not including v and therefore a cycle $[v, b, c, a, v]$. Furthermore, let deg$(c) = 2$. Then include $\{a, b\}$ in the vertex cover (no branching is necessary).
2. If $|N(a, b)| \geq 4$, then branch according to $\{a, b\}$ and $N(a, b)$.
3. Assume there is exactly one cycle of length four containing v. Say the cycle is $[v, a, y, b, a, v]$. Then branch according to $N(y)$ and $N(a)$.
4. Assume there are two cycles of length 4 containing v. W.l.o.g. let $[v, a, y, b, v]$ and $[v, a, z, b, v]$ be these cycles. Then branch according to $N(y)$ and $\{v, y, z\}$.

Case 4 If the above cases do not apply and if the graph is regular, then choose some vertex v and branch according to v and $N(v)$.

Case 5 If the above cases do not apply and if there is a vertex v with deg$(v) = 3$ then let $N(v) = \{a, b, c\}$ and proceed as follows.

1. If there is a cycle $[v, y, z, v]$ then assume $y = a$ and $z = b$. Branch according to $N(v)$ and $N(c)$.
2. If there are at least two different cycles $[v, u_1, x_1, w_1, v]$ and $[v, u_2, x_2, w_2, v]$, then branch according to $N(v)$ and $\{v, x_1, x_2\}$.
3. If there is exactly one cycle $[v, u, x, w, v]$, then assume $u = a$ and $w = b$. Furthermore assume deg$(a) = 3$. Branch according to $N(v)$ and $N(a)$.

4. If there is exactly one cycle $[v, u, x, w, v]$ and the case above does not apply, we again assume $u = a$ and $w = b$. Branch according to $N(v), N(a), \{a, v\} \cup N(b, c)$.
5. If none of the cases above applies in the whole graph, but there is a vertex v with neighbors $N(v) = \{a, b, c\}$, we distinguish as follows.
   (a) Assume at least two vertices of $\{a, b, c\}$ have degree at least four, say a and b. Then we branch according to $N(v)$, $\{v\} \cup N(a, b)$, $\{v, a\} \cup N(b, c)$, and $\{v, b\} \cup N(a, c)$.
   (b) Otherwise, we can assume that there is a vertex v in G with $\deg(v) = 3$, $N(v) = \{a, b, c\}$, and exactly one of $\{a, b, c\}$ have degree at least 4. In this case we refer to [58] due to the complicated branching in this case. We do not use of this case in our algorithm described in the next chapter.

Case 6 If the above cases do not apply all the vertices in G have either degree 4 or 5. Let v be a vertex with $\deg(v) = 4$. Let $y \in N(v)$ and $\deg(y) = 5$. We proceed as follows.

1. Assume there is a cycle $[v, a, b, v]$.
   (a) $a, b \neq y$. Let $c \notin \{a, b, y\}$ be another neighbor of v. Branch according to $N(v)$, $N(y)$, and $\{v, y\} \cup N(c)$.
   (b) $a = y$. W.l.o.g. let $c, d \in N(v)$, $c, d \notin \{a, b\}$ and (c, d) is not an edge. Branch according to $N(v)$, $N(c)$, $\{v, c\} \cup N(d)$.
2. If the above case does not apply and there is no vertex v with degree 4 that has the following two properties at once
   - v has a neighbor x with degree 5.
   - there are at least two different cycles v is contained in but not x.

   Let $N(v) = \{y, b, c, d\}$. Branch according to $N(y)$, $N(v)$, $\{v, y\} \cup N(b, d)$, $\{v, y, d\} \cup N(b, c)$, and $\{v, y, b\} \cup N(c, d)$.
3. If the above case does not apply pick a vertex v having these 2 properties at once.
   - v has a neighbor y with degree 5,
   - there are at least 2 different cycles v is contained in but not y

branch according to $N(y), N(v)$.

Since later (cf. Section 10.3) we use Case 6 of this algorithm, we mention the recurrence equations following from the subcases. Subcases 6 (a) i, 6 (a) ii, and 6 (c) imply the recurrence equation $f(k) \leq f(k-5) + 2f(k-4) + 1$; the recurrence equation following from 6 (b) is $f(k) \leq f(k-9) + 2f(k-8) + f(k-5) + f(k-4) + 1$. The solution corresponding to the resulting polynomials $x^5 - 2x - 1 = 0$ and $x^9 - 2x - x^4 - x^4 - 1 = 0$ is $x \approx 1.2906$.

Chapter 10

An Improved $\mathcal{FPT}$ Algorithm

In this chapter, we present an improved fixed-parameter-tractable algorithm for k-VERTEX COVER with a time complexity of $O(kn + r^k k)$, where $r \approx 1.2906$. We achieve a klam value of 157.

A preliminary version of our algorithm was published in [67] (joint work with Fellows). The algorithm described here is a combination of the algorithm by Downey, Fellows, and Stege (cf. Section 9.4 and [21]) and the algorithm by Niedermeier and Rossmanith (cf. Section 9.5 and [58]). The main improvement is a better kernelization, which is achieved by new reduction rules and an improved structure of the search tree.

The main idea for the new reduction rules is the concept of *adding edges*; the instance $\langle G, k\rangle$ is transformed into $\langle G', k'\rangle$, by deleting vertices and adding edges such that G has a k-vertex cover if and only if G' has a k'-vertex cover.

We restate Observation 2.1 (cf. page 15).

Observation 10.1. *Given a graph $G = (V, E)$. Then for each $v \in V$ and each vertex cover VC of G*

$$v \in VC \text{ or } N(v) \subseteq VC.$$

10.1 Reduction to a Problem Kernel

Starting with $\langle G, k\rangle$, we apply each of the following reduction rules until no further application is possible. The justifications for the reductions are given below (cf. page 108–113). For the sake of completeness, we give also the proofs for the rules we took over from the algorithms described in the previous chapter.

(R 1) If G has a vertex v with $\deg(v) > k$ then replace $\langle G, k\rangle$ with $\langle G - v, k-1\rangle$. Furthermore, v is contained in every k-vertex cover of G.

(R 2) If G has an edge (u, v) with $\deg(u) = 1$, then replace $\langle G, k\rangle$ with $\langle G - \{u, v\}, k-1\rangle$.

Due to complexity reasons, we apply the following rules (R 3), (R 4), and (R 5) only for vertices v of degree at most γ for a given constant $\gamma \in \mathbb{N}$.

(R 3) If G has two adjacent vertices u and v such that $N(u) \subseteq N[v]$, then replace $\langle G, k\rangle$ with $\langle G - v, k-1\rangle$ (cf. Figure 10.3, page 109).

(R 4) If G has a vertex v with even degree, $G|_{N(v)}$ has exactly $\frac{1}{2}\deg(v)(\deg(v)-2)$ edges, and none of the other cases applies, then (up to renaming vertices) $G|_{N(v)}$ has the following form.

Let $N(v) = \{x_1, x_2, \dots, x_{\deg(v)}\}$. Then the edges in the complementary graph of $G|_{N(v)}$ are

$$(x_1, x_{\deg(v)}), (x_2, x_{\deg(v)-1}), \dots, (x_{\frac{1}{2}\deg(v)}, x_{\frac{1}{2}\deg(v)+1})$$

(cf. Figure 10.1).

Replace $\langle G, k\rangle$ with $\langle G', k - \frac{1}{2}\deg(v)\rangle$. G' is obtained from G by

- adding all the possible edges between $x_{\deg(v)}$ and $N(x_1)$;
- adding all the possible edges between $x_{\deg(v)-1}$ and $N(x_2)$,

  $\vdots$
- adding all the possible edges between $x_{\frac{1}{2}\deg(v)+1}$ and $N(x_{\frac{1}{2}\deg(v)})$;
- deleting the vertices $v, x_1, x_2, \dots, x_{\frac{1}{2}\deg(v)}$.

Cf. Figure 10.4 on page 111 as an example.

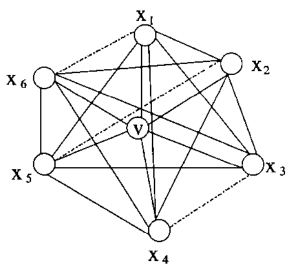


Figure 10.1: (R 4): Every configuration not having a *perfect matching* [9] (shown by the red edges) in its complement would imply case (R 3).

(R 5) If G has a vertex v with odd degree, $G|_{N(v)}$ has exactly $\left(\frac{1}{2}\deg(v)(\deg(v)-2)-\frac{1}{2}\right)$ edges, and none of the other cases applies, then $G|_{N(v)}$ has (up to renaming vertices) the following form. Let $N(v) = \{x_1, x_2, \dots, x_{\deg(v)}\}$. Then the edges in the complementary graph of $G|_{N(v)}$ are $(x_1, x_{\deg(v)})$, $(x_2, x_{\deg(v)-1})$, $\dots$, $(x_{\frac{1}{2}(\deg(v)-1)}, x_{\frac{1}{2}(\deg(v)+3)})$, and $(x_{\frac{1}{2}(\deg(v)-1)}, x_{\frac{1}{2}(\deg(v)+1)})$ (cf. Figure 10.2).

Replace $\langle G, k\rangle$ with $\langle G', k-\frac{1}{2}\deg(v)+\frac{1}{2}\rangle$. G' is obtained from G by

- adding all the possible edges between $x_{\frac{1}{2}(\deg(v)+3)}$ and $N(x_{\frac{1}{2}(\deg(v)-1)})$
- adding all the possible edges between $x_{\frac{1}{2}(\deg(v)+5)}$ and $N(x_{\frac{1}{2}(\deg(v)-3)})$

  $\vdots$

- adding all the possible edges between $x_{\deg(v)}$ and $N(x_1)$
- adding all the possible edges between $x_{\frac{1}{2}(\deg(v)+1)}$ and $N(x_{\frac{1}{2}(\deg(v)-1)})$;

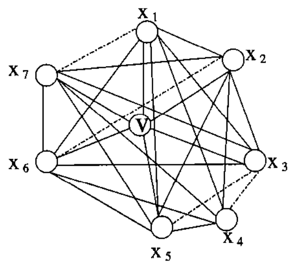


Figure 10.2: (R 5): Every configuration not having such a configuration of red edges in its complement would imply case (R 3).

- deleting the vertices $v, x_1, x_2, \cdots, x_{\frac{1}{2}(\deg(v)-1)}$.

Cf. Figure 10.5 on page 112 as an example.

(R 6) If G has a vertex v with $\deg(v) = 3$ and $N(v) = \{a, b, c\}$, and none of the above cases applies, then $G|_{a,b,c}$ contains no edge. Replace $\langle G, k\rangle$ with $\langle G', k\rangle$. G' is obtained from G by

- deleting the vertex v from G,
- adding all the possible edges between c and the vertices in $N(a)$,
- adding all the possible edges between a and the vertices in $N(b)$,
- adding all the possible edges between b and the vertices in $N(c)$,
- adding the edges (a, b) and (b, c).

Cf. Figure 10.6 on page 114 as an example. Note that this reduction rule is not symmetric.

The reduction rules described above are justified as follows.

(R 1) Any k-vertex cover in G must contain v, since otherwise it would be forced to contain $N(v)$, but $|N(v)| > k$.

(R 2) If G has a k-vertex cover VC with $v \notin VC$, then $u \in VC$. But then $(VC - \{u\}) \cup \{v\}$ is also a k-vertex cover. Thus, G has a k-vertex cover if and only if it has one containing v. That is, G has a k-vertex cover if and only if $G - \{u, v\}$ has a $(k-1)$-vertex cover.

(R 3) If a k-vertex cover VC does not contain v, then it would be forced to contain $N(v)$ and therefore it must contain $N[u] - \{v\}$. But then $(VC - \{u\}) \cup \{v\}$ is also a k-vertex cover (cf. Figure 10.3). Thus,

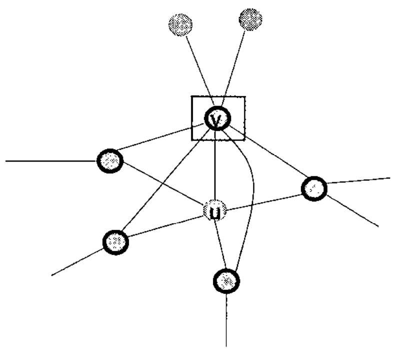


Figure 10.3: Reduction rule (R 3).

G has a k-vertex cover if and only if it has one containing v.

(R 4) If G has a k-vertex cover, then there is a k-vertex cover VC having one of the following forms.

1. $N(v) \subseteq VC$
2. $v \in VC$

In the second case we can assume that at most $\deg(v) - 2$ vertices of $N(v)$ belong to the vertex cover. v is redundant, if all vertices of $N(v)$ belong to the vertex cover. If $\deg(v) - 1$ of the vertices belong to VC, then we can exchange v by the vertex in $N(v)$ not being in the vertex cover.

Thus, we can assume VC has one of the following forms.

- $N(v) \subseteq VC$
- $N(x_{\frac{1}{2}\deg(v)+1}, x_{\frac{1}{2}\deg(v)}) \subseteq VC$
- $N(x_{\frac{1}{2}\deg(v)+2}, x_{\frac{1}{2}\deg(v)-1}) \subseteq VC$

  $\vdots$
- $N(x_{\deg(v)}, x_1) \subseteq VC$

Similarly, if G' has a k'-vertex cover, then there is a k'-vertex cover VC' having one of the following forms.

- $\{x_{\frac{1}{2}\deg(v)+1}, \cdots, x_{\deg(v)}\} \subseteq VC'$
- $N(x_{\frac{1}{2}\deg(v)+1}) \subseteq VC'$
- $N(x_{\frac{1}{2}\deg(v)+2}) \subseteq VC'$

  $\vdots$
- $N(x_{\deg(v)}) \subseteq VC'$.

We show G has a k-vertex cover if and only if G' has a k'-vertex cover. If G has a k-vertex cover of the first form, then G' has a $(k - \frac{1}{2}\deg(v))$-vertex cover containing $\{x_{\frac{1}{2}\deg(v)+1}, \cdots, x_{\deg(v)}\}$ and vice versa. If G has a k-vertex cover VC of the form $v \in VC$, $N(v) - \{x_i, x_{\deg(v)-i+1}\} \subseteq VC$, $N(x_i, x_{\deg(v)-i+1}) \subseteq VC$ for some i $(1 \leq i \leq \frac{1}{2}\deg(v))$, then there is a k'-vertex cover VC' in G' containing $N(x_{\deg(v)-i+1}) \subseteq VC'$. Clearly, in G'

$$\{x_{\frac{1}{2}\deg(v)+1}, \cdots, x_{\deg(v)}\} - \{x_{\deg(v)-i+1}\} \subseteq N(x_{\deg(v)-i+1}).$$

Furthermore, the vertices adjacent to $x_{\deg(v)-i+1}$ in G' agree with the neighbors of x_i and $x_{\deg(v)-i+1}$ in G. Finally a k'-vertex cover VC' in G' of the form $N(x_{\deg(v)-i+1}) \subseteq VC'$ implies a k-vertex cover in G of the form $v \in VC$, $N(v) - \{x_i, x_{\deg(v)-i+1}\} \subseteq VC$ $(1 \leq i \leq \frac{1}{2}\deg(v))$.

(R 5) If G has a k-vertex cover, then there is a k-vertex cover VC having one of the following forms.

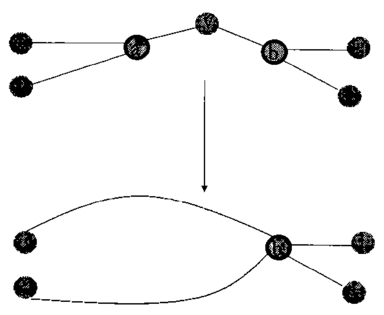


Figure 10.4: Reduction rule (R 4) for $\deg(v) = 2$.

1. $N(v) \subseteq VC$
2. $v \in VC$

In the second case we can assume that at most $\deg(v) - 2$ vertices of $N(v)$ belong to the vertex cover. Thus, we can assume VC has one of the following forms.

- $N(v) \subseteq VC$
- $N(x_{\frac{1}{2}(\deg(v)+3)}, x_{\frac{1}{2}(\deg(v)-1)}) \subseteq VC$
- $N(x_{\frac{1}{2}(\deg(v)+5)}, x_{\frac{1}{2}(\deg(v)-3)}) \subseteq VC$

  $\vdots$
- $N(x_{\frac{1}{2}\deg(v)}, x_1) \subseteq VC$
- $N(x_{\frac{1}{2}(\deg(v)-1)}, x_{\frac{1}{2}(\deg(v)+1)}) \subseteq VC$

If G' has a $k - (\frac{1}{2}(\deg(v) - 1)$-vertex cover VC', then there is one having one of the following forms.

- $\{x_{\frac{1}{2}(\deg(v)+1)}, x_{\frac{1}{2}(\deg(v)+3)}, \dots, x_{\deg(v)}\} \subseteq VC'$
- $N(x_{\frac{1}{2}(\deg(v)+1)}) \subseteq VC'$

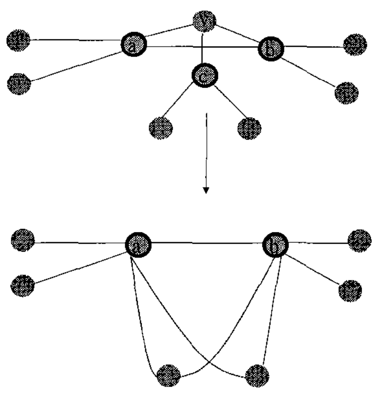


Figure 10.5: Reduction rule (R 5) for deg$(v) = 3$.

- $N(x_{\frac{1}{2}(\deg(v)+3)}) \subseteq VC'$

  $\vdots$
- $N(x_{\deg(v)}) \subseteq VC'$.

We show G has a k-vertex cover if and only if G' has a k'-vertex cover. If G has a k-vertex cover of the first form, then $VC' = VC - \{x_1, x_2, \dots, x_{\frac{1}{2}(\deg(v)-1)}\}$ is a vertex cover of G' and vice versa.

If G has a k-vertex cover VC of the form $N(x_{\deg(v)-i+1}, x_i) \subseteq VC$ for some i $(1 \leq i \leq \frac{1}{2}(\deg(v)-3))$ then $VC' = VC - \{x_1, x_2, \dots, x_{\frac{1}{2}(\deg(v)-1)}\}$ is k'-vertex cover in G'. (All the vertices covered from $N(x_i)$ in G are also covered in G' because of the added edges between $x_{\deg(v)-i+1}$ and $N(x_i)$ in G'.) Conversely, if G' has a vertex cover VC' of the form $N(x_{\frac{1}{2}(\deg(v)-1)}) \subseteq VC'$ then $VC = VC' \cup (\{x_1, x_2, \dots, x_{\frac{1}{2}(\deg(v)-1)}\} - \{x_i\})$ is a vertex cover for G. Finally, if G has a vertex cover of the form

$$N(x_{\frac{1}{2}(\deg(v)-1)}, x_{\frac{1}{2}(\deg(v)+1)}) \subseteq VC,$$

then $VC' = VC - \{x_1, x_2, \dots, x_{\frac{1}{2}(\deg(v)-1)}\}$ is vertex cover in G'.

(R 6) We show that for $\deg(v) = 3$ and $N(v) = \{a, b, c\}$, we can apply (R 2) if $G|_{\{a,b,c\}}$ contains more than one edge. Assume $G|_{\{a,b,c\}}$ contains two edges. Then the two edges must have a vertex in common, say a. But then $N(v) \subseteq N[a]$. We can assume that there is no edge in $G|_{N(v)}$, because otherwise we can apply (R 3) or (R 5) (cf. Figure 10.6). Let VC denote a k-vertex cover in G. We can assume that VC has one of the following forms.

1. $\{a, b, c\} \subseteq VC$
2. $v \in VC$

In the second case, we can assume that at most one of the vertices of $\{a, b, c\}$ belongs to VC.

We show that G' has a k-vertex cover. If VC has form 1 then VC is also a k-vertex cover of G'. Let's consider form 2. If none of the vertices of $\{a, b, c\}$ belongs to VC, then $(VC - \{v\}) \cup \{b\}$ is a k-vertex cover of G'.

Now assume there is exactly one vertex of $\{a, b, c\}$ in VC.

- If $a \in VC$ then $(VC - \{v\}) \cup \{c\}$ is a k-vertex cover of G'.
- If $b \in VC$ then $(VC - \{v\}) \cup \{a\}$ is a k-vertex cover of G'.
- If $c \in VC$ then $(VC - \{v\}) \cup \{b\}$ is a k-vertex cover of G'.

Conversely suppose G' has a k-vertex cover VC. Then we can assume VC has one of the following forms:

1. $\{a, b, c\} \subseteq VC$
2. $b \in VC$
3. $\{a, c\} \subseteq VC$
4. $\{a, b\} \subseteq VC$
5. $\{b, c\} \subseteq VC$

Clearly, if VC has form 1 then it is also a vertex cover of G. If VC has form 2, then $N(a, b, c) \subseteq VC$ and $(VC - \{b\}) \cup \{v\}$ is a k-vertex cover of G'. If VC has form 3 then $N(b, c) \subseteq VC$ and $(VC - \{b\}) \cup \{v\}$ is a k-vertex cover of G. If VC has form 4 then $N(a, c) \subseteq VC$ and $(VC - \{a\}) \cup \{v\}$ is a k-vertex cover of G.

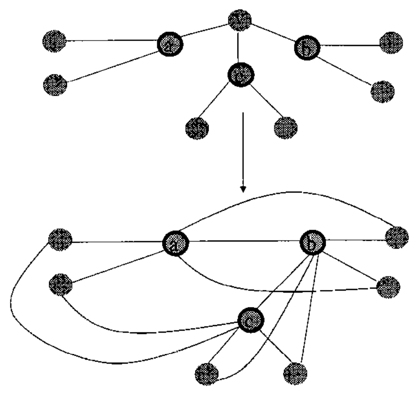


Figure 10.6: Reduction rule (R 6).

10.2 Time-Complexity Analysis of the Reduction to a Problem Kernel

Simply because of reduction rule (R 1) we can conclude, after spending time $O(kn)$, that the answer of input $\langle G, k\rangle$ is *no* if the number of vertices in G' is larger than $k^2 + k$ (cf. footnote page 97).

To analyze the the running time of the other reduction rules we need to consider a graph of size $O(k^2)$ only. With the exception of (R 6), applying a reduction rule leads to decreasing the parameter by at least one. Therefore, we can apply each of (R 1)–(R 5) in time $O(k^3)$.

In the case of reduction rule (R 6), the parameter k is not decreased and we consider the running time of applying (R 6) seperately. We can assume that $\deg(v) > 2$ for all v in G and for each v in G with $\deg(v) = 3$ there is no edge in $G|_{N(v)}$ (otherwise one rule of (R 2) – (R 5) can be applied). Let v be a vertex in G with $\deg(v) = 3$ and $N(v) = \{a, b, c\}$. After applying (R 6) to v, we obtain graph G' with $\deg(x) \geq 4$ for all $x \in \{a, b, c\} \cup N(a, b, c)$. We distinguish two cases.

1. There is no vertex $x \in \{a, b, c\} \cup N(a, b, c)$ with $\deg(x) > k$. In this case, we either apply (*R* 6) to another vertex in G, or no further reduction rule applies. When looking for the next vertex where

(R 6) can be applied, we do not have to reconsider the vertices we checked to find the former one (because only the degree of the vertices $\{a, b, c\} \cup N(a, b, c)$ changes). Therefore, all possible applications of (R 6) for a certain k can be done in time $O(k^3)$.

2. There is a vertex $x \in \{a, b, c\} \cup N(a, b, c)$ with $\deg(x) > k$. Then we apply (R 1) to x and the parameter k is reduced by one. Because we can find x, after applying (R 6) to v, in constant time, here (R 1) does need $O(k)$ time additionally.

Thus, the whole kernelization step can be done in time $O(kn + k^3)$ or simply $O(kn)$.

At the end of this kernelization step we have reduced $\langle G, k\rangle$ to $\langle G', k'\rangle$. G' has minimum degree 4 if we have not already answered the question. If we still have no answer about the original input $\langle G, k\rangle$, then we are left with considering $\langle G', k'\rangle$ where $|V'| \le k^2 + k$ and $k' \le k$.

Combined with a search tree, in the next section we present a further kernelization of the graph s.t. $|G| = O(k)$.

10.3 A Better Search Tree

In this phase of the algorithm we build a search tree of height at most k. The root of the tree is labeled with the output $\langle G', k'\rangle$ of the kernelization. We describe how to build the search tree in three steps. Compared to the algorithms described in the previous chapter, we start branching at vertices of degree 7. This leads us to a smaller graph size of $O(k)$ after applying this first branching step.

Step 1. Following Observation 10.1, we create for every vertex of degree at least 7 two children, one labeled with $\langle G' - v, k' - 1\rangle$, and the other one labeled with $\langle G' - N[v], k' - \deg(v)\rangle$. We repeat this branching procedure, at each step reapplying the reductions of the kernelization.

After Step 1, we can assume that at each leaf of the resulting search tree we are left with an instance $\langle G'', k''\rangle$ consisting of a graph where every vertex has degree four, five or six, and none of the above cases of the reduction rules of the kernelization applies.

What does Step 1 mean to the size of the graph of such an instance? We consider the following graph-theoretical result by Bollobás [9].

Theorem 10.2. *Suppose we are given a graph $G = (V, E)$, $|V| = n$, with $\delta \leq \deg(v) \leq \Delta$ for all $v \in V$. Furthermore $\delta \leq \Delta - 2$ and $n \geq \Delta + \delta$. Then the number of independent edges in G (two edges are* independent *if they have no vertex in common) is at least $\lceil \frac{n\delta}{\delta+\Delta} \rceil$.*

For $\delta = 4$ and $\Delta = 6$, we obtain the following corollary.

Corollary 10.3. *Suppose we are given a graph $G = (V, E)$, $|V| = n$ and $n \geq 10$, with $4 \leq \deg(v) \leq 6$ for all $v \in V$. Then each vertex cover of G has a size of at least $\frac{2}{5}n$.*

Thus, we can answer *no* to the question whether G'' has a k''-vertex cover if $k'' < \frac{2}{5}n$, and therefore we can assume G'' does contain at most $2.5k$ vertices (i.e., $|G''| = O(n)$).

Step 2. For every vertex of degree at least 6 we create two children, one labeled with $\langle G' - v, k'' - 1\rangle$, and the other one labeled with $\langle G'' - N[v], k'' - \deg(v)\rangle$ and repeat this branching procedure, at each step reapplying the reductions of the kernelization.

At each leaf of the search tree we are left with an instance $\langle G^{(3)}, k^{(3)}\rangle$, $G^{(3)}$ having vertices of degree 4 and 5.

Step 3. We apply Case 4 (in the case where $G^{(3)}$ is regular) and Case 6 (in the case where there is a vertex of degree 4 with a neighbor having degree 5) of the algorithm by Niedermeier and Rossmanith (cf. Section 9.5) and repeat this branching, until neither Case 4 nor Case 6 can be applied in $G^{(3)}$, at each step reapplying (R 1), (R 2), and (R 3) and the following branching variants of reduction rules (R 4), (R 5), and (R 6).

These branching variants of the reduction rules are necessary to avoid a repetition of the 4- or 5 regularity of the graph in a branch when applying the algorithm, since in (R 4), (R 5), and (R 6) edges are added to the graph. Of course we only have to apply this variant in the case the graph has been 4 or 5 regular already. In each variant, we combine the reduction rule with a branching such that the resulting recurrence equation for this branching rule in total a polynomial with solution $r \leq 1.290648801$ (i.e., the new branching rules are not more expensive than the Case 6 in the algorithm by Niedermeier and Rossmanith).

We consider (R 4) for a vertex of degree 2 and (R 5) for a vertex of degree 3 only.

(B 4) Let $\langle G, k\rangle$ be an instance where each vertex in G has degree at most 5. Suppose G has a vertex of degree 2 and (R1)-(R3) do not apply to a vertex in G. Let $N(v) = \{a, b\}$. We can assume $(a, b) \notin E$ (otherwise (R 3) applies). We first reduce the graph and do a subsequent branching (cf. Figure 10.7). We reduce as follows.

- delete v
- add all the possible edges between b and $N(a)$.
- Include a in the vertex cover *VC* and delete a.

Consider the resulting instance $\langle G', k-1\rangle$. For every neighbor w of b in G' we have $\deg_{G'}(w) = \deg_G(w)$, that is only the degree of b may be changed. If $\deg(v) < 4$ we are done, because G' contains vertices of degree smaller than 4 and thus, G' is neither 4- nor 5-regular.

If $\deg_{G'}(b) \geq 4$ we branch according to $\langle G' - b, k-2\rangle$ and $\langle G' - N(b), k - (1 + |N(b)|)\rangle$, $|N(b)| \geq 4$. Clearly, no resulting graph G'' is 4- or 5-regular. The vertex in G'' of highest degree has at most the degree of a vertex with the highest degree in G.

(B 5) Suppose G has a vertex of degree 3 and (R 1)-(R 4) and (B 4) do not apply in G (i.e., we can assume that every vertex in G is of degree at least 2 and at most 5). Let $N(v) = \{a, b, c\}$. We can assume that $(a, b) \in E$, but there is no other edge in $G|_{\{a,b,c\}}$ (otherwise (R 3) applies).

We apply (R 5) to v and as a result we obtain graph G'. Only the degrees of a, b and $N(c)$ in G might have changed their degrees in G'.

If G has vertices of degree 5, in G' both a and b have a degree of at least 3 and at most 7. If all the vertices of $\{a, b\} \cup N(a)$ have a degree of 5, we branch according to a and $N(a)$. Otherwise, as long as there are vertices of degree higher than 5 in G' branch according to v and $N(v)$. The reduction and the first branching results in a recurrence equation of the form $f(k) \leq f(k-2) + f(k-6) + 1$.

If there is no vertex of degree 5 in G, in G' both a and b have a degree of at least 3 and at most 5. If a or b have a degree of at least 4, we branch according to a and $N(a)$. If both a and b

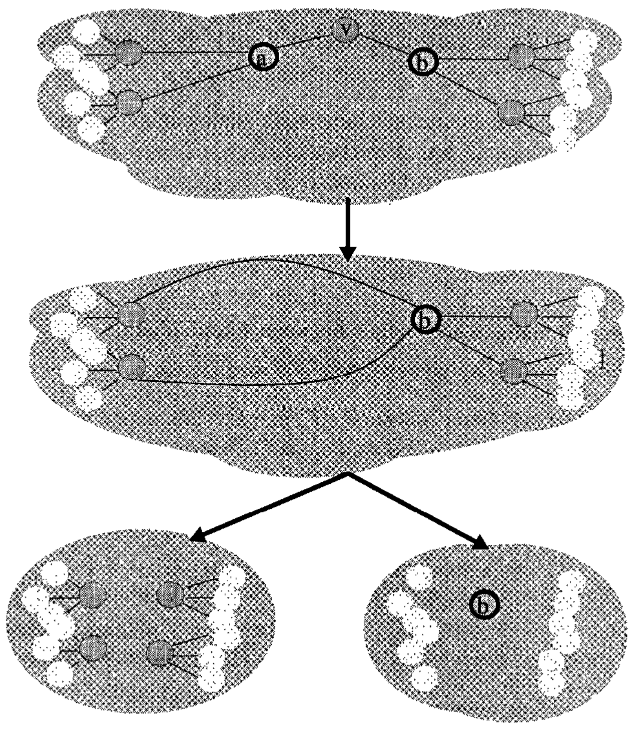


Figure 10.7: Branching rule (B 4).

have a degree of 3 but there is a neighbor v of a and b having a degree of 5, we branch according to v and $N(v)$. The reduction and the first branching result in a recurrence equation of the form $f(k) \leq f(k-2) + f(k-5) + 1$.

(B 6) Suppose G has a vertex of degree 3 and (R 1)-(R 5), (B 4), and (B 5) do not apply in G. Let $N(v) = \{a, b, c\}$. We assume that there is no edge in $G|_{\{a,b,c\}}$. We distinguish 3 cases.

1. There are two vertices of $\{a, b, c\}$, say a, b, such that $N(a) \subseteq N(b)$.
2. The first case does not apply and there is a vertex v of degree

3 having a neighbor of degree at least 4.

3. The first case does not apply and the graph is 3-regular.

In the first case we first apply (R 6) at vertex v. The resulting graph is G'. Now we can reduce G' by b applying (R 3). If the graph is 3-regular, we can further assume that (B 5) does not apply in G. Then we pick a vertex v and branch according to v and $N(v)$. Consider $G-v$. Now there are at least 3 vertices of degree 2 and we apply (B 4). In $G - N(v)$ there are at least two vertices of degree 2 and thus we apply (B 4) to them. The resulting recurrence equation is $f(k) \leq f(k-6)+f(k-7)+2f(k-8)+2f(k-10)+1$.

In the second case let $\deg(b) = 4$. We apply (R 6). In the resulting graph G' w.l.o.g. c has at least 2 neighbors $u, w \notin N(b)$ in G. Branching on c first and then on b (and if necessary also on a) results in a graph being neither 4- nor 5-regular, the recurrence equation is either $f(k) \leq f(k-6)+f(k-1)+1$ or $f(k) \leq 3f(k-7)+f(k-3)+1$.

Finally, branching a 3-regular graph G can be done via branching once, at a vertex v, according to Observation 10.1, and then in the resulting graphs we apply (B 4) at the vertices of degree 2. The resulting recurrence equation is of the form $f(k) \leq 3f(k-15)+3f(k-12)+f(k-9)+f(k-6)+f(k-3)+1$.

10.4 Time-Complexity Analysis

In Step 1, branching a vertex with degree 7 or higher implies the recurrence equation of the form $f(k) \leq f(k-7)+f(k-1)+1$. The resulting polynomial $r^7 - r^6 - 1 = 0$ has the root $r \approx 1.2554$. The graph size is bounded by $O(k^2)$ vertices and therefore so far we have a running time of $O(kn + r^k \cdot k^2)$, $r \approx 1.2554$, for $k > 189$.

The recurrence equation resulting from Step 2 is $f(k) \leq f(k-6)+f(k-1)+1$. The root of its resulting polynoial is $r \approx 1.2852$. The bottleneck recurrence equation resulting from Step 3, Step 4 and Step 5 comes from Case 6 in the algorithm by Niedermeier and Rossmanith (Section 9.5, [58]). The root of the resulting polynomial is $r \approx 1.2906$. Because of Corollary 10.3, the graph size is bounded by $2.5k$ vertices. Thus, the overall time complexity of our algorithm of $O(kn + r^k k)$, $r \approx 1.2906$.

The klam value of 157 is obtained from $\max\{(r_1)^k k^2, (r_2)^k\, 2.5k\} \geq 10^{20}$, $r_1 \approx 1.2554$ and $r_2 \approx 1.2906$, for $k \geq 157.0818$. We want to remark, that independently from this result Chen, Kanj, and Jia recently published an algorithm which solves k-VERTEX COVER in time $O(kn + 1.271^k k^2)$ [14].

10.5 Ideas for Future Work

Taking into account that VERTEX COVER is $\mathcal{NP}$-complete, we presented a very efficient $\mathcal{FPT}$ algorithm solving k-VERTEX COVER. However, difficulties in practice can appear when the input graph is large. One problem is due to memory requirements, since we have to keep copies of the graph for each branch of the search tree. An other reason is the longer practical running time due to the large size of the search tree. Therefore, in the case of large input graphs, it is useful to spend more time in the kernelization step.

For example, in the case of degree-4 vertices, some more reduction rules were found. Because we were not able to find reduction rules for *all* the degree-4 vertices, this does not help improving the running time.

(R 7) If G has a vertex v with $\deg(v) = 4$ and $N(v) = \{a, b, c, d\}$, none of the above cases applies, and if one of the following cases is fulfilled, then $G|_{\{a,b,c,d\}}$ contains at most three edges. Replace $\langle G, k\rangle$ with $\langle G', k'\rangle$ according to one of the following cases depending on the graph $G|_{\{a,b,c,d\}}$ (up to renaming of the vertices a, b, c, d).

(R 7.1) There are exactly three edges in $G|_{\{a,b,c,d\}}$ which we assume to be the edges (a, b), (b, c), and (c, d). In this case, $k' = k$ and G' is obtained from G by

- deleting the vertex v from G,
- adding all the possible edges between d and the vertices in $N(a, b, c)$,
- adding all the possible edges between a and the vertices in $N(d)$,
- adding the edges (a, c), (a, d), and (b, d).

(R 7.2) There are exactly three edges in $G|_{\{a,b,c,d\}}$ which we assume to be the edges (a, b), (a, c), and (b, c). In this case, $k' = k - 2$ and G' is obtained from G by

- adding all possible edges between $\{a, b, c\}$ and $N(d)$,
- deleting vertices v and d.

The rectification is as follows.

(R 7) We show that if $G|_{\{a,b,c,d\}}$ has more than four edges we can apply (R 2). Let $G|_{\{a,b,c,d\}}$ have five edges. But then at least three of them share a common vertex, say a. Then $N(v) \subseteq N[a]$. We can assume there are at most three edges in $G|_{N(v)}$, since otherwise we can apply (R 3) or (R 5).

(R 7.1) If there exist a k-vertex cover in G then there exists a k-vertex cover *VC* of one of the following forms:

1. $\{a, b, c, d\} \subseteq VC$
2. $\{v, a, c\} \subseteq VC$
3. $\{v, b, c\} \subseteq VC$
4. $\{v, b, d\} \subseteq VC$

Clearly, in the first case *VC* is a vertex cover of G'. In the second, third, and fourth case $(VC - \{v\}) \cup \{d\}$ is a k-vertex cover of G'.

Conversely, if G' has a k-vertex cover *VC*, then *VC* has one of the following forms:

1. $\{a, b, c, d\} \subseteq VC$
2. $\{a, c, d\} \subseteq VC$
3. $\{a, b, d\} \subseteq VC$
4. $\{b, c, d\} \subseteq VC$
5. $\{a, b, c\} \subseteq VC$

In the first case *VC* is a vertex cover of G. If *VC* has form 2 then $N(b, d) \subseteq VC$ and $(VC - \{d\}) \cup \{v\}$ is a k-vertex cover of G. In the third case $N(a, c) \subseteq VC$ and $(VC - \{a\}) \cup \{v\}$ is a k-vertex cover of G. In the fourth case, $N(a, d) \subseteq VC$ and $(VC - \{d\}) \cup \{v\}$ is a k-vertex cover of G. In the fifth case, $N(a, b, c, d) \subseteq VC$ and $(VC - \{b\}) \cup \{v\}$ is a k-vertex cover of G.

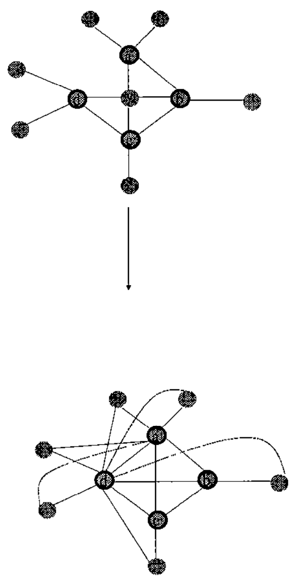


Figure 10.8: Reduction rule (R 7.1).

(R 7.2) Let VC denote a k-vertex cover in G. If $v \notin VC$, then necessarily $\{a, b, c, d\} \subseteq VC$. In this case, $VC - \{d\}$ is a $(k-1)$-vertex cover of G'. Assume that VC contains v. To cover the edges (a, b), (a, c), and (b, c) at least two vertices of $\{a, b, c\}$ belong to VC. That is, in this case we assume $d \notin VC$ (otherwise w.l.o.g. $\{a, b, c, d\} \subseteq VC$). Now $VC - \{v\}$ is a $(k-1)$-vertex cover of G'.

Conversely, if G' contains a $(k-1)$-vertex cover then it must contain one of the following forms.

1. $\{a, b, c\} \subseteq VC$
2. $\{a, b\} \subseteq VC$
3. $\{a, c\} \subseteq VC$
4. $\{b, c\} \subseteq VC$

In the first case $VC \cup \{d\}$ is a k-vertex cover of G. In the other cases $VC \cup \{v\}$ does it.

Reduction rules (R 3), (R 4), and (R 5) should be applied for vertices with high degree γ (cf. page 106).

Even though if it results in squaring the running time of the kernelization step, another useful reduction rule might be:

(R 8) If G has two nonadjacent vertices u and v such that $|N(u,v)| > k$, then replace $\langle G, k\rangle$ with $\langle G + (u,v), k\rangle$.

The rectification of (R 8) is as follows. It is impossible that a k-vertex cover of G does not contain at least one element of $\{u, v\}$, because otherwise the vertex cover must contain all of the vertices of $N(u,v)$. This allows us to add edge (u,v).

Because the kernelization step is independent of the branching rules and the kernelization is possible in polynomial time, kernelization might be a useful preprocessing step for any general method for Vertex Cover.

The presented fixed-parameter-tractable algorithm returns just one solution of a k-vertex cover. But note, that reduction rule (R 1) is applicable for any vertex cover of size k. And since after applying (R 1) we are left with a kernel of size $O(k^2)$ vertices, we can compute all possible solutions for a k-vertex cover in time $O(kn + \binom{k^2}{k}k^2)$ naively. This means also computing all solutions of a k-vertex cover of a given graph is fixed-parameter tractable. To improve on this naive method is left as an open problem, but a good starting point might be the search-tree technique by Fernau and Niedermeier [30] in the case of computing all $(k_1 + k_2)$-vertex covers for a given bipartite graph.

Chapter 11

Experiments

We compare our implementation of the fixed-parameter-tractable algorithm (cf. Section 9.4) with two VERTEX COVER algorithms based on heuristics. That is, we are interested not only in the existence of a k-vertex cover for a fixed k, but in a minimal vertex cover (i.e., the solution of the optimization version of Problem 2.1) for a given input graph $G = (V, E)$, $|V| = n$ and $|E| = m$. The first heuristic algorithm is based on the *greedy heuristic* and the second algorithm is based on a heuristic by Gonnet which we describe below. These vertex cover algorithms[1] have been integrated into the Darwin system (Version 2.0) [35].

Greedy Heuristic. The greedy heuristic always takes the vertex with the highest degree, put it in the vertex cover and removes it from the graph. The greedy heuristic approximates the optimal vertex cover within $1 + \log(k)$ where k is the minimal vertex-cover size [45]. The running time is $O((n + m)n)$.

Gonnet's Heuristic. This heuristic first computes a lower bound of the size of the vertex cover. Therefore it looks for 3-cliques in the graph, enlarges them as much as possible and removes them. The algorithm then attempts $O(n^2)$ iterations in the complete search tree, afterwards the greedy heuristic is applied. If the lower bound coincides with the size of the answer, then one can conclude that the answer is optimal. Empirical evidence shows that this algorithm is $O(n^3)$ for random graphs

[1]The two heuristics have been implemented by G. Gonnet.

with about $n \ln n$ edges.

$\mathcal{FPT}$-Implementation. The fixed-parameter-tractable algorithm which solves k-VERTEX COVER in running time $O(kn+(1.31951)^k \cdot k^2)$ (cf. Section 9.4 and [21]) has been implemented by the author.

The main idea of the implementation for solving VERTEX COVER consists of choosing a "good" k as a starting value. Then we check with our algorithm for k-VERTEX COVER if a solution exists. Depending on whether the answer is *yes* or *no* we decrease or increase the value k and repeat the checking of the procedure until an optimal k is computed.

Due to practical reasons this implementation is focused on input graphs with no more than $|V| \ln |V| + 4\frac{|V|}{10}$ edges. This limit is quite reasonable since dense graphs are rather unlikely as input for conflict graphs. Experiments show, that in random graphs with $|V| \ln |V|$ edges more than half of the vertices are contained in the vertex cover (Figure 11.1–11.9).

As a preprocessing step, we first compute an upper bound using the greedy heuristic. Thus, we exploit its excellent performance for graphs up to $|V| \ln |V|$ edges, as shown in Figures 11.1–11.9.

Using Darwin, we produced a set of 1000 random graphs for 100 vertices with varying number of edges. The vertex-cover sizes, computed with the different methods, and the corresponding CPU times are shown in the following figures. We observed that the greedy heuristic failed to produce an exact solution for approximately 15% of the input graphs whereas the Gonnet heuristic only failed for approximately 1% of the graphs. We obtained similar performance figures for random graphs with 200 vertices. The experiments ran on one processor 336 MHz SUN Enterprise 3500 with 6 processors and 3 GByte RAM.

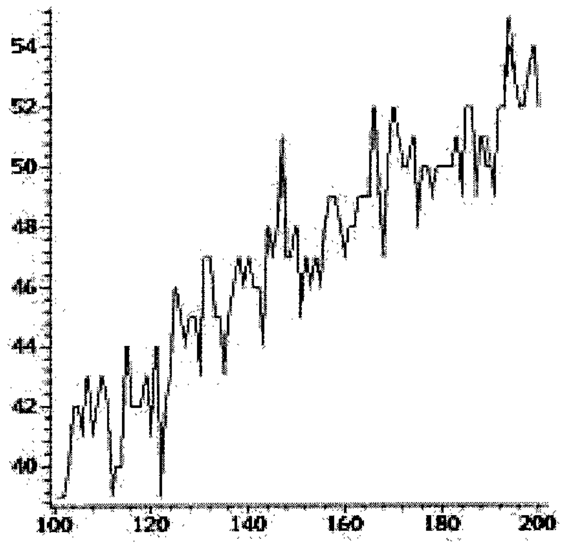


Figure 11.1: The computed size of the vertex covers (i.e., ordinate) for graphs of 100 vertices with an edge number between 100 and 200 (i.e., abscissa). The results of the algorithm based on the greedy heuristic are displayed with red lines, the algorithm based on the heuristic by Gonnet with blue lines, and the results of the fixed-parameter-tractable algorithm are presented in black. If the blue line is not visible then the black and the blue coincide.

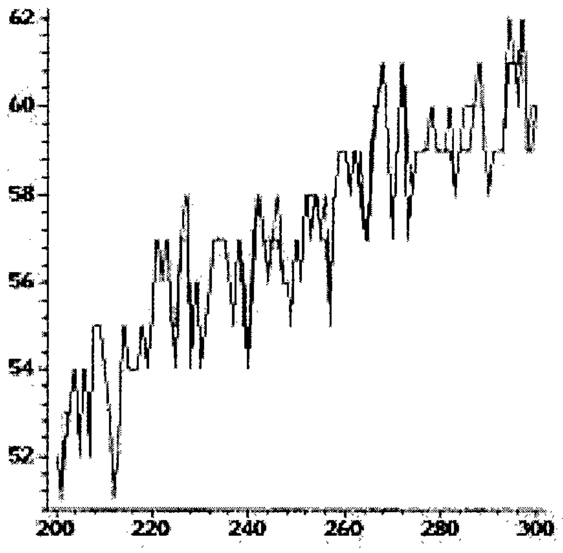


Figure 11.2: The computed size of the vertex covers (i.e., ordinate) for graphs of 100 vertices with an edge number between 200 and 300 (i.e., abscissa). The results of the algorithm based on the greedy heuristic are displayed with red lines, the algorithm based on the heuristic by Gonnet with blue lines, and the results of the fixed-parameter-tractable algorithm are presented in black. If the blue line is not visible then the black and the blue coincide.

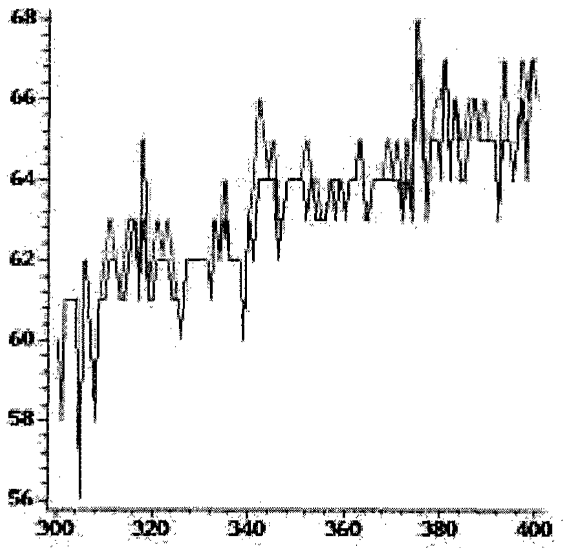


Figure 11.3: The computed size of the vertex covers (i.e., ordinate) for graphs of 100 vertices with an edge number between 300 and 400 (i.e., abscissa). The results of the algorithm based on the greedy heuristic are displayed with red lines, the algorithm based on the heuristic by Gonnet with blue lines, and the results of the fixed-parameter-tractable algorithm are presented in black. If the blue line is not visible then the black and the blue coincide.

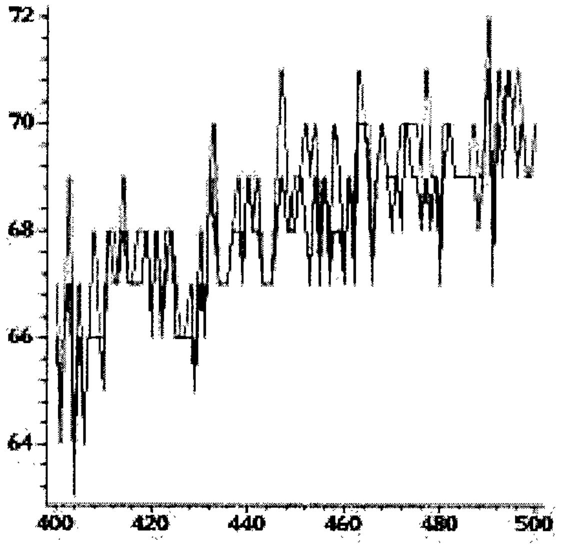


Figure 11.4: The computed size of the vertex covers (i.e., ordinate) for graphs of 100 vertices with an edge number between 400 and 500 (i.e., abscissa). The results of the algorithm based on the greedy heuristic are displayed with red lines, the algorithm based on the heuristic by Gonnet with blue lines, and the results of the fixed-parameter-tractable algorithm are presented in black. If the blue line is not visible then the black and the blue coincide.

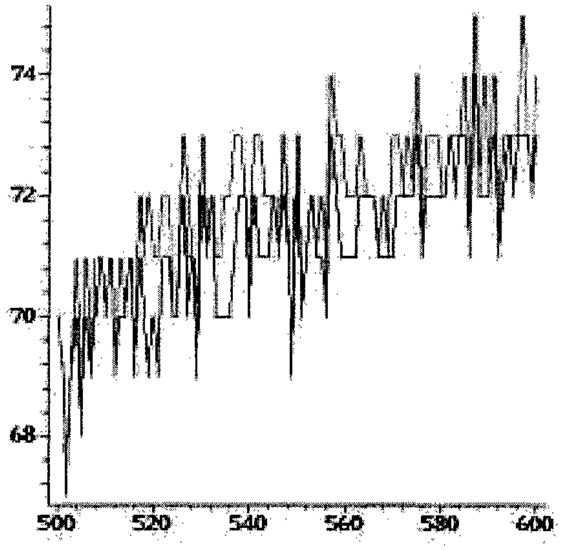


Figure 11.5: The computed size of the vertex covers (i.e., ordinate) for graphs of 100 vertices with an edge number between 500 and 600 (i.e., abscissa). The results of the algorithm based on the greedy heuristic are displayed with red lines, the algorithm based on the heuristic by Gonnet with blue lines, and the results of the fixed-parameter-tractable algorithm are presented in black.

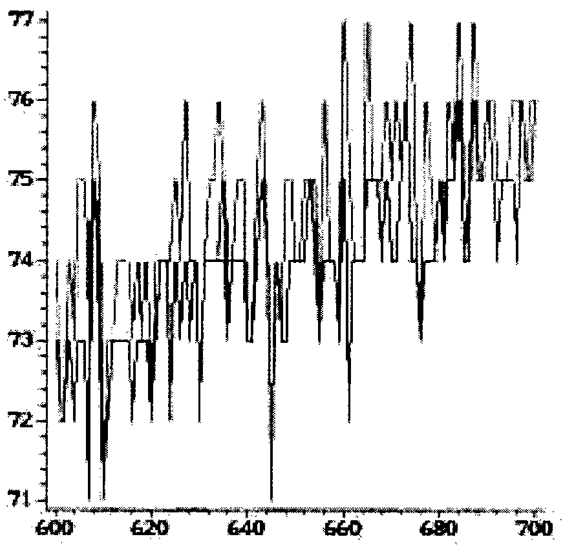


Figure 11.6: The computed size of the vertex covers (i.e., ordinate) for graphs of 100 vertices with an edge number between 600 and 700 (i.e., abscissa). The results of the algorithm based on the greedy heuristic are displayed with red lines, the algorithm based on the heuristic by Gonnet with blue lines, and the results of the fixed-parameter-tractable algorithm are presented in black.

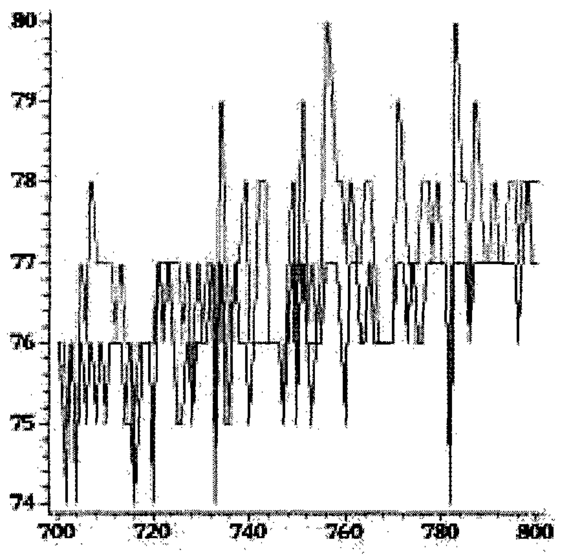


Figure 11.7: The computed size of the vertex covers (i.e., ordinate) for graphs of 100 vertices with an edge number between 700 and 800 (i.e., abscissa). The results of the algorithm based on the greedy heuristic are displayed with red lines, the algorithm based on the heuristic by Gonnet with blue lines, and the results of the fixed-parameter-tractable algorithm are presented in black.

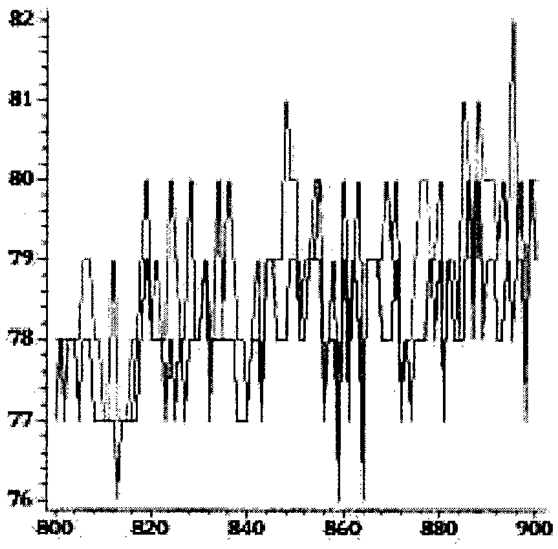


Figure 11.8: The computed size of the vertex covers (i.e., ordinate) for graphs of 100 vertices with an edge number between 800 and 900 (i.e., abscissa). The results of the algorithm based on the greedy heuristic are displayed with red lines, the algorithm based on the heuristic by Gonnet with blue lines, and the results of the fixed-parameter-tractable algorithm are presented in black.

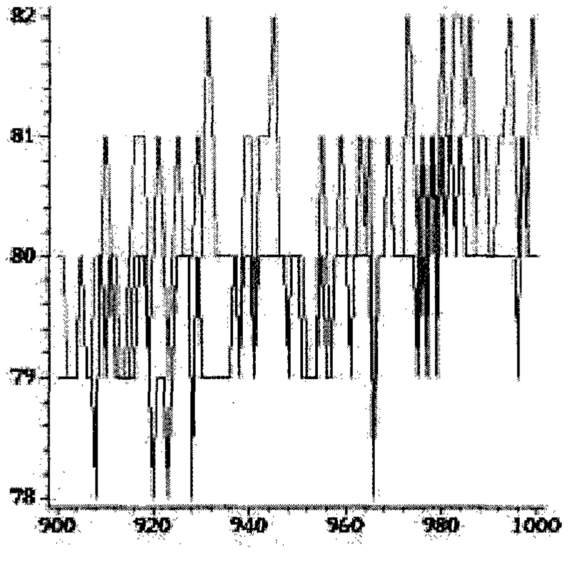


Figure 11.9: The computed size of the vertex covers (i.e., ordinate) for graphs of 100 vertices with an edge number between 900 and 1000 (i.e., abscissa). The results of the algorithm based on the greedy heuristic are displayed with red lines, the algorithm based on the heuristic by Gonnet with blue lines, and the results of the fixed-parameter-tractable algorithm are presented in black.

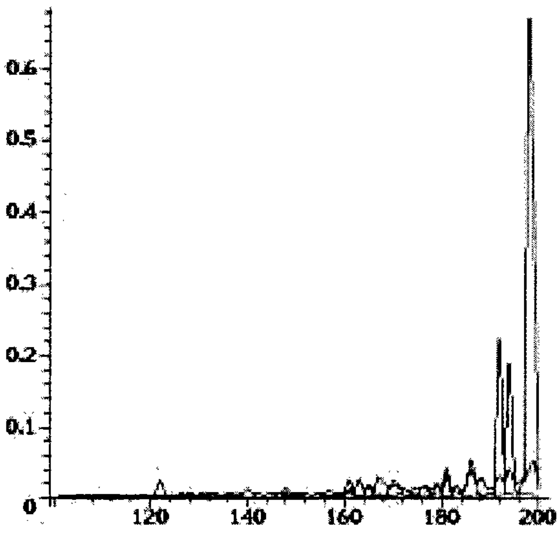


Figure 11.10: The computed CPU times (i.e., ordinate) for graphs of 100 vertices with an edge number between 100 and 200 (i.e., abscissa). The results of the algorithm based on the greedy heuristic are displayed with red lines, the algorithm based on the heuristic by Gonnet with blue lines, and the results of the fixed-parameter-tractable algorithm are presented in black.

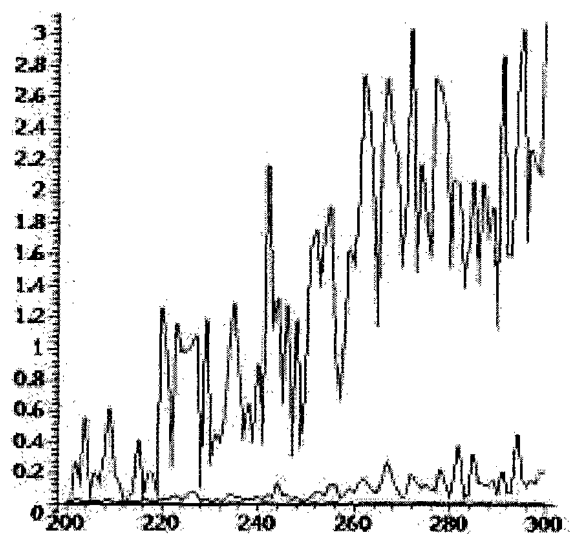


Figure 11.11: The computed CPU times (i.e., ordinate) for graphs of 100 vertices with an edge number between 200 and 300 (i.e., abscissa). The results of the algorithm based on the greedy heuristic are displayed with red lines, the algorithm based on the heuristic by Gonnet with blue lines, and the results of the fixed-parameter-tractable algorithm are presented in black.

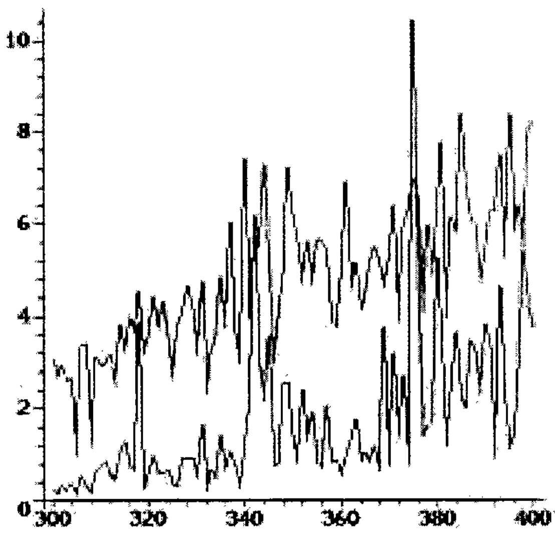


Figure 11.12: The computed CPU times (i.e., ordinate) for graphs of 100 vertices with an edge number between 300 and 400 (i.e., abscissa). The results of the algorithm based on the greedy heuristic are displayed with red lines, the algorithm based on the heuristic by Gonnet with blue lines, and the results of the fixed-parameter-tractable algorithm are presented in black.

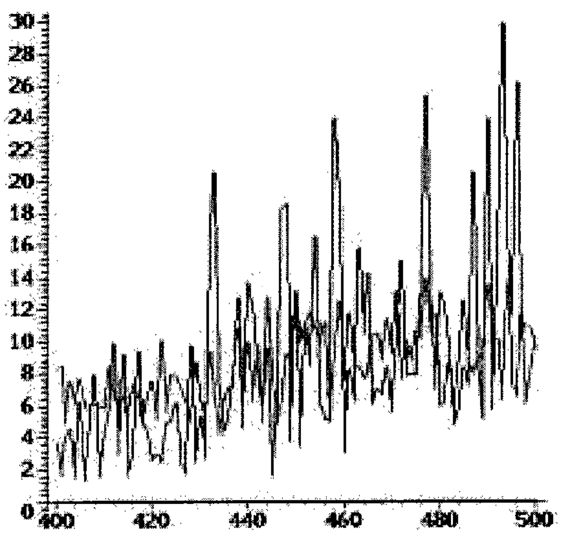


Figure 11.13: The computed CPU times (i.e., ordinate) for graphs of 100 vertices with an edge number between 400 and 500 (i.e., abscissa). The results of the algorithm based on the greedy heuristic are displayed with red lines, the algorithm based on the heuristic by Gonnet with blue lines, and the results of the fixed-parameter-tractable algorithm are presented in black.

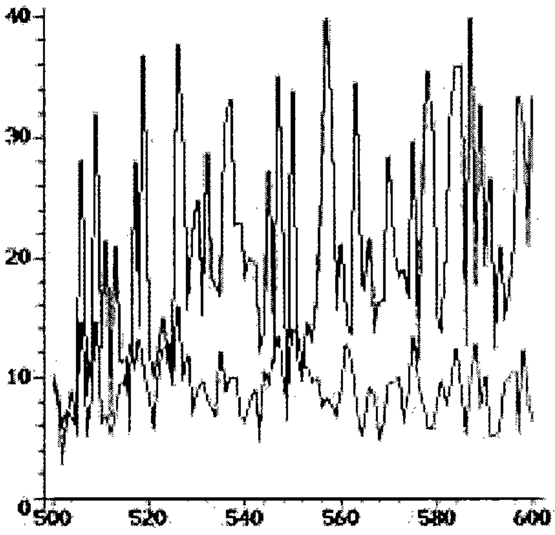


Figure 11.14: The computed CPU times (i.e., ordinate) for graphs of 100 vertices with an edge number between 500 and 600 (i.e., abscissa). The results of the algorithm based on the greedy heuristic are displayed with red lines, the algorithm based on the heuristic by Gonnet with blue lines, and the results of the fixed-parameter-tractable algorithm are presented in black.

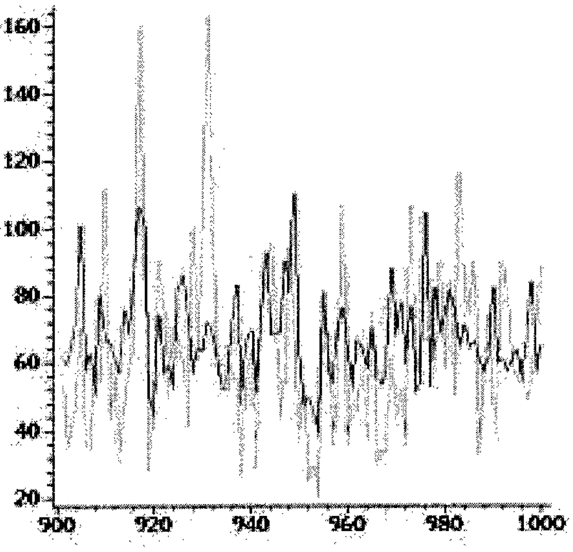


Figure 11.15: Here we compare the CPU time (i.e., ordinate) of the $\mathcal{FPT}$-implementation with the greedy heuristic as preprocessing (green) and Gonnet's heuristic (red) as a preprocessing step for dense graphs ($900 < |E| \leq 1000$; i.e., abscissa). The differences in the running time is attributed to the different starting values resulting from the heuristics. Though the red line looks more stable in the running time, the average is worse. The average CPU time of the algorithm which uses the greedy heuristic as a preprocessing step is 62.256, the average CPU time of the algorithm which uses Gonnet's heuristic as a preprocessing step is 67.087.

Part IV

Conclusions and Open Questions

Chapter 12

Conclusions

The main goal of this research has been to investigate mathematical models and algorithms for conflict resolution in molecular sequence data. Constructing evolutionary trees from DNA sequence data is an important and timely topic in computational biology.

Part I of this thesis provided motivation for this research and introduced definitions and terms from graph theory, classical computational complexity, and parameterized computational complexity used in subsequent chapters. Part II investigated how to resolve inconsistencies between gene trees and species trees. In Part III we developed a new fixed-parameter-tractable algorithm for VERTEX COVER to resolve conflict graphs.

12.1 Summary of Contributions

The main contributions from this thesis are as follows:

- a survey of mathematical models for contradictory trees
- development of the explanation-tree model
- a survey and development of models for gene duplication events
- definition and complexity analysis of the SMALLEST COMMON SUPERTREE problem
- a fixed-parameter-tractable algorithm for the GENE DUPLICATION problem

- definition of the BALL-AND-TRAP GAME and complexity analysis of parameterizations of the BALL-AND-TRAP GAME
- complexity analysis of the MULTIPLE GENE DUPLICATION problem using the -AND-TRAP GAME
- definition of a conflict graph model for multiple sequence alignments
- a survey of known fixed-parameter-tractable algorithms for the k-VERTEX-COVER problem
- an improved kernelization and search tree for the k-VERTEX-COVER problem
- a time complexity of $O(kn + r^k k)$, $r \approx 1.2906$, for our k-VERTEX COVER algorithm
- an implementation of our k-VERTEX COVER algorithm and comparison

12.1.1 A Survey of Mathematical Models for Contradictory Trees

Chapters 4 and 5 present a survey of known methods—consensus methods, agreement methods, and the duplication-loss approach—for computing trees summarizing the common properties of a given set of contradictory trees over a leafset. The most popular consensus tree is the Adams consensus tree [1, 2, 53, 54, 70, 72]. The most famous agreement tree is the Maximum Agreement Subtree (MAST) [70]. While consensus and agreement subtrees preserve mathematical properties of trees, they provide little intuitive biological interpretation. In contrast, the duplication-loss approach [36] explains the differences between a given gene tree and a species tree with the minimum number of gene-duplication events and gene losses necessary to rectify the gene tree with respect to the species tree and, thus, affords an easy to interpret biological explanation.

12.1.2 Development of the Explanation-Tree Model

Chapter 5 surveys and develops models for counting evolutionary events. In Section 5.1 we model the history of a gene tree for a given species tree by means of a cost function. This function assigns a value reflecting the number of evolutionary events in the history of the gene tree with respect to the species tree. The whole history of the gene tree with respect to a given species tree can be viewed by means of the explanation tree. The event function and a tree-stable location function determine the explanation tree. Section 5.1 showed that an explanation tree is leaf-labeled isomorphic to its gene tree.

12.1.3 A Survey and Development of Models for Gene-Duplication Events

Sections 5.2–5.4 formalize the DUPLICATION-AND-LOSS MODEL, the GENE-DUPLICATION MODEL, and the MULTIPLE-GENE-DUPLICATION MODEL. The basis for each model is a cost function, an event function, and a tree-stable location function. Each model results in a problem statement, namely, DUPLICATION AND LOSS, GENE DUPLICATION, and MULTIPLE GENE DUPLICATION.

12.1.4 Definition and Complexity Analysis of the Smallest-Common-Supertree Problem

The smallest common supertree of a set of gene trees implies a lower bound for the number of gene-duplication events necessary to rectify the gene trees with respect to a species tree (cf. Section 6.1). In Section 6.2 we show that the SMALLEST COMMON SUPERTREE problem is $\mathcal{NP}$-complete and $\mathcal{W}[1]$-hard when parameterized by the number of input trees. The hardness is shown by reduction from the SHORTEST COMMON SUPERSEQUENCE problem. The problem becomes fixed-parameter tractable if we allow a bounded number of additional leaves (cf. Theorem 6.5) [25, 27].

12.1.5 An Fixed-Parameter-Tractable Algorithm for the Gene-Duplication Problem

Section 7.2 develops our fixed-parameter tractable algorithm for a parameterized version of the GENE DUPLICATION problem. This algorithm was implemented as a C program and tested using small samples of data.

12.1.6 Definition of the Ball-and-Trap Game and Complexity Analysis of Parameterizations of the Ball-and-Trap Game

Section 8.1 defines the BALL-AND-TRAP GAME, which is a combinatorial game played on a rooted tree decorated with traps and balls. The score of the tree is defined as the sum of the scores over all vertices. The score of a vertex is defined as the maximum number of balls colored with the same color. The goal of the game is to execute defined moves such that the score of the tree is minimized. Several parameterizations of the BALL-AND-TRAP GAME are formalized in Section 8.2. In Section 8.3 the BALL-AND-TRAP GAME is shown to be $\mathcal{NP}$-complete. Furthermore, two parameterizations of the BALL-AND-TRAP GAME are shown to be $\mathcal{W}[1]$-hard.

12.1.7 Complexity Analysis of the Multiple-Gene-Duplication Problem using the Ball-and-Trap Game

The BALL-AND-TRAP GAME is a simplification of the MULTIPLE GENE DUPLICATION problem. We show a reduction from a restricted form of MULTIPLE GENE DUPLICATION to BALL-AND-TRAP (cf. Section 8.1). We then show in Section 8.4 a reduction from a $\mathcal{W}[1]$-hard parameterization of BALL AND TRAP back to a parameterization of MULTIPLE GENE DUPLICATION thereby showing that the MULTIPLE GENE DUPLICATION problem is $\mathcal{NP}$-complete and $\mathcal{W}[1]$-hard [26].

12.1.8 Definition of a Conflict Graph Model for Multiple Sequence Alignments

Constructing Multiple Sequence Alignments (MSAs) is a fundamental problem in computational biology. The known algorithms computing MSAs usually fail to produce an exact solution corresponding to the underlying model due to the $\mathcal{NP}$-hardness of the problem [13, 34, 39, 44, 71]. The main problem is the misplacement of gaps. Therefore we can view the problem of computing MSAs as a problem of inserting gaps at the correct places [48, 49]. We model this problem by means of a conflict graph where the vertices and edges represent gaps and conflicts, respectively. The goal is to identify the the minimum number of gaps which prevent the construction of a unique evolutionary tree. Hence, we have transformed the problem into the VERTEX COVER problem [49].

12.1.9 A Survey of Known Fixed-Parameter-Tractable Algorithms for the k-Vertex-Cover Problem

Chapter 9 presents a survey of known fixed-parameter-tractable algorithms for the k-VERTEX COVER problem, where k is the size of the vertex cover to be determined. The table below is a summary of this survey.

Authors	Refs	Approach	Time Complexity	Klam Value
Fellows	[24]	Bounded search tree	$O(2^k n)$	–
Buss & Goldsmith	[11]	Reduction to a problem kernel	$O(kn + k^{2k+2})$	9
Papadimitriou & Yannakakis	[60]	Maximal matching	$O(3^k n)$	–
Downey & Fellows	[19]	Combining [11] and [60]	$O(kn + 3^k k^2)$	35
Downey & Fellows	[18]	Combining [24] and [11]	$O(kn + 2^k k^2)$	54
Balasubramanian, Fellows & Raman	[4]	Combining [18] with an improved search tree	$O(kn + r^k k^2)$ $r \approx 1.3247$	129
Downey, Fellows & Stege	[21]	Better kernelization and improved search tree	$O(kn + r^k k^2)$ $r \approx 1.3195$	130
Niedermeier & Rossmanith	[58]	Improved search tree	$O(kn + r^k k^2)$ $r \approx 1.2917$	141
Chapter 10 of this thesis		Combining [21] and [58] and an improved kernelization and search tree	$O(kn + r^k k)$ $r \approx 1.2906$	157

12.1.10 An improved Kernelization and Search Tree for the k-Vertex-Cover problem

The main approach to reducing the time complexity of algorithms for k-VERTEX COVER is to improve on the problem kernel and the search tree. In Chapter 10 we presented an improved kernelization, which is accomplished by new reduction rules and an improved structure of the search tree. The main idea of the new kernelization is to bound the degree at each vertex of the search tree to a fixed range (i.e., 4–6). This is accomplished by introducing additional reduction rules and the branching in the search tree. The new reduction rules use the concept of *adding edges*. The instance $\langle G, k\rangle$ is transformed into an instance $\langle G', k'\rangle$ by adding edges and deleting vertices, such that G has a k-vertex cover if and only if G' has a k'-vertex cover. The main idea of the improved search tree is to branch at degree 7, which leads to a graph size linear in k. The result is a new fixed-parameter-tractable algorithm for the k-VERTEX COVER problem.

Section 10.5 presents further reduction rules, which could be applied if the graph input size is dense or large.

12.1.11 A Time Complexity of $O(kn + r^k k)$, where $r \approx 1.2906$, for our k-Vertex-Cover Algorithm

The time complexity of the algorithm is basically determined by Rule 1 of the kernelization phase and Step 3 of the search-tree-building phase. Because we were able to bound the degree of each vertex to a fixed range, we were able to apply a theorem by Bollobás [9] to show that the graph size, after this step, is linear in k. The overall time complexity of our k-VERTEX COVER algorithm is $O(kn + r^k k)$, where $r \approx 1.2906$. The klam value for this algorithm is 157, which is an improvement of 16 over the algorithm by Niedermeier and Rossmanith [58].

12.1.12 An Implementation of our k-Vertex-Cover Algorithm and Comparison

We implemented our new fixed-parameter-tractable algorithm for VERTEX COVER and integrated it into the Darwin system [35]. The main idea is to choose a "good" k as a starting value and then use our algorithm to check whether the solution is optimal. If it is not optimal, decrement k until an optimal solution is found. The implementation is designed for input graphs with at most $|V| \ln |V| + 4\frac{|V|}{10}$ edges, which is a reasonable assumption since conflict graphs are usually sparse.

Using Darwin, we conducted selected experiments to compare our algorithm with respect to optimality and CPU-time with two algorithms which employ the greedy heuristic and the Gonnet heuristic, respectively, to compute the vertex cover of the input graph. We used 1000 random graphs with a fixed number of 100 vertices and a number of edges in the range of 100 to 1000. Of course our fixed-parameter-tractable algorithm produces an exact solution for every input graph. We observed that the greedy heuristic failed to produce an exact solution for approximately 15% of the input graphs, whereas the Gonnet heuristic only failed for approximately 1% of the graphs. For sparse graphs, our fixed-parameter-tractable algorithm performs better than the algorithms based on heuristics.

12.2 Open Problems and Future Work

In this section we formulate open problems for three of the research topics this dissertation investigated: GENE DUPLICATION, MULTIPLE GENE DUPLICATION, and VERTEX COVER.

12.2.1 Open Problems in the Area of Gene Duplication

- The fixed-parameter-tractable algorithm solving GENE DUPLICATION is based on a bounded search tree only. Since a kernelization for each fixed-parameter-tractable problem exists [21], it is worthwhile to investigate an efficient kernelization as a preprocessing step.
- Moreover, the search tree used in the fixed-parameter-tractable algorithm for GENE DUPLICATION can probably be further improved.
- Conduct extensive experiments using real DNA data sets.

12.2.2 Open Problems in the Area of Multiple Gene Duplication

- We did not find reasonable tractable parameterizations of the $\mathcal{NP}$-complete problem MULTIPLE GENE DUPLICATION. One approach worth pursuing is the question whether there is a reasonable tractable parameterization of the BALL-AND-TRAP GAME to develop an algorithm for MULTIPLE GENE DUPLICATION. Another approach is to investigate a more relaxed formulation of this model.

- If no reasonable fixed-parameter-tractable algorithm is found, heuristics could be developed for the MULTIPLE GENE DUPLICATION PROBLEMin order to conduct extensive experiments using real DNA data sets.

12.2.3 Open Problems in the Area of k-Vertex Cover

- Because several reduction rules to get rid of special cases of degree 4 vertices are known, it is an interesting question whether one can develop further reduction rules such that all of the degree 4 vertices can be eliminated. This would further improve the running time of our fixed-parameter-tractable algorithm solving k-VERTEX COVER.

- The efficiency of our current implementation of the fixed-parameter-tractable VERTEX COVER algorithm could be improved by storing intermediate results and reusing them in other branches of the tree using clever bookkeeping or hashing schemes.

- The implementation could also be improved for dense graphs. For dense graphs both the greedy algorithm and the Gonnet heuristic algorithm deviate more from the optimal solution than for sparse graphs. Thus, the amount of work to be done for the k-VERTEX COVER algorithm to find an optimal is prohibitive. Better upper bounds for dense graphs are also needed.

- Moreover, the additional reduction rules presented in Section 10.5 could be applied if the graph input size is dense and/or large. In this cases, it would pay off to spend more time preprocessing to reduce the size of the search tree.

Bibliography

[1] Edward N. Adams. Consensus techniques and the comparison of taxonomic trees. In *Syst. Zool.*, volume 21, pages 390–397, 1972.

[2] Edward N. Adams. N-trees are nestings: Complexity, similarity, and consensus. *Journal of Classification*, 3:299–317, 1986.

[3] Amihood Amir and Dmitry Keselman. Maximum agreement subtrees in a set of evolutionary trees – metrics and efficient algorithms. In *35th Annual Symposium on Foundations of Computer Science*, pages 758–769, 1994.

[4] R. Balasubramanian, Michael R. Fellows, and Venkatesh Raman. An improved fixed-parameter algorithm for Vertex Cover. *Information Processing Letters*, 65:163–168, 1998.

[5] Jean-Pierre Barthélemy. Thresholded Consensus for n-Trees. *Journal of Classification*, 5:229–236, 1988.

[6] Jean-Pierre Barthélemy, Bruno Leclerc, and Bernard Monjardet. On the use of ordered sets in problems of comparison and consensus of classifications. *Journal of Classification*, 3:187–224, 1986.

[7] Jean-Pierre Barthélemy and F.R. McMorris. The median procedure for n-trees. *Journal of Classification*, 3:329–334, 1986.

[8] Steven Benner and Andrew Ellington. Evolution and structural theory. The frontier between chemistry and biochemistry. *Bioorganic Chemistry Frontiers*, 1:1–70, 1990.

[9] Béla Bollobás. *Extremal Graph Theory*. Academic Press, London, 1978.

[10] David Bryant. *Building Trees, Hunting for trees, and Comparing Trees—Theory and Methods in Phylogenetic Analysis.* PhD thesis, University of Canterbury, 1997.

[11] Jonathan F. Buss and Judy Goldsmith. Nondeterminism within P. *SIAM Journal of Computing*, 22:560–572, 1993.

[12] Liming Cai, Jianer Chen, Rodney G. Downey, and Michael R. Fellows. The parameterized complexity of the short computation and factorization. *Arch. for Math. Logic*, 36:321–337, 1997.

[13] Humberto Carillo and David Lipman. The multiple sequence alignment problem in biology. *SIAM J. Appl. Math.*, 48(5):1073–1082, 1988.

[14] Jianer Chen, Iyad A. Kanj, and Weijia Jia. Vertex Cover: Further observations and further improvements. In *25th International Workshop on Graph-Theoretical Concepts in Computer Science (WG'99)*, LNCS, 1999.

[15] Rodney G. Downey and Michael R. Fellows. Fixed parameter intractability. In *IEEE Proc. Structure in Complexity Theory*, pages 36–50, 1992.

[16] Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness. *Congressus Numerantium*, 87:161–187, 1992.

[17] Rodney G. Downey and Michael R. Fellows. Fixed parameter tractability and completeness I: Basic theory. *SIAM Journal of Computing*, 24:873-921, 1995.

[18] Rodney G. Downey and Michael R. Fellows. Fixed parameter tractability and completeness II: Completeness for $W[1]$. *Theoretical Computer Science A*, 141:109–131, 1995.

[19] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity.* Springer–Verlag, 1998.

[20] Rodney G. Downey, Michael R. Fellows, and Ulrike Stege. Computational tractability: A view from Mars. *Bulletin of EATCS*, September 1999.

[21] Rodney G. Downey, Michael R. Fellows, and Ulrike Stege. *Contemporary Trends in Discrete Mathematics: From DIMACS and DIMATIA to the Future*, volume 49 of *AMS-DIMACS*, chapter Parameterized Complexity: A Framework for Systematically Confronting Computational Intractability, pages 49–99. AMS, 1999.

[22] Oliver Eulenstein, Boris Mirkin, and Martin Vingron. Duplication-based measures of difference between gene and species trees. *Journal of Computational Biology*, 5(1):135–148, 1998.

[23] Martin Farach, Teresa Przytycka, and Mikkel Thorup. On the agreement of many trees. *Information Processing Letters*, 55:297–301, 1990.

[24] Michael R. Fellows. On the complexity of vertex set problems. Technical report, Computer Science Department, University of New Mexico, 1988.

[25] Michael R. Fellows, Michael T. Hallett, Chantal Korostensky, and Ulrike Stege. Analogs & duals of the MAST problem for sequences & trees. In *6th Annual European Sympoium on Algorithms (ESA 98)*, volume 1461 of *LNCS*, pages 103–114, 1998.

[26] Michael R. Fellows, Michael T. Hallett, and Ulrike Stege. On the Multiple Gene Duplication problem. In *Algorithms and Computation, 9th International Symposium, ISAAC'98*, volume 1533 of *LNCS*, pages 347–356. Springer–Verlag, 1998.

[27] Michael R. Fellows, Michael T. Hallett, and Ulrike Stege. Analogs & duals of the MAST problem for sequences & trees. journal version, unpublished, 1999.

[28] Michael R. Fellows and Michael A. Langston. Nonconstructive advances in polynomial-time complexity. *Inform. Process. Letters*, 28:157–162, 1987/88.

[29] Joseph Felsenstein. Phylogenies from molecular sequences: Inference and reliability. *Annu. Rev. Genet*, 22:521–565, 1988.

[30] Henning Fernau and Rolf Niedermeier. An efficient exact algorithm for constraint bipartite Vertex Cover. In *MFCS'99*, 1999.

[31] Michael R. Garey and David S. Johnson. *Computers and Intractability—A guide to the Theory of NP-Completeness.* A Series of Books in the Mathematical Sciences. W.H. Freeman and Company, New York, 1979.

[32] Fred W. Glover, editor. *Journal of Heuristics.* Kluwer Academic Publishers, 1995.

[33] W. Goddard, E. Kubicka, G. Kubicki, and F.R. Morris. The agreement metric for labelled binary trees. *Mathematical Biosciences*, 123:215–226, 1994.

[34] Gaston H. Gonnet and Steven A. Benner. Probabilistic ancestral sequences and multiple alignments. In *Fifth Scandinavian Workshop on Algorithm Theory, Reykjevik*, 1996.

[35] Gaston H. Gonnet and Michael T. Hallett. The Darwin manual. to be published, 1999.

[36] M. Goodman, J. Czelusniak, G.W. Moore, A.E. Romero-Herrera, and G. Matsuda. Fitting the gene lineage into its species lineage: A parsimony strategy illustrated by cladograms constructed from globin sequences. *Syst. Zool.*, 28:132–163, 1979.

[37] J.P. Grime and M.A. Mowforth. Variation in genome size and ecological interpretation. *Nature*, 299:151–153, 1982.

[38] R. Guigó, I. Muchnik, and T.F. Smith. Reconstruction of ancient molecular phylogeny. *Molecular Phylogenetics and Evolution*, 6(2):189–213, 1996.

[39] Sandeep K. Gupta, John Kececioglu, and Alejandro A. Schaffer. Improving the practical space and time efficiency of the shortest-paths approach to sum-of-pairs multiple sequence alignment. In *J. Computational Biology*, 1996.

[40] David Harel and Robert Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal of Computing*, 13:338–355, 1984.

[41] Jotun Hein, Tao Jiang, Lusheng Wang, and Kaizhong Zhang. On the complexity of comparing evolutionary trees. In Z. Galil and E. Ukkonen, editors, *Proceedings of the 6th Annual Symposium on Combinatorial Pattern Matching*, number 937 in LNCS, pages 177–190. Springer-Verlag, Berlin, 1995.

[42] D.M. Hillis, C. Moritz, and B.K. Mable, editors. *Molecular Systematics.* Sinauer Associates, Inc., 2nd edition, 1996.

[43] Dorit S. Hochbaum, editor. *Approximation Algorithms for NP-Hard Problems.* PWS Publishing Company, Boston, MA, 1996.

[44] Xiaoqiu Huang. On global sequence alignment. *CABIOS*, 10(3):227–235, 1994.

[45] David S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.

[46] Richard M. Karp. *Reducibility Among Combinatorial Problems*, pages 85–103. Plenum Press, NY, 1972.

[47] Donald E. Knuth. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming.* Addison-Wesley, 3rd edition, 1997.

[48] Chantal Korostensky. Algorithms for multiple sequence alignments and evolutionary trees. PhD-thesis draft, 2000.

[49] Chantal Korostensky, Ulrike Stege, and Gaston Gonnet. Using insertion and deletion events for improving multiple sequence alignments and building the corresponding evolutionary tree. manuscript, September 1999.

[50] Jens Lagergren, 1999. personal communication.

[51] When-Hsiung Li. *Molecular Evolution.* Sinauer Associates, 1997.

[52] B. Ma, M. Li, and L. Zhang. On reconstructing species trees from gene trees in term of duplications and losses. In *Proceedings of the Second Annual International Conference on Computational Molecular Biology(Recomb 98)*, 1998.

[53] T. Margush and F.R. McMorris. Consensus n-trees. *Bulletin of Mathematical Biology*, 43(2):239–244, 1981.

[54] F.R. McMorris, D.B. Meronk, and D.A. Neumann. *Numerical Taxonomy*, chapter A View of Some Consensus Methods for Trees, pages 122–125. Springer–Verlag, Berlin Heidelberg, 1983.

[55] F.R. McMorris and M.A. Steel. The complexity of the median procedure for binary trees. In *Proceedings of the International Federation of Classification Societies, 1993*, New York, 1993.

[56] B. Mirkin, I. Muchnik, and T.F. Smith. A biologically consistent model for comparing molecular phylogenies. *Journal of Computational Biology*, 2(4):493–507, 1995.

[57] D.A. Neumann. Faithful consensus methods for n-trees. *Mathematical Biosciences*, 63:271–287, 1983.

[58] Rolf Niedermeier and Peter Rossmanith. Upper bounds for Vertex Cover further improved. In *Proceedings of the 16th Symposium on Theoretical Aspects in Computer Science (STACS'99)*, LNCS, 1999.

[59] R.D.M. Page. Maps between trees and cladistic analysis of historical associations among genes, organisms, and areas. *Syst. Biol.*, 43:58–77, 1994.

[60] C.H. Papadimitriou and M. Yannakakis. On limited nondeterminism and the complexity of the V-C dimension. *Journal of Computer and System Sciences*, 53(2):161–170, 1996.

[61] Cynthia A. Phillips and Tandy Warnow. The asymmetric median tree—A new model for building consensus trees. In *Combinatorial Pattern Matching, 7th Annual Symposium*, volume 1075 of *LNCS*, pages 234–252, Laguna Beach, California, 10-12 June 1996.

[62] Teresa Przytycka. Sparse dynamic programming for maximum agreement subtree problem. *Mathematical Hierarchies and Biology DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 1997.

[63] H. Rees and R.N. Jones. The origin of the wide species variation in nuclear DNA content. *Int. Rev. Cytol.*, 32:53–92, 1972.

[64] B. Schieber and U. Vishkin. On finding lowest common ancestors: simplifications and parallelizations. *SIAM Journal of Computing*, 17:1253–1262, 1988.

[65] Mike Steel and Tandy Warnow. Kaikoura tree theorems: Computing the maximum agreement subtree. *Information Processing Letters*, 48(2):77–82, November 1993.

[66] Ulrike Stege. Gene trees and species trees: The Gene-Duplication problem is fixed-parameter tractable. In *6th International Workshop on Algorithms and Data Structures (WADS'99)*, volume 1663 of *LNCS*, 1999. (For a long version see: Tech. Rep. 319, Department of Computer Science, ETH Zürich).

[67] Ulrike Stege and Michael R. Fellows. An improved fixed-parameter-tractable algorithm for Vertex Cover. Technical Report 318, Department of Computer Science, ETH Zürich, April 1999.

[68] Ralph Stinebrickner. s-consensus trees and indices. *Bulletin of Mathematical Biology*, 46:923–935, 1984.

[69] Ralph Stinebrickner. s-consensus index methods: An additional axiom. *Journal of Classification*, 3:319–327, 1986.

[70] D.L. Swofford. *Phylogenetic analysis of DNA sequences*, chapter When are phylogeny estimates from molecular and morphological data incongruent?, pages 295–333. Oxford University Press, 1991.

[71] J.D. Thompson, D.G. Higgins, and T.J. Gibson. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, positions-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22:4673–4680, 1994.

[72] M. Wilkinson. Common cladistic information and its consensus representation: reduced Adams and reduced cladistic consensus trees and profiles. *Systematic Biology*, 43(3):343–368, 1994.

[73] L. Zhang. On a Mirkin-Muchnik-Smith conjecture for comparing molecular phylogenies. *Journal of Computational Biology*, 4(2), 1997.

Curriculum Vitae

1969, May 1	Born in Mannheim, Germany
1975–79	Grundschule (Wilhelm–Leuschner–Schule) in Ludwigshafen am Rhein
1979–88	Staatliches Theodor–Heuss–Gymnasium in Ludwigshafen am Rhein (Abitur)
1988–94	Albert-Ludwigs-Universität Freiburg im Breisgau: Diploma in Mathematics
1994–97	Ph. D. student and teaching assistent in the group of Prof. P. Widmayer (Institute of Theoretical Computer Science), Department of Computer Science, ETH Zürich, Switzerland
1997–99	Ph. D. student and teaching assistent in the Computational Biochemistry Research Group (Head: Prof. G. Gonnet), Department of Computer Science, ETH Zürich, Switzerland