Algorithms, Haplotypes and Phylogenetic Networks

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
Rector Magnificus, prof.dr.ir. C.J. van Duijn, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op donderdag 29 januari 2009 om 16.00 uur

door

Leo Jan Joseph van Iersel

geboren te Hellevoetsluis

Preface

Before I started my PhD in *computational biology* in 2005, I had never even heard of this term. Now, almost four years later, I think I have some idea of what is meant by it. One of the goals of my PhD was to explore different topics within computational biology and to see where the biggest opportunities for discrete/combinatorial mathematicians could be found. Roughly speaking, the first two years of my PhD I focussed mainly on problems related to *haplotyping* and *genome rearrangements* and the last two years on *phylogenetic networks*. I must say I really enjoyed learning so much about both mathematics and biology. It was especially amazing to learn how exact, theoretical mathematics can be used to solve complex, practical problems from biology. The topics I studied clearly show how extremely useful mathematics can be for biology. But I also learned that there are many more interesting topics in computational biology than the ones that I could study so far. The number of opportunities for discrete mathematicians is absolutely immense. I did not include my studies on *genome rearrangements* in this thesis, because my most interesting results [Hur07a; Hur07b] are not directly related to biology. This work is nevertheless interesting to mathematicians and I recommend them to read it. I can certainly conclude that also in this field there is a vast number of opportunities for mathematicians and that the topic *genome rearrangements* provides numerous beautiful mathematical problems.

I could never have written this thesis without a great amount of help from many different people. I want to thank my supervisors Leen Stougie and Judith Keijsper for guiding me, for helping me, for correcting my mistakes, for supplying ideas and for the enjoyable time I had while working with them. I also want to thank the Dutch BSIK/BRICKS project for funding my research and Gerhard Woeginger for giving me the opportunity to work in his group and being my second promotor. I want to thank Jens Stoye and Julia Zakotnik for the work we did together and for the great time I had in Bielefeld. I want to thank Ferry Hagen and Teun Boekhout for helping me to make my work relevant for "real" biology. I also want to thank John Tromp, Rudi Cilibrasi, Cor Hurkens and all others I worked with during my PhD. I want to thank Erik de Vink and Mike Steel for reading and commenting my thesis. I want to thank my colleagues from the Combinatorial Optimisation group at the Technische

Universiteit Eindhoven for the pleasant working conditions and the fun things we did besides work. I especially want to thank Matthias Mnich, not only a great colleague but also a good friend, for all his ideas, his humour and our good and fruitful cooperation. I also want to thank Steven Kelk. I must say that I was very lucky to work with Steven during my PhD. He introduced me to problems, had an enormous amount of ideas, found the critical mistakes in my proofs and made my PhD a success both in terms of results and in terms of enjoying work. Finally, I want to thank Conno Hendriksen and Bas Heideveld for assisting me during my PhD defence and I want to thank them and all my other friends and family for helping me with everything in my life but research.

Contents

Chapter 1

Introduction

1.1 Computational Biology

Nowadays, biology is for an important part being studied at the molecular level. A series of important discoveries in the 20th century showed that studying molecules such as DNA, RNA and proteins can help us to describe, predict and manipulate the functioning and development of organisms, as well as their evolution. One of the first important discoveries in this direction by Oswald Avery in 1944 [Ave44] showed that the hereditary information of organisms is (partly) stored in the long DNA molecules present in each cell of an organism. Later, in 1953, a great breakthrough was made when James Watson and Francis Crick [Wat53] discovered the molecular structure of DNA. This discovery eventually led to the ability to find the sequence of nucleotides in a DNA molecule. James Watson also started, in 1990, the well-known Human Genome Project. In 2003 this project completed a description of the human DNA and most of the genes that it encodes [Int01; Ven01]. A good introduction to molecular biology has been written by Lewin [Lew00].

This progress in molecular biology has triggered opportunities for mathematicians and computer scientists. Algorithms translated into computer programs have been the basis of the enormous developments in this field. It is clear that mathematical modelling, problem solving and algorithm design and implementation will continue to play an important role in the description and prediction of the functioning and evolution of organisms on the basis of molecular data. The books by Clote and Backofen [Clo01], Waterman [Wat95] and Pevzner [Pev00] provide an excellent introduction to the field of *computational biology*: the field that applies mathematics and computer science to problems from or inspired by biology.

The problems studied in computational biology are derived from many different areas of biology. Only a selection is studied in this thesis, which consists of two main parts. The first part, Chapters 2 and 3, considers problems from the field called *haplotyping*. See Bonizzoni et al. [Bon03] for an introduction to this field. Haplotypes are sequences describing the most important positions of an individual's DNA and can thus be seen as a kind of fingerprints of that individual. Deducing correct and precise haplotypes from measured data can be difficult and leads to interesting mathematical problems. Chapter 2 considers the problem of reconstructing two haplotypes of an individual from haplotype fragments that may contain errors. Chapter 3 considers reconstructing the haplotypes of a whole population simultaneously, from ambiguous "genotype" data.

The second part of this thesis considers problems from *phylogenetics*, the study of the evolution of species and populations. This rich field has been approached from many different perspectives; see Nei and Kumar [Nei00] for an overview. The traditional way to describe an evolutionary history is to use a tree-shape with leaves representing currently living organisms. The focus in this thesis is on reconstructing *phylogenetic networks*, which display evolutionary histories that cannot be displayed by a tree. Such a situation can for example be the result of evolutionary events like recombinations or hybridisations. These events cause that two lineages that previously diverged from each other combine again later in time. Mathematically, such histories can be displayed by a directed acyclic graph. Phylogenetic networks can also be used in the context of tree-like evolution, when there is uncertainty about the exact shape of this tree. In Chapter 4, algorithms will be developed for the reconstruction of phylogenetic networks.

1.2 Algorithms and Computational Complexity

To be able to describe the problems and summarise the results in this thesis, it is necessary to give a short introduction to complexity theory. For a complete introduction see Garey and Johnson [Gar79] or Ausiello et al. [Aus99].

Polynomial-time Algorithms and NP-hardness

Computational biology and other application areas of mathematics, such as logistics and communication technology, usually offer problems that have to be solved repeatedly for different inputs. A precise list of instructions, to be used for solving the problem for each such input, is called an *algorithm*. There are many different kinds of problems and many different types of algorithms. For an optimisation problem, an algorithm preferably finds an optimal solution in an efficient way. Informally, an algorithm is considered efficient if it computes a solution, for each reasonable sized input, in a reasonable time. The following class of algorithms has been introduced to formalise efficiency.

Definition 1.1. *An algorithm is said to be* polynomial-time *if its running time is bounded by a polynomial function of the input size.*

The class P consists of those problems that can be solved by a polynomial-time algorithm. These problems are called *(computationally) tractable.* For many problems however, there are no polynomial-time algorithms known. To be able to assess for which problems it is unlikely to be fruitful to search for polynomial-time solutions, the notion of NP-hardness has been introduced. Intuitively, the class of NP-hard problems consists of those problems for which the existence of a polynomial-time algorithm is considered very unlikely. The class NP consists of all decision problems for which a solution can be *checked* in polynomial time. An NP-hard problem is at least as hard as any problem in NP (for formal definitions of the classes NP and NP-hard see [Gar79; Aus99]). In fact, if someone would solve an NP-hard problem in polynomial time, then this would imply that P = NP, and it would enable us to efficiently solve *all* problems known to be in the class NP. In computational mathematics and computer science an NP-hardness proof is therefore considered a strong advice not to look for an optimal polynomial-time algorithm. Whether a problem is tractable or NP-hard (or neither) is often called its *(computational) complexity* (although there are many more complexity classes, see [Pap94]).

Solving NP-hard Problems

For NP-hard problems, other types of algorithms are required. One possibility is to settle for an *approximation algorithm.* Such an algorithm produces solutions that are not necessarily optimal, but whose objective value is, even for a worst-case input, at most a bounded distance away from the optimum. For a minimisation problem, an r-approximation algorithm returns a solution with an objective value that is at most r times the objective value of an optimal solution. Here, r is called the *approximation ratio* (defined in a similar way for maximisation problems). An approach that describes a $(1+\epsilon)$-approximation for *all* $\epsilon > 0$ and runs in time polynomial in the input size (but not necessarily in $\frac{1}{\epsilon}$) is called a *polynomial time approximation scheme (PTAS).*

It can be useful to provide evidence that algorithms with certain approximation guarantees are unlikely to be found. For this reason, APX-hardness has been introduced as an approximation counterpart of NP-hardness. Informally, the class APX-hard contains those problems for which a PTAS is unlikely to exist (in the same sense as above). Proving APX-hardness thus forms a stronger result than proving NP-hardness. However, such a result does not provide any information about the existence of an r-approximation algorithm for some fixed r. For some problems, even stronger inapproximability results can be shown, but this will not be done in this thesis.

A different approach to solving NP-hard problems is to maintain the requirement that an optimal solution has to be found, but to allow the algorithm a non-polynomial running time. Such an algorithm is called an *exact algorithm.* The running time of an exact algorithm for an NP-hard problem is usually an exponential function of the input size. If the exponent of that function is small, then such an algorithm can still be practical, in cases where optimal solutions are required.

Finally, an approach that is very popular in practice is what mathematicians call a *heuristic*. Such an algorithm is not guaranteed to find an optimal solution and does not guarantee any approximation ratio. Its quality is evaluated on basis of its performance in practice. Such algorithms thus need to be thoroughly tested on practical data. Heuristics are often shown to perform particularly well in practice but many heuristics hardly provide insights in the structure of a problem and might therefore be of little help for developments in the long run.

1.3 Algorithms for Computational Biology

Objective Functions and the Parsimony Principle
Formulating biological problems in a mathematical way usually involves formulating an objective that needs to be optimised. In application areas as logistics, there is usually a clear objective: maximise profit or minimise cost. On the contrary, a biological problem often does not state an explicit objective to be optimised. The ultimate goal is in general to find the "real" solution and there are various ways to asses how likely it is that a certain solution is the "real" solution. In some cases, this likelihood can be explicitly computed under a certain model of probability. In these cases, one tries to find a solution with a *maximum likelihood*. In other cases, specific properties of the real solution can be formulated and the objective can then be to find a solution that possesses (as many as possible of) these properties.

Another popular approach is to use the *parsimony principle*, also known as Occam's razor, which states "entia non sunt multiplicanda praeter necessitatem", meaning that one should not choose a solution that makes more assumptions than necessary. For example, in phylogenetics this suggests that the best solution is the one that assumes the smallest number of evolutionary events. Sometimes, such a most parsimonious solution can be equal to the most likely one, especially when the probability of the evolutionary events is very small. For many of the problems considered in this thesis, the parsimony principle is part of the motivation for the formulated objective.

Computational Biology and Combinatorial Mathematics
Computational biology and combinatorial mathematics are often related in a natural way. Problems studied by mathematicians can turn out to be relevant in computational biology and problems from computational biology can have applications in other fields. A nice example is the algorithm constructed by Aho et al. [Aho81], which combines a set of small trees (with three leaves each) into a single large tree "consistent" with each of the small trees (cf. Section 1.5). This algorithm has been designed with an application concerning

relational expressions in mind. The application to phylogenetics is due to Mike Steel [Ste92] and this algorithm is now a well-known and important result in this field.

Another fascinating example is the relation between the evolution of DNA and the sorting of pancakes. Differences between the DNA of two species are often caused by evolutionary events in which a part of a DNA molecule gets reversed. Therefore, a common way to measure the evolutionary distance between the DNA of two species (using the parsimony principle) is to calculate the minimum number of such *reversals* needed to turn the first molecule into the other. This problem is called *sorting by reversals* and has already been well-studied before the relation to computational biology became clear, sometimes motivated by the relation to sorting pancakes. Imagine a stack of pancakes of different types or sizes that one wishes to sort by repeatedly flipping a couple of pancakes on top of the stack with a spatula. From a mathematical point of view, sorting the stack of pancakes with a minimum number of flips is almost the same problem as sorting by reversals, the only difference being that a block of pancakes can only be flipped when it is on top of the stack. This problem has been studied already in 1979 by Gates and Papadimitriou [Gat79], but only much later in relation to computational biology, e.g. by Bafna and Pevzner [Baf96]. In continues to be studied both in computational biology [Ber04] and in a pure mathematical setting [Hur07a; Hur07b].

Borders of Tractability
In Sections 1.4 and 1.5, the computational, biological problems are introduced that are investigated in this thesis. In their most general form, these problems are all NP-hard. Thus, after proving NP-hardness, this thesis investigates how these problems *can* be approached. This is done by considering many different restricted versions of the problems. It is shown for different problems that they remain NP-hard even in very restricted cases. In addition, it is shown how imposing other restrictions enables us to design polynomial-time algorithms. This thesis thus investigates the border between tractability and NP-hardness of computational biology problems. It does not only show NP-hardness in the most general case, but attempts to completely characterise where the problems remain NP-hard and where they become tractable. In addition, for the NP-hard versions of the problems, the approximability is explored. Both approximation algorithms and APX-hardness results are given. Finally, an exponential-time exact algorithm is given that solves one of the most general, NP-hard, problems considered in this thesis.

1.4 Problems from Haplotyping

The genetic information of organisms is stored in long molecules, called chromosomes. A simplified model of these molecules describes them as long sequences of the nucleotides A (adenine), C (cytosine) , T (thymine) and G (guanine). Surprisingly, the genomes of for example two humans have identical nucleotides at more than 99% of the positions. The other positions, where variation *is* observed within a species, are called SNPs (Single Nucleotide Polymorphisms). For almost all of these SNPs, only two out of the four nucleotides are observed. That implies that, mathematically, chromosomes can be viewed as binary strings, where each position of the string describes the nucleotide that this individual has at a certain SNP. These binary strings are called *haplotypes*.

The situation is slightly more complicated, because the DNA of so called *diploid* organisms (such as humans) consists of pairs of chromosomes. The general goal is therefore to obtain two haplotypes per individual. However, the two chromosomes of a pair are difficult to sequence separately, which leads to interesting problems. Firstly, the problem of reconstructing the two haplotypes of an individual from a collection of haplotype fragments. This problem, called Single Individual Haplotyping, is explained in detail in Section 1.4.1. Another interesting problem arising in this field is to reconstruct all haplotypes of a population from a collection of "genotypes", where each genotype describes the two haplotypes of an individual, but in an ambiguous way. This Population Haplotyping problem is explained in Section 1.4.2. Each of these practical problems leads to a number of mathematical problem formulations, which will all be described shortly.

1.4.1 Single Individual Haplotyping

The Single Individual Haplotyping problem arises when trying to obtain the two haplotypes of a chromosome-pair of an individual. The laboratory investigations usually return fragments of the two sequences, and our general goal is to merge these fragments into the two haplotypes. The first main complication is that it is not known which fragment comes from which chromosome. Secondly, the merging process can be disturbed by measurement errors in the fragments.

To formalise the problem, haplotype fragments are described by sequences of the symbols '0', '1' and '$-$', where a '$-$' is called a *hole* and denotes that the fragment does not give any information about this specific SNP. A *gap* in a haplotype fragment is a maximal contiguous block of holes that is flanked on both sides by non-hole symbols. For example, the fragment `---0010---` has no gaps, `-0--10-111` has two gaps, and `-0-----1--` has one gap. The input to Single Individual Haplotyping problems is an *SNP matrix* M having entries from $\{0, 1, -\}$, with each row of the matrix corresponding to a haplotype fragment. An input matrix is said to be *gapless* if there are no rows with gaps. For example, a possible gapless input matrix of haplotype fragments is in Figure 1.1.

If in a certain column one row has a 1 and another row has a 0, then these two

$$\begin{pmatrix} 1 & 0 & 0 & 1 & - & - & - & - & - \\ 0 & 1 & 1 & - & - & - & - & - & - \\ - & - & 0^* & 0 & 0 & 1 & 0 & 1 & - \\ - & - & 0 & 1 & 1 & 0 & - & - & - \\ - & - & - & - & 0 & 1 & 1^* & 1 & 1 \\ - & - & - & - & - & 0 & 1 & 1 & 0 \end{pmatrix}$$

Figure 1.1: A gapless input matrix of haplotype fragments. This matrix can be made feasible by either flipping the elements with a * or by removing the third row.

rows can not represent fragments of the same haplotype (unless one of these two entries is erroneous). In this case, these two rows are said to *conflict*. An SNP matrix M is *feasible* if the rows of M can be partitioned into two sets such that all rows within each set are pairwise non-conflicting. If this is the case, then there exist two haplotypes (binary strings) such that each row of M does not conflict with at least one of the two haplotypes. On the other hand, when a matrix is not feasible, then it is still possible that all fragments come from two haplotypes, if errors have been made during sequencing. To model such errors, define a *flip* as converting a 0 entry to a 1, or vice-versa. The first concrete problem considered in this thesis is Minimum Error Correction (MEC), introduced by Greenberg et al. [Gre04]. The objective in MEC is to flip as few entries of the input matrix as possible to arrive at a feasible matrix.

Minimum Error Correction (MEC)	
Input:	An SNP matrix M.
Output:	The smallest number of flips needed to make M feasible.

For example, consider the gapless SNP matrix from Figure 1.1 and suppose we flip the element in the 3rd row and 3rd column to 1 and that we also flip the element in the 5th row and 7th column to 0. Then the 2nd, 3rd and 5th row become pairwise non-conflicting while this was already true for the 1st, 4th and 6th row. Thus, this matrix can be made feasible by only two flips. The two resulting haplotypes are `011001011` (for the 1st, 4th and 6th row) and `100110110` (for the 2nd, 3rd and 5th row). Since it turns out that it is not possible to make this matrix feasible with just one flip, the optimal value of MEC is in this case two.

There are various interesting versions of this problem. In Gapless-MEC, the input matrix is gapless; in 1-Gap-MEC, each row of the input matrix has at most one gap; and, finally, in Binary-MEC, the input matrix does not contain any holes (i.e. it only contains 0s and 1s). This last version is only of theoretical interest, while the first two versions are also of practical interest since they correspond to possible data generated by two different sequencing techniques.

The second problem considered in this thesis is Longest Haplotype Reconstruction (LHR), introduced by Lancia et al. [Lan01]. This problem has the same input as MEC but a different objective. Only in relation to LHR, haplotypes potentially contain holes. If M is a feasible SNP matrix, then a *feasible partition* of the rows of M is a bipartition of these rows into two sets, M_l and M_r, such that the rows within each set are pairwise non-conflicting. For each M_i ($i \in \{l, r\}$), let the *associated* haplotype H_i be defined as the result of combining the rows of M_i as follows: the j-th column of H_i is set to 1 if at least one row from M_i has a 1 in column j, is set to 0 if at least one row from M_i has a 0 in column j and is set to a hole if all rows in M_i have a hole in column j. Two haplotypes H_1 and H_2 are said to *explain* M if they are the associated haplotypes of a feasible partition (M_l, M_r) of the rows of M.

For example, suppose one side of the partition contains rows `10--`, `-0--` and `---1`; then the associated haplotype we get from this is `10-1`. The *length* of a haplotype is defined as the number of positions where it does not contain a hole; the haplotype `10-1` thus has length three, for example. The objective of LHR is to find two haplotypes that explain a subset of the rows of M and to maximise the sum of the lengths of the two haplotypes.

For example, consider again the input matrix from Figure 1.1 and remember that the 1st, 4th and 6th row were already pairwise non-conflicting (giving the associated haplotype $H_1 =$ `100110110`). Now, observe that also de 2nd and 5th row are non-conflicting and that from these rows we can build the haplotype $H_2 =$ `011-01111`. This input matrix can thus be made feasible by removing the 3rd row, and the haplotypes H_1 and H_2 explain the resulting matrix. The value of this solution, which turns out to be optimal, is 17: the sum of the lengths of H_1 and H_2.

Longest Haplotype Reconstruction (LHR)

Input:	An SNP matrix M.
Output:	Two haplotypes that explain a feasible subset of the rows of M, maximising the sum of the lengths of the two haplotypes.

For LHR, the same restricted versions are considered as for MEC. The most important results in this thesis with respect to the complexity of MEC and LHR are summarised in Table 1.1.

	MEC	LHR
Binary	Open (Section 2.2.3)	Trivial
Gapless	NP-hard (Section 2.2.1)	P (Section 2.3.1)
1-Gap	APX-hard (Section 2.2.2)	APX-hard (Section 2.3.2)

Table 1.1: The new state of knowledge regarding MEC and LHR.

From the APX-hardness of the 1-Gap case of MEC and LHR, it follows directly that these problems are both also APX-hard (and thus NP-hard) in general. The main open problem is the complexity of BINARY-MEC, which has therefore been studied in a more general context. In Section 2.2.4, it is shown that BINARY-MEC becomes NP-hard when the number of haplotypes is not fixed at two, but part of the input. The complexity of BINARY-MEC itself remains open.

All these results will be presented in Chapter 2 and have been published previously in [Cil05; Cil07].

1.4.2 Population Haplotyping

It is often considered too expensive to obtain experimentally (fragments of) the two haplotypes of an individual. In these cases, the experiments only provide *genotypes*. For a certain SNP, such a genotype describes which two nucleotides are present on the two chromosomes, but it does *not* describe which nucleotide is on which chromosome. Mathematically, these genotypes can be seen as sequences of three different symbols. For example, if the two haplotypes are `011` and `010`, then the corresponding genotype is `012`: the genotype is equal to the two haplotypes wherever they are equal to each other, and the genotype gets a 2 at positions where the haplotypes differ.

Given two haplotypes, it is thus easy to find the corresponding genotype. A more interesting problem arises when the data consists of *genotypes* and ones wishes to reconstruct the *haplotypes*. For a single genotype, this would be an impossible task since there is simply not enough information available. However, interesting mathematical problems arise when the genotypes of a whole population are considered simultaneously and the goal is to find all the corresponding haplotypes of that population.

Various criteria have been introduced to assess which set of haplotypes is most likely to be the set corresponding to these genotypes. The first, well-studied, criterion asks to find the *smallest* possible set of haplotypes that is able to "resolve" all genotypes. This criterion is motivated by the parsimony principle combined with the observation that in practice the number of different haplotypes in a population is much smaller than the number of individuals [Gus03]. A different criterion is to find a set of haplotypes that resolves the genotypes and can, in addition, be embedded as the leaves of a "perfect phylogeny"; an evolutionary tree with biologically-motivated restrictions. Finally, also a combination of the previous two criteria has been studied.

To formalise the problems, we introduce the input as a *genotype matrix* G with entries from $\{0, 1, 2\}$, rows corresponding to genotypes and columns corresponding to SNPs. The output is a *haplotype matrix* H with elements in $\{0, 1\}$; the columns of H also correspond to SNPs, but its rows correspond to haplotypes. Two rows h_1 and h_2 of H are said to *resolve* a row g of G if $g(j) = h_1(j)$ for all j with $h_1(j) = h_2(j)$ and $g(j) = 2$ otherwise. A haplotype matrix H *resolves* a genotype matrix G if for each row g of G, containing at least one 2, there are two rows h_1 and h_2 of H resolving g and each row g of G without 2s is also a row of H. The first population haplotyping problem studied in this thesis can now be formulated. The formulation was first suggested by Earl Hubbell, who proved the problem to be NP-hard in general (see [Gus03]).

Parsimony Haplotyping (PH)
Input: A genotype matrix G.
Output: A haplotype matrix H with a minimum number of rows that resolves G.

A *perfect phylogeny* is a tree describing the evolution of a set of haplotypes with the restriction that each SNP mutates at most once in the whole tree. To formalise this, each vertex of the tree is labelled by a haplotype and each edge is labelled by the SNPs that have a different value in the two haplotypes connected by this edge. That each SNP mutates at most once means that each SNP labels at most one edge. A haplotype matrix H is said to *admit* a perfect phylogeny if there exists a perfect phylogeny with the rows of H labelling its leaves. This definition enables us to formally define the second population haplotyping problem considered in this thesis, introduced by Bafna et al. [Baf04].

Minimum Perfect Phylogeny Haplotyping (MPPH)
Input: A genotype matrix G.
Output: A haplotype matrix H with a minimum number of rows that resolves G and admits a perfect phylogeny.

Like PH, also MPPH is NP-hard in general [Baf04]. In response to this intractability, this thesis investigates restricted cases of both these problems and explores in which cases these problems remain NP-hard and when they become polynomial-time solvable. In a (k, ℓ)-*bounded instance*, the input genotype matrix G has at most k 2s per row and at most ℓ 2s per column (cf. [Sha06]). When k or ℓ is a "$*$", this means that these instances are bounded only by the number of 2s per column or per row, respectively.

Previous work on these (k, ℓ)-bounded instances has shown that PH$(3, *)$ and PH$(4, 3)$ are APX-hard [Lan04; Sha06]. In this thesis it is shown that even PH$(3, 3)$ is APX-hard. Furthermore, polynomial-time algorithms are given for PH$(2, *)$ and PH$(*, 1)$. As far as MPPH is concerned, there have been no prior results beyond the above mentioned NP-hardness result. Here it is shown that MPPH$(3, 3)$ is APX-hard and that, like their PH counterparts, MPPH$(2, *)$ and MPPH$(*, 1)$ are polynomial-time solvable. The complexity of both PH(k, ℓ) as MPPH(k, ℓ) can thus be summarised as in Figure 1.2.

The main open problem, for both PH and MPPH, is the complexity of $(*, 2)$-bounded instances. Therefore, this problem has been studied in the restricted version in which the *compatibility graph* of the input genotype matrix is a clique. (Informally, the compatibility graph shows for every pair of genotypes

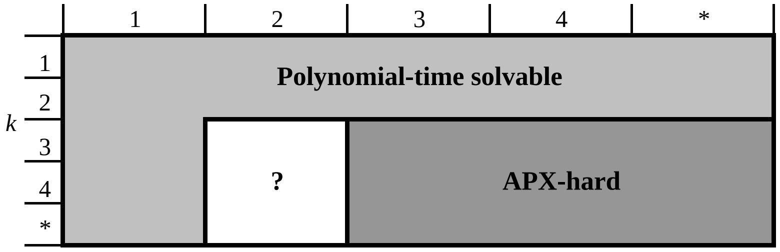


Figure 1.2: The complexity landscape of both PH(k, ℓ) and MPPH(k, ℓ).

whether those two genotypes can use common haplotypes in their resolution.) Sharan et al. showed that PH($*, 2$) is polynomial-time solvable in this special case [Sha06]. This thesis shows how also this special case of MPPH($*, 2$) can be solved in polynomial time.

The fact that both PH and MPPH already become APX-hard for $(3, 3)$-bounded instances means that, in terms of deterministic approximation algorithms, the best that one can in general hope for is constant approximation ratios. This thesis gives approximation algorithms for both PH($*, \ell$) and MPPH($*, \ell$) that have a constant approximation ratio for each fixed ℓ. The approximation-ratios are summarised in Table 1.2.

	Approximation ratio
PH($*, \ell$)	$\frac{3}{2}\ell + \frac{1}{2}$
PH($*, \ell$) with at least one 2 per genotype	$\frac{3}{4}\ell + \frac{7}{4} - \frac{3}{2}\frac{1}{\ell+1}$
MPPH($*, \ell$)	2ℓ
MPPH($*, \ell$) with at least one 2 per genotype	$\ell + 2 - \frac{2}{\ell+1}$

Table 1.2: Approximation ratios achieved in this thesis (for $\ell \geq 2$).

All these results will be presented in Chapter 3 and have been published previously in [Cil05; Ier06; Ier08a].

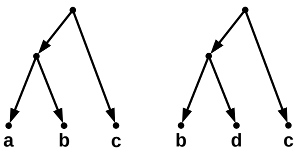
1.5 Problems from Phylogenetics

Phylogenetics studies the evolutionary relationships between groups of organisms. These groups are called *taxa* and can for example be species or different variants of a species. The simplest form of an evolutionary history is a tree-shape, called a *phylogenetic tree*, see Figure 1.4. However, biological processes as recombination, hybridisation and horizontal gene transfer can obstruct description of an evolutionary history by a tree. Such evolutionary events are called *reticulation* events and in a mathematical graph model they can be displayed as vertices with indegree two and outdegree one. This leads to the notion of *phylogenetic networks*, directed acyclic graphs where the leaves (vertices with outdegree zero) represent the taxa, there is one root and all other vertices have either indegree one and outdegree two (*split vertices*) or indegree two and outdegree one (*reticulations*).[1]

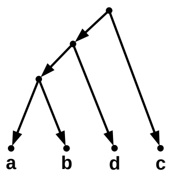
The second part of this thesis studies the reconstruction of such phylogenetic networks from *triplets*; phylogenetic trees for three taxa. These triplets form the basic building blocks of phylogenetic trees in the sense that each phylogenetic tree can be uniquely described by a set of triplets. Thus, if phylogenetic trees are available as input, triplets can be induced from the trees. Moreover, if sequence data is available as input, one can compute a triplet for each combination of three sequences by using well-known methods as Maximum Parsimony or Maximum Likelihood. Here it is studied how the obtained triplets can be combined into a phylogenetic network.

A simple example of this problem is displayed in Figure 1.3. Suppose we consider four species named a, b, c and d. In Figure 1.3(a) is a possible input triplet set; the first triplet representing the evolutionary relation among species a, b and c and the second triplet among species b, c and d. In this case a possible evolution for all four species is depicted in Figure 1.3(b), which in some sense respects the evolutionary relationships defined by the triplets. This is formalised in the following definition.

Definition 1.2. *A network N is* consistent *with a triplet $xy|z$ if N contains a subdivision of $xy|z$, i.e. if N contains distinct vertices u and v and pairwise internally vertex-disjoint paths $u \to x$, $u \to y$, $v \to u$ and $v \to z$.*



(a) Evolutionary relationships of two triples of species.



(b) Possible evolutionary history of all four species.

Figure 1.3: A fundamental problem in phylogenetics: how can we construct one large evolutionary history from various smaller ones?

[1]Many other definitions of phylogenetic networks have been proposed, sometimes using undirected graphs or directed graphs with vertices with outdegree greater than two.

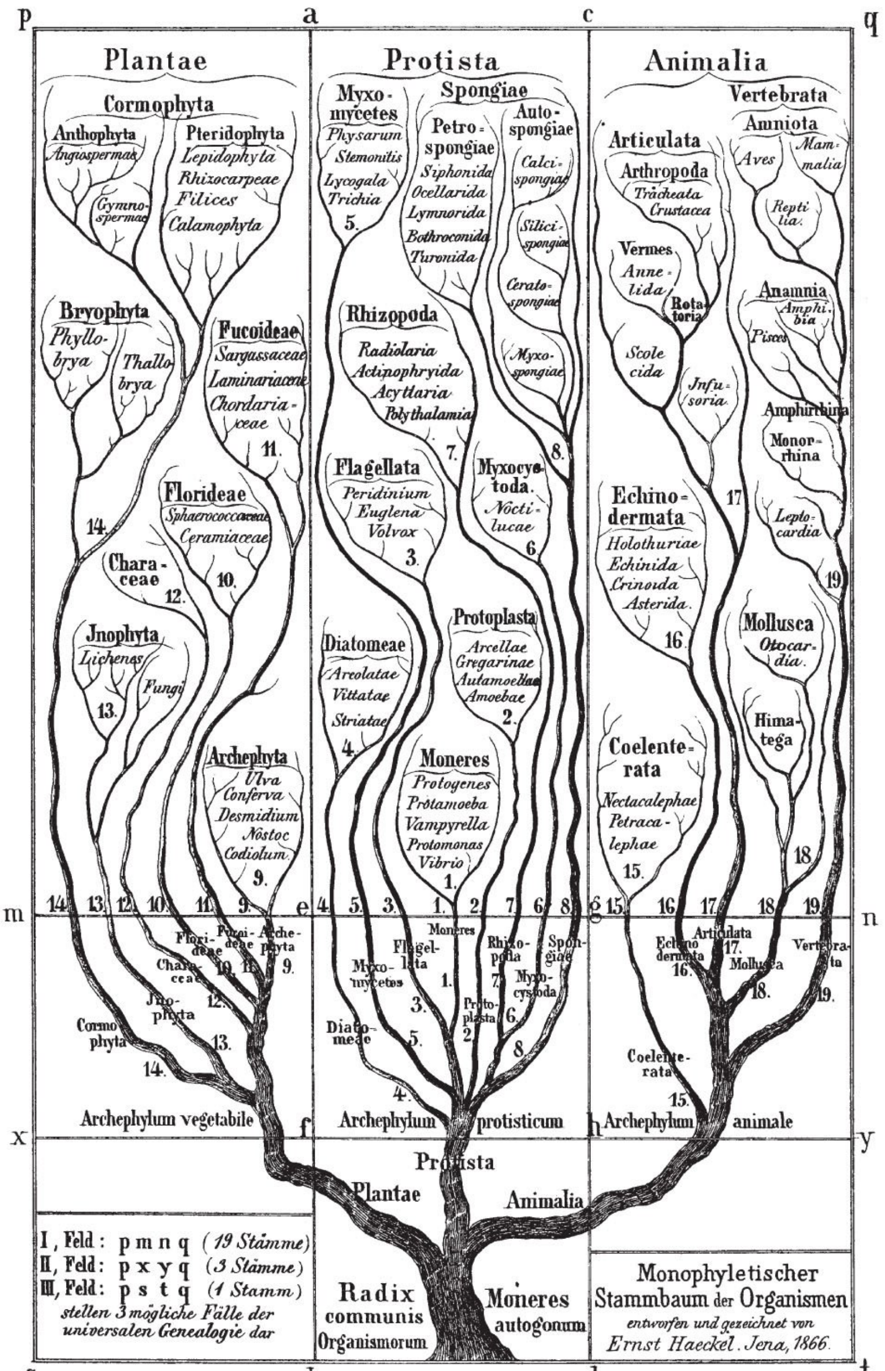
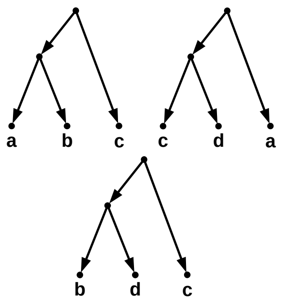
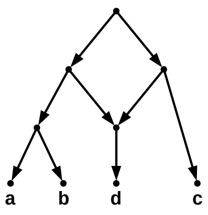


Figure 1.4: Ernst Haeckel's Monophyletic tree of organisms, 1866.

If we add one triplet, as in Figure 1.5(a), there is no tree-like evolution possible anymore. One of the input triplets could be incorrect. However, it is also possible that the real evolutionary history is not tree-like. For example, if species d originated from a hybridisation between an ancestor of c and a common ancestor of a and b, meaning that the evolutionary history of these four species is the one visualised in Figure 1.5(b). This latter evolutionary history is an example of a phylogenetic network; one that is consistent with all input triplets.



(a) Evolutionary relationships of three triples of species.



(b) Possible evolutionary history of all four species.

Figure 1.5: Example input set of triplets that is consistent with a phylogenetic network.

The above model of a phylogenetic network allows for many different degrees of complexity, ranging from trees to complex webs of frequently diverging and recombining lineages. In Section 4.3.1, it will be shown that, if we do not put any further restrictions on the constructed network, then there is a single network which is consistent with any input triplet set. However, this does not communicate any useful information. Therefore, this thesis puts restrictions on the complexity of a network. Two distinct measures of complexity are considered. Firstly, the total number of reticulations in the network is considered. The second measure of complexity considers the non-treelike parts of the network; the biconnected components (formally defined in Section 4.2). A network is said to be a *level-k network* if each biconnected component contains at most k reticulations. For example, the tree in Figure 1.3(b) is a level-0 network and in Figure 1.5(b) is a level-1 network. One of the motivations for introducing the level of a network is the hope that, for small k, constructing level-k networks might be tractable. This leads to the first problem from phylogenetics considered in this thesis.

Consistent Level-k Network (CL-k)

Input: A triplet set T.

Output: A level-k network consistent with all triplets in T (if such a network exists).

This problem CL-k was shown to be NP-hard for $k = 1$ [Jan06a] and in Section 4.3 it is shown that this problem is in fact NP-hard for all $k > 0$. However, Jansson and Sung [Jan06b] also showed that this problem becomes polynomial-time solvable for $k = 1$ if the input triplet set is *dense*, i.e. if it contains at least one triplet for each combination of three taxa. In Section 4.6 of this thesis, it is shown that it is even possible to construct level-2 networks from dense triplet sets in polynomial time.

The dense level-2 algorithm described in Section 4.6 has been tested and applied to biological data consisting of sequences of different isolates of the yeast *Cryptococcus gattii*. The evolutionary relationships between these isolates are of special interest since one version of this yeast turned out to be dangerous. Normally, *Cryptococcus gattii* is only seen in tropical and subtropical regions. However, since 1999, the yeast is also seen on the Westcoast of Canada, where it caused many human infections and even some fatalities [Kid04]. Because recombinations could be the cause of this outbreak, constructing phylogenetic networks for this yeast is particularly interesting.

Not surprisingly, triplet data can, like any kind of biological data, contain errors. The next problem considered in this thesis also attempts to construct level-k networks from triplets. However, this problem also takes into account that some of the input triplets might not be derived correctly.

Maximum Consistent Level-k Network (MaxCL-k)

Input: A triplet set T.

Output: A level-k network consistent with the maximum number of triplets in T that any level-k network is consistent with.

Because CL-k is NP-hard, it follows directly that also the more general problem MaxCL-k is NP-hard for all $k > 0$. Moreover, also MaxCL-0 is NP-hard [Bry97; Jan01; Wu04] and, in Section 4.3, it is shown that MaxCL-k even remains NP-hard for dense input triplet sets, for all $k \geq 0$. It is therefore interesting to explore exponential-time approaches to this problem. In Section 4.4, an exponential-time exact algorithm is presented, which solves this problem for level-1.

Given that a level-k network consistent with all input triplets exists, a next question to ask is whether it is also possible to find such a network with a minimum number of reticulations. Minimising reticulations is well-studied in (slightly) different contexts [Son04; Bar05; Bor07b]. It derives its legitimacy from the parsimony principle, discussed in Section 1.2.

Minimum Reticulation Level-k Network (MinRL-k)

Input: A triplet set T.

Output: A level-k network consistent with all triplets in T (if such a network exists) and containing a minimum number of reticulations over all such networks.

As a direct result of the NP-hardness of CL-k, also MinRL-k is NP-hard in general for all $k > 0$. However, in Section 4.7, it is shown that for dense triplet sets MinRL-1 and MinRL-2 are both polynomial-time solvable. For $k > 2$, the complexity of the problem MinRL-k remains, like the complexity of CL-k, open.

This thesis concludes with a positive result that holds for all k. This result is interesting under a stronger assumption on the quality of the input triplets. Namely, under the assumption that the input triplet set contains *all* triplets consistent with the network. This implies that we are looking for networks that are not only required to be consistent with all input triplets but that are, in addition, *not* consistent with any other triplets. If this is indeed the case, we say that the network *reflects* the triplet set. It turns out that constructing a level-k network reflecting an input triplet set (if such a network exists) can be done in polynomial time for each fixed k. It is even possible to construct such a network that minimises simultaneously both the level of the network and the total number of reticulations in the network. Formally, the last algorithm in this thesis, in Section 4.8, solves the following problem in polynomial time.

Minimum Reflective Level-k Network (RefL-k)

Input: A triplet set T.

Output: A level-k network N that reflects T (if such a network exists) and, ranging over all such networks, minimises both the level and the number of reticulations used.

Table 1.3 summarises all complexity results in this thesis regarding CL-k, MaxCL-k, MinRL-k and RefL-k. The results will be presented in Chapter 4 and have been published previously in [Ier08b; Ier08c; Ier08d].

	General	Dense
CL-k	In P for $k = 0$ [Aho81] NP-hard for $k = 1$ [Jan06a] NP-hard for $k \geq 2$ (Section 4.3)	In P for $k = 1$ [Jan06b] In P for $k = 2$ (Section 4.6) Open for $k \geq 3$
MaxCL-k	NP-hard for $k = 0$ [Bry97; Jan01; Wu04] NP-hard for $k \geq 1$ ($\leftarrow$)	NP-hard for all k (Section 4.3)
MinRL-k	NP-hard for $k \geq 1$ (implicitly in [Jan06a])	In P for $k \leq 2$ (Section 4.7) Open for $k \geq 3$
RefL-k	In P for all k (Section 4.8)	Identical to the general case

Table 1.3: The new state of knowledge regarding constructing level-k networks from triplets. The $\leftarrow$ denotes that this result follows from the one in the cell to the right.

Chapter 2

Single Individual Haplotyping

2.1 Introduction

Haplotypes are binary strings, representing the most interesting sites (SNPs) of an individual's DNA, and can thus be thought of as a "fingerprint" for that individual (see Section 1.4). This chapter considers the reconstruction of the haplotypes of an individual using (potentially) incomplete and/or imperfect fragments of sequencing data. This biologically-motivated field of SNP and haplotype analysis has spawned a rich variety of combinatorial problems, which are well described in surveys such as [Bon03] and [Hal03].

This chapter focusses on two such combinatorial problems, both variants of the Single Individual Haplotyping Problem (SIH) [Lan01]. This problem arises from the fact that *diploid* organisms, such as humans, have two versions of each chromosome; one each from the individual's mother and father. SIH amounts to determining the two haplotypes of an individual given fragments of sequencing data where the fragments potentially have read errors and, crucially, where it is *not* known which of the two chromosomes each fragment was read from. This chapter considers two well-known variants of the problem: *Minimum Error Correction* (MEC) and *Longest Haplotype Reconstruction* (LHR), both defined in Section 1.4.1.

These problems have been discussed - sometimes under different names - in papers such as [Bon03], [Pan04], [Gre04] and (implicitly) [Lan01]. One question arising from this discussion is how the distribution of holes in the input data affects computational complexity. Two special cases of MEC and LHR that are considered to be practically relevant are the gapless case and the 1-gap case. In the gapless case each haplotype fragment consists of a contiguous block of 0s and 1s, while in the 1-gap case there can be two disjoint such blocks, separated by a block of holes. In both cases there can be holes to the left and to the right of these blocks.

Section 2.2.1 describes the first proof that Gapless-MEC (and hence 1-Gap-MEC and also the general MEC) is NP-hard. This is done by reduction from MAX-CUT. (As far we are am aware, other claims of this result are based explicitly or implicitly on results found in [Kle98b]; as will be discussed in Section 2.2.3, the results in [Kle98b] cannot be used for this purpose.)

The NP-hardness of 1-Gap-MEC (and general MEC) follows immediately from the proof that Gapless-MEC is NP-hard. However, the NP-hardness proof for Gapless-MEC is not approximation-preserving, and consequently tells us little about the (in)approximability of Gapless-MEC, 1-Gap-MEC and general MEC. In this respect, Section 2.2.2 provides a proof that 1-Gap-MEC is APX-hard, thus excluding (unless P=NP) the existence of a *Polynomial Time Approximation Scheme* (PTAS) for 1-Gap-MEC (and general MEC).

In Section 2.2.3 the problem Binary-MEC is defined, where the input matrix contains no holes. It is argued that the complexity of this problem is still - intriguingly - open. Subsequently, a version of binary-MEC is considered where the number of haplotypes is not fixed at two, but is part of the input. It is shown that this problem is NP-hard in Section 2.2.4.

In Section 2.3.1 it is shown that Gapless-LHR is polynomial-time solvable and a dynamic programming algorithm for this problem is given which runs in time $O(n^2m + n^3)$ for an $n \times m$ input matrix. This improves upon the result by Lancia et al. [Lan01] which also showed a polynomial-time algorithm for Gapless-LHR but under the restricting assumption of non-nested input rows.

Finally, Section 2.3.2 proves that LHR is APX-hard (and thus also NP-hard) in the general case, by proving the much stronger result that 1-Gap-LHR is APX-hard. Although there is a claim in [Lan01], made very briefly, that LHR is NP-hard in general, this is not substantiated. Therefore, our result is the first proof of hardness for both 1-Gap-LHR and general LHR.

2.2 Minimum Error Correction (MEC)

For a length-m string $X \in \{0, 1, -\}^m$, and a length-m string $Y \in \{0, 1\}^m$, we define $d(X, Y)$ as the number of *mismatches* between the strings i.e. positions where X is 0 and Y is 1, or vice-versa; holes do not contribute to the mismatch count. Recall the definition of *feasible* from Section 1.4.1; an alternative, and equivalent, definition (which is used in the proofs in this section) is as follows. An $n \times m$ SNP matrix M is *feasible* if there exist two strings (haplotypes) $H_1, H_2 \in \{0, 1\}^m$, such that for all rows r of M, $d(r, H_1) = 0$ or $d(r, H_2) = 0$.

In addition, recall that a *flip* is where a 0 entry is converted to a 1, or vice-versa. Flipping to or from holes is not allowed and the haplotypes H_1 and H_2 may not contain holes.

2.2.1 Complexity of Gapless-MEC

Gapless-MEC

Input:	A gapless SNP matrix M.
Output:	The smallest number of flips needed to make M feasible.

Theorem 2.1. Gapless-MEC *is NP-hard.*

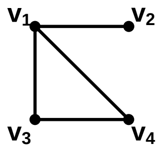
Proof. We give a reduction from MAX-CUT, which is the problem of computing the size of a maximum cardinality cut in a graph. Let $G = (V, E)$ be the input to MAX-CUT, where E is undirected. (We identify, without loss of generality, V with $\{1, 2, ..., |V|\}$.) We construct an input matrix M for Gapless-MEC with $2k|V| + |E|$ rows and $2|V|$ columns where $k = 2|E||V|$. We use M_0 to refer to the first $k|V|$ rows of M, M_1 to refer to the second $k|V|$ rows of M, and M_G to refer to the remaining $|E|$ rows. M_0 consists of $|V|$ consecutive blocks of k identical rows. Each row in the i-th block (for $1 \leq i \leq |V|$) contains a 0 at columns $2i - 1$ and $2i$ and holes at all other columns. M_1 is defined similar to M_0 with 1-entries instead of 0-entries. Each row of M_G encodes an edge from E: for edge $\{i, j\}$ (with $i < j$) we specify that columns $2i - 1$ and $2i$ contain 0s, columns $2j - 1$ and $2j$ contain 1s, and for all $h \neq i, j$, column $2h - 1$ contains 0 and column $2h$ contains 1. (See Figures 2.1(a) and 2.2(b) for an example of how M is constructed.)

Suppose t is the largest cut possible in G and s is the minimum number of flips needed to make M feasible. We claim that the following holds:

$$s = |E|(|V| - 2) + 2(|E| - t). \tag{2.1}$$

From this t, the optimal solution of MAX-CUT, can easily be computed. First, note that the solution to Gapless-MEC is trivially upper bounded by $|V||E|$. This follows because we could simply flip every 1 entry in M_G to 0; the resulting overall matrix would be feasible because we could just take H_1 as the all-0 string and H_2 as the all-1 string. Now, we say a haplotype H has the *double-entry* property if, for all odd-indexed positions (i.e. columns) j in H, the entry at position j of H is the same as the entry at position $j + 1$. We argue that a minimal number of feasibility-inducing flips will *always* lead to two haplotypes H_1, H_2 such that both haplotypes have the double-entry property and, further, H_1 is the bitwise complement of H_2. (We describe such a pair of haplotypes as *partition-encoding.*) This is because, if H_1, H_2 are not partition-encoding, then at least $k > |V||E|$ (in contrast with zero) entries in M_0 and/or M_1 will have to be flipped, meaning this strategy is doomed to begin with.

Now, for a given partition-encoding pair of haplotypes, it follows that - for each row in M_G - we will have to flip either $|V| - 2$ or $|V|$ entries to reach its



(a) Example input to MAX-CUT.

$$\left(\begin{array}{cccccccc}
0 & 0 & - & - & - & - & - & - \\
- & - & 0 & 0 & - & - & - & - \\
- & - & - & - & 0 & 0 & - & - \\
- & - & - & - & - & - & 0 & 0 \\
1 & 1 & - & - & - & - & - & - \\
- & - & 1 & 1 & - & - & - & - \\
- & - & - & - & 1 & 1 & - & - \\
- & - & - & - & - & - & 1 & 1 \\
0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 & 0 & 1 & 1
\end{array}\right)
\begin{array}{l}
\left.\vphantom{\begin{array}{c}0\\0\\0\\0\\0\\0\\0\\0\end{array}}\right\}\text{32 copies}\\
\left.\vphantom{\begin{array}{c}0\\0\\0\\0\end{array}}\right\}M_G
\end{array}$$

(b) Constructed matrix M.

Figure 2.1: Example of the reduction in Theorem 2.1.

nearest haplotype. This is because, irrespective of which haplotype we move a row to, the $|V|-2$ pairs of columns *not* encoding end-points (for a given row) will always cost 1 flip each to fix. Then either 2 or 0 of the 4 "endpoint-encoding" entries will also need to be flipped; 4 flips will never be necessary because then the row could move to the other haplotype, requiring no extra flips. Gapless-MEC thus maximises the number of rows which require $|V|-2$ rather than $|V|$ flips. If we think of H_1 and H_2 as encoding a partition of the vertices of V (i.e. a vertex i is on one side of the partition if H_1 has 1s in columns $2i-1$ and $2i$, and on the other side if H_2 has 1s in those columns), it follows that each row requiring $|V|-2$ flips corresponds to a cut-edge in the vertex partition defined by H_1 and H_2. The expression (2.1) follows. □

2.2.2 Approximability of 1-Gap-MEC

1-Gap-MEC

Input: An SNP matrix M with at most 1 gap per row.

Output: The smallest number of flips needed to make M feasible.

To prove that 1-Gap-MEC is APX-hard an *L-reduction* will be given. This is a specific type of *approximation-preserving* reduction, first introduced in [Pap91]. If there exists an L-reduction from a problem X to a problem Y, then a PTAS for Y can be used to build a PTAS for X. Conversely, if there exists an

L-reduction from X to Y and X is APX-hard then so is Y. See (for example) [Hoo01] for a succinct discussion of this. We will reduce from CUBIC-MIN-UNCUT, which is the problem of finding the minimum number of edges that have to be removed from a 3-regular graph in order to make it bipartite. Our first goal is thus to prove the APX-hardness of CUBIC-MIN-UNCUT, which itself will be proven using an L-reduction from the APX-hard problem CUBIC-MAX-CUT.

To help the reader, we reproduce here the definition of an L-reduction.

Definition 2.1. *(Papadimitriou and Yannakakis [Pap91]) Let A and B be two optimisation problems. An* L-reduction *from A to B is a pair of functions R and S, both computable in polynomial time, such that for any instance I of A with optimum cost Opt(I), R(I) is an instance of B with optimum cost Opt(R(I)) and for every feasible solution s of R(I), S(s) is a feasible solution of I such that:*

$$Opt(R(I)) \leq \alpha Opt(I), \tag{2.2}$$

for some positive constant α and:

$$|Opt(I) - c(S(s))| \leq \beta |Opt(R(I)) - c(s)|, \tag{2.3}$$

for some positive constant β, where c(S(s)) and c(s) represent the costs of S(s) and s, respectively.

Observation 2.1. CUBIC-MIN-UNCUT *is APX-hard.*

Proof. We give an L-reduction from CUBIC-MAX-CUT, the problem of finding the maximum cardinality of a cut in a 3-regular graph. (This problem is shown to be APX-hard in [Ali97]; see also [Ber99].) Let $G = (V, E)$ be the input to CUBIC-MAX-CUT.

Note that CUBIC-MIN-UNCUT is the "complement" of CUBIC-MAX-CUT, as expressed by the following relationship:

$$\text{CUBIC-MAX-CUT(G)} = |E| - \text{CUBIC-MIN-UNCUT(G)}. \tag{2.4}$$

Here, and throughout this chapter, $P(I)$ is used to denote the optimal value of problem P on input I.

To see why (2.4) holds, note that for every cut C, the removal of the edges in $E \setminus C$ will lead to a bipartite graph. On the other hand, given a set of edges E' whose removal makes G bipartite, the complement is not necessarily a cut. However, given a bipartition induced by the removal of E', the edges from the original graph that cross this bipartition form a cut C', such that $|C'| \geq |E \setminus E'|$. This proves (2.4), and the mapping (just described) from E' to C' is the mapping we use in the L-reduction.

Now, note that property (2.2) of the L-reduction is easily satisfied (taking $\alpha = 1$) because the optimal value of CUBIC-MIN-UNCUT is always less than or equal to the optimal value of CUBIC-MAX-CUT. This follows from the combination of (2.4) with the fact that a maximum cut in a 3-regular graph always contains at least 2/3 of the edges: if a vertex has less than two incident edges in the cut then we can get a larger cut by moving this vertex to the other side of the partition.

To see that property (2.3) of the L-reduction is easily satisfied (taking $\beta = 1$), let E' be any set of edges whose removal makes G bipartite. Property (2.3) is satisfied because E' gets mapped to a cut C', as defined above, and combined with (2.4) this gives:

$$\begin{aligned}\text{CUBIC-MAX-CUT(G)} - |C'| &\leq \text{CUBIC-MAX-CUT(G)} - |E \setminus E'| \\ &= |E'| - \text{CUBIC-MIN-UNCUT(G)}.\end{aligned}$$

This completes the L-reduction from CUBIC-MAX-CUT to CUBIC-MIN-UNCUT, proving the APX-hardness of CUBIC-MIN-UNCUT.

□

We also need the following observation.

Observation 2.2. *Let $G = (V, E)$ be an undirected, 3-regular graph. Then we can find, in polynomial time, an orientation of the edges of G so that each vertex has either in-degree 2 and out-degree 1 ("in-in-out") or out-degree 2 and in-degree 1 ("out-out-in").*

Proof. (We assume that G is connected; if G is not connected, we can apply the following argument to each component of G in turn, and the overall result still holds.) Every cubic graph has an even number of vertices, because every graph must have an even number of odd-degree vertices. We add an arbitrary perfect matching to the graph, which may create multiple edges. The graph is now 4-regular and therefore has an Euler tour. We direct the edges following the Euler-tour; every vertex is now in-in-out-out. If we remove the perfect matching edges we added, we are left with an oriented version of G where every vertex is in-in-out or out-out-in. This can all be done in polynomial time. □

Theorem 2.2. 1-Gap-MEC *is APX-hard.*

Proof. We give a reduction from CUBIC-MIN-UNCUT. Consider an arbitrary 3-regular graph $G = (V, E)$ and orient the edges as described in Observation 2.2 to obtain an oriented version of G, $\overrightarrow{G} = (V, \overrightarrow{E})$, where each vertex is either in-in-out or out-out-in. We construct an $|E| \times |V|$ input matrix M for 1-Gap-MEC as follows. The columns of M correspond to the vertices of $\overrightarrow{G}$ and every row of M encodes an oriented edge of $\overrightarrow{G}$; it has a 0 in the column corresponding to the tail of the edge (i.e. the vertex from which the edge leaves), a 1 in the column corresponding to the head of the edge and it has holes in the remaining columns.

We prove the following:

$$\text{CUBIC-MIN-UNCUT(G)} = \text{1-G}\textsc{ap}\text{-MEC(M)}. \tag{2.5}$$

We first prove that:

$$\text{1-G}\textsc{ap}\text{-MEC(M)} \leq \text{CUBIC-MIN-UNCUT(G)}. \tag{2.6}$$

To see this, let E' be a minimal set of edges whose removal makes G bipartite, and let $|E'| = k$. Let $B = (L \cup R, E \setminus E')$ be the bipartite graph (with bipartition $L \cup R$) obtained from G by removing the edges E'. Let H_1 (respectively, H_2) be the haplotype that has 1s in the columns representing vertices of L (respectively, R) and 0s elsewhere. It is possible to make M feasible with k flips, by the following process: for each edge in E', flip the 0 bit in the corresponding row of M to 1. For each row r of M it is now true that $d(r, H_1) = 0$ or $d(r, H_2) = 0$, proving the feasibility of M.

The proof that

$$\text{CUBIC-MIN-UNCUT(G)} \leq \text{1-G}\textsc{ap}\text{-MEC(M)} \tag{2.7}$$

is more subtle. Suppose we can render M feasible using j flips, and let H_1 and H_2 be any two haplotypes such that, after the j flips, each row of M is distance 0 from either H_1 or H_2. If H_1 and H_2 are bitwise complementary then we can make G bipartite by removing an edge whenever we had to flip a bit in the corresponding row. The idea is, namely, that the 1s in H_1 (respectively, H_2) represent the vertices L (respectively, R) in the resulting bipartition $L \cup R$.

However, suppose the two haplotypes H_1 and H_2 are not bitwise complementary. In this case it is sufficient to demonstrate that there also exists bitwise complementary haplotypes H'_1 and H'_2 such that, after j (or fewer) flips, every row of M is distance 0 from either H'_1 or H'_2. Consider thus a column of H_1 and H_2 where the two haplotypes are not complementary. Crucially, the orientation of $\overrightarrow{G}$ ensures that every column of M contains *either* one 1 and two 0s *or* two 1s and one 0 (and the rest holes). A simple case analysis shows that, because of this, we can always change the value of one of the haplotypes in that column, without increasing the number of flips. (The number of flips might decrease.) Repeating this process for all columns of H_1 and H_2 where the same value is observed thus creates complementary haplotypes H'_1 and H'_2, and - as described in the previous paragraph - these haplotypes then determine which edges of G should be removed to make G bipartite. This completes the proof of (2.5).

The above reduction can be computed in polynomial time and is an L-reduction. From (2.5) it follows directly that property (2.2) of an L-reduction is satisfied with $\alpha = 1$. Property (2.3), with $\beta = 1$, follows from the proof of (2.7), combined with (2.5). Namely, whenever we use (say) t flips to make M feasible, we can find $s \leq t$ edges of G that can be removed to make G bipartite. Combined with (2.5) this gives:

$$|\text{CUBIC-MIN-UNCUT(G)} - s| \leq |\text{1-G}\textsc{ap}\text{-MEC(M)} - t|.$$

□

2.2.3 Binary-MEC

From a mathematical point of view it is interesting to determine whether MEC remains NP-hard when the input matrix is further restricted. Let us therefore define the following problem.

Binary-MEC

Input:	An SNP matrix M that does not contain any holes.
Output:	The smallest number of flips needed to make M feasible.

To elaborate, it is claimed in several papers (e.g. [Alo99]) that a problem equivalent to Binary-MEC is NP-hard. Such claims inevitably refer to the seminal paper *Segmentation Problems* by Kleinberg, Papadimitriou, and Raghavan (KPR), which has appeared in multiple different forms since 1998 [Kle98b; Kle98a; Kle04]. However, the KPR papers actually discuss two superficially similar, but essentially different, problems: one problem is essentially equivalent to Binary-MEC, and the other is a more general (and thus, potentially, a more difficult) problem. This more general problem allows the entries of the input matrix to be drawn arbitrarily from $\mathbb{R}$, which makes it much easier to prove NP-hardness. Communication with the authors [Pap05] has confirmed that they have no proof of hardness for the former problem, i.e. the problem that is essentially equivalent to Binary-MEC.

Thus we conclude that the complexity of Binary-MEC is still open. From an approximation viewpoint the problem has been quite well-studied; the problem has a *Polynomial Time Approximation Scheme* (PTAS) because it is a special form of the *Hamming 2-Median Clustering Problem.* A randomised PTAS was demonstrated in [Ost02] and later a deterministic PTAS in [Jia04]. Other approximation results appear in [Kle98b], [Alo99], [Kle04] and a heuristic for a similar problem appears in [Pan04]. We also know that, if the number of haplotypes to be found is specified as part of the input (and not fixed as 2), the problem becomes NP-hard; we prove this in the following section. Finally, it may also be relevant that the "geometric" version of the problem (where rows of the input matrix are not drawn from $\{0,1\}^m$ but from $\mathbb{R}^m$, and Euclidean distance is used instead of Hamming distance) is also open from a complexity viewpoint [Ost02]. (However, the version using Euclidean-distance-squared *is* known to be NP-hard [Dri04].)

2.2.4 Binary-MEC with more than two Haplotypes

Let us now consider a generalisation of the problem Binary-MEC, where the number of haplotypes is not fixed as two, but part of the input.

Parameterised-Binary-MEC (PBMEC)

Input: An SNP matrix M that contains no holes, and a natural number $k \geq 1$.

Output: The smallest number of flips needed to make M feasible under k haplotypes.

The notion of *feasible* generalises easily to $k \geq 1$ haplotypes: an SNP matrix M is *feasible* under k haplotypes if the rows of M can be partitioned into k groups such that all the rows within each group are pairwise non-conflicting. Given an SNP matrix M and a natural number k, the problem PBMEC thus aims to find haplotypes $H_1, \ldots, H_k$ minimising

$$D_{M,k}(H_1, \ldots, H_k) = \sum_{\text{rows } r \text{ of } M} \min(d(r, H_1), d(r, H_2), ..., d(r, H_k)). \qquad (2.8)$$

We define $OptTuples(M, k)$ as the set of unordered optimal k-tuples of haplotypes for M i.e. those k-tuples of haplotypes which have an optimal $D_{M,k}$ score. Denote this optimal score by PBMEC(M, k).

Theorem 2.3. PBMEC *is NP-hard.*

Proof. We reduce from the NP-hard problem MIN-VERTEX-COVER. Let $G = (V, E)$ be an undirected graph. A subset $V' \subseteq V$ is said to *cover* an edge $(u, v) \in E$ if $u \in V'$ or $v \in V'$. A *vertex cover* of an undirected graph $G = (V, E)$ is a subset U of the vertices such that every edge in E is covered by U. Given a graph G, MIN-VERTEX-COVER is the problem of computing the size of a minimum cardinality vertex cover U of G.

Let $G = (V, E)$ be the input to MIN-VERTEX-COVER. We construct an SNP matrix M as follows. M has $|V|$ columns and $3|E||V|+|E|$ rows. We name the first $3|E||V|$ rows M_0 and the remaining $|E|$ rows M_G. M_0 is the matrix obtained by taking the $|V| \times |V|$ identity matrix (i.e. 1s on the diagonal, 0s everywhere else) and making $3|E|$ copies of each row. Each row in M_G encodes an edge of G: the row has 1-entries at the endpoints of the edge, and the rest of the row is 0. We argue shortly that, to compute the size of the smallest vertex cover in G, we call PBMEC(M, k) for increasing values of k (starting with $k = 2$) until we first encounter a k such that:

$$PBMEC(M, k) = 3|E|(|V| - (k - 1)) + |E|. \qquad (2.9)$$

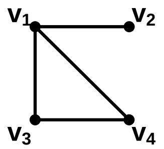
Once the smallest such k has been found, we can output that the size of the smallest vertex cover in G is $k-1$. Actually, if we haven't yet found a value $k < |V| - 2$ satisfying the above equation, we can check by brute force in polynomial-time whether G has a vertex cover of size $|V|-3$, $|V|-2$, $|V|-1$, or $|V|$. The reason for wanting to ensure that PBMEC(M, k) is not called with $k \geq |V| - 2$ is explained later in the analysis. Note that, should we wish to build a Karp reduction from the decision version of MIN-VERTEX-COVER to the decision version of PBMEC, it is not a problem to make this brute force checking fit into the framework of a Karp reduction. The Karp reduction can do the brute force checking itself and use trivial inputs to the decision version of PBMEC to communicate its "yes" or "no" answer.

It remains only to prove that (for $k < |V| - 2$) (2.9) holds if and only if G has a vertex cover of size $k-1$.

To prove this we need to first analyse $OptTuples(M_0, k)$. Recall that M_0 was obtained by duplicating the rows of the $|V| \times |V|$ identity matrix. Let $I_{|V|}$ be shorthand for the $|V| \times |V|$ identity matrix. Given that M_0 is simply a "scaled up" version of $I_{|V|}$, it follows that:

$$OptTuples(M_0, k) = OptTuples(I_{|V|}, k). \tag{2.10}$$

Now, we argue that all the k-tuples in $OptTuples(I_{|V|}, k)$ (for $k < |V| - 2$) have the following form: one haplotype from the tuple contains only 0s, and the remaining $k-1$ haplotypes from the tuple each have precisely one entry set to 1. Let us name such a k-tuple a *candidate* tuple.



(a) Example input graph to MIN-VERTEX-COVER.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix} \begin{matrix} \left.\vphantom{\begin{matrix}1\\1\\1\\1\end{matrix}}\right\} \text{12 copies} \\ \left.\vphantom{\begin{matrix}1\\1\\1\\1\end{matrix}}\right\} M_G \end{matrix}$$

(b) Constructed matrix M.

Figure 2.2: Example of the reduction in Theorem 2.3.

First, note that $PBMEC(I_{|V|}, k) \leq |V| - (k-1)$, because $|V| - (k-1)$ is the value of the D measure - defined in (2.8) - under any candidate tuple.

Secondly, under an arbitrary k-tuple there can be at most k rows of $I_{|V|}$ which contribute 0 to the D measure. However, if precisely k rows of $I_{|V|}$ contribute 0 to the D measure (i.e., every haplotype has precisely one entry set to 1, and the haplotypes are all distinct) then there are $|V| - k$ rows which each contribute 2 to the D measure; such a k-tuple cannot be optimal because it has a D measure of $2(|V| - k) > |V| - (k-1)$. So we reason that at most $k-1$ rows contribute 0 to the D measure. In fact, *precisely* $k-1$ rows must contribute 0 to the D measure because, otherwise, there would be at least $|V| - (k-2)$ rows contributing at least 1, and this is not possible because $PBMEC(I_{|V|}, k) \leq |V| - (k-1)$. So $k-1$ of the haplotypes correspond to rows of $I_{|V|}$, and the remaining $|V| - (k-1)$ rows of $I_{|V|}$ must each contribute 1 to the D measure. But the only way to do this (given that $|V| - (k-1) > 2$) is to make the kth haplotype the haplotype where every entry is 0. Hence:

$$PBMEC(I_{|V|}, k) = |V| - (k-1) \tag{2.11}$$

and:

$$PBMEC(M_0, k) = 3|E|(|V| - (k-1)). \tag{2.12}$$

$OptTuples(I_{|V|}, k)$ ($= OptTuples(M_0, k)$) is, by extension, precisely the set of candidate k-tuples.

The next step is to observe that $OptTuples(M, k) \subseteq OptTuples(M_0, k)$. To see this, suppose (by way of contradiction) that it is not true, and there exists a k-tuple $H^* \in OptTuples(M, k)$ that is not in $OptTuples(M_0, k)$. But then replacing H^* by any k-tuple out of $OptTuples(M_0, k)$ would reduce the number of flips needed in M_0 by at least $3|E|$, in contrast to an increase in the number of flips needed in M_G of at most $2|E|$, thus leading to an overall reduction in the number of flips; contradiction! (The $2|E|$ figure is the number of flips required to make all rows in M_G equal to the all-0 haplotype.)

Because $OptTuples(M, k) \subseteq OptTuples(M_0, k)$, we can restrict our attention to the k-tuples in $OptTuples(M_0, k)$. Observe that there is a natural 1-1 correspondence between the elements of $OptTuples(M_0, k)$ and all size $k-1$ subsets of V: a vertex $v \in V$ is in the subset corresponding to $H^* \in OptTuples(M_0, k)$ if and only if one of the haplotypes in H^* has a 1 in the column corresponding to vertex v.

Now, for a k-tuple $H^* \in OptTuples(M_0, k)$ we let $Cov(G, H^*)$ be the set of edges in G which are covered by the subset of V corresponding to H^*. (Thus, $|Cov(G, H^*)| = |E|$ if and only if H^* represents a vertex cover of G.) It is easy to check that, for $H^* \in OptTuples(M_0, k)$:

$$\begin{aligned} D_{M,k}(H^*) &= 3|E|(|V| - (k-1)) + |Cov(G, H^*)| + 2(|E| - |Cov(G, H^*|)) \\ &= 3|E|(|V| - (k-1)) + 2|E| - |Cov(G, H^*)|. \end{aligned}$$

Hence, for $H^* \in OptTuples(M_0, k)$, $D_{M,k}(H^*)$ equals $3|E|(|V| - (k-1)) + |E|$ if and only if H^* represents a size $k-1$ vertex cover of G. □

2.3 Longest Haplotype Reconstruction (LHR)

Recall that the rows of a feasible SNP matrix M can be partitioned into two sets, M_l and M_r, such that the rows within each set are pairwise non-conflicting. Note that this partition does not have to be unique. As explained in Section 1.4.1, from M_i ($i \in \{l, r\}$) one can build a haplotype H_i by combining the rows of M_i as follows: the jth column of H_i is set to 1 if at least one row from M_i has a 1 in column j, is set to 0 if at least one row from M_i has a 0 in column j, and is set to a hole if all rows in M_i have a hole in column j. Also recall that, in contrast to MEC, this leads to haplotypes that potentially contain holes. Only in this section haplotypes are defined as strings over $\{0, 1, -\}$, while in all other sections haplotypes are strings over $\{0, 1\}$.

Let $|H|$ denote the length of haplotype H, defined as the number of positions where it does not contain a hole; the haplotype `10-1` thus has length three, for example. The objective with LHR is to remove *rows* from M to make it feasible but also such that the sum of the lengths of the two resulting haplotypes is maximised. We define the function LHR(M) (which gives a natural number as output) as the largest value this sum-of-lengths value can take, ranging over all feasibility-inducing row-removals and subsequent partitions.

In Section 2.3.1 we provide a polynomial-time dynamic programming algorithm for the gapless variant of LHR, Gapless-LHR. In Section 2.3.2 we show that LHR becomes APX-hard and NP-hard when at most one gap per input row is allowed, automatically also proving the hardness of LHR in the general case.

2.3.1 Polynomial-time Algorithm for Gapless-LHR

Gapless-LHR

Input:	A gapless SNP matrix M.
Output:	The value LHR(M), as defined above.

The LHR problem for gapless matrices was proved to be polynomial-time solvable by Lancia et al. [Lan01], but only with the genuine restriction that no fragments are included in other fragments. Our algorithm improves this in the sense that it works for all gapless input matrices; our algorithm is similar in style to the algorithm by Bafna et al. ([Baf05]) that solves MFR (minimum fragment removal), where the objective is not to maximise the length of the haplotypes, but to minimise the number of rows removed. Note that our dynamic-programming algorithm computes Gapless-LHR(M) but it can easily be adapted to generate the rows that must be removed (and subsequently, the partition that must be made) to achieve this value.

Theorem 2.4. Gapless-LHR *can be solved in time* $O(n^2m + n^3)$.

Proof. Let M be the input to Gapless-LHR, and assume the matrix has size $n \times m$. For row i define $l(i)$ as the leftmost column that is not a hole and define $r(i)$ as the rightmost column that is not a hole. The rows of M are ordered such that $l(i) \leq l(j)$ if $i < j$. Define the matrix M_i as the matrix consisting of the first i rows of M and two extra rows at the top: row 0 and row -1, both consisting of all holes. Define $W(i)$ as the set of rows $j < i$ that are not in conflict with row i.

For $h, k \leq i$ and $h, k \geq -1$ and $r(h) \leq r(k)$ define $D[h, k; i]$ as the maximum sum of lengths of two haplotypes such that:

- each haplotype is built up as a combination of rows from M_i (in the sense explained above);
- each row from M_i can be used to build at most one haplotype (i.e. it cannot be used for both haplotypes);
- row k is one of the rows used to build a haplotype and among such rows maximises $r(\cdot)$;
- row h is one of the rows used to build the haplotype for which k is not used and among such rows maximises $r(\cdot)$.

The optimal solution of the problem, LHR(M), is given by:

$$\max_{h,k \mid r(h) \leq r(k)} D[h, k; n]. \tag{2.13}$$

This optimal solution can be calculated by starting with $D[h, k, 0] = 0$ for $h, k \in -1, 0$ and using the following recursive formulas. We distinguish three different cases, the first is that $h, k < i$. Under these circumstances:

$$D[h, k; i] = D[h, k; i - 1]. \tag{2.14}$$

This equation results from the following. First of all, if $r(i) > r(k)$, then row i cannot be used for the haplotype that row k is used for, because row k has maximal $r(\cdot)$ among all rows that are used for a haplotype. Secondly, if $r(i) \leq r(k)$: row i cannot increase the length of the haplotype that row k is used for (because also $l(i) \geq l(k)$). Finally, the same arguments hold for h.

The second case is when $h = i$; $D[i, k; i]$ is equal to:

$$\max_{\substack{j \in W(i), \quad j \neq k \\ r(j) \leq r(i)}} D[j, k; i - 1] + f(i, j). \tag{2.15}$$

Where $f(i,j) = r(i) - \max\{r(j), l(i) - 1\}$ is the increase of the haplotype's length. Equation (2.15) results from the following. The definition of $D[i,k;i]$ says that row i has to be used for the haplotype for which k is not used and amongst such rows maximises $r(\cdot)$. Therefore, the optimal solution is achieved by adding row i to some solution that has a row j as the most-right-ending row, for some j that agrees with i, is not equal to k and ends before i. Adding row i to the haplotype leads to an increase of its length of $f(i,j) = r(i) - \max\{r(j), l(i) - 1\}$. This term is fixed, for fixed i and j and therefore we only have to consider extensions of solutions that were already optimal. Note that this reasoning does not hold for more general, "gapped", data.

The last case is when $k = i$; $D[h,i;i]$ is equal to:

$$\max_{\substack{j \in W(i),\ j \neq h \\ r(j) \leq r(i)}} \begin{cases} D[j,h;i-1] + f(i,j) & \text{if } r(h) \geq r(j), \\ D[h,j;i-1] + f(i,j) & \text{if } r(h) < r(j). \end{cases}$$

The above algorithm can be sped up by using the fact that, as a direct consequence of (2.14), $D[h,k;i] = D[h,k;max(h,k)]$ for all $h,k \leq i \leq n$. It is thus unnecessary to calculate the values $D[h,k;i]$ for $h,k < i$.

The time for calculating all the $W(i)$ is $O(n^2m)$. When all the $W(i)$ are known, it takes $O(n^3)$ time to calculate all the $D[h,k;max(h,k)]$. This is because we need to calculate $O(n^2)$ values $D[i,k;i]$ and also $O(n^2)$ values $D[h,i;i]$ that take $O(n)$ time each. This leads to an overall time complexity of $O(n^2m + n^3)$. □

2.3.2 Complexity and Approximability of 1-Gap-LHR

1-Gap-LHR

Input:	An SNP matrix M with at most one gap per row.
Output:	The value LHR(M), as defined earlier.

In this section we prove that 1-Gap-LHR is APX-hard (and thus also NP-hard). We prove this by demonstrating (indirectly) an L-reduction from the problem CUBIC-MAX-INDEPENDENT-SET - the problem of computing the maximum cardinality of an independent set in a cubic graph - which is itself proven APX-hard in [Ali97].

We reduce via the intermediate problem *Single Haplotype* LHR (SH-LHR). In this version of the problem rows must be removed from the input matrix until the remaining rows are mutually non-conflicting. The objective is to maximise the number of columns that have at least one non-hole entry in the remaining rows.

The reduction chain looks as follows. We first show an L-reduction from SH-LHR to LHR, such that the number of gaps per row is unchanged. We then show an L-reduction from CUBIC-MAX-INDEPENDENT-SET to 2-gap SH-LHR. Next, using an observation pertaining to the structure of cubic graphs, we show how this reduction can be adapted to give an L-reduction from CUBIC-MAX-INDEPENDENT-SET to 1-GAP-SH-LHR. This proves the APX-hardness of 1-GAP-SH-LHR and thus (by transitivity of L-reductions) also 1-GAP-LHR.

Lemma 2.1. SH-LHR *is L-reducible to* LHR*, such that the number of gaps per row is unchanged.*

Proof. Let M be the $n \times m$ input to SH-LHR. We may assume that M contains no duplicate rows, because duplicate rows are redundant when working with only one haplotype. We map the SH-LHR input, M, to the $2n \times m$ LHR input, M', by taking each row of M and making a copy of it. Informally, the idea is that the influence of the second haplotype can be neutralised by doubling the rows of the input matrix. Note that this construction clearly preserves the maximum number of gaps per row.

Now, let $SOL(M')$ be the set that contains all pairs of haplotypes (H_1, H_2) that can be induced by removing some rows of M', partitioning the remaining rows of M' into two mutually non-conflicting sets, and then reading off the two induced haplotypes. Similarly, let $SOL(M)$ be the set that contains all haplotypes H that can be induced by removing some rows of M (such that the remaining rows are mutually non-conflicting) and then reading off the single, induced haplotype. Note the following pair of observations, which both follow directly from the construction of M':

$$(H_1, H_2) \in SOL(M') \Rightarrow H_1, H_2 \in SOL(M), \tag{2.16}$$

$$H \in SOL(M) \Rightarrow (H, H) \in SOL(M'). \tag{2.17}$$

To satisfy the L-reduction we need to show how elements from $SOL(M')$ are mapped back to elements of $SOL(M)$ in polynomial time. So, let (H_1, H_2) be any pair from $SOL(M')$. If $|H_1| \geq |H_2|$ map the pair (H_1, H_2) to H_1, otherwise to H_2. This completes the L-reduction, and we now prove its correctness. Central to this is the proof of the following:

$$\text{SH-LHR}(M) = \frac{1}{2}\text{LHR}(M'). \tag{2.18}$$

The fact that $\text{SH-LHR}(M) \geq \frac{1}{2}\text{LHR}(M')$ follows immediately from (2.16) and the mapping described above. This lets us fulfil condition (2.2) of the L-reduction definition, taking $\alpha = 2$. The fact that $\text{SH-LHR}(M) \leq \frac{1}{2}\text{LHR}(M')$ follows because, by (2.17), every element in $SOL(M)$ is guaranteed to have a counterpart in $SOL(M')$ which has a total length twice as large.

We can fulfil condition (2.3) of the L-reduction by taking $\beta = \frac{1}{2}$. To see this, let (H_1, H_2) be any pair from $SOL(M')$, and (without loss of generality) assume that $|H_1| \geq |H_2|$. Let $r = \text{LHR}(M')$, the distance of (H_1, H_2) from optimal is then:

$$r - (|H_1| + |H_2|) \geq r - 2|H_1|. \tag{2.19}$$

Let $l = \text{LHR}(M)$, then:

$$\begin{aligned} l - |H_1| &= \tfrac{r}{2} - |H_1| \\ &= \tfrac{1}{2}\Big(r - 2|H_1|\Big) \\ &\leq \tfrac{1}{2}\Big(r - (|H_1| + |H_2|)\Big). \end{aligned} \tag{2.20}$$

Thus, taking $\beta = \frac{1}{2}$ satisfies condition (2.3) of the L-reduction. □

Lemma 2.2. 2-Gap-SH-LHR *is APX-hard.*

Proof. We reduce from CUBIC-MAX-INDEPENDENT-SET. Let $G = (V, E)$ be the undirected, cubic input to CUBIC-MAX-INDEPENDENT-SET. We direct the edges of G in the manner described by Observation 2.2, to give $\overrightarrow{G} = (V, \overrightarrow{E})$. Thus, every vertex of $\overrightarrow{G}$ is now out-out-in or in-in-out. A vertex w is a *child* of a vertex v if there is an edge leaving v in the direction of w i.e. $(v, w) \in \overrightarrow{E}$, and in this case v is said to be the *parent* of w.

Let v_{in} be the number of vertices in $\overrightarrow{G}$ that are in-in-out, and v_{out} be the number of vertices that are out-out-in. We build a matrix M, to be used as input to 2-Gap-SH-LHR, which has $|V|$ rows and $2v_{in} + v_{out}$ columns. The construction of M is as follows. (Each row of M will represent a vertex from V, so we henceforth index the rows of M using vertices of V.) Now, to each in-in-out vertex of $\overrightarrow{G}$, we allocate two *adjacent* columns of M, and for each out-out-in vertex, we allocate one column of M. (A column may not be allocated to more than one vertex.) Note that, for this lemma, it is not important how the columns are allocated; in the proof of Theorem 2.5, the ordering is crucial. For simplicity, we also impose an arbitrary total order P on the vertices of V.

Now, for each vertex $v \in V$, we build row v as follows. Firstly, we put 1(s) in the column(s) representing v. Secondly, consider each child w of v. If w is an out-out-in vertex, we put a 0 in the column representing w. Alternatively, w is an in-in-out vertex, so w is represented by two columns; in this case we put a 0 in the left such column (if v comes before the other parent of w in the total order P) or, alternatively, in the right column (if v comes after the other parent of w in the total order P). The rest of the row consists of holes.

This completes the construction of M. Note that rows encoding in-in-out vertices contain two adjacent 1s and one 0, with at most one gap in the row, and rows encoding out-out-in vertices contain one 1 and two 0s, with at most two gaps in the row. In either case there are precisely 3 non-hole elements per

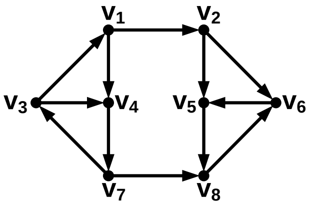


Figure 2.3: Example input graph to CUBIC-MAX-INDEPENDENT-SET (see Lemmas 2.2 and 2.3) after an appropriate edge orientation has been applied.

$$\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \end{array} \begin{array}{c} \begin{array}{cccccccccccc} v_3 & v_1 & v_2 & v_5 & v_5 & v_7 & v_8 & v_8 & v_4 & v_4 & v_6 & v_6 \end{array} \\ \left(\begin{array}{cccccccccccc} - & 1 & 0 & - & - & - & - & - & 0 & - & - & - \\ - & - & 1 & 0 & - & - & - & - & - & - & 0 & - \\ 1 & 0 & - & - & - & - & - & - & - & 0 & - & - \\ - & - & - & - & - & 0 & - & - & 1 & 1 & - & - \\ - & - & - & 1 & 1 & - & - & 0 & - & - & - & - \\ - & - & - & - & 0 & - & - & - & - & - & 1 & 1 \\ 0 & - & - & - & - & 1 & 0 & - & - & - & - & - \\ - & - & - & - & - & - & 1 & 1 & - & - & - & 0 \end{array} \right) \end{array}$$

Figure 2.4: Construction of matrix M (from Lemma 2.2 and 2.3) for graph in Figure 2.3.

row. It is also crucial to note that, reading down any one column of M, one sees exactly one 1 and exactly one 0.

Let K be any submatrix of M obtained by removing rows from M, and let $V[K] \subseteq V$ be the set of vertices whose rows appear in K. If the rows of K are mutually non-conflicting, then the haplotype induced by K has length $3r$ where r is the number of rows in K. This follows from the aforementioned facts that every column of M contains exactly one 1 and one 0. and that every row has exactly 3 non-hole elements.

We now prove that the rows of K are in conflict if and only if $V[K]$ is not an independent set. First, suppose $V[K]$ is not an independent set. Then there exist $u, v \in V[K]$ such that $(u, v) \in \overrightarrow{E}$. In row v of K there are thus 1(s) in the column(s) representing vertex v. However, there is also (in row u) a 0 in the column (or one of the columns) representing vertex v, causing a conflict. Hence, if $V[K]$ is not an independent set, K is in conflict. Now consider the other direction. Suppose K is in conflict. Then in some column of K there is a 0 and a 1. Let u be the row where the 0 is seen, and v be the row where the 1 is seen. So both u and v are in $V[K]$. Further, we know that there is an out-edge (u, v) in $\overrightarrow{E}$, and thus an edge between u and v in E, proving that $V[K]$ is not an independent set. This completes the proof of the equivalence relationship.

It follows that:

$$\text{CUBIC-MAX-INDEPENDENT-SET}(G) = \frac{1}{3}\text{LHR}(M). \tag{2.21}$$

The conditions of the L-reduction definition are now easily satisfied, because of the 1-1 correspondence between haplotypes induced (after row-removals) and independent sets in G, and the fact that a size-r independent set of G corresponds to a length-$3r$ haplotype (or, equivalently, to r mutually non-conflicting rows of M.) The L-reduction is formally satisfied by taking $\alpha = 3$ and $\beta = \frac{1}{3}$. The two functions that comprise the L-reduction are both polynomial time computable. □

Lemma 2.3. 1-Gap-SH-LHR *is APX-hard.*

Proof. This proof is almost identical to the proof of Lemma 2.2; the difference is the manner in which columns of M are assigned to vertices of G. The informal motivation is follows. In the previous allocation of columns to vertices, it was possible for a row corresponding to an out-out-in vertex to have 2 gaps. Suppose, for each out-out-in vertex, we could ensure that one of the 0s in its row was adjacent to the 1 in the row, with no holes in between. Then every row of the matrix would have (at most) 1 gap, and we would be finished. We now show that, by exploiting a rather subtle property of cubic graphs, it is indeed possible to allocate columns to vertices such that this is possible.

Assume, that we have ordered the edges of G as before to obtain $\overrightarrow{G}$. Let $V_{out} \subseteq V$ be those vertices in V that are out-out-in. Now, suppose we could compute (in polynomial time) an injective function $favourite : V_{out} \rightarrow V$ with the following properties:

- for every $v \in V_{out}$, $(v, favourite(v)) \in \overrightarrow{E}$;
- the subgraph of $\overrightarrow{G}$ induced by edges of the form $(v, favourite(v))$, henceforth called the *favourite-induced subgraph*, is acyclic.

Given such a function it is easy to create a total enumeration of the vertices of V such that every out-out-in vertex is immediately followed by its *favourite* vertex. This enumeration can then be used to allocate the columns of M to the vertices of V, such that every row of M has at most one gap. To ensure this property, it is necessary to stipulate that, where $favourite(v)$ is an in-in-out vertex, the 0 encoding the edge $(v, favourite(v))$ is placed in the *left* of the two columns encoding $favourite(v)$. This is not a problem because every vertex is the favourite of at most one other vertex.

It remains to prove that the function *favourite* exists and that it can be constructed in polynomial time. This is equivalent to finding vertex disjoint directed paths in $\overrightarrow{G}$ such that every out-out-in vertex is on such a path and all paths end in an in-in-out vertex. Lemma 2.4 tells us how to find such paths. We thank Bert Gerards for invaluable help with this.

This completes the proof that 1-Gap-SH-LHR is APX-hard. (See Figures 2.3 and 2.4 for an example of the whole reduction in action.) $\square$

Lemma 2.4. *Let $\overrightarrow{G}$ be a directed, cubic graph with a partition (V_{out}, V_{in}) of the vertices such that the vertices in V_{out} are out-out-in and the vertices in V_{in} are in-in-out. Then V_{out} can be covered, in polynomial time, by vertex-disjoint directed paths ending in V_{in}.*

Proof. Observe that any two directed circuits contained entirely within V_{out} are pairwise vertex disjoint. Let V'_{out} be obtained from V_{out} by shrinking each directed circuit in V_{out} to a single vertex, and let $\overrightarrow{G'}$ be the resulting new graph. (Note that each vertex in V'_{out} has outdegree at least 2 and indegree at most 1 and that the indegree of each node in V_{in} is still 2, because we do not delete multiple edges.) We now argue that it is possible to find a set of edges F' in $\overrightarrow{G'}$, with $|F'| = |V'_{out}|$, such that, for each $v \in V'_{out}$, precisely one edge from F' begins at v, and such that no two edges in F' have the same endpoint. We prove this by construction. For each vertex $u \in V'_{out}$ that has a child v in V'_{out}, we can add the edge (u, v) to F', because v has indegree 1 and therefore no other edges can end at v. (In case u has two such children, we can choose one of the edges to add to F'). Thus we are left to deal with a subset of vertices $L \subseteq V'_{out}$ where every vertex in L has all its children in V_{in}. Now consider the bipartite graph B with bipartition (L, V_{in}) and an edge for every directed edge of $\overrightarrow{G'}$ going from L to V_{in}. If we can find a matching in B of size $|L|$, we can complete the construction of F' by adding the edges from the perfect matching. Hall's Theorem states that a bipartite graph with bipartition (X, Y) has a matching of size $|X|$ if and only if, for all $X' \subseteq X$, $|N(X')| \geq |X'|$, where $N(X')$ is the set of all neighbours of X'. Now, note that each vertex in L sends at least two edges across the partition of B, and each vertex in V_{in} can accept at most two such edges, so for each $L' \subseteq L$ it is clear that $|N(L')| \geq |L'|$. Hence, the graph (L, V_{in}) does indeed have a matching of size $|L|$ and the construction of F' can be completed.

Now, given that the graph induced by V'_{out} is acyclic, so is F'. Let F be the set of edges in $\overrightarrow{G}$ corresponding to those in F'. F is acyclic and each directed circuit C in V_{out} has exactly one vertex v_C that is a tail of an edge of F and no vertex that is a head of an edge in F. Let P_C be the longest directed path in C that ends in v_C. Then the union of F and all P_C over all directed circuits C in V_{out} is a collection of paths ending in V_{in} and covering V_{out}.

Finding cycles in a graph and finding a maximum matching in a bipartite graph are both polynomial-time computable, so the whole process described above is polynomial-time computable. $\square$

Theorem 2.5. 1-GAP-LHR *is APX-hard.*

Proof. Follows from Lemma 2.3 and Lemma 2.1. □

2.4 Conclusion and Open Problems

This chapter has studied the complexity (under various different input restrictions) of the haplotyping problems Minimum Error Correction (MEC) and Longest Haplotype Reconstruction (LHR). Two strongly related problems are Minimum Fragment Removal (MFR) and Minimum SNP Removal (MSR). MSR (MFR) is the problem of removing the minimum number of columns (rows) from an SNP matrix in order to make it feasible. The state of knowledge about MEC, LHR, MFR and MSR after this work is demonstrated in Table 2.1.

MEC	Binary	Open (Section 2.2.3) PTAS known [Jia04]
	Gapless	NP-hard (Section 2.2.1)
	1-Gap	APX-hard (Section 2.2.2),
LHR	Gapless	P (Section 2.3.1)
	1-Gap	APX-hard (Section 2.3.2)
MFR	Gapless	P [Baf05]
	1-Gap	APX-hard [Baf05]
MSR	Gapless	P [Lan01]
	1-Gap	APX-hard [Baf05]

Table 2.1: The new state of knowledge following this work

Indeed, from a complexity perspective, the most intriguing open problem is to ascertain the complexity of the "re-opened" problem BINARY-MEC. It would also be interesting to study the approximability of GAPLESS-MEC.

From a more practical perspective, the next logical step is to study the complexity of these problems under more restricted classes of input, ideally under classes of input that have direct biological relevance. It would also be of interest to study some of these problems in a "weighted" context i.e. where the cost of the operation in question (row removal, column removal, error correction) is some function of (for example) an *a priori* specified confidence in the correctness of the data being changed.

Chapter 3

Population Haplotyping

3.1 Introduction

This chapter studies the computational problem of inferring biologically meaningful haplotype data from the genotype data of a population (see Section 1.4.2). A popular underlying abstraction for this model (in the context of diploid organisms) represents a genotype as a string over the $\{0, 1, 2\}$ alphabet, and a haplotype as a string over $\{0, 1\}$. The exact goal depends on the biological model being applied but a common, minimal algorithmic requirement is that, given a set of genotypes, a set of haplotypes must be produced which resolves the genotypes.

To be precise, we are given an $n \times m$ *genotype matrix* G with elements in $\{0, 1, 2\}$, the n rows of which correspond to genotypes, while its m columns correspond to sites on the genome, called SNPs. Without loss of generality we assume that any two rows of the input genotype matrix G are distinct. The goal is to construct an $n' \times m$ *haplotype matrix* H with elements in $\{0, 1\}$; the m columns of H also correspond to SNPs, but its n' rows correspond to haplotypes. Two rows h_1 and h_2 of H are said to *resolve* a row g of G if $g(j) = h_1(j)$ for all j with $h_1(j) = h_2(j)$ and $g(j) = 2$ otherwise. If this is the case, then we write $g = h_1 + h_2$, and we call h_1 the *complement* of h_2 with respect to g, and vice versa.

A haplotype matrix H *resolves* a genotype matrix G if for each row g of G, containing at least one 2, there are two rows h_1 and h_2 of H resolving g and each row g of G without 2s is also a row of H.

The first concrete mathematical problem studied in this chapter is Parsimony Haplotyping (PH).

Parsimony Haplotyping (PH)

Input: A genotype matrix G.

Output: A haplotype matrix H with a minimum number of rows that resolves G.

There is a rich literature in this area, of which recent papers such as [Bro06] give a good overview. The problem is APX-hard [Lan04; Sha06] and, in terms of approximation algorithms with performance *guarantees*, existing methods remain rather unsatisfactory, as will be shortly explained. This has led many authors to consider methods based on Integer Linear Programming (ILP) [Bro06; Gus03; Hal03; Lan04].

A different response to the hardness is to search for "islands of tractability" amongst special, restricted cases of the problem, exploring the frontier between hardness and polynomial-time solvability. In the literature available in this direction [Lan04; Lan06; Sha06], this investigation has specified classes of (k, ℓ)-*bounded instances*, in which the input genotype matrix G has at most k 2s per row and at most ℓ 2s per column (cf. [Sha06]). If k or ℓ is a "$*$" we mean instances that are bounded only by the number of 2s per column or per row, respectively. In this chapter we supplement this "tractability" literature with mainly positive results, and in doing so almost complete the bounded instance complexity landscape.

Next to the PH problem we study the *Minimum Perfect Phylogeny Haplotyping* (MPPH) model. Again a minimum-size set of resolving haplotypes is required but this time under the additional, biologically-motivated restriction that the produced haplotypes permit a *perfect phylogeny*, i.e., they can be placed at the leaves of an evolutionary tree within which each site mutates at most once. Haplotype matrices admitting a perfect phylogeny are completely characterised [Gus91; Gus97] by the absence of the forbidden submatrix:

$$F = \begin{bmatrix} 1 & 1 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Minimum Perfect Phylogeny Haplotyping (MPPH)

Input: A genotype matrix G.

Output: A haplotype matrix H with a minimum number of rows that resolves G and admits a perfect phylogeny.

The feasibility question (PPH) - given a genotype matrix G, find any haplotype matrix H that resolves G and admits a perfect phylogeny, or state that no such H exists - is solvable in linear time [Din06; Vij06]. Researchers in this area are now moving on to explore the PPH question on phylogenetic *networks* [Son05b] (cf. Chapter 4).

The MPPH problem, however, has so far hardly been studied beyond an NP-hardness result [Baf04] and occasional comments within PH and PPH literature [Bon03; Vij06; Zha06]. In this chapter we thus provide what is one of the first attempts to analyse the parsimony optimisation criteria within a well-defined and widely applicable biological framework. We seek namely to map the MPPH complexity landscape in the same way as the PH complexity landscape: using the concept of (k, ℓ)-boundedness. We write PH(k, ℓ) and MPPH(k, ℓ) for these problems restricted to (k, ℓ)-bounded instances.

Previous work and new results

Lancia et al. showed that PH$(3, *)$ is APX-hard [Lan04]. Recently, it was shown by Sharan et al. [Sha06] (amongst other results) that PH$(4, 3)$ is APX-hard. In this publication it was also proven that the restricted subcase of PH$(*, 2)$ is polynomial-time solvable where the *compatibility graph* of the input genotype matrix is a clique. (Informally, the compatibility graph shows for every pair of genotypes whether those two genotypes can use common haplotypes in their resolution.)

In this chapter, we bring the boundaries between hard and easy classes closer by showing that PH$(3, 3)$ is APX-hard and that PH$(*, 1)$ and PH$(2, *)$ are polynomial-time solvable.

As far as MPPH is concerned, there have been, prior to this work, no concrete results beyond the above mentioned NP-hardness result. We show that MPPH$(3, 3)$ is APX-hard and that, like their PH counterparts, MPPH$(2, *)$ and MPPH$(*, 1)$ are polynomial-time solvable (in both cases using a reduction to the PH counterpart). We also show that the clique result from [Sha06] holds in the case of MPPH$(*, 2)$ as well. As for its PH counterpart, the complexity of MPPH$(*, 2)$ remains open, see Figure 1.2 on page 11.

The fact that both PH and MPPH already become APX-hard for $(3, 3)$-bounded instances means that, in terms of deterministic approximation algorithms, the best that we can in general hope for is constant approximation ratios. Lancia et al. [Lan04; Lan06] have given two separate approximation algorithms for PH with approximation ratios of $\sqrt{n}$ and 2^{k-1} respectively, where n is the number of genotypes in the input, and k is the maximum number of 2s appearing in a row of the genotype matrix[1]. An $O(\log n)$ approximation algorithm has also been given [Hua05] but this only runs in polynomial time if the set of all possible haplotypes that can participate in feasible solutions, can be enumerated in polynomial time. The obvious problem with the 2^{k-1} and the $O(\log n)$ approximation algorithms is thus that either the accuracy decays exponentially (as in the former case) or the running time increases exponentially (as in the latter case) with an increasing number of 2s per row. Here we offer a simple, alternative approach which achieves (in polynomial time) approximation ratios linear in ℓ for PH$(*, \ell)$ and MPPH$(*, \ell)$ instances, and

[1]It would be overly restrictive to write PH$(k, *)$ here because their algorithm runs in polynomial time even if k is not a constant.

actually also achieves these ratios in polynomial time when ℓ is not constant. These ratios are shown in the Table 3.1; note how improved ratios can be obtained if every genotype is guaranteed to have at least one 2.

	Approximation ratio
$PH(*, \ell)$	$\frac{3}{2}\ell + \frac{1}{2}$
$PH(*, \ell)$ at least one 2 per genotype	$\frac{3}{4}\ell + \frac{7}{4} - \frac{3}{2}\frac{1}{\ell+1}$
$MPPH(*, \ell)$	2ℓ
$MPPH(*, \ell)$ at least one 2 per genotype	$\ell + 2 - \frac{2}{\ell+1}$

Table 3.1: Approximation ratios achieved in this thesis (for $\ell \geq 2$).

We have thus decoupled the approximation ratio from the maximum number of 2s per row, and instead made the ratio conditional on the maximum number of 2s per column. Our approximation scheme is hence an improvement to the 2^{k-1}-approximation algorithm except in cases where the maximum number of 2s per row is exponentially small compared to the maximum number of 2s per column. Our approximation scheme yields also the first approximation results for MPPH.

As explained by Sharan et al. in their "islands of tractability" paper [Sha06], identifying tractable special classes can be practically useful for constructing high-speed subroutines within ILP solvers, but perhaps the most significant aspect of this chapter is the analysis underpinning the results, which - by deepening our understanding of how this problem behaves - assists the search for better, faster approximation algorithms and for determining the exact shorelines of the islands of tractability.

Furthermore, the fact that - prior to this work - concrete and positive results for MPPH had not been obtained (except for rather pessimistic modifications to ILP models [Bro06]), means that the algorithms given here for the MPPH cases, and the data structures used in their analysis (e.g. the *restricted compatibility graph* in Section 3.3.2), assume particular importance.

Finally, this chapter yields some interesting open problems, of which the outstanding $(*, 2)$ case (for both PH and MPPH) is only one. Prominent amongst these questions (which are discussed in Section 3.5) is the question of whether MPPH and PH instances are inter-reducible, at least within the bounded-instance framework.

3.2 Complexity of Population Haplotyping Problems

Theorem 3.1. *MPPH*$(3,3)$ *is APX-hard.*

Proof. The proof in [Baf04] that MPPH is NP-hard uses a reduction from Vertex Cover, which can be modified to yield NP-hardness and APX-hardness for (3,3)-bounded instances. Given a graph $T = (V, E)$ the reduction in [Baf04] constructs a genotype matrix $G(T)$ of MPPH with $|V|+|E|$ rows and $2|V|+|E|$ columns. For every vertex $v_i \in V$ there is a genotype (row) g_i in $G(T)$ with $g_i(i) = 1$, $g_i(i + |V|) = 1$ and $g_i(j) = 0$ for every other position j. In addition, for every edge $e_k = \{v_h, v_l\}$ there is a genotype g_k with $g_k(h) = 2$, $g_k(l) = 2$, $g_k(2|V| + k) = 2$ and $g_k(j) = 0$ for every other position j. Bafna et al. [Baf04] prove that an optimal solution for MPPH with input $G(T)$ contains $|V| + |E| + VC(T)$ haplotypes, where $VC(T)$ is the size of the smallest vertex cover in T.

3-Vertex Cover is the vertex cover problem when every vertex in the input graph has at most degree 3. It is known to be APX-hard [Pap91; Ali97]. Let T be an instance of 3-Vertex Cover. We assume that T is connected. Observe that for such a T the reduction described above yields an MPPH instance $G(T)$ that is $(3, 3)$-bounded. It is left to show that the described reduction is an L-reduction (see Definition 2.1 on page 21) when reducing from 3-Vertex Cover.

Connectedness of T implies that $|V| - 1 \leq |E|$. In 3-Vertex Cover, a single vertex can cover at most three edges in T, implying that $VC(T) \geq \frac{1}{3}|E|$. The optimal value for $MPPH$ on instance $G(T)$ is (if $|E| \geq 1$):

$$|V| + |E| + VC(T) \leq 2|E| + 1 + VC(T) \leq 7|VC(T)| + 1 \leq 8|VC(T)|.$$

Therefore, the first property of an L-reduction holds with $\alpha = 8$. Now consider any feasible solution of $MPPH$ containing t haplotypes. This can easily be converted into a solution of 3-Vertex Cover containing $d = t-|V|-|E|$ vertices, as shown in [Baf04]. It follows that the second property of an L-reduction holds with $\beta = 1$, since $||VC(T)| - d|$ is equal to $||V| + |E| + VC(T) - t|$. □

Theorem 3.2. *PH*$(3,3)$ *is APX-hard.*

Proof. The proof by Sharan et al. [Sha06] that PH$(4, 3)$ is APX-hard can be modified slightly to obtain APX-hardness of PH$(3, 3)$. The reduction is from 3-Dimensional Matching with each element occurring in at most three triples (3DM3): given disjoint sets X, Y and Z containing ν elements each and a set $C = \{c_0, \dots, c_{\mu-1}\}$ of μ triples in $X \times Y \times Z$ such that each element occurs in at most three triples in C, find a maximum cardinality set $C' \subseteq C$ of disjoint triples.

From an instance of 3DM3 we build a genotype matrix G with $3\nu+3\mu$ rows and $6\nu+4\mu$ columns. The first 3ν rows are called *element-genotypes* and the last 3μ rows are called *matching-genotypes.* We specify only the non-zero entries of the genotypes[2]. Let $X = \{x_0, \ldots, x_{\nu-1}\}$, $Y = \{y_0, \ldots, y_{\nu-1}\}$, $Z = \{z_0, \ldots, z_{\nu-1}\}$. For every element $x_i \in X$ define element-genotype g_i^x with $g_i^x(3\nu + i) = 1$; $g_i^x(6\nu + 4k) = 2$ for all k with $x_i \in c_k$. If x_i occurs in at most two triples we set $g_i^x(i) = 2$. For every element $y_i \in Y$ there is an element-genotype g_i^y with $g_i^y(4\nu + i) = 1$; $g_i^y(6\nu + 4k) = 2$ for all k with $y_i \in c_k$ and if y_i occurs in at most two triples then we set $g_i^y(\nu + i) = 2$. For every element $z_i \in Z$ there is an element-genotype g_i^z with $g_i^z(5\nu + i) = 1$; $g_i^z(6\nu + 4k) = 2$ for all k with $z_i \in c_k$ and if z_i occurs in at most two triples then we set $g_i^z(2\nu + i) = 2$. For each triple $c_k = \{x_{i_1}, y_{i_2}, z_{i_3}\} \in C$ there are three matching-genotypes c_k^x, c_k^y and c_k^z: c_k^x has $c_k^x(3\nu+i_1) = 2$, $c_k^x(6\nu+4k) = 1$ and $c_k^x(6\nu+4k+1) = 2$; c_k^y has $c_k^y(4\nu+i_2) = 2$, $c_k^y(6\nu+4k) = 1$ and $c_k^y(6\nu+4k+2) = 2$; c_k^z has $c_k^z(5\nu+i_3) = 2$, $c_k^z(6\nu + 4k) = 1$ and $c_k^z(6\nu + 4k + 3) = 2$.

Notice that the element-genotypes only have a 2 in the first 3ν columns if the element occurs in at most two triples. This is the only difference with the reduction from [Sha06], where every element-genotype has a 2 in the first 3ν columns: i.e., for elements $x_i \in X$, $y_i \in Y$ or $z_i \in Z$ a 2 in column i, $\nu + i$ or $2\nu + i$, respectively. As a direct consequence our genotype matrix has only three 2s per row in contrast to the four 2s per row in the original reduction.

We claim that for this (3,3)-bounded instance exactly the same arguments can be used as for the (4,3)-bounded instance. In the original reduction the left-most 2s ensured that, for each element-genotype, at most one of the two resolving haplotypes was used in the resolution of other genotypes. Clearly this remains true in our modified reduction for elements appearing in two or fewer triples, because the corresponding left-most 2s have been retained. So consider an element x_i appearing in three triples and suppose, by way of contradiction, that *both* haplotypes used to resolve g_i^x are used in the resolution of other genotypes. Now, the 1 in position $3\nu + i$ prevents this element-genotype from sharing haplotypes with other element-genotypes, so genotype g_i^x must share both its haplotypes with matching-genotypes. Note that, because $g_i^x(3\nu+i) = 1$, the genotype g_i^x can only possibly share haplotypes with matching-genotypes corresponding to triples that contain x_i. Indeed, if x_i is in triples c_{k_1}, c_{k_2} and c_{k_3} then the only genotypes with which g_i^x can potentially share haplotypes are $c_{k_1}^x$, $c_{k_2}^x$ and $c_{k_3}^x$. Genotype g_i^x cannot share both its haplotypes with the same matching-genotype (e.g. $c_{k_1}^x$) because both haplotypes of g_i^x will have a 1 in column $3\nu + i$ whilst only one of the two haplotypes for $c_{k_1}^x$ will have a 1 in that column. So, without loss of generality, g_i^x is resolved by a haplotype that is shared with $c_{k_1}^x$ and a haplotype that is shared with $c_{k_2}^x$. However, this is not possible, because g_i^x has a 2 in the column corresponding to c_{k_3} (i.e. $g_i^x(6\nu + 4k_3) = 2$), whilst both $c_{k_1}^x$ and $c_{k_2}^x$ have a 0 in that column

[2]Only in this proof we index haplotypes, genotypes and matrices starting with 0, which makes notation consistent with [Sha06].

$(c^x_{k_1}(6\nu + 4k_3) = c^x_{k_2}(6\nu + 4k_3) = 0)$, yielding a contradiction.

Note that, in the original reduction, it was not only true that each element-genotype shared at most one of its haplotypes, but - more strongly - it was also true that such a shared haplotype was used by exactly one other genotype (i.e. the genotype corresponding to the triple the element gets assigned to). To see that this property is also retained in the modified reduction observe that if (say) g^x_i shares one haplotype with two genotypes $c^x_{k_1}$ and $c^x_{k_2}$ then x_i must be in both triples c_{k_1} and c_{k_2}, but this is not possible because, in the two columns corresponding to triples c_{k_1} and c_{k_2}, $c^x_{k_1}$ has 1 and 0 whilst $c^x_{k_2}$ has 0 and 1 (i.e. $c^x_{k_1}(6\nu + 4k_1) = 1$, $c^x_{k_1}(6\nu + 4k_2) = 0$ whilst $c^x_{k_2}(6\nu + 4k_1) = 0$, $c^x_{k_2}(6\nu + 4k_2) = 1$). □

3.3 Polynomial-time Algorithms

3.3.1 Parsimony Haplotyping

This section shows the polynomial-time solvability of PH on $(*, 1)$- and $(2, *)$-bounded instances. We start with PH$(*, 1)$.

We say that two genotypes g_1 and g_2 are *compatible*, denoted as $g_1 \sim g_2$, if $g_1(j) = g_2(j)$ or $g_1(j) = 2$ or $g_2(j) = 2$ for all j. A genotype g and a haplotype h are *consistent* if h can be used to resolve g, ie. if $g(j) = h(j)$ or $g(j) = 2$ for all j. The following observation follows directly from these definitions.

Observation 3.1. *If g_1 and g_2 are distinct compatible rows of a genotype matrix with at most one 2 per column, then there exists exactly one haplotype that is consistent with both g_1 and g_2.*

The *compatibility graph* associated with a given genotype matrix is the graph with vertices for the genotypes and an edge between two genotypes if they are compatible. Each edge $\{g_1, g_2\}$ of the compatibility graph of a PH$(*, 1)$-instance is labelled by the unique haplotype that is consistent with both g_1 and g_2, namely h with $h(j) = g_1(j)$ for all j with $g_1(j) \neq 2$ and $h(j) = g_2(j)$ for all j with $g_2(j) \neq 2$.

We use the notation $g_1 \sim_h g_2$ if g_1 and g_2 are compatible and h is consistent with both (i.e. h labels edge $\{g_1, g_2\}$). We prove that the compatibility graph of a PH$(*, 1)$-instance has a specific structure. A *1-sum* of two graphs is the graph obtained by identifying a vertex of one graph with a vertex of the other graph. A *1-sum* of $n+1$ graphs is inductively obtained by identifying a vertex of a graph with a vertex of a 1-sum of n graphs. See Figure 3.1 for an example of a 1-sum of three cliques (K_3, K_4 and K_2).

It follows directly that a graph C is a 1-sum of cliques if and only if for each cycle $v_1, \ldots, v_k$ in C all edges $\{v_i, v_j\}$ (with $1 \leq i, j \leq k$, $i \neq j$) are in C. If $1 < |i - j| < k - 1$ then such an arc $\{v_i, v_j\}$ is called a *chord* of the cycle. A graph is called *chordal* if each cycle of at least four vertices has at least one chord. A 1-sum of cliques is thus always chordal. However, the reverse is not always the case.

Lemma 3.1. *If G is a genotype matrix with at most one 2 per column, then every connected component of the compatibility graph of G is a 1-sum of cliques, where edges in the same clique are labelled with the same haplotype.*

Proof. Let C be the compatibility graph of G and let $g_1, g_2, \dots, g_k$ be a cycle in C. It suffices to show that there exists a haplotype h_c such that $g_i \sim_{h_c} g_{i'}$ for all $i, i' \in \{1, ..., k\}$. Consider an arbitrary column j. If there is no genotype with a 2 in this column then $g_1 \sim g_2 \sim \dots \sim g_k$ implies that $g_1(j) = g_2(j) = \dots = g_k(j)$. Otherwise, let g_{i_j} be the unique genotype with a 2 in column j. Then $g_1 \sim g_2 \sim \dots \sim g_{i_j - 1}$ together with $g_1 \sim g_k \sim g_{k-1} \sim \dots \sim g_{i_j + 1}$ implies that $g_i(j) = g_{i'}(j)$ for all $i, i' \in \{1, ..., k\} \setminus \{i_j\}$. Set $h_c(j) = g_i(j)$, $i \neq i_j$. Repeating this for each column j produces a haplotype h_c such that indeed $g_i \sim_{h_c} g_{i'}$ for all $i, i' \in \{1, ..., k\}$. □

$$\begin{array}{c} g_1 \\ g_2 \\ g_3 \\ g_4 \\ g_5 \\ g_6 \\ g_7 \end{array} \quad \begin{bmatrix} 0 & 0 & 1 & 0 & 2 & 0 & 1 \\ 2 & 0 & 2 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 2 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 2 \\ 0 & 0 & 1 & 1 & 0 & 2 & 1 \\ 1 & 2 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

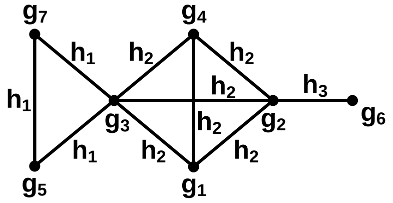


Figure 3.1: Example of of a genotype matrix with corresponding compatibility graph, with $h_1 = (0,0,1,1,0,0,1)$, $h_2 = (0,0,1,0,0,0,1)$ and $h_3 = (1,0,0,0,0,0,1)$.

From this lemma, it follows directly that in PH$(*, 1)$ the compatibility graph is chordal. Every chordal graph has a *simplicial* vertex, a vertex whose (closed) neighbourhood is a clique. Deleting a vertex in a chordal graph gives again a chordal graph (see for example [Bla93] for an introduction to chordal graphs). The following lemma leads almost immediately to polynomial solvability of PH$(*, 1)$. In this lemma, we use H' to denote the genotypes in the input that do not contain 2s and can thus be seen as haplotypes that have to be contained in the output matrix H. We use set-operations for the rows of matrices: thus, e.g., $h \in H$ means h is a row of matrix H, $H \cup h$ says h is added to H as a row, and $H' \subset H$ says H' is a submatrix consisting of rows of H.

Lemma 3.2. *Given a haplotype matrix H' and a genotype matrix G with at most one 2 per column it is possible to find, in polynomial time, a haplotype matrix H that resolves G, has H' as a submatrix and has a minimum number of rows.*

Proof. The proof is constructive. Let problem (G, H') denote the above problem on input matrices G and H'. Let C be the compatibility graph of G, which, as implied by Lemma 3.1, is chordal. Suppose g corresponds to a simplicial vertex of C. If g is not an isolated vertex then there exists a unique haplotype consistent with any genotype in the closed neighbourhood clique of g. Denote this haplotype by h_c. We extend matrix H' to H'' and update graph C as follows.

1. If g has no 2s it can be resolved by just the haplotype $h = g$. We set $H'' = H' \cup h$ and delete g from C.
2. Else, if there exist rows $h_1 \in H'$ and $h_2 \in H'$ such that $g = h_1 + h_2$ we set $H'' = H'$ and delete g from C.
3. Else, if the complement of h_c w.r.t. g is in H' we set $H'' = H' \cup h_c$ and delete g from C.
4. Else, if there exists $h_1 \in H'$ such that $g = h_1 + h_2$ for some $h_2 \notin H'$ we set $H'' = H' \cup h_2$ and delete g from C.
5. Else, if g is not an isolated vertex in C then we set $H'' = H' \cup \{h_1, h_c\}$, with h_1 the complement of h_c w.r.t. g, and delete g from C.
6. Otherwise, g is an isolated vertex in C and we set $H'' = H' \cup \{h_1, h_2\}$ for any h_1 and h_2 such that $g = h_1 + h_2$ and delete g from C.

The resulting graph is again chordal and we repeat the above procedure for $H' = H''$ until all vertices are removed from C. Let H be the final haplotype matrix H''. It is clear from the construction that H resolves G.

We prove that H has a minimum number of rows by induction on the number of genotypes. Clearly, if G has only one genotype the algorithm constructs an optimal solution. The induction hypothesis is that the algorithm finds an optimal solution to the problem (G, H') for any haplotype matrix H' if G has at most $n-1$ rows. Now consider haplotype matrix H' and genotype matrix G with n rows. The first step of the algorithm selects a simplicial vertex g and proceeds with one of the cases 1 to 6. The algorithm then finds (by the induction hypothesis) an optimal solution H to problem $(G \setminus \{g\}, H'')$. It remains to prove that H is also an optimal solution to problem (G, H'). We do this by showing that an optimal solution H^* to problem (G, H') can be modified to include H''. We prove this for every case of the algorithm separately.

1. In this case $h \in H^*$, since g can only be resolved by h.
2. In this case $H'' = H'$ and hence $H'' \subseteq H^*$.

3. Suppose that $h_c \notin H^*$. Because we are not in case 2 we know that there are two rows in H^* that resolve g and at least one of the two, say h^*, is not a row of H'. Since h_c is the unique haplotype consistent with (the simplicial) g and any compatible genotype, h^* can not be consistent with any other genotype than g. Thus, replacing h^* by h_c gives a solution with the same number of rows but containing H''.

4. Suppose that $h_2 \notin H^*$. Because we are not in case 2 or 3 we know that there is a haplotype $h^* \in H^*$ consistent with g, $h^* \notin H'$ and $h^* \neq h_c$. Hence, h^* is not consistent with any other genotypes than g and we can replace h^* by h_2 to obtain an optimal solution containing H''.

5. Suppose that $h_1 \notin H^*$ or $h_c \notin H^*$. Because we are not in case 2, 3 or 4, there are haplotypes $h^* \in H \backslash H'$ and $h^{**} \in H \backslash H'$ that resolve g. If h^* and h^{**} are both not equal to h_c then they are not consistent with any other genotype than g. Replacing h^* and h^{**} by h_1 and h_c leads to another optimal solution. If one of h^* and h^{**} is equal to h_c then we can replace the other one by h_1.

6. Suppose that $h_1 \notin H^*$ or $h_2 \notin H^*$. There are haplotypes $h^*, h^{**} \in H^* \backslash H'$ that resolve g and just g since g is an isolated vertex. Replacing h^* and h^{**} by h_1 and h_2 gives an optimal solution containing H''.

□

Theorem 3.3. *The problem PH$(*, 1)$ can be solved in polynomial time.*

Proof. The proof follows from Lemma 3.2. Construction of the compatibility graph, including labels h_c, takes $O(n^2m)$ time, for an n times m input matrix. Finding a *simplicial ordering* (i.e. an ordering of the vertices such that each vertex is simplicial in the subgraph obtained by deleting the previous vertices in the ordering) can be done in time $O(n^2)$ [Ros76] and resolving each vertex takes $O(n^2m)$ time. The overall running time of the algorithm is therefore $O(n^3m)$. □

Now we will show polynomial-time solvability of PH$(2, *)$. This has also been discovered independently by Lancia et al. [Lan06]. Given a graph G, let PH(G) denote the minimum number of haplotypes needed to resolve G.

Theorem 3.4. *The problem PH$(2, *)$ can be solved in polynomial time.*

Proof. We let $n = |G|$ denote the number of genotypes in G and let m denote the length of each genotype in G. An *independent set* of a graph is a collection of mutually non-adjacent vertices. We will compute the solution of *PH* on input G, by reduction to the polynomial-time solvable problem MaxBIS, which is the problem of finding an independent set of maximum cardinality in a bipartite graph.

First, some notation. A 2 in a genotype is called an *ambiguous position* and a genotype is *i-ambiguous* if it contains i ambiguous positions. Each genotype in G is thus either 0-ambiguous, 1-ambiguous, or 2-ambiguous. For a 0-ambiguous genotype g, we define h_g as the haplotype equal to g. For a 1-ambiguous genotype g we let $h_{g:0}$ (respectively, $h_{g:1}$) be the haplotype obtained by replacing the ambiguous position in g with 0 (respectively, 1). For a 2-ambiguous genotype g we let $h_{g:i,j}$ - where $i, j \in \{0, 1\}$ - be the haplotype obtained by replacing the first (i.e. leftmost) ambiguous position in g with i, and the second ambiguous position with j. A haplotype is said to have *even (odd) parity* if it contains an even (odd) number of 1s.

Now, observe that there are two ways to resolve a 2-ambiguous genotype g: (1) with haplotypes $h_{g:0,0}$ and $h_{g:1,1}$ and (2) with $h_{g:0,1}$ and $h_{g:1,0}$. Note that - depending on h - one of the ways uses two even parity haplotypes, and the other uses two odd parity haplotypes.

We build a set H of haplotypes by stepping through the list of genotypes and, for each genotype, adding the 1, 2 or 4 corresponding haplotypes to the set H. (Note that, because H is a set, we discard duplicate haplotypes.) That is, for a 0-ambiguous genotype g add h_g, for a 1-ambiguous genotype g add $h_{g:0}$ and $h_{g:1}$, and for a 2-ambiguous genotype g add $h_{g:0,0}, h_{g:0,1}, h_{g:1,0}$ and $h_{g:1,1}$.

We are now ready to build a bipartite graph $B = (V, E)$ as follows, where V has bipartition $V^+ \cup V^-$. For each $h \in H$ we introduce a vertex, which we also refer to as h; all h with even parity are put into V^+ and all h with odd parity are put into V^-. For each 0-ambiguous genotype $g \in G$ we introduce a set $I_0(g)$ of four vertices and we connect each vertex in $I_0(g)$ to h_g. For each 1-ambiguous genotype $g \in G$ we introduce two sets of vertices $I_1(g, 0)$ and $I_1(g, 1)$, both containing two vertices. Each vertex in $I_1(g, 0)$ is connected to $h_{g:0}$ and each vertex in $I_1(g, 1)$ is connected to $h_{g:1}$. Finally, for each 2-ambiguous $g \in G$ we add (to V^+ and V^- respectively) two sets of vertices $I_2(g, +)$ and $I_2(g, -)$, each containing 4 vertices. We connect every vertex in $I_2(g, +)$ to every vertex in $I_2(g, -)$, connect every vertex in $I_2(g, +)$ to the two odd parity haplotypes resolving g, and connect every vertex in $I_2(g, -)$ to the two even parity haplotypes resolving g. This completes the construction of B.

Let M be an independent set of B with maximum cardinality. Observe that all the vertices of $I_0(g)$ must be in M, for all 0-ambiguous g. To see this, suppose there exists a 0-ambiguous g such that at least one of the vertices in $I_0(g)$ is not in M. This leads to a contradiction because then $M \setminus \{h_g\} \cup I_0(g)$ would be a larger independent set. By a similar argument we see that, for all 1-ambiguous $g \in G$, all of $I_1(g, 0)$ and $I_1(g, 1)$ must be in M. Now, consider $I_2(g, +)$ and $I_2(g, -)$, for a 2-ambiguous $g \in G$. We argue that either $I_2(g, +)$ is contained in M, or $I_2(g, -)$ is contained in M. Suppose, by way of argument, that there exists a g such that both $I_2(g, +)$ and $I_2(g, -)$ are completely outside M. If we are (without loss of generality) free to add all the vertices in $I_2(g,+)$ to the independent set we have an immediate contradiction. So $I_2(g,+)$ is prevented from being in M by the fact that one or two of the haplotypes to which it is connected are already in M. But we could then build a bigger independent set by removing those (at most) two haplotypes from M and adding the four vertices $I_2(g,+)$; contradiction.

We can think of the presence of an I-set in the independent set as denoting that the genotype it represents is resolved using the haplotypes to which it is attached. Hence, every haplotype that is used for at least one resolution will *not* be in the independent set. In addition, unused haplotypes *will* be in the independent set. Hence, a maximum independent set will try and minimise the number of haplotypes used to resolve the given genotypes. Thus:

$$\textsc{MaxBIS}(B) = 4n + (|H| - PH(G)). \tag{3.1}$$

We can thus use a polynomial-time algorithm for MaxBIS to compute PH(G). □

Running time

The above algorithm can be implemented in time $O(mn\log(n) + n^{\frac{3}{2}})$.

First we build the graph B. We can without too much trouble build a graph representation of B - that combines adjacency-matrix and adjacency-list features - in $O(mn\log(n))$ time. For each $g \in G$, add its corresponding I-set(s) and add the (at most) four haplotypes corresponding to g, without eliminating duplicates, and at all times efficiently maintaining adjacency information. Then sort, in $O(mn\log(n))$ time, the list of haplotypes and eliminate duplicate haplotypes (by merging their adjacency information into one single haplotype).

A maximum independent set in a bipartite graph can be constructed from a maximum matching. A maximum matching in B can be found in time $O(n^{\frac{3}{2}})$ by the algorithm in [Hop73] (this algorithm runs in time $O(\sqrt{|V|}|E|)$ and in our case $|V| = O(n)$ and $|E| = O(n)$). Once the maximum matching is found, one needs $O(|E|+|V|)$ time to find a maximum independent set [Gav77]. Thus finding a maximum independent set takes $O(n^{\frac{3}{2}})$ time overall.

3.3.2 Minimum Perfect Phylogeny Haplotyping

This section shows that also MPPH$(2,*)$ can be solved in polynomial time.

Minimum Perfect Phylogeny Haplotyping (MPPH)

Input: A genotype matrix G.

Output: A haplotype matrix H with a minimum number of rows that resolves G and admits a perfect phylogeny.

Recall that a haplotype matrices admits a perfect phylogeny if and only if it does not contain the forbidden submatrix:

$$F = \begin{bmatrix} 1 & 1 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

We start with a definition.

Definition 3.1. *For two columns of a genotype matrix a* reduced resolution *of these columns is the result of applying to the submatrix induced by these columns the following operations until none is applicable: (1) delete one of two identical rows and (2) apply one of the replacement rules*

$$\begin{bmatrix} 2 & a \end{bmatrix} \rightarrow \begin{bmatrix} 1 & a \\ 0 & a \end{bmatrix}, \begin{bmatrix} 2 & 2 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \text{ and } \begin{bmatrix} 2 & 2 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

for $a \in \{0, 1\}$.

Two reduced resolutions are regarded as identical if one of them can be obtained from the other by permuting its rows. Two columns can have different reduced resolutions if there is a genotype with a 2 in both these columns. The reduced resolutions of a column pair of a genotype matrix G are submatrices of (or equal to) the forbidden matrix F. They represent all possibilities for the resolution of G w.r.t. that column pair. More precisely, if H resolves G and $H[i,j]$ denotes the submatrix of H containing its ith and jth column, then collapsing identical rows in $H[i,j]$ gives a reduced resolution of the ith and jth column of G.

Theorem 3.5. *The problem MPPH*$(2,*)$ *can be solved in polynomial time.*

Proof. We reduce MPPH$(2,*)$ to PH(2,*), which can be solved in polynomial time by the previous section (and by [Lan06]). Let G be an instance of MPPH$(2,*)$. We may assume that all rows of G are distinct.

Take the submatrix of any two columns of G. If it does not contain a [2 2] row, then in terms of Definition 3.1 there is a unique reduced resolution. If G contains two or more [2 2] rows then, since by assumption all genotypes are distinct, G has $\begin{bmatrix} 2 & 2 & 0 \\ 2 & 2 & 1 \end{bmatrix}$ and therefore $\begin{bmatrix} 2 & 0 \\ 2 & 1 \end{bmatrix}$ as a submatrix, which can only be resolved by a haplotype matrix containing the forbidden submatrix F. It follows that in this case the instance is infeasible. If it contains exactly one [2 2] row, then there are clearly exactly two reduced resolutions. Thus we may assume that for each column pair there are at most two reduced solutions.

Observe that if for some column pair all reduced resolutions are equal to F the instance is again infeasible. On the other hand, if no column pair has F as a reduced resolution then MPPH$(2,*)$ is equivalent to PH$(2,*)$ because any minimal haplotype matrix H that resolves G admits a perfect phylogeny. Finally, consider a column pair with two reduced resolutions, one of them containing F. Because there are two reduced resolutions there is a genotype g with a 2 in both columns (and in no other columns). Let h_1 and h_2 be the haplotypes that correspond to the resolution of g that does not lead to F in these two columns. Then we replace g in G by h_1 and h_2, ensuring that a minimal haplotype matrix H resolving G does not have F as a submatrix in these two columns.

Repeating this procedure for every column pair either tells us that the instance G is infeasible or creates a genotype matrix G' such that any haplotype matrix H resolves G' if and only if H resolves G and H admits a perfect phylogeny (assuming that H does not contains rows that are not used in the resolution of G').

The reduction takes $O(nm^2)$ time since all reduced resolutions have to be computed for each column-pair. Using the algorithm from [Lan06] as a subroutine to solve the resulting $PH(2,*)$ problem instance, we get an overall running time of $O(m^2n + n^{\frac{3}{2}})$, for an $n \times m$ input matrix. □

Theorem 3.6. *The problem MPPH$(*,1)$ can be solved in polynomial time.*

Proof. Similar to the proof of Theorem 3.5 we reduce MPPH$(*,1)$ to PH$(*,1)$. As there, consider for any pair of columns of the input genotype matrix G its reduced resolutions, according to Definition 3.1. Since G has at most one 2 per column there is at most one genotype with 2s in both columns. Hence there are at most two reduced resolutions. If all reduced resolutions are equal to the forbidden submatrix F the instance is infeasible. If on the other hand no column pair has F as a reduced resolution then in fact MPPH$(*,1)$ is equivalent to PH$(*,1)$, because any haplotype matrix H resolving G admits a perfect phylogeny (again assuming that H does not contain rows that are not used in the resolution of G).

As in the proof of Theorem 3.5 we are left with considering column pairs for which one of the two reduced resolutions is equal to F. For such a column pair there must be a genotype g that has 2s in both these columns. The other genotypes have only 0s and 1s in them. Suppose we get a forbidden submatrix F in these columns of the solution if g is resolved by haplotypes h_1 and h_2, where h_1 has a and b and therefore h_2 has $1-a$ and $1-b$ in these columns, $a, b \in \{0,1\}$. We will change the input matrix G such that if g gets resolved by such a *forbidden resolution* the haplotypes involved in that resolution are not consistent with any other genotypes. We do this by adding an extra column to G as follows. The genotype g gets a 1 in this new column. Every genotype with a and b or with $1-a$ and $1-b$ in the considered columns gets a 0 in the new column. Every other genotype gets a 1 in the new column. For example, the matrix

$$\begin{bmatrix} 2 & 2 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \text{ gets one extra column and becomes } \begin{bmatrix} 2 & 2 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}.$$

Denote by G_{mod} the result of modifying G by adding such a column for every pair of columns with exactly one 'bad' and one 'good' reduced resolution. It is not hard to see that any optimal solution to PH$(*, 1)$ on G_{mod} can be transformed into a solution to MPPH$(*, 1)$ on G of the same cardinality. Indeed, any two haplotypes used in a forbidden resolution of a genotype g are not consistent with any other genotype of G_{mod}, and hence may be replaced by two other haplotypes resolving g in a non-forbidden way, not increasing the total number of haplotypes. Now, let H be an optimal solution to MPPH$(*, 1)$ on G. We can modify H to obtain a solution to PH$(*, 1)$ on G_{mod} of the same cardinality as follows. We modify every haplotype in H in the same way as the genotypes it resolves (if a genotype can be resolved in several ways then we choose one resolution arbitrarily). From the construction of G_{mod} it follows that two compatible genotypes are only modified distinctly if the haplotype they are both consistent with is in a forbidden resolution. However, in H no genotypes are resolved with a forbidden resolution since H is a solution to MPPH$(*, 1)$. We conclude that optimal solutions to PH$(*, 1)$ on G_{mod} correspond to optimal solutions to MPPH$(*, 1)$ on G and hence the latter problem can be solved in polynomial time, by Theorem 3.3.

The reduction takes $O(nm^2)$ time and increases the number of columns (in the worst case) quadratically. If we use the algorithm from the proof of Lemma 3.2 as a subroutine to solve the resulting $PH(*, 1)$ problem instance we thus get an overall running time of $O(n^3m^2)$, for an $n \times m$ input matrix. □

The questions remain whether PH$(*, 2)$ and MPPH$(*, 2)$ are polynomial-time solvable. Unfortunately, we have not found the answer to these complexity questions. However, the borders have been pushed slightly further. In [Sha06] PH$(*, 2)$ is shown to be polynomially solvable if the input genotypes have the complete graph as compatibility graph, we call this problem PH$(*, 2)$-C1. We will give the counterpart result for MPPH$(*, 2)$-C1.

Let G be an $n \times m$ MPPH$(*, 2)$-C1 input matrix. Since the compatibility graph is a clique, every column of G contains either only 1s and 2s (a *1-column*) or only 0s and 2s. If we replace in every 1-column of G the 1s by 0s and mark the SNP corresponding to this column 'flipped', then we obtain a problem on a $\{0, 2\}$-matrix G'. This problem is equivalent because of the following. If we take any solution H to MPPH$(*, 2)$-C1 on input G' and change all 0s in columns corresponding to flipped SNPs to 1s then we obtain a solution of

MPPH$(*, 2)$-C1 on input G, and vice versa. From now on we assume that the input matrix G contains only 0s and 2s.

If we assume moreover that $n \geq 3$, which we do from here on, the *trivial haplotype* h_t defined as the all-0 haplotype of length m is the only haplotype consistent with all genotypes in G.

We define the *restricted* compatibility graph $C_R(G)$ of G as follows. As in the normal compatibility graph, the vertices of $C_R(G)$ are the genotypes of G. However, there are fewer edges: an edge $\{g, g'\}$ is only in C_R(G) if $g \sim_h g'$ for some $h \neq h_t$, or, equivalently, if there is a column where both g and g' have a 2. The difference with the normal compatibility graph is thus that there is no edge between two genotypes if they share only the trivial haplotype.

Lemma 3.3. *If G is a feasible instance of MPPH$(*, 2)$-C1 then every vertex in $C_R(G)$ has degree at most 2.*

Proof. Any vertex of degree higher than 2 in $C_R(G)$ implies the existence in G of submatrix:

$$B = \begin{bmatrix} 2 & 2 & 2 \\ 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}.$$

It is easy to verify that no resolution of this submatrix permits a perfect phylogeny. □

Suppose that instance G of any PH or MPPH problem has two identical columns. It is easy to see that any haplotype matrix H resolving G can be modified, without introducing a forbidden submatrix, to make the corresponding columns in H equal as well (simply delete one column and duplicate another). It follows that in solving any PH or MPPH problem we can start with collapsing identical columns. In particular this is true for MPPH$(*, 2)$-C1, which leads to the first step of the algorithm **A** that we propose for solving MPPH$(*, 2)$-C1:

Step 1 of A: Collapse identical columns in G.

From now on, we assume that there are no identical columns. Denote by G_0, G_1 and G_2 the sets of genotypes in G with, respectively, degree 0,1, and 2 in $C_R(G)$. For any genotype g of degree 1 in $C_R(G)$ there is exactly one genotype with a 2 in the same column as g. Because there are no identical columns, it follows that any genotype g of degree 1 in $C_R(G)$ can have at most two 2s. Similarly any genotype of degree 2 in $C_R(G)$ has at most three 2s. Accordingly we define G_1^1 and G_1^2 as the genotypes in G_1 that have one 2 and two 2s, respectively, and similarly G_2^2 and G_2^3 as the genotypes in G_2 with two and three 2s, respectively.

The following lemma states how genotypes in these sets must be resolved if no submatrix F is allowed in the solution. If genotype g has k 2s we denote by $g[a_1, a_2, \ldots, a_k]$ the haplotype with entry a_i in the position where g has its i-th 2 and 0 everywhere else.

Lemma 3.4. *A haplotype matrix is a feasible solution to MPPH$(*, 2)$-C1 if and only if all genotypes are resolved in one of the following ways:*

(i) *A genotype $g \in G_0 \cup G_1^1$ is resolved by the haplotypes $g[1]$ and $g[0] = h_t$ (or just by h_t if $g = h_t$).*
(ii) *A genotype $g \in G_2^2$ is resolved by $g[0,1]$ and $g[1,0]$.*
(iii) *A genotype $g \in G_1^2$ is either resolved by $g[0,0] = h_t$ and $g[1,1]$ or by $g[0,1]$ and $g[1,0]$.*
(iv) *A genotype $g \in G_2^3$ is either resolved by $g[1,0,0]$ and $g[0,1,1]$ or by $g[0,1,0]$ and $g[1,0,1]$ (assuming that the two neighbours of g have a 2 in the first two positions where g has a 2).*

Proof. A genotype $g \in G_2^2$ has degree 2 in $C_R(G)$, which implies the existence in G of a submatrix:

$$D = \begin{array}{c} g \\ g' \\ g'' \end{array} \begin{bmatrix} 2 & 2 \\ 2 & 0 \\ 0 & 2 \end{bmatrix} .$$

Resolving g with $g[0,0]$ and $g[1,1]$ clearly leads to the forbidden submatrix F in the haplotype matrix because g' needs a haplotype with 1 and 0 and g'' needs a haplotype with 0 and 1 in these columns. Similarly, resolving a genotype $g \in G_2^3$ with $g[0,0,1]$ and $g[1,1,0]$ or with $g[0,0,0]$ and $g[1,1,1]$ leads to a forbidden submatrix in the first two columns where g has a 2. It follows for the cases (ii) and (iv) that resolving the genotypes in a way other than described in the lemma yields a haplotype matrix which does not admit a perfect phylogeny. The cases (i) and (iii) just give the unique resolution.

Now suppose that all genotypes are resolved as described in the lemma and assume that there is a forbidden submatrix F in the solution. Without loss of generality, we assume F can be found in the first two columns of the solution matrix. We may also assume that no haplotype can be deleted from the solution. Then, since F contains [1 1], there is a genotype $g \in G$ starting with [2 2]. Since there are no identical columns there are only two possibilities. The first possibility is that there is exactly one other genotype g' with a 2 in exactly one of the first two columns. Since all genotypes different from g and g' start with [0 0], none of the resolutions of $g \in G$ creates the complete submatrix F. Contradiction. The other possibility is that there is exactly one genotype with a 2 in the first column and exactly one other genotype with a 2 in the second column, i.e. we have the submatrix D. Then $g \in G_2^3$ or $g \in G_2^2$

and, according to (ii) and (iv), g is resolved by a haplotype with 0 and 1 and a haplotype with 1 and 0 in the first two columns. Because no other genotype can be resolved by a haplotype with 1s in the first two columns, a resolution does not contain the forbidden submatrix in the first two columns. □

Lemma 3.5. *Let G be an instance of MPPH$(*,2)$ and G_1^2, G_2^3 as defined above.*
(i) *Any nontrivial haplotype is consistent with at most two genotypes in G.*
(ii) *A genotype $g \in G_1^2 \cup G_2^3$ must be resolved using at least one haplotype that is not consistent with any other genotype.*

Proof. *(i)* Let h be a nontrivial haplotype. There is a column where h has a 1 and there are at most two genotypes with a 2 in that column.
(ii) A genotype $g \in G_1^2 \cup G_2^3$ has a 2 in a column that has no other 2s. For its resolution it needs a haplotype with a 1 in this column and this haplotype is not consistent with any other genotypes. □

A haplotype that is only consistent with g is called a *private haplotype* of g. Based on (i) and (ii) of Lemma 3.4 we propose the next step of **A**:

Step 2 of A: Resolve all $g \in G_1^1 \cup G_2^2$ by the unique haplotypes allowed to resolve them according to Lemma 3.4. Also resolve each $g \in G_0$ with h_t and the complement of h_t with respect to g. This leads to a partial haplotype matrix H_2^p.

The next step of **A** is based on Lemma 3.5 (ii).

Step 3 of A: For each $g \in G_1^2 \cup G_2^3$ with $g \sim_{h'} g'$ for some g' and some $h' \in H_2^p$ that is allowed to resolve g according to Lemma 3.4, resolve g by adding the complement h'' of h' w.r.t. g to the set of haplotypes, i.e. set $H_2^p := H_2^p \cup \{h''\}$, and repeat this step as long as new haplotypes get added. This leads to partial haplotype matrix H_3^p.

Notice that if after Step 3 there are any unresolved genotypes left, then they can not be resolved by haplotypes from H_3^p. Let us denote this set of leftover, unresolved genotypes by GL, the degree 1 vertices among those by $GL_1 \subseteq G_1^2$, and the degree 2 vertices among those by $GL_2 \subseteq G_2^3$. The restricted compatibility graph induced by GL, which we denote by $C_R(GL)$ consists of paths and circuits. We first give the final steps of algorithm A and argue optimality afterwards.

Step 4 of A: Resolve each cycle in $C_R(GL)$, necessarily consisting of GL_2-vertices, by starting with an arbitrary vertex and, following the cycle, resolving each next pair g, g' of vertices by haplotype $h \neq h_t$ such that $g \sim_h g'$ and the two complements of h w.r.t. g and g' respectively. Note that h has a 1 in the column where both g and g' have a 2 and otherwise 0. It follows easily that g and g' are both allowed to use h (and its complement) according to (iv) of Lemma 3.4. In case of an odd cycle the last vertex is resolved by any pair of haplotypes that is allowed to resolve it.

Step 5 of A: Resolve each path in $C_R(GL)$ with both endpoints in GL_1 by first resolving the GL_1 endpoints by the trivial haplotype h_t and the complements of h_t w.r.t. the two endpoint genotypes, respectively. The remaining path contains only GL_2-vertices and is resolved according to Step 6.

Step 6 of A: Resolve each remaining path by starting in (one of) its GL_2-endpoint(s), and following the path, resolving each next pair g, g' of vertices as in Step 4 (i.e. by haplotype $h \neq h_t$ such that $g \sim_h g'$ and the two complements of h w.r.t. g and g' respectively). In case of a path with an odd number of vertices, resolve the last vertex by any pair of haplotypes that is allowed to resolve it in case it is a GL_2-vertex, and resolve it by the trivial haplotype and its complement w.r.t. the vertex in case it is a GL_1 vertex.

By construction the haplotype matrix H resulting from **A** resolves G. In addition, from Lemma 3.4 it follows that H admits a perfect phylogeny.

To argue minimality of the solution, first observe that the haplotypes added in Step 2 and Step 3 are unavoidable by Lemma 3.4 (i) and (ii) and by Lemma 3.5 (ii). Lemma 3.5 tells us moreover that the resolution of a cycle of k genotypes in GL_2 requires at least $k + \lceil \frac{k}{2} \rceil$ haplotypes that can not be used to resolve any other genotypes in GL. This proves optimality of Step 4. To prove optimality of the last two steps we need to take into account that genotypes in GL_1 can potentially share the trivial haplotype. Observe that to resolve a path with k vertices one needs at least $k + \lceil \frac{k}{2} \rceil$ haplotypes. Indeed **A** does not use more than that number of haplotypes in Steps 5 and 6. Moreover, since these paths are disjoint, they cannot share haplotypes for resolving their genotypes except for the endpoints if they are in GL_1, which can share the trivial haplotype. Indeed, **A** exploits the possibility of sharing the trivial haplotype in a maximal way, except on a path with an even number of vertices and one endpoint in GL_1. Such a path, with k (even) vertices, is resolved in **A** by $\frac{3}{2}k$ haplotypes that can not be used to resolve any other genotypes. The degree 1 endpoint might alternatively be resolved by the trivial haplotype and its complement w.r.t. the corresponding genotype, adding the latter private haplotype, but then for resolving the remaining path with $k-1$ (odd) vertices only from GL_2 we still need $k - 1 + \lceil \frac{k-1}{2} \rceil$, which together with the private haplotype of the degree 1 vertex gives $3\frac{k}{2}$ haplotypes as well (not even counting h_t).

As a result we have polynomial-time solvability of MPPH$(*, 2)$-C1.

Theorem 3.7. *MPPH$(*, 2)$ is solvable in polynomial time if the compatibility graph is a clique.*

□

3.4 Approximation Algorithms

In this section we construct polynomial time approximation algorithms for PH and MPPH, where the accuracy depends on the number of 2s per column of the input matrix. We describe genotypes without 2s as *trivial* genotypes, since they have to be resolved in a trivial way by one haplotype. Genotypes with at least one 2 will be described as *nontrivial* genotypes. We write PH^{nt} and MPPH^{nt} to denote the restricted versions of the problems where each genotype is nontrivial. We make this distinction between the problems because we have better lower bounds (and thus approximation ratios) for the restricted variants.

PH and MPPH where all Input Genotypes are Nontrivial

To prove approximation guarantees we need good lower bounds on the number of haplotypes in the solution. We start with two bounds from [Sha06], whose proof we give because the first one is short but based on a crucial observation, and the second one was incomplete in [Sha06]. We use these bounds to obtain a different lower bound that we need for our approximation algorithms.

Lemma 3.6. [Sha06] *Let G be an $n \times m$ instance of PH^{nt} (or MPPH^{nt}). Then at least*

$$LB_{sqrt}(n) = \left\lceil \frac{1+\sqrt{1+8n}}{2} \right\rceil$$

haplotypes are required to resolve G.

Proof. The proof follows directly from the observation that q haplotypes can resolve at most $\binom{q}{2} = q(q-1)/2$ nontrivial genotypes. □

Lemma 3.7. [Sha06] *Let G be an $n \times m$ instance of PH$^{nt}(*, \ell)$, for some $\ell \geq 1$, such that the compatibility graph of G is a clique. Then at least*

$$LB_{sha}(n,\ell) = \left\lceil \frac{2n}{\ell+1} + 1 \right\rceil$$

haplotypes are required to resolve G.

Proof. Recall from the previous section that, after relabelling if necessary (i.e. flipping SNPs), the trivial haplotype h_t is the all-0 haplotype and is consistent with all genotypes. Suppose a solution of G has q non-trivial haplotypes. Observe that h_t can be used in the resolution of at most q genotypes. Also observe that each non-trivial haplotype can be used in the resolution of at most ℓ genotypes. Now distinguish two cases. First consider the case where h_t is in the solution. Then from the two observations above it follows that $n \leq (q + \ell q)/2$ and hence the solution consists of at least $q+1 \geq 2n/(\ell+1)+1$ haplotypes and the result follows. Now consider the second case i.e. where

h_t is not in the solution. Then we have that $n \leq \frac{\ell q}{2}$ and hence that the solution consists of at least $\frac{2n}{\ell}$ haplotypes. If $n \geq \frac{1}{2}\ell(\ell+1)$ we have that $1 \leq \frac{2n}{\ell(\ell+1)} = \frac{2n}{\ell} - \frac{2n}{\ell+1}$. This implies that $\frac{2n}{\ell} \geq \frac{2n}{\ell+1}+1$, and the claim follows. If $n < \frac{1}{2}\ell(\ell+1)$ then this implies that $\ell > \frac{\sqrt{1+8n}-1}{2}$. Combining this with that by Lemma 3.6 $q \geq \frac{\sqrt{1+8n}+1}{2}$ gives that $(\ell+1)(q-1) > \frac{1}{4}(\sqrt{1+8n}+1)(\sqrt{1+8n}-1)$, which is equal to $2n$. It follows that $q > \frac{2n}{\ell+1} + 1$. □

The LB_{sha} bound has been proven only for PH^{nt} (and MPPH^{nt}) instances where the compatibility graph is a clique. We now prove a different bound which, in terms of cliques, is slightly weaker (for large n) than LB_{sha}, but which allows us to generalise the bound to more general inputs. (Indeed it remains an open question whether LB_{sha} applies as a lower bound not just for cliques but also for general instances.)

Lemma 3.8. *Let G be an $n \times m$ instance of PH$^{nt}(*, \ell)$, for some $\ell \geq 1$. Then at least*

$$LB^{nt}_{mid}(n, \ell) = \left\lceil \frac{2(n+\ell)(\ell+1)}{\ell(\ell+3)} \right\rceil$$

haplotypes are required to resolve G.

Proof. Let $C(G)$ be the compatibility graph of G. We may assume without loss of generality that $C(G)$ is connected. First consider the case where $C(G)$ is a clique. If $n \geq \ell(\ell+1)/2$, it suffices to notice that $LB^{nt}_{mid}(n,\ell) \leq LB_{sha}(n,\ell)$ for each value of $\ell \geq 1$, since the function

$$f(n) = \frac{2n}{\ell+1} + 1 - \frac{2(n+\ell)(\ell+1)}{\ell(\ell+3)}$$

is equal to 0 if $n = \ell(\ell+1)/2$ and has nonnegative derivative $f'(n) = \frac{2}{\ell+1} - 2\frac{\ell+1}{\ell(\ell+3)} \geq 0$.
Secondly, if $1 \leq n \leq \ell(\ell+1)/2$, straightforward but tedious calculations show that for all $\ell \geq 1$ the function

$$F(n) = \frac{1+\sqrt{1+8n}}{2} - \frac{2(n+\ell)(\ell+1)}{\ell(\ell+3)}$$

has value 0 for $n = \ell(\ell+1)/2$ and for some n in the interval $[0,1]$, whereas in between these values it has positive value. Hence, $LB^{nt}_{mid}(n,\ell) \leq LB_{sqrt}(n)$ for $1 \leq n \leq \ell(\ell+1)/2$.

To prove that the bound also holds if $C(G)$ is not a clique we use induction on n. Suppose that for each $n' < n$ the lemma holds for all $n' \times m$ instances G' of PH$^{nt}(*, \ell')$ for every m and ℓ'. Since $C(G)$ is not a clique there exist two genotypes g_1 and g_2 in G and a column j such that $g_1(j) = 0$ and $g_2(j) = 1$. Given that G is a PH$^{nt}(*, \ell)$ instance $t \leq \ell$ genotypes have a 2 in column j. Deleting these t genotypes yields an instance G^d with disconnected compatibility graph $C(G^d)$, since the absence of a 2 in column j prevents the existence of any path from g_1 to g_2. Let $C(G^d)$ have $p \geq 2$ components $C(G_1), ..., C(G_p)$, and let $n_i \geq 1$ denote the number of genotypes in G_i. Thus, $n = n_1 + ... + n_p + t$. We use the induction hypothesis on $G_1, \ldots, G_p$ to conclude that the number of haplotypes required to resolve G is at least

$$\begin{aligned}
\sum_{i=1}^{p} \left\lceil \frac{2(n_i+\ell)(\ell+1)}{\ell(\ell+3)} \right\rceil &\geq \left\lceil \frac{2(\sum_{i=1}^{p} n_i + p\ell)(\ell+1)}{\ell(\ell+3)} \right\rceil \\
&\geq \left\lceil \frac{2(\sum_{i=1}^{p} n_i + 2\ell)(\ell+1)}{\ell(\ell+3)} \right\rceil \\
&\geq \left\lceil \frac{2(\sum_{i=1}^{p} n_i + t + \ell)(\ell+1)}{\ell(\ell+3)} \right\rceil \\
&= \left\lceil \frac{2(n+\ell)(\ell+1)}{\ell(\ell+3)} \right\rceil .
\end{aligned}$$

□

Corollary 3.1. *Let G be an $n \times m$ instance of PH$^{nt}(*, \ell)$ or MPPH$^{nt}(*, \ell)$, for some $\ell \geq 1$. Any feasible solution for G is within a ratio $\ell + 2 - \frac{2}{\ell+1}$ from optimal.*

Proof. Any solution for G has at most $2n$ haplotypes. By the previous lemma such a solution achieves a ratio of $\frac{2n\ell(\ell+3)}{2(n+\ell)(\ell+1)} \leq \frac{\ell(\ell+3)}{\ell+1} = \ell + 2 - \frac{2}{\ell+1}$. In the case of MPPH we can check whether feasible solutions exist, and if so obtain such a solution, by using the algorithm in for example [Din06]. □

Not surprisingly, better approximation ratios can be achieved. The following simple algorithm computes approximations of PH$^{nt}(*, \ell)$. (The algorithm does not work for MPPH, however.)

Algorithm: PH^{nt}M
Step 1: construct the compatibility graph $C(G)$.
Step 2: find a maximal matching M in $C(G)$ (i.e. a matching M such that there are no edges between two vertices not covered by M).
Step 3: for every edge $\{g_1, g_2\} \in M$, resolve g_1 and g_2 by in total 3 haplotypes: any haplotype consistent with both g_1 and g_2, and its complements with respect to g_1 and g_2.
Step 4: resolve each remaining genotype by two haplotypes.

Theorem 3.8. *PH^{nt}M computes a solution to PH$^{nt}(*, \ell)$ in polynomial time within an approximation ratio of $c(\ell) = \frac{3}{4}\ell + \frac{7}{4} - \frac{3}{2}\frac{1}{\ell+1}$, for every $\ell \geq 1$.*

Proof. Since constructing $C(G)$ given G takes $O(n^2m)$ time and finding a maximal matching in any graph takes linear time, $O(n^2m)$ running time follows directly.

Let q be the size of the maximal matching. Then PH^{nt}M gives a solution with $3q + 2(n - 2q) = 2n - q$ haplotypes. Since the vertices not covered by M form an independent set of size $n - 2q$, any solution must contain at least $2(n - 2q)$ haplotypes to resolve the genotypes in this independent set. The theorem thus holds if $\frac{2n-q}{2n-4q} \leq c(\ell)$. If $\frac{2n-q}{2n-4q} > c(\ell)$, implying that $q > \frac{2-2c(\ell)}{1-4c(\ell)}n$, we use the lower bound of Lemma 3.8 to obtain

$$\begin{aligned}
\frac{2n-q}{LB^{nt}_{mid}(n,\ell)} &< \frac{2n - \frac{2-2c(\ell)}{1-4c(\ell)}n}{LB^{nt}_{mid}(n,\ell)} \\
&< \frac{(2n - \frac{2-2c(\ell)}{1-4c(\ell)}n)\ell(\ell+3)}{2n(\ell+1)} \\
&= \frac{3\ell c(\ell)}{4c(\ell)-1}\frac{\ell+3}{\ell+1} \\
&= c(\ell).
\end{aligned}$$

The last equality follows directly from $(4c(\ell) - 1)(\ell + 1) = 3\ell(\ell + 3)$. □

PH and MPPH where not all Input Genotypes are Nontrivial

Given an instance G of PH or MPPH containing n genotypes, n_{nt} denotes the number of nontrivial genotypes in G and n_t the number of trivial genotypes; clearly $n = n_{nt} + n_t$.

Lemma 3.9. *Let G be an $n \times m$ instance of PH$(*, \ell)$, for some $\ell \geq 2$, where the compatibility graph of the nontrivial genotypes in G is a clique, G is not equal to a single trivial genotype, and no nontrivial genotype in G is the sum of two trivial genotypes in G. Then at least*

$$LB_{mid}(n,\ell) = \left\lceil \frac{n}{\ell} + 1 \right\rceil$$

haplotypes are needed to resolve G.

Proof. Note that the lemma holds if $n_t \geq \frac{n}{\ell} + 1$. So we assume from now on that $n_t < \frac{n}{\ell} + 1$.

We first prove that the bound holds for $n_{nt} \leq \ell$. Combining this with $n_t < \frac{1}{2}n+1$ (since $\ell \geq 2$) gives that $n < 2\ell+2$. Thus $\frac{n}{\ell}+1 < 4$. Hence if $n_t \geq 4$ then we are done. Thus we only have to consider cases where both $n_t \in \{0, 1, 2, 3\}$ and $\ell \geq \max\{2, n_{nt}\}$. We verify these cases in Table 3.2; note the importance of the fact that no nontrivial genotype is the sum of two trivial genotypes in verifying that these are correct lower bounds. (Also, there is no $n_t = 1, n_{nt} = 0$ case because of the lemma's precondition.)

We now prove the lemma for $n_{nt} > \ell$. Note that in this case there exists a unique haplotype (the trivial haplotype h_t) consistent with all nontrivial

Table 3.2: Case $n_t < 4$, $n_{nt} \leq \ell$ in proof of Lemma 3.9

n_t	n_{nt}	$\lceil n/\ell + 1 \rceil$
0	1	2
0	$z \geq 2$	$\leq \lceil z/z + 1 \rceil = 2$
1	1	2
1	$z \geq 2$	$\leq \lceil (z+1)/z + 1 \rceil = 3$
2	0	2
2	1	≤ 3
2	$z \geq 2$	$\leq \lceil (z+2)/z + 1 \rceil = 3$
3	0	≤ 3
3	1	≤ 3
3	2	≤ 4
3	$z \geq 3$	$\leq \lceil (z+3)/z + 1 \rceil = 3$

genotypes. Suppose, by way of contradiction, that $N = N_t + N_{nt}$ is the size of the smallest instance G' for which the bound does not hold. Let H be an optimal solution for G' and let $h = |H|$.

Observe firstly that $N = 1 \pmod{\ell}$, because if this is not true we have that $LB_{mid}(N-1, \ell) = LB_{mid}(N, \ell)$ and we can find a smaller instance for which the bound does not hold, simply by removing an arbitrary genotype from G', contradicting the minimal choice of N.

Similarly we argue that $h = LB_{mid}(N, \ell) - 1$, since if $h \leq LB_{mid}(N, \ell) - 2$ we could remove an arbitrary genotype to yield a size $N-1$ instance and still have that $h < LB_{mid}(N-1, \ell)$.

We choose a specific resolution of G' using H and represent it as a *haplotype graph*. The vertices of this graph are the haplotypes in H. For each nontrivial genotype $g \in G'$ there is an edge between the two haplotypes that resolve it. For each trivial genotype $g \in G'$ there is a loop on the corresponding haplotype. There are no edges between looped haplotypes because of the precondition that no nontrivial genotype is the sum of two trivial genotypes.

From Lemma 5 of [Sha06] it follows that, with the exception of the possibly present trivial haplotype and disregarding loops, each haplotype in the graph has degree at most ℓ. In addition, if an unlooped haplotype has degree less than or equal to ℓ, or a looped haplotype has degree (excluding its loop) strictly smaller than ℓ, then deleting this haplotype and all its at most ℓ incident edges (genotypes) creates an instance G'' containing at least $N - \ell$ genotypes that can be resolved using $h-1$ haplotypes, yielding a contradiction to the minimality of N. (Note that, because $N_{nt} > \ell$, it is not possible that the instance G'' is empty or equal to a single trivial genotype.)

The only case that remains is when, apart from the possibly present trivial haplotype, every haplotype in the haplotype graph is looped and has degree ℓ (excluding its loop). However, there are no edges between looped vertices and they can therefore only be adjacent to the trivial haplotype, yielding a contradiction. □

Lemma 3.10. *Let G be an $n \times m$ instance of PH$(*, \ell)$, for some $\ell \geq 2$, where G is not equal to a single trivial genotype, and no nontrivial genotype in G is the sum of two trivial genotypes in G. Then at least $LB_{mid}(n, \ell)$ haplotypes are needed to resolve G.*

Proof. Essentially the same inductive argument as used in Lemma 3.8 works: it is always possible to disconnect the compatibility graph of G into at least two components by removing at most ℓ nontrivial genotypes, and using cliques as the base of the induction. The presence of trivial genotypes in the input (which we can actually simply exclude from the compatibility graph) does not alter the analysis. The fact that (in the inductive step) at least two components are created, each of which contains at least one nontrivial genotype, ensures that the inductive argument is not harmed by the presence of single trivial genotypes (for which the bound does not hold). □

Corollary 3.2. *Let G be an $n \times m$ instance of PH$(*, \ell)$ or MPPH$(*, \ell)$, for some $\ell \geq 2$. Any feasible solution for G is within a ratio of 2ℓ from optimal.*

Proof. Immediate because $2n/(n/\ell + 1) < 2\ell$. (As before the algorithm from e.g. [Din06] can be used to generate feasible solutions for MPPH, or to determine that they do not exist.) □

The algorithm PH^{nt}M can easily be adapted to solve PH$(*, \ell)$ approximately.

Algorithm: PHM
Step 1: remove from G all genotypes that are the sum of two trivial genotypes
Step 2: construct the compatibility graph $C(G')$ of the leftover instance G'.
Step 3: find a maximal matching M in $C(G')$.
Step 4: for every edge $\{g_1, g_2\} \in M$, resolve g_1 and g_2 by three haplotypes if g_1 and g_2 are both nontrivial and by two haplotypes if one of them is trivial.
Step 5: resolve each remaining nontrivial genotype by two haplotypes and each remaining trivial genotype by its corresponding haplotype.

Theorem 3.9. *PHM computes a solution to PH$(*, \ell)$ in polynomial time within an approximation ratio of $d(\ell) = \frac{3}{2}\ell + \frac{1}{2}$, for every $\ell \geq 2$.*

Proof. Since constructing $C(G)$ given G takes $O(n^2m)$ time and finding a maximal matching in any graph takes linear time, $O(n^2m)$ running time follows immediately.

Let q be the size of the maximal matching, n the number of genotypes after Step 1 and n_t the number of trivial genotypes in G'. Then PHM gives a solution with $2n - q - n_t$ haplotypes. Since the vertices not covered by the maximal matching form an independent set of size $n - 2q$ in $C(G')$, any solution must contain at least $n-2q$ haplotypes to resolve the genotypes in this independent set. The theorem thus holds if $\frac{2n-q-n_t}{n-2q} \leq d(\ell)$. If $\frac{2n-q-n_t}{n-2q} > d(\ell)$, implying that $q > \frac{(d(\ell)-2)n+n_t}{2d(\ell)-1}$, we use the lower bound of Lemma 3.10 and obtain

$$\begin{aligned} \frac{2n-q-n_t}{LB_{mid}(n,\ell)} &< \frac{2n - \frac{(d(\ell)-2)n+n_t}{2d(\ell)-1}}{\lceil \frac{n}{\ell}+1 \rceil} \\ &< \frac{2n - \frac{(d(\ell)-2)n}{2d(\ell)-1}}{\frac{n}{\ell}} \\ &= \frac{3d(\ell)\ell}{2d(\ell)-1} \\ &= d(\ell). \end{aligned}$$

The last equality follows from the observation that $2d(\ell) - 1 = 3\ell$. □

3.5 Conclusion and Open Problems

There remain a number of open problems to be solved. The complexity of PH$(*, 2)$ and MPPH$(*, 2)$ is still unknown. An approach that might raise the necessary insight is to study the PH$(*, 2)$-Cq and MPPH$(*, 2)$-Cq variants of these problems (i.e. where the compatibility graph can be obtained from q cliques, by identifying some of the vertices) for small q. If a complexity result nevertheless continues to be elusive then it would be interesting to try and improve approximation ratios for PH$(*, 2)$ and MPPH$(*, 2)$; might it even be possible to find a PTAS (*Polynomial-time Approximation Scheme*) for each of these problems? Note also that the complexity of PH$(k, 2)$ and MPPH$(k, 2)$ remains open for constant $k \geq 3$.

Another intriguing open question concerns the relative complexity of PH and MPPH instances. Has PH(k, ℓ) always the same complexity as MPPH(k, ℓ) and are both problems always equally hard to approximate? A related question is whether it is possible to directly encode PH instances as MPPH instances, and/or vice-versa, and if so whether/how this affects the bounds on the number of 2s in columns and rows.

For hard PH(k, ℓ) instances it would also be interesting to see if those approximation algorithms that yield approximation ratios as functions of k, can be intelligently combined with the approximation algorithms in this chapter (having approximation ratios determined by ℓ), perhaps with superior approximation ratios as a consequence. In terms of approximation algorithms for MPPH there is a lot of work to be done because the approximation algorithms presented in this chapter actually do little more than return an arbitrary feasible solution. It is also not clear if the 2^{k-1}-approximation algorithms for

PH($k, *$) can be attained (or improved) for MPPH. More generally, it seems likely that big improvements in approximation ratios (for both PH and MPPH) will require more sophisticated, input-sensitive lower bounds and algorithms. What are the limits of approximability for these problems, and how far will algorithms with formal performance-guarantees (such as in this chapter) have to improve to make them competitive with dominant ILP-based methods?

Finally, with respect to MPPH, it could be good to explore how parsimonious the solutions are that are produced by the various PPH feasibility algorithms, and whether searching through the entire space of PPH solutions (as proposed in [Vij06]) yields practical algorithms for solving MPPH.

Chapter 4

Phylogenetic Networks

4.1 Introduction

One of the ultimate goals in computational biology is to create methods that can reconstruct evolutionary histories from biological data of currently living organisms. The immense complexity of biological evolution makes this task almost a hopeless one [Mor05], which has motivated researchers to focus first on the simplest possible pattern of evolution. This least complicated shape of an evolutionary history is the tree-shape. Now that treelike evolution has been studied intensively, a logical next step is to consider slightly more complicated evolutionary scenarios, gradually extending the complexity that our models can describe. At the same time we also wish to take into account the parsimony principle, which tells us that amongst all equally good explanations of our data, one prefers the simplest one (see Section 1.3 and e.g. [Hei90]).

For a set of taxa (e.g. species or strains), a *phylogenetic tree* describes (a hypothesis of) the evolution that these taxa have undergone. The taxa form the leaves of the tree while the internal vertices represent events of genetic divergence: one incoming branch splits into two (or more) outgoing branches. These internal vertices can thus be seen as representing a common ancestor of all taxa below it. This thesis focusses on binary trees, in which internal vertices have two outgoing branches. More general trees can be used to describe evolutionary histories in which the precise order of divergence is unclear, but these will not be used in this thesis.

Phylogenetic networks extend the phylogenetic tree model with the extra possibility that two branches combine into one new branch. We call such an event a *reticulation*, which can model any kind of non-treelike (also called "reticulate") evolutionary process such as recombination, hybridisation or lateral gene transfer. In addition, phylogenetic networks can also be used to describe evolutions that are believed to be tree-like when the precise shape of this tree is not clear. In this case, reticulations can be used to display different possible treelike evolutions in one figure. The recent years have seen a rapidly growing interest in phylogenetic networks and their application [Bar04; Mor04; Mor05; Hus06; Mak06; Gam08].

Level-k Networks

This model of a phylogenetic network allows for many different degrees of complexity, ranging from networks that are equal, or almost equal, to a tree to complex webs of frequently diverging and recombining lineages. We consider two different measures for the complexity of a network. The first of these measures is the total number of reticulations in the network. Secondly, we consider the "level" of the network, which is (informally) an upper bound on the number of reticulations per non-treelike part of the network. In graph theory, a subgraph is called *biconnected* if it cannot be disconnected by deleting a single vertex. A *biconnected component* is a maximal biconnected subgraph. Formally, a network is a *level-k network* if it contains at most k reticulations per biconnected component. Thus the *level* of networks restricts how interwoven the reticulations can be. In trees (i.e. level-0 networks) no reticulation events occur; in level-1 networks all reticulation cycles must be disjoint. The higher the level of the network, the more freedom in reticulation is allowed. See for example Figure 4.1 for a level-2 network. Its nontrivial (i.e. containing at least three vertices) biconnected components are coloured grey. Level-1 networks have also been called *galled trees* [Gus04], *gt-networks* [Nak05] and *galled networks* [Jan06a]. General level-k networks were first introduced by Choy, Jansson, Sadakane and Sung [Cho05].

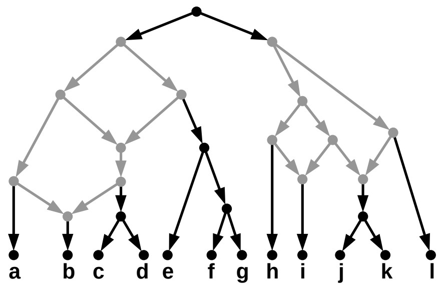


Figure 4.1: An example of a level-2 network.

The focus on level is motivated by several factors. Firstly, level induces a hierarchy on the space of networks with lower level networks being more "tree-like" than higher-level networks. Identifying the position of candidate solutions (i.e. networks) within this hierarchy, or finding the minimum level at which candidate solutions exist, communicates important structural information about the solution space. Level minimisation, which derives its legitimacy from the parsimony principle, can also be used in an implicit context e.g. to measure the accuracy of input data. For example, if we expect the solution to be a tree, but only obtain higher level networks, this suggests that data errors lie in the regions corresponding to the biconnected components. Secondly, from an algorithmic perspective, focussing on lower-level networks can potentially lead to polynomial-time solvability, better running-times and/or clearer mathematical analysis. Finally, restricting level is useful for avoiding trivial (and unrealistic) solutions. Indeed, several of the problems that are discussed in this chapter can be trivially solved if we allow solutions with high enough level, but (as we shall see) such solutions communicate no useful information.

Triplets

This chapter considers a triplet-based approach to construct directed phylogenetic networks. As input we take a collection of *triplets*: rooted phylogenetic trees on size-3 subsets of the taxa, see Figure 4.2 for an example. These triplets can be constructed by methods such as Maximum Parsimony [Swo98] or Maximum Likelihood [Gui03], that work accurately and fast for small numbers of taxa. Another possibility is to infer the triplets from a set of phylogenetic trees, possibly originating from different sources. However way the triplets are obtained, the next step is to combine them into a single, large phylogenetic network for all taxa. Designing algorithms for the latter task forms the subject of this chapter. Triplet methods have become popular since they allow us to solve certain problems in polynomial time, as will be elaborated on shortly. Another advantage of these methods is that they provide the possibility to combine different sorts of biological data.

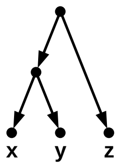


Figure 4.2: One of three possible triplets on leaves x, y and z.

Triplet-based methods have been extensively studied in the literature. Aho et al. [Aho81] gave a polynomial-time algorithm that constructs a tree from triplets if there exists a tree that is consistent with all input triplets. This positive result provided the stimulus for studying the applicability of triplet-based methods to networks. Unfortunately, it has been shown that for level-1 networks the corresponding problem becomes NP-hard [Jan06a]. On the positive side, the same paper gives a polynomial-time algorithm for the problem where the input is *dense*, i.e. there is at least one triplet in the input for every size-3 subset of the taxa. A related problem that accommodates errors in the triplets is finding a tree consistent with as many input triplets as possible. This problem is NP-hard [Bry97; Jan01; Wu04], and approximation algorithms have been explored both for the construction of trees [Gąs99] and level-1 networks [Byr08; Jan06a]. For the construction of trees, also efficient heuristics (without approximation guarantees) have been designed by Semple and Steel [Sem00], Page [Pag02], Wu [Wu04] and Snir and Rao [Sni06]. The last algorithm (MAX CUT triplets) is shown to outperform the (not triplet-based) method Matrix Representation with Parsimony (MRP) [Rag92], which is popular in practice [Bau92; Rag92; San98].

Complexity results
Section 4.3.1 of this thesis starts by giving a level-$(n-1)$ network that is consistent with any triplet set on n leaves. This shows that, when there is no bound on the level nor on the total number of reticulations, constructing networks from triplets is not an interesting problem (unless different restrictions are imposed). The result is also used to explore an interesting uniqueness question. Biologists are in general not only interested in finding *a* solution to their problem, but are also eager to know whether this is the only possible solution (and if not, how dissimilar different solutions are). It is therefore an important open question to characterise which triplet sets uniquely define a phylogenetic network. Section 4.3.2 gives some initial progress into answering this question by giving, for each k, a level-k network that is uniquely defined by the set of triplets it is consistent with. Section 4.3.3 shows how useful this result is in the analysis of triplet methods. Firstly, it is used to generalise the result by Jansson et al. that it is NP-hard to construct level-1 networks from general triplet sets [Jan06a]. It is shown that this problem is in fact NP-hard for the construction of level-k networks, for *all* $k \geq 1$. Secondly, the unique networks are used to prove that it is also NP-hard to construct level-k networks consistent with a maximum number of input triplets from a dense triplet set, for all $k \geq 0$. This generalises the known result for level-0 [Bry97; Jan01; Wu04] to all levels and in addition strengthens it by showing that the problem is even NP-hard for dense triplet sets.

Algorithmic results
Sections 4.4 - 4.8 respond to the aforementioned intractability with a series of positive results. These sections present various algorithms that can be used to construct phylogenetic networks. Since it is NP-hard to construct networks consistent with a maximum number of input triplets, it is interesting to develop algorithms that run in exponential time. Wu described an exact algorithm [Wu04] that finds a tree consistent with a maximum number of input triplets in $O(3^n(n^2+m))$ time, with m the number of triplets and n the number of leaves. Section 4.4 extends this approach by giving an exact algorithm that constructs level-1 networks and runs in time $O(m4^n)$.

Informally, *simple* networks are networks that consist of just one biconnected component with leaves hanging from it. Jansson et al. showed how simple level-1 networks can be constructed from dense triplet sets. In addition, they showed how (in the dense case) general level-1 networks can be built by recursively building simple level-1 networks [Jan06a]. It is still an important open problem whether level-k networks can be built from dense triplet sets in polynomial time for each fixed k. However, Section 4.5 shows that this problem is indeed polynomial-time solvable if we restrict to simple networks. In addition, Section 4.6 shows that even general level-2 networks can be built from dense triplet sets in polynomial time. The latter algorithm has been applied in practice to help identify the origin of an outbreak of the yeast *Cryptococcus gattii*, which led to many infections and several fatalities on the Westcoast of Canada.

Minimising reticulations has been well studied in the setting where the input consists of (binary) sequences [Hei90; Son04; Son05a; Gus07]. For example, Wang et al. considered the problem of finding a "perfect phylogeny" with a minimum number of reticulations and showed that this problem is NP-hard [Wan01]. Gusfield et al. showed that this problem can be solved in polynomial time if restricted to level-1 networks [Gus04]. There are also several results already about the version of the problem where the input consists of a set of trees and the objective is to construct a network that is "consistent" with each of the input trees. Baroni et al. give bounds on the minimum number of reticulations needed to combine two trees [Bar05] and Bordewich and Semple showed that it is APX-hard to compute this minimum number [Bor07b]. However, there exists an exact algorithm [Bor07a] that runs reasonably fast in many practical situations. If restricted to level-1 networks, the problem becomes polynomial-time solvable even if there are more than two (but a fixed number of) input trees [Huy05].

In relation to triplet-methods, minimising reticulations has been less well studied. In the reduction proving NP-hardness of constructing level-1 networks from triplets [Jan06a, Theorem 7] only one reticulation is used. Thus, NP-hardness of finding a network consistent with a non-dense triplet set that contains a minimum number of reticulations follows immediately, if there are no restrictions on the level of the constructed network. It is unknown whether this problem becomes easier if the input triplet set is dense. However, in Section 4.7 of this thesis the restriction of this problem to level-k networks is considered and polynomial-time algorithms are given that construct level-1 and level-2 networks with a minimum number of reticulations from a dense triplet set. The level-1 version MARLON (Minimum Amount of Reticulation Level One Network) has been implemented, tested and applied to simulated data, with promising results.

The last result of this thesis is presented in Section 4.8, where it is shown that level-k networks can indeed be constructed in polynomial time for all fixed k in the case that the input triplet set is exactly equal to the set of triplets consistent with some level-k network. If that is the case then algorithm MINPITS can find such a network that simultaneously minimises level *and* the total number of reticulations used. The fact that in this case optimal solutions for both measures coincide, is an interesting consequence of the extra assumption about the input triplets.

The chapter is concluded in Section 4.9 with a list of interesting open problems concerning the construction of level-k phylogenetic networks.

4.2 Preliminaries

A *phylogenetic network* (*network* for short) is defined as a directed acyclic graph in which a single vertex has indegree 0 and outdegree 2 (the *root*) and all other vertices have either indegree 1 and outdegree 2 (*split vertices*), indegree 2 and outdegree 1 (*reticulations*) or indegree 1 and outdegree 0 (*leaves*), where the leaves are distinctly labelled.

A directed acyclic graph is *connected* (also called “weakly connected”) if there is an undirected path (ignoring arc orientations) between any two vertices and is *biconnected* if it contains no vertex whose removal disconnects the graph. A biconnected subgraph H of a graph G is said to be a *biconnected component* if there is no biconnected subgraph $H' \neq H$ of G that contains H. A biconnected component is called *trivial* if it is equal to two vertices connected by an arc. An arc $a = (u, v)$ of a network N is a *cut-arc* if its removal disconnects N and it is *trivial* if v is a leaf. A vertex w is *below* an arc $a = (u, v)$ (and *below* vertex v) if there is a directed path from v to w.

Definition 4.1. *A network is said to be a* level-k network *if each biconnected component contains at most k reticulations.*

To avoid “redundant” networks, we require every non-trivial biconnected component of a network to have at least three outgoing arcs. The reason for this is that a nontrivial biconnected component with two outgoing arcs can simply be replaced by a single split-vertex (without loosing consistency with any triplet).

A level-k network is a *strict* level-k network if it is not a level-$(k-1)$ network. Level-0 networks are called *phylogenetic trees* (*trees* for short); they have no reticulations.

A *triplet* $xy|z$ is a tree on leaves x, y and z such that the lowest common ancestor of x and y is a proper descendant of the lowest common ancestor of x and z, see Figure 4.2 on page 67.

For a network N, we use $L(N)$ to denote the set of leaves of N. Thus, for a triplet t, $L(t)$ denotes the set of leaves of t. For a set of triplets T we use (by extension) $L(T)$ to denote the set of all leaves of all triplets in T, i.e. $L(T) = \bigcup_{t \in T} L(t)$. Furthermore, we use n to denote the number of leaves in T. For $L' \subseteq L(T)$ denote by $T|L'$ the set of triplets $t \in T$ with $L(t) \subseteq L'$. A set T of triplets is *dense* if it contains at least one triplet for each size-3 subset of $L(T)$. Whenever we refer to a *path* in a network we mean a directed path.

Definition 4.2. *A triplet $xy|z$ is* consistent *with a network N (interchangeably: N is consistent with $xy|z$) if N contains a subdivision of $xy|z$, i.e. if N contains distinct vertices u and v and pairwise internally vertex-disjoint paths $u \to x$, $u \to y$, $v \to u$ and $v \to z$.*

By extension, a set T of triplets is *consistent* with N (interchangeably: N is consistent with T) if every triplet in T is consistent with N and $L(T) = L(N)$. For example, Figure 4.1 on page 66 is a level-2 network with two non-trivial biconnected components, each of them containing two reticulations. This network is consistent with (amongst others) the triplets $bc|a$, $bd|h$, $hi|k$ and $ki|h$, but is not consistent with $dk|g$, $ac|b$, $gk|i$ or $cg|e$.

We introduce the class of *simple* level-k networks. Intuitively, these are the basic building blocks of level-k networks in the sense that each non-trivial biconnected component of a level-k network is in essence a simple level-ℓ network, for some $\ell \leq k$. These simple networks will be built by adding leaves to "generators", which we define formally as follows.

Definition 4.3. *A* simple level-k generator*, for $k \geq 1$, is a directed acyclic biconnected multigraph, which has a single root (indegree 0 and outdegree 2), precisely k reticulations (indegree 2 and outdegree at most 1) and apart from that only split vertices (indegree 1 and outdegree 2).*

The arcs of a generator and its vertices with indegree 2 and outdegree 0 are labelled and called *sides*. Note that reticulations in generators are allowed to have outdegree 0, while in networks reticulations always have outdegree 1. The following lemma shows that the graphs in Figure 4.3 are the only simple level-1 and simple level-2 generators. Here and in all subsequent figures, all arcs are directed downwards. For ease of viewing, arrow heads will from now on be omitted.

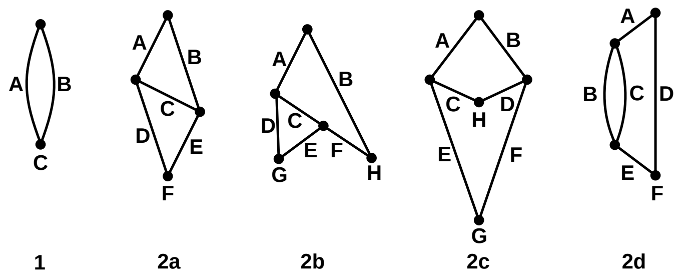


Figure 4.3: The unique simple level-1 generator and the four simple level-2 generators.

Lemma 4.1. *There is one simple level-1 generator and there are four simple level-2 generators and these are depicted in Figure 4.3.*

Proof. To see that graph 1 in Figure 4.3 is the only simple level-1 generator, note firstly that a generator does (by definition) not contain leaves. Hence, the head of each arc is either a split vertex or a reticulation vertex. A reticulation can be the head of at most two arcs and a split vertex increases the number of arcs that still need a head by one. The root vertex is the tail of two arcs and there is precisely one reticulation, so the simple level-1 generator does not contain any split vertices. The uniqueness of the simple level-1 generator follows.

There remains the case of the simple level-2 generators. By the above reasoning a simple level-2 generator has at most two split vertices; three or more split vertices would mean that at least five arcs would need a reticulation vertex as head, and two reticulations can be the head of at most four arcs. Similarly, a level-2 generator must have at least one split vertex.

Case 1: one split vertex. Consider the two arcs leaving the root. It is not possible that they both have the same reticulation r as head because then the removal of r would disconnect the graph. So precisely one of these arcs has a split vertex s as head and the other a reticulation. There are no other split vertices so both arcs leaving s must have reticulations as head. The two possibilities for this lead to $2a$ and $2d$ from Figure 4.3.

Case 2: two split vertices. Let (r, x) and (r, y) be the two arcs leaving the root. It is again not possible that x and y are both equal to the same reticulation. Consider the case where x and y are both equal to split vertices. This creates four arcs that need a head, so all the arcs leaving x and y need a reticulation as head. There is only one way to do this such that the graph is biconnected; this creates $2c$. There remains only the case when (without loss of generality) x is a split vertex and y is a reticulation. Consider the two arcs leaving x: (x, p) and (x, q) say. Then p and q are not both equal to the same reticulation, because then the two arcs leaving the second split vertex still need a head, and y can be the head of only one other arc. So (without loss of generality) p is also a split vertex. Vertex q is not equal to y because then the resulting graph would not be biconnected. So q and y are the children of p. This gives $2b$. □

Computer calculations revealed the 65 simple level-3 generators [Kel08]. For high k, the number of different simple level-k generators is huge. In Figure 4.4 are two examples; one with a maximum and one with a minimum number of vertices and arcs.

Lemma 4.2. *If $G = (V, A)$ is a simple level-k generator then $2k \leq |V| \leq 3k-1$ and $3k - 1 \leq |A| \leq 4k - 2$.*

Proof. Suppose G contains r_0 vertices with indegree 2 and outdegree 0, r_1 vertices with indegree 2 and outdegree 1 and s vertices with indegree 1 and outdegree 2. The sum of the indegrees of all vertices is $s + 2r_1 + 2r_0$, while the sum of their outdegrees is $2 + 2s + r_1$. Clearly, the sum of all outdegrees equals the sum of all indegrees. It follows that $s = r_1 + 2r_0 - 2$. Because $r_0 + r_1 = k$, it follows that that the total number of vertices equals:

$$|V| = 1 + s + r_1 + r_0 = 2k - 1 + r_0 \ .$$

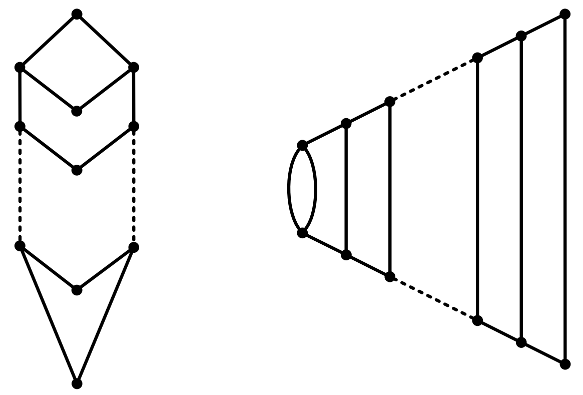


Figure 4.4: Two examples of simple level-k generators; the left one has a maximum number of vertices and arcs and the right one a minimum number. Remember that all arcs are directed downwards.

The total indegree, hence the total number of arcs of G is:

$$|A| = s + 2r_1 + 2r_0 = 3k - 2 + r_0 \ .$$

The number of vertices and arcs in a simple level-k generator thus only depends on k and r_0. Since $1 \leq r_0 \leq k$ the lemma follows. □

Definition 4.4. *A simple level-k network, for $k \geq 1$, is a network obtained by applying the following transformation (called "leaf hanging") to some simple level-k generator G:*

1. *first, for each pair u, v of vertices in G connected by a single arc (u, v), replace (u, v) by a path with $\ell \geq 0$ internal vertices and for each such internal vertex w add a new leaf x and an arc (w, x);*

2. *second, for each pair u, v of vertices in G connected by two arcs replace one such arc by a path with $\ell \geq 1$ internal vertices and for each such internal vertex w add a new leaf x and an arc (w, x); and treat the other arc between u and v as in step 1;*

3. *third, for each vertex v of G with indegree 2 and outdegree 0 add a new leaf y and an arc (v, y).*

Note that it is not necessary to hang leaves on all sides since in step 1 it is allowed to replace an arc by a path with no internal vertices, which does not change the network. The reason that step 2 is slightly different from step 1 is that the generator might contain multiple arcs while networks are not allowed to contain multiple arcs. Therefore, for each pair of multiple arcs, at least

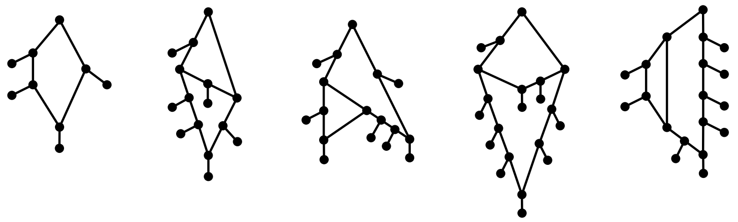


Figure 4.5: Examples of a simple level-1 network and four simple level-2 networks. From left to right, these networks are of type 1, 2a, 2b, 2c and 2d.

one of them needs to be replaced by a path with at least one internal vertex. Moreover, we notice that at least three leaves have to be added to G, to avoid redundancy of the constructed network. A network is *simple* if it is a simple level-k network for some k. A simple level-k network built by hanging leaves from generator G is called a *network of type* G. See Figure 4.5 for an example of a simple level-1 network and simple level-2 networks of type 2a, 2b, 2c and 2d.

We do not attempt to define simple level-0 networks; instead we introduce the *basic tree* which we define as the directed graph on three vertices $\{v_1, v_2, v_3\}$ with arc set $\{(v_1, v_2), (v_1, v_3)\}$.

There is a nice and simple characterisation of simple level-k networks ($k \geq 1$):

Lemma 4.3. *A strict level-k network is a simple level-k network if and only if it contains no nontrivial cut-arcs.*

Proof. A simple level-k network contains no nontrivial cut-arcs because simple level-k generators are biconnected. Now take a strict level-k network N with no nontrivial cut-arcs. It remains to show that N is a simple level-k network. Delete all leaves from N and subsequently suppress all vertices with indegree and outdegree equal to one. The resulting graph still contains exactly k reticulations and contains no cut-arcs. It follows that this graph is biconnected, because any graph with degree at most three containing a cut-vertex also contains a cut-arc. Therefore, this graph is a simple level-k generator and hence N is a simple level-k network. □

Lemma 4.4. *Any simple level-k network on n leaves contains* $2n + 2k - 1$ *vertices and* $2n + 3k - 2$ *arcs.*

Proof. Let $N = (V, A)$ be a simple level-k network with n leaves and s split vertices. The sum of the indegrees of all vertices is $s + 2k + n$, while the sum of their outdegrees is $2 + 2s + k$. Since the sum of all outdegrees equals the sum of all indegrees, $s = n + k - 2$. Hence, we obtain that the total number of vertices equals

$$|V| = s + k + n + 1 = (n + k - 2) + k + n + 1 = 2n + 2k - 1 \ . \qquad (4.1)$$

The total indegree, hence the total number of arcs in N is

$$|A| = n + 2k + s = 2n + 3k - 2 \ .$$

□

This lemma enables us to bound the number of vertices and arcs in general level-k networks as $O(n)$, for fixed k.

Lemma 4.5. *Any level-k network on n leaves contains at most $2n-1+k(n-1)$ vertices and at most $2n-2+\frac{3}{2}k(n-1)$ arcs.*

Proof. Consider a level-k network $N=(V,A)$ with q nontrivial biconnected components. Let $B(N)$ be the result of replacing each nontrivial biconnected component C by a single vertex (i.e. contracting all arcs in C). Thus $B(N)$ is a tree, q of the internal vertices of $B(N)$ represent biconnected components of N and the other internal vertices of $B(N)$ represent split-vertices of N and its root. Let $B(N)$ contain b internal vertices with $n_1,\ldots,n_b$ outgoing arcs respectively. Then $B(N)$ contains $b+n$ vertices in total and $b+n-1$ arcs. Denote by k_i the number of reticulations in the biconnected component of N represented by the i-th internal vertex of $B(N)$ and let $k_i=0$ if this internal vertex represents a split-vertex or the root of N.

Assume the i-th internal vertex of $B(N)$ represents a biconnected component C_i of N. Consider C_i, the cut-arcs leaving C_i and the n_i vertices that have a parent on C_i. This forms a simple level-k_i network with n_i leaves. By (4.1) it has $2n_i+2k_i-1$ vertices. Replacing C_i by a single vertex thus reduces the number of vertices by n_i+2k_i-2.

The number of vertices of N is equal to the number of vertices in $B(N)$ plus the number of vertices that have been deleted while replacing each C_i by a single vertex:

$$\begin{aligned} |V| &= b+n+\sum_{i=1}^{b}(n_i+2k_i-2) \\ &\leq b+n+(b+n-1)+2q\cdot k-2b \\ &\leq 2n-1+k(n-1) \ . \end{aligned}$$

For the first inequality we use that $n_1+\ldots+n_b$ is equal to the number of arcs of $B(N)$, which is $b+n-1$, that $k_i \leq k$ for all i and that $k_i=0$ for $b-q$ values of i. For the second inequality we use that each nontrivial biconnected component of N has at least three outgoing arcs. This implies that the sum of all outdegrees is at least $3q+2(b-q)$. The sum of all outdegree is equal to the sum of all indegrees, which is $b+n-1$. It follows that $q+b \leq n-1$ and hence that $2q \leq n-1$. Similarly, the number of arcs of N is at most:

$$\begin{aligned} |A| &= b+n-1+\sum_{i=1}^{b}(n_i+3k_i-2) \\ &\leq b+n-1+(b+n-1)+3q\cdot k-2b \\ &\leq 2n-2+\frac{3}{2}k(n-1) \ . \end{aligned}$$

□

Note that if we allow networks where nontrivial biconnected components have two (or more) outgoing arcs, then the above proof can be adapted to show that there are at most $2n-1+2k(n-1)$ vertices and $2n-2+3k(n-1)$ arcs. If there is no restriction on the number of outgoing arcs per biconnected component then the size of the graph is unbounded. However, this thesis only considers networks with at least three arcs going out of any nontrivial biconnected component.

In the proofs in this chapter, frequently leaves will be deleted from a network. This might result in a graph that is not a valid network. Therefore, we define *tidying up* a directed acyclic graph as repeatedly applying the following five steps until none is applicable: (1) delete unlabelled vertices with outdegree 0; (2) suppress vertices with indegree and outdegree 1 (i.e. contract one arc incident to the vertex); (3) replace multiple arcs by single arcs; (4) remove the root if it has outdegree 1 and (5) replace nontrivial biconnected components with at most two outgoing arcs by a single vertex (i.e. contract all arcs in the biconnected component). Observe that if N' is the result of removing leaves L' from network N and tidying up the resulting graph, then N' is a valid network. In addition, observe that in this case N' is consistent with exactly the same triplets as N, except for triplets containing leaves from L'.

We will now define SN-sets (short for "Side Network" sets), introduced in [Jan06b], which will play a crucial role in Sections 4.5 - 4.8. For a triplet set T, a subset X of $L(T)$ is an *SN-set* if there is no triplet $xy|z \in T$ with $x, z \in X$ and $y \notin X$. An SN-set is called *nontrivial* if it is not equal to $L(T)$. Furthermore, we say that an SN-set X is *maximal* (*under a restriction* R) if there is no nontrivial SN-set (satisfying restriction R) that is a strict superset of X. An SN-set of size one is called a *singleton* SN-set.

For any two SN-sets of a dense triplet set holds that either they are disjoint or one is included in the other [Jan06b, Lemma 8]. Such a set is called *laminar*. This leads to the following definition. The *SN-tree* is the directed tree in which vertices have outdegree either zero or at least two, such that the SN-sets of T correspond exactly to the sets of leaves reachable from a vertex of the SN-tree. It follows that there are at most $2(n-1)$ nontrivial SN-sets of a dense triplet set T. All these SN-sets can be found by constructing the SN-tree in $O(n^3)$ time [Jan06a]. If a network is consistent with a dense triplet set T, then the set of leaves S below any cut-arc is always an SN-set, since triplets of the form $xy|z$ with $x, z \in S$, $y \notin S$, are not consistent with such a network.

A cut-arc (u, v) is *highest* if there does not exist a cut-arc (u', v') such that u is reachable from v'. A little thought should make it clear that the sets of leaves below highest cut-arcs partition $L(N)$. The following lemma reveals a crucial characteristic that we exploit in our algorithms.

Lemma 4.6. *Let N be a network consistent with a dense triplet set T. Then each maximal SN-set S of T is the union of sets of leaves below highest cut-arcs in N.*

Proof. Suppose that S is a maximal SN-set. Since the set of leaves below a cut-arc is always an SN-set, maximality of S implies that S is not a strict subset of the leaves below some single highest cut-arc a. If S is equal to the set of leaves below a single cut-arc, we are done. Now suppose that S intersects with the leaves below at least two highest cut-arcs. Let leaves $x, z \in S$ be below highest cut-arcs a_1, a_2 respectively. For any leaf y below a_1 now follows that $y \in S$, since the only triplet on x, y, z consistent with N is $xy|z$. Similarly, each leaf y below a_2 is also in S. The lemma follows. □

4.3 Complexity of Constructing Networks from Triplets

This section considers the complexity of constructing level-k phylogenetic networks from triplets. To do so, Sections 4.3.1 and 4.3.2 first give several auxiliary results that enable us to prove the intractability results in Section 4.3.3. However, these auxiliary results are indeed also interesting in their own right. In Section 4.3.1 a network is given that is consistent with any triplet set. Furthermore, it is shown that among all networks with this property the given network has the smallest possible level and the smallest possible number of vertices and arcs. This is then used in Section 4.3.2 to give, for each k, a level-k network that is uniquely defined by the set of triplets consistent with it. Finally, this uniqueness result is used in Section 4.3.3 to give two intractability results with respect to constructing level-k phylogenetic networks from triplets.

4.3.1 Sufficiency and Necessity of Network Level

In this section we prove that any triplet set on n leaves is consistent with a level-$(n-1)$ network. Subsequently, we show that this bound is tight by giving a triplet set on n leaves that is not consistent with any network of level smaller than $n - 1$.

Let $T_F(n)$ be the set of all $3\binom{n}{3}$ triplets possible on n leaves. Call $T_F(n)$ the *full triplet set* on n leaves.

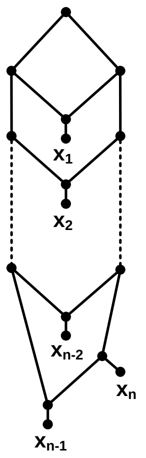


Figure 4.6: The level-$(n-1)$ network $N_F(n)$ is consistent with every triplet set on n leaves, and has the minimum number of arcs and vertices among all such networks.

Theorem 4.1. *For any triplet set T on n leaves there exists a level-$(n-1)$ network consistent with T.*

Proof. Let $n \geq 3$ and let $N_F(n)$ be the network in Figure 4.6. We will show that $N_F(n)$ is consistent with $T_F(n)$. Given that $T \subseteq T_F(n)$, the result follows.

First, consider triplets $x_h x_i | x_j \in T_F(n)$ with $h, i \neq n$. There exists a unique split vertex v below the left child of the root, from which there are two paths to x_h and x_i that have only v in common. On the other hand, there is a path from the root to x_j, via the right child of the root, making the network consistent with $x_h x_i | x_j$.

Secondly, consider triplets $x_i x_n | x_j \in T_F(n)$. There exists a unique split vertex v below the right child of the root, from which there are two paths to x_i and x_n that have only v in common. As there is also a path from the root to x_j via the left child of the root, the network is consistent with $x_i x_n | x_j$. □

Lemma 4.7. *Any network consistent with the full triplet set must be simple.*

Proof. Let $n \geq 3$ and let N be consistent with $T_F(n)$. If N is not simple then, by Lemma 4.3, it contains a non-trivial cut-arc $a = (u, v)$. If there is only one leaf below a, then N is not a valid network because it contains either a vertex with indegree and outdegree one or a biconnected component with only one outgoing arc, which we excluded by assumption. If all leaves are below a, then again N is not a valid network for the same reason. Hence there are leaves x and y below a and a leaf z not below a. This implies that the triplet $xz|y$ is not consistent with N, a contradiction. □

Let a *reticulation leaf* be defined as a leaf whose parent is a reticulation. Simple level-k networks have, as a consequence of their definition, at least one and at most k reticulation leaves ($k \geq 1$).

It was already shown by Jansson and Sung that there exists a network consistent with the full triplet set [Jan06b]. However, the network $N_F(n)$ is much simpler than the complicated sorting network proposed by Jansson and Sung. In fact, below it will be shown that $N_F(n)$ has minimum level and a minimum number of vertices and arcs over all networks consistent with the full triplet set.

Theorem 4.2. *The full triplet set on $n \geq 3$ leaves is not consistent with any level-k network with $k < n - 1$.*

Proof. The theorem holds for $n = 3$: any network consistent with $T_F(3)$ must be simple by Lemma 4.7, and any simple level-1 network on three leaves is consistent with only two triplets. Since there are three possible triplets on a set of three leaves, there is no level-1 network consistent with $T_F(3)$. Now assume that the theorem is not true in general and let N be a smallest counter example, i.e. N is a level-k network consistent with $T_F(n)$, $k < n-1$ and $n > 3$ is as small as possible. By Lemma 4.7, N must be a simple level-k network and thus contains a reticulation leaf x. Delete x and tidy up the resulting graph. This decreases the level of the network since the parent of x is a reticulation and gets removed when tidying up the graph. Thus, this yields a level-$(n - 3)$ network consistent with $T_F(n - 1)$, yielding a smaller counter example. $\square$

Lemma 4.8. *For $n \geq 3$, the network $N_F(n)$ has the minimum number of arcs and vertices over all networks consistent with the full triplet set $T_F(n)$.*

Proof. Let N_n be a network consistent with $T_F(n)$. By Lemma 4.7 and Theorem 4.2, N_n is simple and has level at least $n-1$. Then Lemma 4.4 yields that N_n has at least $2n+2(n-1)-1 = 4n-3$ vertices and $2n+3(n-1)-2 = 5(n-1)$ arcs. The proof is complete by noting that $N_F(n)$ has exactly $4n - 3$ vertices and $5(n - 1)$ arcs. $\square$

4.3.2 A Unique Level-k Network

In the construction and analysis of triplet methods, it is often important to know that a certain network is uniquely defined by a set of triplets. Characterising such networks is an important open problem. In this section we present a partial solution to this question by giving, for each k, a level-k network N^k that is unique in the sense that it is the only level-k network that is consistent with all triplets that are consistent with N^k. How useful this unique network is will be demonstrated in the next section, where we use it to show the intractability of constructing level-k networks from triplets.

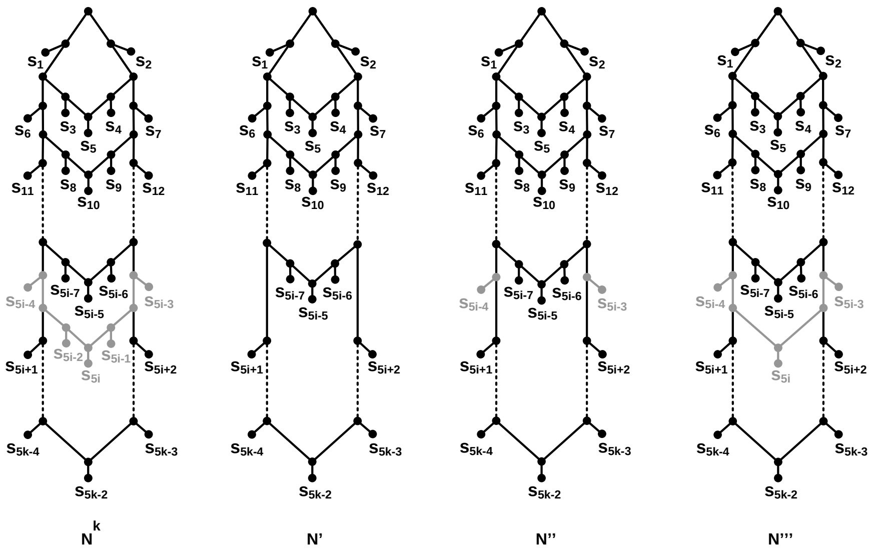


Figure 4.7: Removing leaves $s_{5i-4}, \dots, s_{5i}$ from N^k and tidying up the graph gives N'. Adding s_{5i-4} and s_{5i-3} to N' gives N''; adding s_{5i} to N'' gives N'''; subsequently adding s_{5i-2} and s_{5i-1} gives the original network N^k to the left again.

Let N^k be the network to the left in Figure 4.7 and let T^k be the set of triplets that are consistent with N^k. By *hanging leaves* $x_1, \dots, x_p$ on an arc (u, v) (for some $p \geq 1$) we mean replacing (u, v) by a path $u, w_1, \dots, w_p, v$ and adding arcs (w_i, x_i) for all $i = 1, \dots, p$. Recall that a *side* of a simple level-k generator is either an arc or a vertex with indegree 2 and outdegree 0. *Hanging a leaf* x on a side S of a simple level-k generator either means that it is hung on arc S (if S is an arc) or that an arc (S, x) is added (if S is a vertex with indegree 2 and outdegree 0).

Theorem 4.3. *For each $k \geq 2$, the network N^k is the unique level-k network consistent with T^k.*

Proof. Let R be the set of reticulation leaves of N^k, that is
$R = \{s_5, s_{10}, \dots, s_{5k-5}, s_{5k-2}\}$. We start by proving the following claims.

Claim 4.1. *Any level-k network consistent with T^k is a simple level-k network.*

Proof of Claim 4.1. First observe that all triplets over the leaves $R \cup \{s_{5k-4}\}$ are in T^k. Let N be a level-k network consistent with T^k. From Theorem 4.2 it follows that N is a strict level-k network. Now suppose for contradiction that N is not simple. Then by Lemma 4.3, N contains a non-trivial cut-arc a. Let $B \subsetneq L(N)$ be the set of leaves below a and let $A = L(N) \setminus B$. Because

a is non-trivial, B contains at least two leaves. For every two leaves x, y in B and every leaf z in A, there is only one triplet in T^k on leaves x, y, z that is consistent with N. However, for s_{5k-2} there are no two leaves x', y' such that there is only one triplet in T^k with leaves s_{5k-2}, x', y'. It follows that s_{5k-2} belongs to neither A nor B, a contradiction. □

Claim 4.2. *In any level-k network consistent with T^k, at least one of the leaves in R is a reticulation leaf.*

Proof of Claim 4.2. Let N be a network consistent with T^k. Recall that T^k contains all possible triplets over leaves $R \cup \{s_{5k-4}\}$. Theorem 4.2 says that any network consistent with all triplets over $R \cup \{s_{5k-4}\}$ cannot have level smaller than k, so N is a strict level-k network. By Claim 1, N is a simple level-k network and hence contains a reticulation leaf x. First suppose x does not belong to $R \cup \{s_{5k-4}\}$. Then removing x and tidying up the resulting graph yields a level-$(k-1)$ network consistent with all triplets over $R \cup \{s_{5k-4}\}$. A contradiction, thus N contains no leaves outside $R \cup \{s_{5k-4}\}$ as reticulation leaf. Symmetrically, no leaf outside $R \cup \{s_{5k-3}\}$ is a reticulation leaf of N. It follows that only leaves from R can be reticulation leaves of N, so x belongs to R. □

The proof is by induction on k. The induction basis for $k = 2$ is proven in the following claim. Network N^2, which is shown to be the only network consistent with T^2, is displayed in Figure 4.8 on the right.

Claim 4.3. *Network N^2 is the unique level-2 network consistent with T^2.*

Proof of Claim 4.3. Let N be a network consistent with T^2. From Claim 4.1 we know that N is a simple level-2 network. Recall the four simple level-2 generators, which have been repeated in Figure 4.8 on the left. We will show that N is of type $2c$ (i.e. that N can be obtained by hanging leaves from generator $2c$) and that s_5 and s_8 are the reticulation leaves.

We first argue that if N is of type $2b$ or $2c$ then s_5 and s_8 are reticulation leaves. Observe that whenever a leaf x is not reachable from any reticulation, there exists a unique path from the root to x. Hence, if three leaves x, y and z are all not reachable from any reticulation, then there is only one triplet on $\{x, y, z\}$ consistent with the network. It follows that if for any three leaves there are two triplets in the triplet set (a *double triplet*) then at least one of these leaves must be reachable from a reticulation. In networks of type $2b$ and $2c$ there are precisely two leaves that are reachable from a reticulation. Therefore, if N is of type $2b$ or $2c$ then it is clear that s_5 and s_8 have to be reticulation leaves, since they are the only two leaves that together appear in all double triplets in T^2.

The next step is to exclude that N is of type $2a$, $2b$ or $2d$. First consider networks of type $2b$ and observe that if for any three leaves there are three

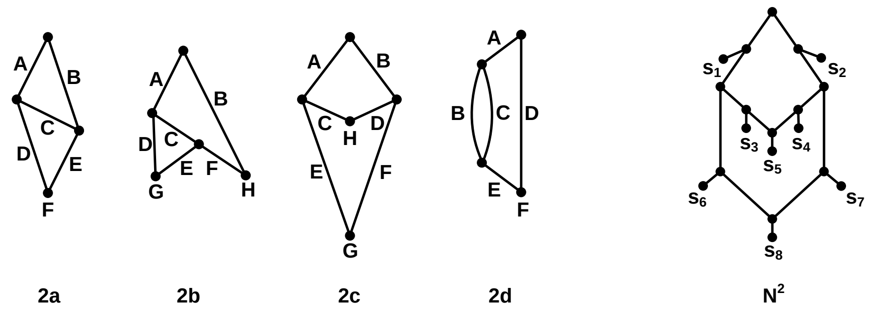


Figure 4.8: The four simple level-2 generators $2a$, $2b$, $2c$ and $2d$ and the network N^2 which is the unique network consistent with T^2.

triplets in the triplet set (a *triple triplet*), then these triplets can only be consistent with a network of type $2b$ if these leaves are on sides G, H and C. Because $\{s_5, s_8, x\}$ is a triple triplet in T^2 for every $x \neq s_5, s_8$, all leaves but s_5 and s_8 have to be on side C. But in this case it is not possible for both triplets $s_6s_8|s_7$ and $s_7s_8|s_6$ to be simultaneously consistent with the network, since s_5 is on side G or H and s_6 and s_7 are both on side C.

Now consider networks of type $2a$ and observe that such networks can only be consistent with triple triplets if its leaves are on sides C, E and F or on sides C, D and F. T^2 contains a triple triplet $\{s_5, s_8, x\}$ for all $x \neq s_5, s_8$ and thus there are only two possibilities for the network to look like. The first possibility is that s_5 and s_8 are on the sides F and D or the sides F and E and all other leaves are on side C. But in this case the triplets $s_6s_8|s_7$ and $s_7s_8|s_6$ cannot simultaneously be consistent with the network, since s_6 and s_7 are both on side C. The other possibility is that s_5 and s_8 are on the sides F and C and all other leaves are on the sides D and E. From the triplet $s_6s_3|s_8$ it follows that s_6 and s_3 are on the same side. But in that case $s_8s_6|s_3$ and $s_5s_3|s_6$ cannot simultaneously be consistent with the network. This excludes that N is of type $2a$.

Finally, consider networks of type $2d$. The only way for a triple triplet to be consistent with this type of network is to put the leaves in the triple triplet on the sides B, C and F. Since s_5 and s_8 are in a triple triplet in T^2 with every other leaf we know that s_5 and s_8 are on the sides F and (without loss of generality) B and all other leaves are on side C. But in this case it is not possible that triplets $s_6s_8|s_7$ and $s_7s_8|s_6$ are simultaneously consistent with the network, since s_6 and s_7 are both on side C.

We are now ready to prove the claim. From the above we know that N is of type $2c$ and that s_5 and s_8 are the two reticulation leaves. Since there is no triplet $s_1s_2|s_8$ we know that s_1 and s_2 are on different sides of the root (one on the left and one on the right side). Assume without loss of generality that

s_1 is on side A, C or E, s_2 is on side B, D or F, s_5 is on side G and s_8 on side H.

From the triplets $s_1s_3|s_8$ and $s_1s_6|s_8$ it follows that s_3 and s_6 are both on one of the sides A, C or E. And from the triplets $s_2s_4|s_8$ and $s_2s_7|s_8$ it follows that s_4 and s_7 are both on one of the sides B, D or F.

From the triplets $s_3s_5|s_6$ and $s_6s_8|s_3$ it now follows that s_3 is on side C and s_6 on side E. And from the triplet $s_3s_6|s_1$ then follows that s_1 is on side A. Similarly, from the triplets $s_4s_5|s_7$ and $s_7s_8|s_4$ it follows that s_4 is on side D and s_7 on side F. And from the triplet $s_4s_7|s_2$ then follows that s_2 is on side B. Therefore, $N = N^2$. □

We will now show the induction step. Let $k > 2$ and assume the theorem holds for all $k' = 2, \ldots, k-1$. In the induction step, we will show that any level-k network consistent with T^k and with reticulation leaf s_{5i} (for any $i \in \{1, \ldots, k-1\}$), equals the network N^k. The case that s_{5k-2} is a reticulation leaf is symmetric to the case that s_{5k-5} is a reticulation leaf. Since by Claim 4.2 at least one leaf from R must be a reticulation leaf, the theorem will follow.

Let N be a simple level-k network consistent with T^k and let $i \in \{1, \ldots, k-1\}$ be such that s_{5i} is a reticulation leaf in N. Let T' be the triplet set obtained from T^k by removing all triplets containing some leaf from $\{s_{5i-4}, \ldots, s_{5i}\}$, i.e. $T' = T^k|_{(L\setminus\{s_{5i-4},\ldots,s_{5i}\})}$. Then T' is consistent with network N', the second network from the left in Figure 4.7. Because T' equals the set of all triplets that are consistent with N' (which is a relabelling of N^{k-1}), by the induction hypothesis N' is the unique level-$(k-1)$ network consistent with T'.

Consider the network obtained from N by removing the leaves s_{5i-1}, s_{5i-2}, s_{5i}, s_{5i-3} and s_{5i-4} (in this order) from N and tidying up the resulting graph. This decreases the level of the network, since the parent of s_{5i} was a reticulation and gets removed when tidying up the graph. Hence this gives a level-$(k-1)$ network consistent with T', which by the induction hypothesis equals N'.

To show that N equals N^k, consider the network N' and apply the reverse of the operation that removed the leaves $s_{5i-1}, s_{5i-2}, s_{5i}, s_{5i-3}$ and s_{5i-4} from N. This process is illustrated in Figure 4.7, and we will show that the such obtained network will equal N^k. Process the leaves in reverse order, so add s_{5i-4} to N' first. Since N' has $k-1$ reticulation leaves and s_{5i} also has to become a reticulation leaf, s_{5i-4} must be a leaf below a split vertex. Hence s_{5i-4} is added to the network by hanging s_{5i-4} on some arc of N'. The same holds for s_{5i-3}. Since s_{5i} was a reticulation leaf in N, it is added to the network choosing two arcs $(u_1, v_1), (u_2, v_2)$, subdividing them into $(u_1, w_1), (w_1, v_1)$ and $(u_2, w_2), (w_2, v_2)$, respectively, and adding a new reticulation x and arcs $(w_1, x), (w_2, x), (x, s_{5i})$. Subsequently, s_{5i-2} and s_{5i-1} are added to the network by hanging them on arcs to be specified. It remains to determine which arcs to subdivide, as to add the leaves $s_{5i-1}, s_{5i-2}, s_{5i}, s_{5i-3}$ and s_{5i-4}.

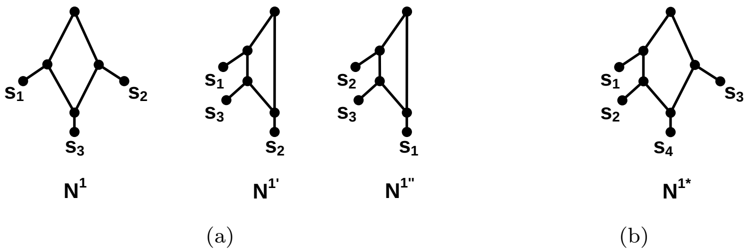


Figure 4.9: (a) The three networks $N^1, N^{1'}$ and $N^{1''}$ that are consistent with $T^1 = \{s_1s_3|s_2, s_2s_3|s_1\}$ and (b) network N^{1^*}, which is the unique network that is consistent with the set of triplets consistent with N^{1^*}.

First consider the case $i > 1$. Because $s_{5k-4}s_{5i+1}|s_{5i-4}$ and $s_{5i-4}s_{5i+1}|s_{5i-7}$ are triplets in T^k, it follows that s_{5i-4} is added to N' by hanging it on the arc entering the parent of s_{5i+1}. Symmetrically, s_{5i-3} is hung on the arc entering the parent of s_{5i+2}. This leads to network N'' in Figure 4.7. Next we discuss how to add s_{5i} to network N''. Triplets $s_{5i}s_{5i+1}|s_{5i-4}$ and $s_{5k-4}s_{5i+1}|s_{5i}$ force a subdivision of the arc between the parents of s_{5i-4} and s_{5i+1}. For symmetric reasons, also the arc between the parents of s_{5i+2} and s_{5i-3} has to be subdivided. So subdivide these arcs and make s_{5i} a reticulation leaf below them (as described in detail in the previous paragraph). This leads to the network N''' in Figure 4.7. Now s_{5i-2} and s_{5i-1} can only be added to the network by hanging them on the arcs entering the parent of s_{5i}, since $s_{5i+1}s_{5i-2}|s_{5i}, s_{5i}s_{5i-2}|s_{5i+1} \in T^k$ and $s_{5i+2}s_{5i-1}|s_{5i}, s_{5i}s_{5i-1}|s_{5i+2} \in T^k$. This leads to the leftmost network in Figure 4.7, which is the network N^k.

The case $i = 1$ is slightly different, since a leaf s_{5i-7} does not exist in this case. However, the triplets $s_{5k-4}s_6|s_1$ and $s_6s_1|s_7$ enforce that $s_1 = s_{5i-4}$ is added to N' by hanging it on the arc entering the parent of s_6. Symmetrically, $s_2 = s_{5i-3}$ must be hung on the arc entering the parent of s_7. The same arguments as in the case $i > 1$ show how to add the leaves $s_{5i}, s_{5i-1}, s_{5i-2}$. Also in this case we obtain the network N^k.

It follows that N equals N^k, completing the proof of Theorem 4.3. □

For level-1 the situation is slightly different since the network N^1 is not the only network consistent with T^1. Figure 4.9(a) shows the three networks that are consistent with T^1. However, there does exist a level-1 network that is unique in this sense. It is not too difficult to argue that the network N^{1^*} in Figure 4.9(b) is the only level-1 network that is consistent with all triplets that are consistent with N^{1^*}. For level-0, each tree is uniquely defined by the set of triplets consistent with it [Jan06b].

4.3.3 From Uniqueness to the Intractability of Constructing Level-k Networks from Triplets

In this section we show how to use the unique networks from the previous section in the complexity analysis of network reconstruction methods, based on triplets. We demonstrate this in two NP-hardness proofs. First, we show that it is NP-hard, for each $k \geq 1$, to decide whether a given triplet set is consistent with some level-k network. Secondly, we show that the maximisation version of this problem is NP-hard for each $k \geq 0$ even for dense triplet sets.

We start with the proof that it is NP-hard to construct a level-k network consistent with all input triplets. Hardness was already known for $k = 1$ [Jan06a]. Note that the NP-hardness for general k is not a consequence of the hardness for level one.

In the following proofs, we will often say we "hang" a leaf or "caterpillar" from a "side" of a simple level-k generator. A network is a *caterpillar* if deleting all leaves gives a directed path. Recall that in simple level-k generators, a *side* is either an arc or a vertex with indegree 2 and outdegree 0. *Hanging a caterpillar from arc* S_i means subdividing S_i and connecting the new vertex to the root of the caterpillar. Similarly defined is *hanging a caterpillar from a vertex with outdegree zero*, which gets connected to the root of the caterpillar. In addition, a leaf x *is on side* S_i if there exists a cut-arc (u, v) such that u is on a subdivision of S_i (if S_i is an arc) or u is a reticulation (if S_i is a reticulation), and there is a directed path from v to x (possibly $v = x$). A leaf x is said to *hang between* vertices w and q if there is a cut-arc (u, v) such that u is on a directed path from w to q and there is a directed path from v to x.

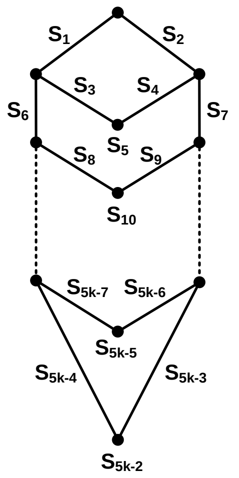
Theorem 4.4. *For each $k \geq 2$, it is NP-hard to decide whether for a triplet set T there exists some level-k network N consistent with T.*

Proof. Reduce from the following NP-hard problem [Gar79]:

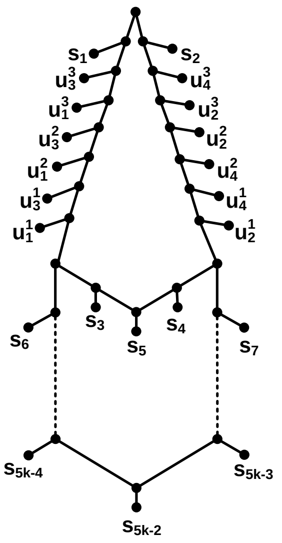
SET SPLITTING

Instance:	A set $U = \{u_1, \ldots, u_n\}$ and a collection $\mathcal{C} = \{C_1, \ldots, C_m\}$ of size-3 subsets of U.
Question:	Can U be partitioned into sets U_1 and U_2 (a set splitting) such that $C_j \not\subseteq U_1$ and $C_j \not\subseteq U_2$, for all $1 \leq j \leq m$?

From an instance $(U, \mathcal{C})$ of SET SPLITTING construct a set T of triplets as follows. Start with triplet set T^k (see previous Section), and for each set $C_j = \{u_a, u_b, u_c\} \in \mathcal{C}$ (with $1 \leq a < b < c \leq n$) add triplets $u_a^j s_5 | u_b^j$, $u_b^j s_5 | u_c^j$ and $u_c^j s_5 | u_a^j$. In addition, for every $u_i \in U$ and $1 \leq j \leq m$ add triplets $s_5 u_i^j | s_1$, $s_5 u_i^j | s_2$, $s_5 s_6 | u_i^j$, $s_5 s_7 | u_i^j$ and (if $j \neq m$) $u_i^j u_i^{j+1} | s_5$. This completes the construction of T. We will prove that T is consistent with some level-k network if and only if there exists a set splitting $\{U_1, U_2\}$ of $(U, \mathcal{C})$.



(a) The simple level-k generator G^k.



(b) The network N.

Figure 4.10: Auxiliary networks in the proof of Theorems 4.4 and 4.5.

First suppose that there exists a set splitting $\{U_1, U_2\}$ of $(U, \mathcal{C})$. Construct the network N by starting with the network N^k, which is obtained from the simple level-k generator G^k in Figure 4.10 by hanging a leaf s_i on each side S_i. For each element $u_i \in U_1$, hang all leaves $u_i^1, \ldots, u_i^m$ on side S_1 below the parent of s_1; for each element $u_i \in U_2$ hang all leaves $u_i^1, \ldots, u_i^m$ on side S_2 below the parent of s_2. To determine the order in which to put these leaves consider a set $C_j = \{u_a, u_b, u_c\} \in \mathcal{C}$. If u_a and u_b are in the same class of the partition, then put leaf u_a^j below u_b^j; if u_b and u_c are in the same class of the partition put u_b^j below u_c^j; and if u_a and u_c are in the same class put u_c^j below u_a^j. The rest of the ordering is arbitrary. It is easy to check that N is consistent with all triplets in T. For an example of this construction see the network to the right in Figure 4.10.

Conversely, suppose that T is consistent with some level-k network N. Since $T^k \subset T$, Theorem 4.3 says that N must be equal to N^k with the leaves not in $L(N^k)$ added. Triplets $s_5 u_i^j | s_1$ and $s_5 u_i^j | s_2$ imply that none of the leaves u_i^j can hang between the root and s_1, or between the root and s_2. Further, triplets $s_5 s_6 | u_i^j$ and $s_5 s_7 | u_i^j$ imply that u_i^j must be on either side S_1 or S_2. Triplets $u_i^j u_i^{j+1} | s_5$ yield that for each $1 \leq i \leq n$, all leaves $u_i^1, \ldots, u_i^m$ have to hang on the same side. For $h \in \{1, 2\}$, let U_h be the set of elements $u_i \in U$ for which all leaves $u_i^1, \ldots, u_i^m$ hang on side S_h. It remains to prove that (U_1, U_2) is a set splitting of $(U, \mathcal{C})$. Consider a set $C_j = \{u_a, u_b, u_c\}$ and suppose for contradiction that $u_a, u_b, u_c \in U_h$ for some $h \in \{1, 2\}$, meaning that all leaves u_a^j, u_b^j, u_c^j hang on side S_h. This is impossible, as T contains triplets $u_a^j s_5|u_b^j$, $u_b^j s_5|u_c^j$, $u_c^j s_5|u_a^j$. □

For dense triplet sets, it can be decided in polynomial time whether there exists a level-1 [Jan06a] or level-2 (Section 4.6) network consistent with all input triplets. Using the uniqueness result from the previous section, we will prove that the maximisation versions of these problems are NP-hard, even for dense triplet sets and for all $k \geq 0$.

MaxCL-k-Dense

Input: A dense triplet set T.

Output: A level-k network consistent with the maximum number of triplets in T that any level-k network is consistent with.

Theorem 4.5. *The problem* MaxCL-k-Dense *is NP-hard, for all* $k \geq 0$.

Proof. Reduce from the following NP-hard problem [Alo06; Cha07]. A *tournament* is a directed graph in which for each pair of vertices u and v there is either an arc (u, v) or an arc (v, u).

Feedback Arc Set in Tournaments (FAST)

Instance: A tournament $G = (V, A)$ and an integer $q \in \mathbb{N}$.

Question: Is there a set $A' \subseteq A$ of q arcs (a feedback arc set) such that $G' = (V, A \setminus A')$ is acyclic?

For $k = 0$, we imitate the reduction of the non-dense case [Bry97; Jan01; Wu04]. The difference is that the constructed instance of MaxCL-0-Dense contains more triplets, to become dense. Given an instance $G = (V, A)$ and $q \in \mathbb{N}$ of FAST, construct an instance T of MaxCL-0-Dense as follows. Introduce a leaf $x \notin V$ and for each arc $(z, y) \in A$, add a triplet $xy|z$ to T. In addition, for each combination of three leaves $v_1, v_2, v_3 \in V$ (thus $v_1, v_2, v_3 \neq x$), add all three triplets $v_1v_2|v_3$, $v_1v_3|v_2$ and $v_2v_3|v_1$ to T. Instead of providing a full proof we will now only describe the differences to the proof for the non-dense case [Bry97; Wu04] and give the intuition behind that proof. The differences with the reduction of Wu [Wu04] are that (1) we reduce from FAST instead of FAS and that (2) we add all triplets containing three leaves from V. The combination of these two modifications makes the instances dense. The extra triplets do not change the reduction since any level-0 network is consistent with exactly one triplet for every combination of three leaves. The intuition of the reduction is as follows: the vertices of an acyclic graph can be uniquely labelled such that arcs point only from vertices with higher label to vertices with lower label. In a phylogenetic tree, this ordering of the vertices corresponds to an ordering of the leaves on the unique path from the tree root to leaf x. Along the lines of the proof for the non-dense case [Wu04], it can be argued that G contains a feedback arc set of size q if and only if there exists a tree consistent with $|T| - q - 2\binom{|V|}{3}$ triplets from T. This completes the proof that MaxCL-0 is NP-hard for dense triplet sets.

For $k \geq 2$, use a similar reduction but start from the simple level-k generator G^k in Figure 4.10(a). Theorem 4.3 tells us that a network N^k obtained by hanging a leaf from each side of G^k is the unique level-k network consistent with T^k; the set of triplets consistent with N^k.

Given a tournament $G = (V, A)$ and integer $q \in \mathbb{N}$, construct a corresponding instance T of MaxCL-k-Dense as follows. First construct a network N' from G^k. From each side S_i of G^k hang a caterpillar with leaves $S_i^1, \ldots, S_i^p$, with $p = 2(q + 2\binom{|V|}{3}) + 1$. The intuition is that p is large enough to force a specific structure of the networks consistent with many triplets in T. For simplicity denote S_{5k-2}^1 by x. Hang $|V|$ leaves on side S_{5k-4}, distinctly labelled by the vertices of V, below the root of the caterpillar, in arbitrary order. This gives the network N'. For an example, see the network on the right in Figure 4.11. Let T' be the set of triplets consistent with N', except for triplets $ab|c$ with $a, c \in V$ and $b \notin V$. For each arc $(z, y) \in A$, add a triplet $xy|z$ to T', informally encoding the arc (z, y) as a constraint "z hangs between the root of the caterpillar and y". Finally, for each 3-set of vertices from V add all three triplets over the three leaves labelled by the vertices, that are not yet present. Denote the resulting (dense) triplet set by T, which forms an instance of MaxCL-k-Dense. We will show that there exists a level-k network N consistent with $|T| - q - 2\binom{|V|}{3}$ triplets from T if and only there exists a feedback arc set A' of size q.

First suppose G has a feedback arc set A' of size q. Thus the graph $G' = (V, A \setminus A')$ is acyclic, and each vertex $v \in V$ can receive a label $f(v)$ such that there are no arcs $(z, y) \in A \setminus A'$ with $f(y) \geq f(z)$. Construct the network N from N' by rearranging the leaves from V by sorting them with respect to their labels such that the highest leaf has the largest label. For any arc $(z, y) \in A \setminus A'$ it holds that $f(y) < f(z)$ and hence the triplet $xy|z$ is consistent with N. For every vertex pair $\{z, y\}$, the triplet $yz|x$ is consistent with N. For each combination of three leaves from V there is exactly one triplet over these leaves consistent with N. It follows that the only triplets in T that are not consistent with N are (1) the triplets corresponding to the arcs in A', and (2) exactly two-thirds of the triplets that have only leaves in V. That means that in total $|T| - q - 2\binom{|V|}{3}$ triplets from T are consistent with N.

For the converse, suppose there exists some level-k network N consistent with $|T| - q - 2\binom{|V|}{3}$ triplets from T. For each $1 \leq j \leq p$, there exists a unique network with leaf set $L_j = \{S_i^j \mid 1 \leq i \leq 5k - 2\}$ that is consistent with all triplets from $T_j = T|L_j$. There are at most $q + 2\binom{|V|}{3}$ triplets not consistent with N, and the sets T_j are pairwise disjoint, so at least one of the sets L_j is placed on a simple level-k network of type G^k. Take any i and observe that for each j such that s_i^j is not on side S_i of N, there exists a triplet $t \in T$, with leaves $L(t) = \{s_i^j, \ell_1, \ell_2\}$, that is not consistent with N and with $\ell_1, \ell_2 \notin \{s_i^1, \ldots, s_i^p\}$. If there would be more than $q + 2\binom{|V|}{3}$ such j then there

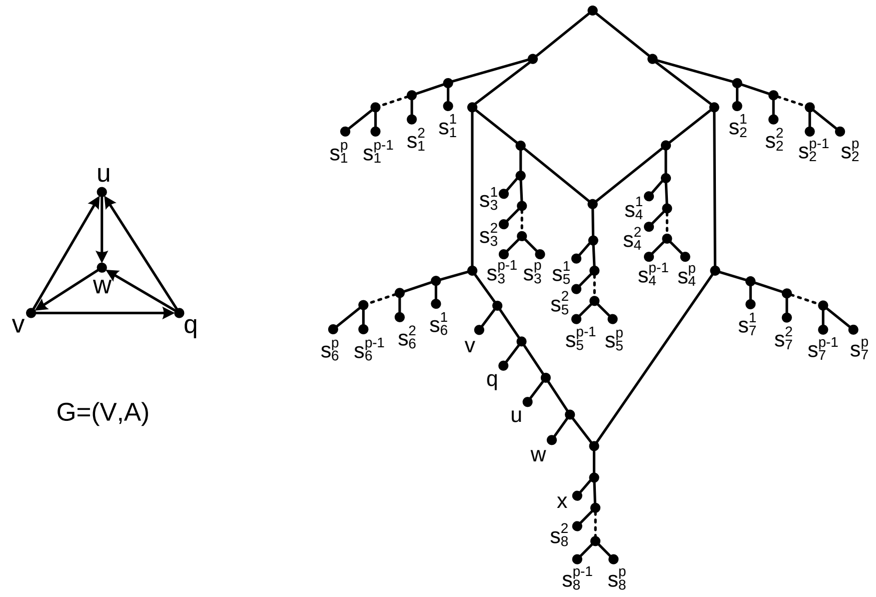

u
w
v
q
G=(V,A)
S_1^p S_1^{p-1} S_1^2 S_1^1
S_2^1 S_2^2 S_2^{p-1} S_2^p
S_3^1 S_3^2 S_3^{p-1} S_3^p
S_4^1 S_4^2 S_4^{p-1} S_4^p
S_5^1 S_5^2 S_5^{p-1} S_5^p
S_6^p S_6^{p-1} S_6^2 S_6^1
S_7^1 S_7^2 S_7^{p-1} S_7^p
v
q
u
w
x
S_8^2
S_8^{p-1} S_8^p

Figure 4.11: An example input $G = (V, A)$ of FAST on the left and the network N constructed in the proof of Theorem 4.5, for $k = 2$, to the right.

would be more than $q + 2\binom{|V|}{3}$ distinct triplets from T not consistent with N. Hence for each i there are at least $p' = q + 2\binom{|V|}{3} + 1$ indices j such that S_i^j is on side S_i. Let $L_1^*, \dots, L_{p'}^*$ be pairwise disjoint sets each containing exactly one leaf S_i^j that is on side S_i, for each $1 \leq i \leq 5k - 2$.

The next claim is that all leaves labelled by vertices from V have to be on side S_{5k-4}, below the root of the caterpillar. Suppose for contradiction this were not the case for some leaf $v \in V$. Then for each of the leaf sets $L_1^* \cup \{v\}, \dots, L_{p'}^* \cup \{v\}$ there exists a triplet in T not consistent with N. Since the sets $L_1^*, \dots, L_{p'}^*$ are pairwise disjoint and $p' > q + 2\binom{|V|}{3}$, we obtain a contradiction.

Since the leaves corresponding to vertices from V all hang on the same side S_{5k-4}, they can be uniquely labelled by their order on side S_{5k-4}, such that the highest leaf has the largest label. If some leaves are below the same cut-arc, they receive the same label. Let A' be the set of arcs (z, y) corresponding to the triplets $xy|z$ that are not consistent with N, and for every $v \in V$ let $f(v)$ be the label of the leaf corresponding to v. Then the graph $G' = (V, A \setminus A')$ is acyclic, because all arcs $(z, y) \in A \setminus A'$ satisfy the relation $f(y) < f(z)$.

An example for $k = 2$ is displayed in Figure 4.11. The graph on the left is an example instance $G = (V, A)$ of FAST. The arcs of G are encoded as triplets $xw|u$, $xw|q$, $xu|v$, $xv|w$, $xu|q$ and $xq|v$. The network N to the right is consistent with all these triplets except $xv|w$. The arc (w, v) is indeed a feedback arc set of the graph G. Other triplets in T enforce this specific level-2 network N and make T dense.

For $k = 1$ the same reduction as for $k \geq 2$ works, when hanging two caterpillars from side S_1. □

4.4 Constructing Level-1 Networks in Exponential Time

Given the intractability results from the previous section for constructing networks consistent with a maximum number of input triplets, there is no hope (unless P = NP) for algorithms solving these problems exactly and in polynomial time. In this section we consider superpolynomial-time exact algorithms.

Wu described an exact algorithm [Wu04] that finds a tree consistent with a maximum number of input triplets in $O(3^n(n^2 + m))$ time, with m the number of triplets and n the number of leaves. We extend this approach for reconstructing evolutions that are not tree-like, but where reticulation cycles are disjoint. We do this by describing an exact algorithm that runs in $O(m4^n)$ time and solves the MaxCL-1 problem, which is NP-hard by Theorem 4.5.

Note that the problem MaxCL-1 does not just ask if there exists a level-1 network consistent with *all* input triplets; it asks us to find a level-1 network that is consistent with a maximum number of them. Hence an algorithm for MaxCL-1 always outputs a solution, no matter how bad the data is the algorithm is confronted with. This contrasts with algorithms that only find a solution if a network exists that is consistent with all triplets of a (dense) input (Sections 4.6-4.8 and [Aho81; Jan06a; Jan06b]). The algorithm described in this section is also more powerful in that it also works for non-dense triplet sets. It can thus be used even if for some combinations of three taxa it is difficult to find the right triplet, which is likely to be the case in practice. The algorithm can also be used for the weighted version of the problem. In addition, it can also be used to choose, among all level-1 networks consistent with a maximum number of input triplets, one with a minimum number of reticulations. However, its exponential running time means that it can only be used for a relatively small number of leaves at a time.

The intuition behind our algorithm is the following. There are three different shapes possible for an optimal network. Either the arcs leaving the root are cut-arcs, like in Figure 4.14(b), or the root is part of a cycle, which can be "skew" like the cycle in Figure 4.15(a) or "non-skew" like in Figure 4.15(b). We construct a candidate network of each type separately. Given the tripartition (X', Y', Z') (with possibly $Z = \emptyset$) of the leaves indicated in the figures, it turns out to be possible to reconstruct the optimal network by merging optimal smaller networks for X', Y', $X' \cup Z'$ and $Y' \cup Z'$. The three candidate networks for a partition π are denoted N^1_π, N^2_π, N^3_π. For partitions with $Z = \emptyset$, only one candidate network N^2_π is created.

Critical is that the smaller networks for $X' \cup Z'$ and $Y' \cup Z'$ must be such that merging two networks does not create a biconnected component with more than one reticulation. To achieve this, we introduce the notion of "non-cycle-reachable"-arc, or ncr-arc for short. An arc $a = (u, v)$ is an *ncr-arc* if u is not in a cycle and no ancestor of u is in a cycle. Note that this implies that also u is not in a cycle. In addition, for any arc $a = (u, v)$ write $R[a]$ to denote the set of leaves below v. Use $f(N)$ to denote the number of triplets in T consistent with N and $g(N, Z)$ to denote the number of triplets in T that are consistent with N and that are not of the form $xy|z$ with $z \in Z$ and $x, y \notin Z$. It will become clear later that the definition of g ensures that merging networks that are optimal with respect to g leads to networks optimal with respect to f.

A description of the algorithm, called ELONA (Exact Level One Network Algorithm) is displayed in Algorithm 1. As will be shown below, for $L' \subseteq L$, the computed network $N(L')$ is a network maximising $f(N)$ over all networks with $L(N) = L'$. In addition, for $Z \subset L' \subseteq L$, the computed network $N^2(L', Z)$ will be shown to maximise $g(N, Z)$ over all level-1 networks N with $L(N) = L'$ that contain an ncr-arc a with $Z = R[a]$. If there are various such networks then the algorithm picks one arbitrarily.

Because the arcs $a = (u, v)$ and $a' = (u', v')$ in lines 9 and 11 are ncr-arcs, we know that (in $N^2(X \cup Z, Z)$ and $N^2(Y \cup Z, Z)$) neither u, nor u', nor any of their ancestors is contained in a cycle. It follows that the newly created cycles do not overlap with any of the original cycles and hence that the constructed networks are indeed level-1 networks. It now also becomes intuitively clear why networks $N^2(X \cup Z, Z)$ and $N^2(Y \cup Z, Z)$ are used that are optimal with respect to $g(\cdot, Z)$ (rather than $f(\cdot)$). Consider for example the construction of N^1_π from $N^2(X \cup Z, Z)$ and $N^2(Y \cup Z, Z)$ in Figure 4.12. It does not matter whether a triplet $x_1x_2|z$ with $x_1, x_2 \in X$ and $z \in Z$ is consistent with $N^2(X \cup Z, Z)$ or not, since such a triplet will be consistent with N^1_π anyhow. Similarly, it does not matter whether triplets of the form $y_1y_2|z$ with $y_1, y_2 \in Y$ and $z \in Z$ are consistent with $N^2(Y \cup Z, Z)$, since the creation of a new cycle as in Figures 4.12 and 4.13 makes all triplets of this form consistent with the network.

Algorithm 1 ELONA (Exact Level One Network Algorithm)

```
1: for each $\ell \in L$ do
2:   Let $N(\ell)$ be a single leaf labelled $\ell$.
3: for $i = 2 \ldots n$ do
4:   for each $L' \subseteq L$ of cardinality $i$ do
5:     for each tripartition $\pi = (X, Y, Z)$ of $L'$ with $X, Y \neq \emptyset$ do
6:       if $Z = \emptyset$ then
7:         Combine $N(X)$ and $N(Y)$ into a new network $N^2_\pi$ by adding a new root and connecting it to the roots of $N(X)$ and $N(Y)$.
8:       else
9:         Construct first candidate. Combine networks $N^2(X \cup Z, Z)$ and $N^2(Y \cup Z, Z)$ into a new network $N^1_\pi$ by creating a "non-skew" cycle as follows. Add a new root and connect it to the roots of $N^2(X \cup Z, Z)$ and $N^2(Y \cup Z, Z)$. Let $a = (u, v)$ and $a' = (u', v')$ be the (unique) ncr-arcs such that $Z = R[a]$ in $N^2(X \cup Z, Z)$ and $Z = R[a']$ in $N^2(Y \cup Z, Z)$. Subdivide $a$ into $(u, w)$ and $(w, v)$, delete $v'$ and all arcs and vertices reachable from $v'$, and add an arc $(u', w)$ (see Figure 4.12).
10:        Construct second candidate. Combine networks $N(X)$ and $N^2(Y \cup Z, Z)$ into a new network $N^2_\pi$ by adding a new root and connecting it to the roots of $N(X)$ and $N^2(Y \cup Z, Z)$.
11:        Construct third candidate. Create $N^3_\pi$ from $N^2_\pi$ by creating a "skew" cycle as follows: let $a = (u, v)$ be the (unique) ncr-arc with $Z = R[a]$. Subdivide $a$ into $(u, w)$ and $(w, v)$, add a new root and connect it to the old root and to $w$ (see Figure 4.13).
12:    Let $N(L')$ be a network that maximises $f(\cdot)$ over all computed networks $N^1_\pi, N^2_\pi$ and $N^3_\pi$ over all tripartitions $\pi$ of $L'$.
13:    for each $\bar{Z} \subset L'$ do
14:      Let $N^2(L', \bar{Z})$ be a network that maximises $g(N^2_\pi, \bar{Z})$ over all tripartitions $\pi = (X, Y, Z)$ of $L'$ with $Z = \bar{Z}$.
15: output $N(L)$
```

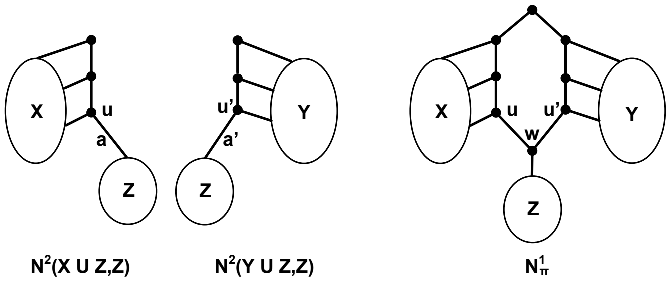


Figure 4.12: Construction of N^1_π from $N^2(X \cup Z, Z)$ and $N^2(Y \cup Z, Z)$.

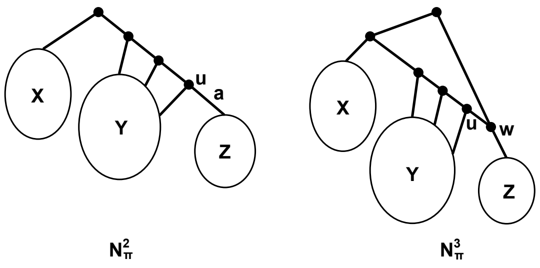


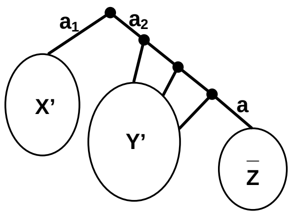
Figure 4.13: Construction of N^3_π from N^2_π.

Correctness of the algorithm is shown by induction on i. The induction basis for $i = 1$ is trivial. For the induction step, consider a leaf set L' with $|L'| > 1$ and assume that the algorithm works correctly for all leaf sets of smaller size. Thus, for each tripartition (X, Y, Z) of L' with $X, Y \neq \emptyset$ the algorithm has correctly computed networks $N(X)$ and $N(Y)$ that are optimal with respect to $f(\cdot)$ and networks $N^2(X \cup Z, Z)$ and $N^2(Y \cup Z, Z)$ that are optimal with respect to $g(\cdot, Z)$ (over all level-1 networks containing an ncr-arc a with $Z = R[a]$). The following lemma shows that the computed network $N^2(L', \bar{Z})$ is then also optimal with respect to $g(\cdot, \bar{Z})$ (over all level-1 networks containing an ncr-arc a with $\bar{Z} = R[a]$), for each $\bar{Z} \subset L'$. Subsequently, Lemma 4.10 shows that the computed network $N(L')$ is optimal with respect to $f(\cdot)$.

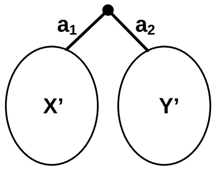
Lemma 4.9. *For every $\bar{Z} \neq \emptyset$, the network $N^2(L', \bar{Z})$ computed by the algorithm maximises $g(N, \bar{Z})$ over all level-1 networks N with $L(N) = L'$ that contain an ncr-arc a with $\bar{Z} = R[a]$.*

Proof. Let N' be any network with $L(N') = L'$ and containing an ncr-arc a' such that $\bar{Z} = R[a']$. We show that $g(N', \bar{Z}) \leq g(N^2(L', \bar{Z}), \bar{Z})$. The root of N' is not in a cycle since it is an ancestor of the ncr-arc a'. Let a_1 and a_2 be the two cut-arcs leaving the root such that the leaves in $\bar{Z}$ are reachable from a_2. Let $X' = R[a_1]$ and $Y' = R[a_2] \setminus \bar{Z}$, see Figure 4.14(a). Because $N^2(L', \bar{Z})$ maximises $g(N^2_\pi, \bar{Z})$ over all tripartitions $\pi = (X, Y, Z)$ of L' with $Z = \bar{Z}$, we have $g(N^2(L', \bar{Z}), \bar{Z}) \geq g(N^2_{(X',Y',\bar{Z})}, \bar{Z})$. Write $N^{2\prime}$ as short for $N^2_{(X',Y',\bar{Z})}$. Compare triplets consistent with N', with those consistent with $N^{2\prime}$.

- There are at least as many triplets in $T|_{X'}$ consistent with $N^{2\prime}$ as with N', because $N(X')$ is a subgraph of $N^{2\prime}$ and by induction $N(X')$ maximises $f(N)$ over all networks N with $L(N) = X'$.
- There are at least as many triplets in $T|_{(Y' \cup \bar{Z})}$ that are not of the form $y_1 y_2 | z$ for $y_1, y_2 \in Y'$ and $z \in \bar{Z}$ that are consistent with $N^{2\prime}$



(a) In the proof of Lemma 4.9.



(b) The first case in the proof of Lemma 4.10.

Figure 4.14: Networks N' in the proofs of Lemmas 4.9 and 4.10.

as with N', because $N^2(Y' \cup \bar{Z}, \bar{Z})$ is a subgraph of $N^{2\prime}$ and by induction $N^2(Y' \cup \bar{Z}, \bar{Z})$ maximises $g(N, \bar{Z})$ over all networks N with $L(N) = Y' \cup \bar{Z}$ that contain an ncr-arc a with $\bar{Z} = R[a]$.

- All triplets of the form $bc|d$ with $b, c \in X'$, $d \in Y' \cup \bar{Z}$ or $b, c \in Y' \cup \bar{Z}$, $d \in X'$ are consistent with both $N^{2\prime}$ and N'.
- All triplets of the form $bc|d$ with $b, d \in X'$, $c \in Y' \cup \bar{Z}$ or $b, d \in Y' \cup \bar{Z}$, $c \in X'$ are consistent with neither $N^{2\prime}$ nor N'.

Thus $g(N', \bar{Z}) \leq g(N^{2\prime}, \bar{Z}) \leq g(N^2(L', \bar{Z}), \bar{Z})$. □

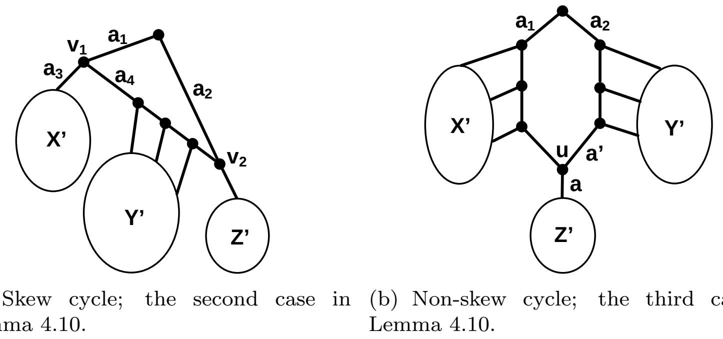
Lemma 4.10. *The computed network $N(L')$ maximises $f(N)$ over all level-1 networks N with $L(N) = L'$.*

Proof. For contradiction, suppose that some network $N' \neq N(L')$ with $L(N') = L'$ is consistent with more triplets in T than $N(L')$. Distinguish three cases, depending on the shape of N'.

The first case is that the two arcs leaving the root of N' are cut-arcs a_1 and a_2. Let $X' = R[a_1]$, $Y' = R[a_2]$ and $Z' = \emptyset$, see Figure 4.14(b), and compare N' to $N^2_{(X',Y',Z')}$.

The latter network is consistent with at least as many triplets from $T|_{X'}$ because it contains $N(X')$ as a subnetwork, and by induction $N(X')$ maximises $f(N)$ over all networks N with $L(N) = X'$. Similarly, $N^2_{(X',Y',Z')}$ is consistent with as least as many triplets from $T|_{Y'}$ as N'. All other triplets are either consistent with both or with none of these networks. Hence, $N^2_{(X',Y',Z')}$ is consistent with at least as many triplets as N'. Because $N(L')$ is consistent with at least as many triplets as $N^2_{(X',Y',Z')}$, it follows that $N(L')$ is also consistent with at least as many triplets as N'; a contradiction.

The second case is that one child of the root of N' is a reticulation. Let $a_1 = (r, v_1)$ and $a_2 = (r, v_2)$ be the two arcs leaving the root of N' and suppose that v_2 is a reticulation. Let a_3 and a_4 be the two arcs leaving v_1. Because N'



(a) Skew cycle; the second case in Lemma 4.10.
(b) Non-skew cycle; the third case in Lemma 4.10.

Figure 4.15: Networks N' in the proof of Lemma 4.10.

is a level-1 network, one of a_3 and a_4 is a cut-arc, say a_3. Let $X' = R[a_3]$, $Y' = R[a_4] \setminus R[a_2]$ and $Z' = R[a_2]$, see Figure 4.15(a). Compare the networks N' and $N^3_{(X',Y',Z')}$ with respect to the number of triplets in T these networks are consistent with. First, consider triplets in $T|_{X'}$: Network $N^3_{(X',Y',Z')}$ is consistent with at least as many of these as N', because it contains $N(X')$ as a subgraph. Secondly, consider triplets in $T|_{(Y' \cup Z')}$ that are not of the form $y_1y_2|z$ for $y_1, y_2 \in Y'$ and $z \in Z'$. We will show that $N^3_{(X',Y',Z')}$ is consistent with at least as many of these triplets as N'. Recall that $N^3_{(X',Y',Z')}$ contains a subdivision of $N^2(Y' \cup Z', Z')$, which maximises $g(N, Z')$ over all networks with $L(N) = Y' \cup Z'$ containing an ncr-arc a with $Z' = R[a]$. The network N' does not contain such an ncr-arc, but we will modify it to a network that does contain such an ncr-arc and is consistent with the same number of the considered triplets. Let N'' be the result of removing the arc a_2 from N' (and tidying up the resulting graph). Observe that all triplets in $T|_{(Y' \cup Z')}$ that are consistent with N' and are not of the form $y_1y_2|z$ with $y_1, y_2 \in Y'$ and $z \in Z'$ are also consistent with N''. So it follows that $N^3_{(X',Y',Z')}$ is consistent with at least as many of these triplets as N'. All other triplets are either consistent with both $N^3_{(X',Y',Z')}$ and N' or with none, since both networks have the structure from Figure 4.15(a): only the internal structure inside X', Y' and Z' might be different in the two networks. Hence $N^3_{(X',Y',Z')}$ is consistent with at least as many triplets as N'. Because $N(L')$ is consistent with at least as many triplets as $N^3_{(X',Y',Z')}$ it follows that $N(L')$ is also consistent with at least as many triplets as N'; a contradiction.

The last case is that the two arcs a_1 and a_2 leaving the root of N' are not cut-arcs and are also not leading to reticulations. Let $X' = R[a_1] \setminus R[a_2]$, $Y' = R[a_2] \setminus R[a_1]$ and $Z' = R[a_1] \cap R[a_2]$, see Figure 4.15(b). Compare the networks N' and $N^1_{(X',Y',Z')}$, with respect to the number of triplets in T these networks are consistent with. First, consider triplets in $T|_{(X' \cup Z')}$ that are not of the form $x_1x_2|z$ for $x_1, x_2 \in X'$ and $z \in Z'$. We will show that $N^1_{(X',Y',Z')}$ is consistent with at least as many of these triplets as N'. Recall that $N^1_{(X',Y',Z')}$ contains a subdivision of $N^2(X' \cup Z', Z')$, which maximises $g(N, Z')$ over all networks with $L(N) = X' \cup Z'$ containing an ncr-arc a with $Z' = R[a]$. The network N' does not contain such an ncr-arc, but we will modify it to a network that does contain such an ncr-arc and is consistent with the same number of the considered triplets. Let $a = (u, v)$ be the cut-arc in N' with $Z' = R[a]$, and let a' be the arc that leads to u and is reachable from a_2. Let N'' be the result of removing arc a' from N' (and tidying up the resulting graph). Now N'' is consistent with the same number of the considered triplets as N', and so it follows that $N^1_{(X',Y',Z')}$ is consistent with at least as many of the considered triplets as N'. In a similar way, it follows that $N^1_{(X',Y',Z')}$ is consistent with as least as many triplets in $T|_{(Y' \cup Z')}$ that are not of the form $y_1y_2|z$ for $y_1, y_2 \in Y'$ and $z \in Z'$). All other triplets are either consistent with both networks or with none. Hence $N^1_{(X',Y',Z')}$ is consistent with at least as many triplets as N'. Because $N(L')$ is consistent with at least as many triplets as $N^1_{(X',Y',Z')}$ it follows that $N(L')$ is also consistent with at least as many triplets as N'; a contradiction. □

Theorem 4.6. *Given a set T of m triplets over n leaves, a level-1 network consistent with a maximum number of triplets in T can be constructed in $O(m4^n)$ time and $O(n3^n)$ space.*

Proof. Correctness of the algorithm follows from the above. To achieve a small polynomial factor in the complexity, we use dynamic programming to compute the optimal value of the solution as well as the partitions we have to choose in each step. Then a traceback algorithm constructs a network consistent with the maximum number of triplets. To be precise, the dynamic programming algorithm finds, for all $L' \subseteq L$, the maximum number $\hat{f}(L')$ of triplets in T consistent with a level-1 network with leaves $L' \subseteq L$. It also computes, for all $\bar{Z} \subset L'$, the maximum value $\hat{g}(L', \bar{Z})$ of $g(N, \bar{Z})$ over all level-1 networks N with leaves L' that contain an ncr-arc a with $\bar{Z} = R[a]$. The algorithm loops through all the subsets $L' \subseteq L$ from small to large and considers all tripartitions $\pi = (X, Y, Z)$ of L'. For each such partition, the values $\hat{f}(X)$, $\hat{f}(Y)$, $\hat{f}(Z)$, $\hat{g}(X \cup Z, Z)$ and $\hat{g}(Y \cup Z, Z)$ are readily available from previous iterations. To compute the values $\hat{f}(L')$ and $\hat{g}(L', Z)$ it only remains to count certain triplets in T, whose consistency with a network only depends on the tripartition (X, Y, Z) and the network type (N^1_π, N^2_π or N^3_π). This can be done by first checking membership of X, Y and Z for each leaf in L' (in $O(n)$ time) and then looping through all triplets only once. Hence this counting can be done in $O(n + m) = O(m)$ time. The algorithm's overall running time is thus bounded by $O(m) \sum_{\ell=1}^{n} \binom{n}{\ell} O\left(3^\ell\right) = O(m4^n)$.

For each leaf set $L' \subseteq L$, store the optimal tripartition and the optimal type of network (N^1_π, N^2_π or N^3_π). In addition, store an optimal bipartition for all $L' \subseteq L$ and $\bar{Z} \subset L'$. This yields a total space complexity of $O(n3^n)$.

Once the values $\hat{f}(L')$ and $\hat{g}(L', \bar{Z})$ have been computed and all optimal tripartitions and bipartitions have been stored, a level-1 network N consistent with $\hat{f}(L)$ many triplets can be constructed by traceback, in polynomial time. □

4.5 Constructing Simple Level-k Networks

This section shows how simple level-k networks can be build from dense triplet sets in polynomial time, for each fixed k. This is done by Algorithm SL-k, which runs in time $O(|T|^{k+1})$. Subsequently it is shown that for level-2 the running time can be optimised to $O(|T|^{\frac{8}{3}})$, which will be used in the algorithm to construct general level-2 networks in the next section.

Let N be a network with at least one reticulation, and let v be the child of a reticulation in N. If v has no reticulation as a descendant, then we call the subnetwork rooted at v a *Tree hanging Below a Reticulation (TBR)*. We additionally introduce the notion of the *empty* TBR, which corresponds to the situation when a reticulation has no outgoing arcs. This cannot happen in a normal network but as explained shortly it will prove a useful abstraction.

Observation 4.1. *Every network N containing a reticulation contains at least one TBR.*

Proof. Suppose this is not true. Let v be the child of a reticulation in N maximising the longest path from the root to v. If v is not the root of a TBR then there must exist some vertex $v' \neq v$ which is a child of a reticulation and which is a descendent of v. But then the longest path from the root to v' is greater than to v, contradiction. □

Note that, because a TBR is (as a consequence of its definition) below a cut-arc, there exists an SN-set S of T such that $T|S$ is consistent with (only) the TBR. An SN-set S such that $T|S$ is consistent with a tree, we call a *CandidateTBR* SN-Set. Every TBR of N corresponds to some CandidateTBR SN-Set of T, but the opposite is not necessarily true. For example, a singleton SN-set is a CandidateTBR SN-Set, but it might not be the child of a reticulation in N.

We abuse definitions slightly by defining the empty CandidateTBR SN-Set, which will correspond to the empty TBR. (This is abusive because the empty set is not an SN-set.) Furthermore we define that every triplet set T has an empty CandidateTBR SN-Set.

Observation 4.2. *Let T be a dense set of triplets on n leaves. There are at most $O(n)$ CandidateTBR SN-sets. All such sets, and the tree that each such set represents, can be found in total time $O(n^3)$.*

Proof. First we construct the SN-tree (see Section 4.2) in $O(n^3)$ time [Jan06a]. There is a bijection between the SN-sets of T and the vertices of the SN-tree. (In the SN-tree, the children of an SN-set S are the maximal SN-sets of $T|S$.) Observe that a vertex of the SN-tree is a CandidateTBR SN-set if and only if it is a singleton SN-set *or* it has in total two children and both are CandidateTBR SN-sets. We can thus use depth first search to construct all the CandidateTBR SN-sets; note that this is also sufficient to obtain the trees that the CandidateTBR SN-sets represent, because (for trees) the structure of the tree is identical to the nested structure of its SN-sets. Given that there are only $O(n)$ SN-sets, the running time is dominated by construction of the SN-tree. □

Algorithm 2 SL-k (Construct all Simple Level-k networks)

```
1: $Net := \emptyset$
2: $TBR_1 := L(T)$
3: for each leaf $b_1 \in TBR_1$ do
4:   $L'_1 := L(T) \setminus \{b_1\}$
5:   $T'_1 := T|L'_1$
6:   $TBR_2 := FindCandidateTBRs(T'_1)$
7:   for each $b_2 \in TBR_2$ do
8:     ...
       {Continue nesting for loops to a depth of $k$.}
9:     ...
10:    $TBR_k := FindCandidateTBRs(T'_{k-1})$
11:    for each $b_k \in TBR_k$ do
12:      $L'_k := L'_{k-1} \setminus b_k$
13:      $T'_k := T'_{k-1}|L'_k$
         {At this point we have finished "guessing" what the TBRs are, and
         $(\{b_1\}, b_2, ..., b_k)$ is a vector of (possibly empty) subsets of $L(T)$. We
         now try all possible ways of hanging the TBRs back in.}
14:      if $L'_k$ contains two or more leaves then
15:        build the unique tree $N'_k = (V, A)$ consistent with $T'_k$ if it exists
           (see [Hen99])
16:      else
17:        If $L'_k$ contains one leaf $\{x\}$, let $N'_k$ be the network comprising the
           single leaf $\{x\}$
18:        If $L'_k$ contains zero leaves, let $N'_k$ be the network comprising a
           single, new dummy leaf
19:      $V := V \cup \{r'\}; A := A \cup \{(r', r)\}$ { with $r$ the root of $N'_k$ and $r'$ a
         new dummy root }
20:      Let $H(b_k)$ be the unique tree consistent with $b_k$
         { Note: $H(b_k)$ is a single vertex if $|b_k| = 1$ and empty if $|b_k| = 0$. }
21:      for every two arcs $a^1_k$, $a^2_k$ in $N'_k$ (not necessarily distinct) do
```

```
22:          Let $p$ (respectively $q$) be a new vertex obtained by subdividing
             $a_k^1$ (respectively $a_k^2$)
23:          Connect $p$ and $q$ to a new reticulation $ret_k$
24:          Hang $H(b_k)$ (or a new dummy leaf if $H(b_k)$ is empty) from $ret_k$
25:          **if** $a_k^1$ (or $a_k^2$) was the arc above a dummy leaf $d$ **then**
26:            Remove $d$ and if its former parent has indegree and outdegree
               1, suppress that
27:          Let $N'_{k-1}$ be the resulting network
28:          Let $H(b_{k-1})$ be the unique tree consistent with $b_{k-1}$
29:          **for** every two arcs $a_{k-1}^1$, $a_{k-1}^2$ in $N'_{k-1}$ (not necessarily distinct)
             **do**
30:            ...
               { Continue nesting **for** loops to a depth of $k$. }
31:            ...
32:            Let $N'_1$ be the resulting network
33:            Let $H(b_1)$ be the tree consisting of only the single vertex $b_1$
34:            **for** every two arcs $a_1^1$, $a_1^2$ in $N'_1$ (not necessarily distinct) **do**
35:              Let $p$ (respectively $q$) be a new vertex obtained by subdivid-
                 ing $a_1^1$ (respectively $a_1^2$)
36:              Connect $p$ and $q$ to a new reticulation $ret_1$
37:              Hang $H(b_1)$ from $ret_1$
38:              **if** $a_1^1$ (or $a_1^2$) was the arc above a dummy leaf $d$ **then**
39:                Remove $d$ and if its former parent has indegree and out-
                   degree 1, suppress that
                 {This is the innermost loop of the algorithm}
40:              Let $N'$ be the resulting network
41:              Remove the dummy root $r'$ from $N'$
42:              Remove (and if needed suppress former parents of) any re-
                 maining dummy leaves in $N'$
43:              **if** $N'$ is a simple level-$k$ network consistent with $T$ **then**
44:                $Net := Net \cup \{N'\}$
45: **return** $Net$
```

Theorem 4.7. *Given a dense set of triplets T, it is possible to construct all simple level-k networks consistent with T in time $O(|T|^{k+1})$.*

Proof. We claim that algorithm SL-k does this for us. First we prove correctness. The high-level idea is as follows. Consider a simple level-k network N. From Observation 4.1 we know that N contains at least one TBR. (Given that N is simple we know that all TBRs are equal to single leaves. That is why the outermost loop of the algorithm can restrict itself to considering only single-leaf TBRs.) By looping through all CandidateTBR SN-sets we will eventually find one that corresponds to a real TBR. If we remove this TBR and the reticulation from which it hangs, and then suppress any resulting vertices with both indegree and outdegree equal to one, we obtain a new graph with one fewer reticulation than N. Note that this new graph might not be a valid network in the sense that it might have reticulations with no outgoing arcs. Repeating this k times in total we eventually reach a tree which we can construct using the algorithm of Aho et al. (and is unique, as shown in [Jan06b]). From this tree we can reconstruct the network N by reintroducing the TBRs back into the network (each TBR below a reticulation) in the reverse order in which we found them. We don't, however, know exactly where the reticulations were in N, so every time we reintroduce a TBR back into the network we exhaustively try every pair of arcs (as the arcs which will be subdivided to hang the reticulation, and thus the TBR, from.) Because we try every possible way of removing TBRs from the network N, and every possible way of adding them back, we will eventually reconstruct N.

The role of the dummy leaves in SL-k is linked to the empty TBRs (and their corresponding empty CandidateTBR SN-Sets). When a TBR is removed, it can happen (as mentioned above) that a graph is created containing reticulations with no outgoing arcs. (For example: when one of the parents of a reticulation from which the TBR hangs, is also a reticulation.) Conceptually we say that there *is* a TBR hanging below such a reticulation, but that it is empty. Hence the need in the algorithm to also consider removing the empty TBR. If this happens, we will also encounter the phenomenon in the second phase of the algorithm, when we are re-introducing TBRs into the graph. What do we insert into the graph when we reintroduce an empty TBR? We use a dummy leaf as a place-holder, ensuring that every reticulation always has an outgoing arc. The dummy leaves can be removed once that outgoing arc is subdivided later in the algorithm, or at the end of the algorithm, whichever happens sooner. The dummy root, finally, is needed for when there are no leaves on a side connected to the root.

It remains to analyse the running time. From Observation 4.2 we know that each execution of $FindCandidateTBRs$ (which computes all TBRs in a dense triplet set plus the empty TBR) takes $O(n^3)$ time and returns at most $O(n)$ TBRs. Operations such as computing T'_i, and the construction of the tree N'_k, all require time bounded above by $O(n^3)$. The for-loops when we loop though the CandidateTBR SN-sets are nested to a depth of k. The for-loops when we loop through pairs of arcs from which to hang TBRs, are also nested to a depth of k. (There will only be $O(n)$ arcs to choose from by Lemma 4.5.) Checking whether N' is consistent with T, which we do inside the innermost loop of the entire algorithm, takes time $O(n^3)$ [Byr08, Lemma 2]. So the running time is $O(n(n^3 + n(n^3 + \ldots + n(n^3 + n^{2k+3}))))$ which is $O(n^{3k+3})$. $\square$

Lemma 4.11. *Given a dense triplet set T, all simple level-2 networks consistent with T can be constructed in $O(n^8)$ time.*

Proof. We say that z is a *low leaf* of a network N if its parent has outdegree zero in $N \setminus L$. A low leaf is thus either a reticulation leaf or the child of a split vertex where both children of the split vertex are leaves. If arc a enters a low leaf z we say that a is a *low arc* of N. An arc leaving the root or a child of the root is called a *high arc*.

Using Algorithm SL-k all simple level-2 networks consistent with T can be constructed in time $O(n^9)$. However, we will show below that when we subdivide two arcs for the first time we can always choose one of these arcs to be a high or a low arc. This improves the running time to $O(n^8)$.

It remains to be shown that all simple level-2 networks are still considered by the Algorithm. First consider a network of type $2a$. We can first remove the leaf on side F and then the TBR consisting of all leaves (possibly none) on side E. If there are at least two leaves on side B then one of the arcs we choose to subdivide is a low arc (we subdivide the arc leading to the lowest leaf on side B). If on the other hand there is at most one leaf on side B then one of the arcs we subdivide is a high arc (we subdivide the arc leaving the dummy root if there are no leaves on side B and we subdivide an arc leaving the child of the dummy root if there is exactly one leaf on side B).

Now consider a network of type $2b$. We first remove the leaf on side G and the TBR consisting of the leaf on side H. Then we can argue just like with $2a$ that if there are at least two leaves on side B we subdivide a low arc and otherwise a high arc. In a network of type $2c$ we remove the leaf on side G and the TBR consisting of the leaf on side H. If on one of the sides C, D, E or F there are at least two leaves we can subdivide a low arc on this side. Otherwise there is at most one leaf on each of the sides C, D, E and F and therefore all arcs we want to subdivide are low. Finally, consider a network of type $2d$. We remove the leaf on side F and the TBR consisting of the leaves on side E. Then we can always subdivide a low arc unless there are no leaves on sides B and C, which is not allowed. This concludes the proof. □

4.6 Constructing Level-2 Networks from Dense Triplet Sets in Polynomial Time

This section describes a polynomial time algorithm that constructs a level-2 network from a dense triplet set T if such a network exists. The algorithm is recursive. The main idea is visualised in Figure 4.16. Suppose that we know the correct partition $L = S_1 \cup \ldots \cup S_7$ of the leaves. Then the algorithm replaces each set S_i by a single (meta-)leaf and constructs a simple level-2 network (in black) on these meta-leaves. How simple networks can be constructed was shown in the previous section. The complete level-2 network can subsequently be obtained by replacing each meta-leaf S_i by a recursively created level-2 network. In spirit, this procedure resembles that for level-1 networks [Jan06b]. However, besides the fact that the simple level-2 networks are more complex, it also turns out that finding the right partition, and proving the correctness of the overall algorithm, is far more involved than in the level-1 version of the problem. Unlike in the level-1 case there does not, for example, always exist a level-2 network where the sets of leaves below highest cut-arcs correspond to the maximal SN-sets.

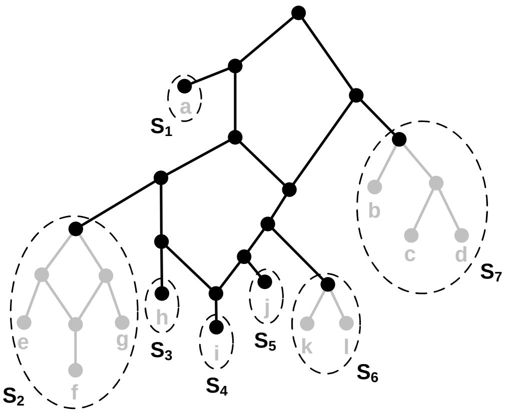


Figure 4.16: Constructing a level-2 network by recursively constructing simple networks, given the partition $L = S_1 \cup \ldots \cup S_7$.

4.6.1 Algorithm and Proof

For any network N, denote by Collapse(N) the network obtained from N by collapsing the sets of leaves below highest cut-arcs. More precisely, for each highest cut-arc $a = (u, v)$ in N, replace v and everything reachable from it by a single new leaf V, which we identify with the set of leaves below a in N. Thus if N has q highest cut-arcs $a_1, \ldots, a_q$ then Collapse(N) has leaf set $\mathcal{C} = \{C_1, ..., C_q\}$, where C_i is the set of leaves below a_i in N (and hence $\mathcal{C}$ is a partition of $L(N)$).

For the sake of convenience we say that the basic tree, simple level-1 networks and simple level-2 networks are all *simple level-≤2 networks.* Note that if N is a level-2 network, then Collapse(N) is a simple level-≤ 2 network, by Lemma 4.3.

For a dense triplet set T on leaf set L, a partition $\mathcal{C}$ of L is a *correct partition* for T if there exists a level-2 network N consistent with T, such that the leaf sets below highest cut arcs in N are given by $\mathcal{C}$, or in other words if there exists a level-2 network N such that Collapse(N) has leaf set $\mathcal{C}$.

For any triplet set T and any partition $\mathcal{C} = \{C_1, \ldots, C_q\}$ of L, define the *induced* triplet set $T\nabla\mathcal{C}$ as the set of triplets $C_iC_j|C_k$ such that there exist $x \in C_i$, $y \in C_j$, $z \in C_k$ with $xy|z \in T$ and i, j and k all distinct. Note that $T\nabla\mathcal{C}$ is a triplet set on $\mathcal{C}$, and that it is dense if T is dense. Note that if N is a network consistent with T such that the leaf sets below highest cut arcs in N are given by a partition $\mathcal{C}$ of L, then Collapse(N) is consistent with $T\nabla\mathcal{C}$.

The rough idea of our algorithm is to first determine, if it exists, a correct partition $\mathcal{C} = \{C_1, \dots, C_q\}$ for the given dense triplet set T, then construct a simple level-≤ 2 network $N_\mathcal{C}$ consistent with $T\nabla\mathcal{C}$ using Algorithm SL-2, and then replace every leaf C_i of $N_\mathcal{C}$ by a recursively created level-2 network consistent with $T|C_i$. The resulting network N is consistent with T by the following lemma. Note that in the lemma we do not require $N_\mathcal{C}$ to be simple.

Lemma 4.12. *Let $\mathcal{C} = \{C_1, \dots, C_q\}$ be a correct partition for a dense triplet set T and $N_\mathcal{C}$ a level-2 network consistent with $T\nabla\mathcal{C}$. Let N be a network obtained from $N_\mathcal{C}$ by replacing each leaf C_i of $N_\mathcal{C}$ by a level-2 network N_i consistent with $T|C_i$. Then N is a level-2 network consistent with T.*

Proof. Consider a triplet $xy|z \in T$. First suppose that $x, y, z \in C_i$ for some i. Then $xy|z \in T|C_i$ and hence N is consistent with $xy|z$ because its subnetwork N_i is.

Since $\mathcal{C}$ is a correct partition for T, there exists some level-2 network N' consistent with T such that the C_i are the sets of leaves below highest cut arcs of N'. This means that for no $i \neq j$ it holds that $x, z \in C_i$ while $y \in C_j$, as N' is consistent with $xy|z$.

If $x, y \in C_i$ and $z \in C_j$ for some $i \neq j$, then $xy|z$ is consistent with N since C_i and C_j are below cut-arcs of N that are not reachable from each other (as they are both expanded leaves of $N_\mathcal{C}$).

Finally, suppose that $x \in C_i$, $y \in C_j$ and $z \in C_k$ with i, j, k distinct. From $xy|z \in T$ it follows that $C_iC_j|C_k \in T\nabla\mathcal{C}$ and hence that $N_\mathcal{C}$ is consistent with $C_iC_j|C_k$. But then N is consistent with $xy|z$. $\square$

It remains to describe how to find a correct partition of the leaf set. This is the main subject of the remainder of this section. If the given triplet set is consistent with a level-1 network, the collection of maximal SN-sets forms a correct partition $\mathcal{S}$, as follows from the algorithm in [Jan06a]. For level-2 the situation is more complicated, but maximal SN-sets can still be used to obtain correct partitions.

Suppose that a dense triplet set T is consistent with a level-2 network N. We will show in Theorem 4.8 that we have to split at most one maximal SN-set of T to obtain a correct partition of the leaves. To prove this we first show in Lemma 4.13 that, if N is a simple level-≤ 2 network, we can “push” any single maximal SN-set, apart from one exception, below a highest cut-arc, while maintaining consistency with T. In Lemma 4.15 this is extended to *all* maximal SN-sets and in Theorem 4.8 to general level-2 networks. For the latter two results we need Lemma 4.14, which describes the relation between the maximal SN-sets of T and the maximal SN-sets of $T\nabla\mathcal{C}$, for a correct partition $\mathcal{C}$.

Lemma 4.13. *Let N be a simple level-≤ 2 network consistent with a dense triplet set T and S a maximal SN-set of T. Then, unless N is of type $2c$ and S consists of the two reticulation leaves in N, the partition $\mathcal{C}' := \{S\} \cup \{\{l\} \mid l \in L \setminus S\}$ of L is a correct partition for T. In fact, in this case there exists a level-2 network N' consistent with T and with the same set of reticulation leaves as N such that Collapse(N') has leaf set $\mathcal{C}'$ and is not of type $2c$ if N is not.*

Proof. We explicitly construct a level-2 network N' consistent with T such that S equals the set of leaves below a highest cut-arc of N', and all other sets of leaves below highest cut-arcs are the same as in N, i.e. they are singletons (as N is simple). Moreover, the reticulation leaves in N' are the same as in N, and if N is not of type $2c$ then neither is Collapse(N').

The case that S is a singleton is trivial and hence we assume from now on that S contains at least two leaves. By assumption, if N is of type $2c$ then S is not equal to the leaves on sides G and H. We say that u is a *lowest common ancestor (LCA)* of x and y if u is an ancestor of both x and y and no proper descendant of u has this property. We start with three critical observations.

Observation 4.3. *No two leaves in S have the root as lowest common ancestor.*

Viz., if two leaves in S have the root as LCA then all leaves are in S, since S is an SN-set.

Observation 4.4. *If $x \in S$ and there is a path not containing any reticulations from the parent of x to another leaf l, then l is also in S.*

Viz., if $y \neq x$ is any other leaf in S, then since N is not consistent with $xy|l$, it follows that l is in S.

Observation 4.5. *If two leaves $x, y \in S$ have exactly one LCA u, then all leaves z that have a parent on a path from u to x are in S, unless N is of type $2a$, x is on side F, y is on side C and z on side E. In this case no leaves of S are on sides E or B.*

Proof. Assume that x and y have a unique LCA u, as in Figure 4.17.

If $z \notin S$ then neither $xz|y$ nor $yz|x$ is in T and therefore $xy|z \in T$ because T is dense. This means that there are vertices v and v' in N and internally vertex disjoint paths $v' \to x$, $v' \to y$, $v \to v'$ and $v \to z$ denoted by $P_{v'x}$, $P_{v'y}$, $P_{vv'}$ and P_{vz}, respectively. Because u is the only LCA of x and y it follows that v' is an ancestor of u. Since $P_{v'x}$ is vertex disjoint from P_{vz}, it does not contain the parent of z. But by assumption there is a path Q_{uzx} from u to x which does contain the parent of z and hence Q_{uzx} joins with $P_{v'x}$ at a reticulation r_1 that is between the parent of z and x.

Suppose that u is contained in P_{vz}. Because $u = \mathrm{LCA}(x, y)$, u is a split vertex (not a reticulation) and there is a u–y path Q_{uy} internally vertex disjoint from P_{vz}. The fact that u is below $v' \notin P_{vz}$ implies the existence of a reticulation on P_{vz} above u, so different from r_1. Since $P_{v'y}$ is vertex disjoint from $P_{vz} \ni u$, it joins the path Q_{uy} at a third reticulation between u and y, which is clearly different from the second one (above u) and also different from r_1 because there are no common ancestors of x and y below u. Thus, the simple level-≤ 2 network N contains three distinct reticulations. A contradiction. So P_{vz} does not contain u.

But then P_{vz} joins with Q_{uzx} at a reticulation r_2 which is between u and z. Since u is an LCA of x and y, there is a path Q_{uy} which is internally vertex disjoint from Q_{uzx}. Because there are at most two reticulations in N, both located on Q_{uzx}, the path $P_{v'y}$ simply contains the entire path Q_{uy}. Now the unique picture emerges of a type $2a$ network, with x the reticulation leaf (on side F), y on side C and z on side E, see Figure 4.17. It follows immediately that no leaf l of S is on side B or E, because N is not consistent with $yl|z$ for such l. $\square$

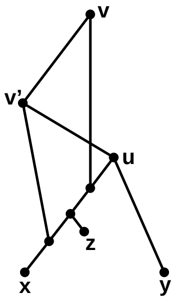


Figure 4.17: Illustration of the proof of Observation 4.5.

As a warming-up for the reader, we first prove the lemma for the case that N is a simple level-1 network. The proof is constructive. The construction in the proof for simple level-2 networks is essentially the same but more involved. In case N is a simple level-1 network, Observations 4.3 and 4.4 imply that the parents of S form a path P starting below the root and ending in either the reticulation or in a parent of the reticulation. We construct the network N' as follows. Let N_c denote the result of contracting in N the path P to a single vertex v_c. Let N_c^* denote the result of removing all leaves in S from N_c and let N^* denote the result of removing all leaves not in S from N. We combine N_c^* and N^* by adding an arc from v_c to the root of N^*. In order to obtain a valid network N' with only labelled leaves we remove any unlabelled vertices with outdegree zero and suppress any vertices with indegree and outdegree one. Clearly, N' has the same reticulation leaf as N and Collapse(N') is of type 1 so not of type $2c$. Furthermore, it is not too difficult to check that N' is consistent with all triplets that are consistent with N, except for triplets of the form $xy|z$ with $x, z \in S$ and $y \notin S$. Since the latter type of triplets are not in T, this concludes the level-1 case.

From now on we assume that N is a simple level-2 network. The main differences with the level-1 case are that in the level-2 case the parents of S form a connected subgraph which is not necessarily a path, and that there are some exceptional cases, which complicate the analysis.

Claim 4.4. *There are at most two (directed) paths in N such that each leaf in S has a parent on one of these paths, unless N is of type $2b$ and there is a leaf on side D, a leaf on side E and a leaf on side F or H in S.*

Proof. First consider a simple level-2 network of type $2a$ and suppose that a leaf on side B is in S. Then by Observation 4.3 no leaves on sides A, C and D are in S and the claim follows. If no leaf on side B is in S then the claim is also clearly true. For $2b$ we can argue similarly that if a leaf of side B is in S only leaves on side B and H can be in S. If B is not in S then the claim also holds, since we excluded the case that there are leaves on sides D, E and F in S. Now consider $2c$ and suppose that a leaf on one of the sides A, C or E is in S. Then no leaves on sides B, D and F can be in S by Observation 4.3 and the claim follows. The case that there is a leaf on one of the sides B, D and F is symmetric. This covers all cases in a network of type $2c$ since we assumed that for networks of this type, S does not consist of leaves on sides G and H. In $2d$, if a leaf on side D is in S there can only be leaves from sides D and F in S. If there is no leaf on side D in S the claim is also clearly true. $\square$

Claim 4.5. *There exists a connected subgraph of N containing the parents of leaves in S and no parents of other leaves, unless N is of type $2a$ and (some) leaves on sides C and F are in S and no leaves on side B or E are in S. Let $2P$ denote such a connected subgraph of minimum size. Let $\delta^-(2P)$ denote the number of arcs entering $2P$ and let $\delta^+(2P)$ denote the number of arcs going out of $2P$ not leading to leaves in S. Then either $\delta^-(2P) \leq 3$ and $\delta^+(2P) \leq 1$, or $\delta^-(2P) = 1$ and $\delta^+(2P) = 2$.*

Proof. If N is of type $2b$ and (some) leaves on sides D, E and F are in S, then all leaves on sides D, E and F and C are in S by Observations 4.4 and 4.5 and all leaves on side G are in S because N is not consistent with $de|g$ for $d \in S$ on side D, $e \in S$ on side E and g on side G. Thus, a minimal connected subgraph $2P$ covering the parents of S in this case consists of sides C, D, E, F, G and possibly H and possibly (the lower part of) side A. It has $\delta^-(2P) = 2$ and $\delta^+(2P) \leq 1$.

If we are not in the above situation, the previous claim says that N contains at most two paths covering all parents of S. Suppose that these paths necessarily also cover parents of leaves not in S. Then there are leaves $x, z \in S$ and $y \notin S$

such that there is a path from the parent of x, passing the parent of y and going to the parent of z. In this case the parent of x is a LCA of x and z. In simple level-2 networks there is only one possible situation where two leaves have two distinct different LCAs and this is if these leaves are on sides G and H in a network of type $2c$. Since x is not a reticulation leaf we can thus conclude that the parent of x is the unique LCA of x and z. This leads to a contradiction by Observation 4.5.

Thus, there are at most two paths in N such that S consists of exactly those leaves that have a parent on one of these paths. The union of the (at most) two paths and their unique LCA is, by Observation 4.5, always a connected subgraph.

Thus we can define $2P$ to be a connected subgraph of N of minimum size covering the parents of S and no parents of other leaves. The claim about the possible indegrees and outdegrees of this connected subgraph $2P$ follows from examining the structure of the four simple level-2 generators and Observations 4.3, 4.4 and 4.5. Note that, by minimality of $2P$ and by Observations 4.3 and 4.5, $2P$ does not contain the root. Also note that if N is of type $2b$ and leaves on D and C (and possibly F and H) are in S, then also all leaves on side E are in S (because $cd|e$ is not consistent with N for c on side C, d on D, e on E). and the leaf on side G is in S (because $cd|g$ is not consistent with N for c on side C, d on D, g on G). So here $\delta^-(2P) \leq 3$ and $\delta^+(2P) \leq 1$. $\square$

If N is of type $2a$ and leaves on sides C and F are in S and no leaves on side E or B are in S we construct a network N' as indicated in Figure 4.18, where all the leaves in S (including possibly leaves on side A and/or D) are put below the non-trivial cut-arc and all other leaves above it. Note that if there are leaves on side A or D in S then all leaves on side D must be in S. It is not too difficult to check that in this case Collapse(N') is not of type $2c$, N' has the same reticulation leaves as N and is consistent with all triplets that are consistent with N and not of the form $xy|z$ with $x, z \in S$ and $y \notin S$.

In all other cases, let N_c denote the result of contracting in N the connected subgraph $2P$ to a single vertex v_c. Let N_c^* denote the result of removing all leaves in S from N_c. Let N^* denote the result of removing all leaves not in S from N. We construct N' by combining N_c^* and N^*. How this is done depends on the indegree and outdegree of v_c in N_c^*. The idea is, as in the level-1 case, to create a new outgoing arc from v_c to the root of N^*. But in order to obtain a valid network, we only add such an arc if v_c has outdegree zero or indegree and outdegree one. If v_c has indegree at least two and outdegree one we subdivide the arc leaving v_c by a new vertex v_n and create an arc from v_n to the root of N^*. If v_c has indegree one and outdegree two we subdivide the arc entering v_c by a new vertex v_n and create an arc from v_n to the root of N^*, unless N is of type $2a$ with all leaves on sides C and D and some on side A in S. In the latter excluded case we apply the construction from Figure 4.19.

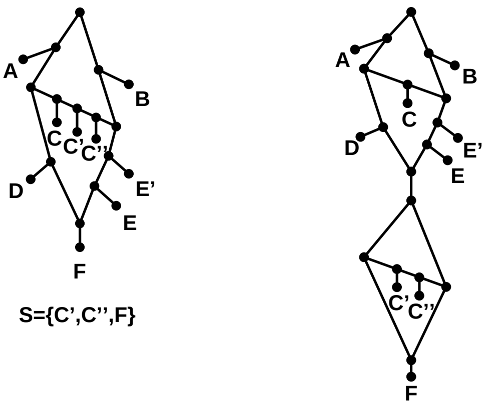


Figure 4.18: Construction of N' (the network on the right) in case N (on the left) is of type $2a$ and leaves on sides C and F are in S and no leaves on side E or B are in S.

To obtain a valid network we remove again all unlabelled vertices with outdegree zero and suppress vertices with indegree and outdegree one. Moreover if v_c has indegree three, we replace it by two vertices v_c^1 and v_c^2 and connect two former parents of v_c to v_c^1, one former parent of v_c to v_c^2 and v_c^1 to v_c^2.

This whole procedure is illustrated in Figure 4.20. An example network N is displayed on the left with $2P$ in grey. After contracting $2P$ to a vertex v_c (in grey) and removing the leaves from $S = \{H, D, G, F\}$ we get the network N_c^* in the middle. Since v_c has indegree three and outdegree zero we replace it by two vertices of indegree two and create an outgoing arc to N^*. After suppressing all vertices with indegree and outdegree both one we get the network N' on the right. Note that N is of type $2c$, but Collapse(N') is of type $2d$ in this example. If G would not have been included in S, then Collapse(N') would have been of type $2a$, and if H would not have been included either, then Collapse(N') would have been of type $2c$. So in general, a change of type is possible in this construction, but the reader may verify that Collapse(N') is only of type $2c$ if N is of that type already.

To check that N' is consistent with T, consider three leaves x, y, z. If $x, y, z \in S$ or $x, y, z \notin S$ then any triplet on these leaves that is consistent with N is clearly also consistent with N'. If $x, y \in S$ and $z \notin S$ then, by the definition of SN-set, $xy|z$ is the only triplet in T on these three leaves and this triplet is consistent with N', as x and y appear below one cut-arc in N' together, whereas z is not below that arc.

Now consider the case that $x \in S$ and $y, z \notin S$ and suppose that a triplet t over $\{x, y, z\}$ is consistent with N but not with N'. Notice that since t is not consistent with N' it is also not consistent with N_c. First suppose that $t = yz|x$. Because t is consistent with N, there are vertices u and v of N such that there exist internally vertex disjoint paths $u \to v$, $v \to y$, $v \to z$

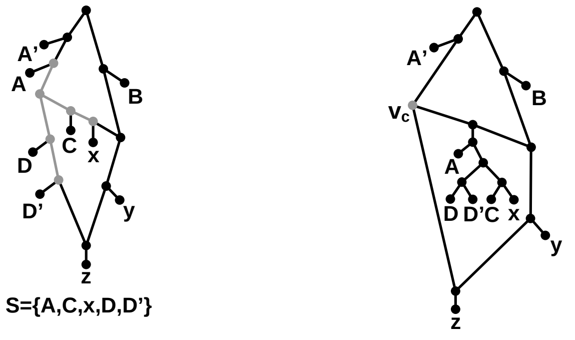


Figure 4.19: Construction of N' (on the right) if N (on the left) is of type $2a$ with all leaves on sides C and D and some on side A in S.

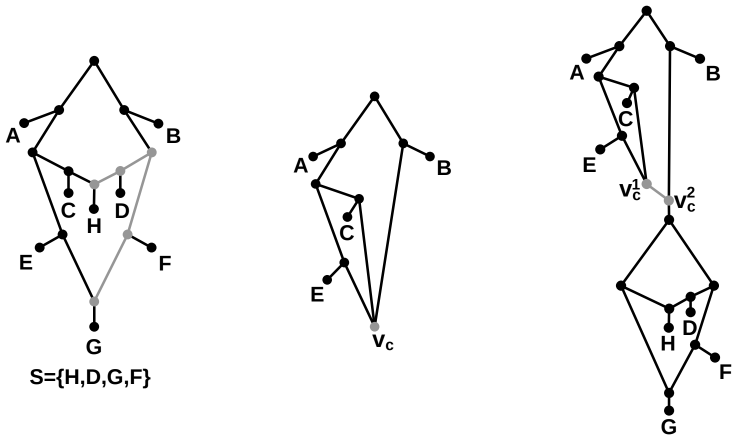


Figure 4.20: Networks N (on the left), N_c^* (in the middle) and N' (on the right) as an example of the construction in the proof of Lemma 4.13.

and $u \to x$. Because t is not consistent with N_c, one of these paths has been contracted and is thus contained in $2P$. This must be the path $u \to v$ since $2P$ does not contain any leaves. Hence in N_c there are three internally disjoint paths from v_c to x, y and z. It follows that v_c has outdegree two and thus indegree one in N_c^*. This means that network N' is either constructed as in Figure 4.19, or the arc leading to v_c is subdivided by v_n and there is an arc from v_n to the root of N^*. In both cases inconsistency of N' with $yz|x$ is contradicted.

Now consider the case that $t = xy|z$ (the case $= xz|y$ is symmetric). In the same way as above we argue that, because t is consistent with N but not with N_c, there exist vertices u and v of N, internally vertex disjoint paths $u \to v$, $v \to x$, $v \to y$ and $u \to z$, denoted by P_{uv}, P_{vx}, P_{vy} and P_{uz} respectively, and that $2P$ contains the path P_{uv}. The fact that $2P$ is a minimal subgraph covering all parents of leaves in S implies that u is either a LCA of two leaves in S or a descendant of such a LCA. As argued before $2P$ does not contain the root, hence u is not the root. Observe that u is also not the parent of a leaf since $z \notin S$ would be the only candidate for such a leaf and $2P$ contains no parents of leaves that are not in S. Hence u is a split vertex which is not equal to the root and does not have a leaf as child.

Below we prove a claim stating that there exists a path Q_{rz} from the root of N to z that does not intersect $2P$. This path also exists in N_c. If it does not intersect the path from v to y in N, and hence the path from v_c to y in N', then this implies immediately that $xy|z$ is consistent with N', a contradiction. Otherwise Q_{rz} joins P_{vy} at a reticulation, r_1 say. Q_{rz} must join P_{uz} at a reticulation r_2, which is a descendant of r_1. The facts that u is a split vertex without a leaf child and that there exist internally vertex disjoint paths $u \to r_1$ (consisting of P_{uv} and part of P_{vy}) and $u \to r_2$ (consisting of the first part of P_{uz}) imply that N is of type $2a$ with z on side F and y on side E. Moreover, u is its unique split vertex without leaf child and v is on side C (and hence the upper part of C is in S, and by Observation 4.4 all of C is in S). Because $2P$ contains the parent of $x \in S$, and because Q_{rz} is completely disjoint from $2P$, it follows that x is on C and no leaf of E is in S. Since $2P$ is a minimal subgraph covering S, and since u is contained in $2P$, also the lower part of A or the upper part of D is in S. In any case, this implies that all of D is in S (by Observation 4.4 or by the fact that $ac|d$ is not consistent with N for a on A, c on C and d on D). In this case N' is constructed like in Figure 4.19, again contradicting inconsistency with $xy|z$.

Remains to prove the claim

Claim 4.6. *There exists a path from the root of N to z that does not intersect $2P$.*

Proof. Observation 4.4 immediately implies that the claim is true if z is not below any reticulation. In case $z \notin S$ is on side E or F of $2a$, side H of $2b$, side G or H of $2c$ or on side F of $2d$, the claim follows basically from the fact that no two leaves in S have the root as LCA (Observation 4.3). For $2a$ we need the additional argument that if $2P$ contains part of E (above z) and part of either A or D, then it also contains the whole lower part of E (by Observation 4.4), and it contains F (as $ae|f$, $de|f$ are not consistent) contradicting the assumption that $z \notin S$.

Now suppose that z is on side G of $2b$. Then $2P$ cannot contain part of side A since through Observation 4.4 this would imply that all of D,C and E would be in S and therefore also all of G, contradicting $z \notin S$. If both D and C or E contain vertices in S then for $a \in D$, $b \in C$ or E the triplet $ab|z$ cannot be consistent with N, implying that $z \in S$, a contradiction.

Finally suppose that z is on side E of $2d$. If there is no path from the root to z with is disjoint from $2P$ then $2P$ contains part of A or part of C and part of B or both. In any case, no triplet $ab|z$ with $a, b \in S$ is consistent with N, a contradiction. $\square$

$\square$

The above lemma essentially shows that any single maximal SN-set (with one well-specified exception) can be "pushed" below a highest cut-arc in a simple level-≤ 2 network without losing consistency with the given triplet set. We would like to argue by induction that all maximal SN-sets (with the same exception) can be "pushed" below highest cut-arcs. For this we first need the following lemma.

Lemma 4.14. *Let $\mathcal{C}$ be a correct partition for dense triplet set T. There is a bijection between the maximal SN-sets of T and the maximal SN-sets of $T\nabla\mathcal{C}$ mapping a maximal SN-set $\{C_1, \ldots, C_\ell\}$ of $T\nabla\mathcal{C}$ to the maximal SN-set $C_1 \cup \ldots \cup C_\ell$ of T.*

Proof. Let N be a level-2 network consistent with T whose sets of leaves below highest cut arcs are given by $\mathcal{C}$. Firstly, consider an SN-set of T that is a union $S = C_1 \cup \ldots \cup C_\ell$ of sets from $\mathcal{C}$. Define S' as $\{C_1, \ldots, C_\ell\}$ and suppose that S' is not an SN-set of $T\nabla\mathcal{C}$. Then there exist distinct $C_i, C_k \in S'$ and $C_j \notin S'$ such that $C_iC_j|C_k \in T\nabla\mathcal{C}$. This implies existence of leaves $x \in C_i, y \in C_j, z \in C_k$ and a triplet $xy|z \in T$ such that $x, z \in S$ and $y \notin S$, contradicting the fact that S is an SN-set. Thus, S' is an SN-set of $T\nabla\mathcal{C}$.

Secondly, consider an SN-set $S' = \{C_1, \ldots, C_\ell\}$ of $T\nabla\mathcal{C}$. If $S := C_1 \cup \ldots \cup C_\ell$ is not an SN-set of T then there exists a triplet $xy|z \in T$ with $x, z \in S$ and $y \notin S$. Because $y \notin S$, y is not in the same partition element C_j as x or z, or in other words y is not below the same highest cut-arc as x or z in N. If x and z were both in the same partition element C_i, then they were both below the same highest cut arc in the network N. But this contradicts the fact that N is consistent with $xy|z \in T$. So $x \in C_i$, $y \in C_j$ and $z \in C_k$, where $C_i, C_k \in S'$ distinct and $C_j \notin S'$. But then $xy|z \in T$ implies $C_iC_j|C_k \in T\nabla\mathcal{C}$, contradicting the fact that S' is an SN-set.

Finally, the 1-1 correspondence we have just established between SN-sets of $T\nabla\mathcal{C}$ and SN-sets of T of the form $S = C_1 \cup \ldots \cup C_\ell$ together with the fact that, by Lemma 4.6, any maximal SN-set of T is of that form, proves the lemma. $\square$

We are ready to extend Lemma 4.13 to the collection of all SN-sets.

Lemma 4.15. *Let N be a simple level-≤ 2 network consistent with a dense triplet set T and $\mathcal{C}$ the collection of maximal SN-sets of T. Then $\mathcal{C}$ is a correct partition for T, unless N is of type $2c$ and T has a maximal SN-set $S = \{G, H\}$*

consisting of the two reticulation leaves of N: in the latter case $\mathcal{C}' := \mathcal{C} \setminus \{S\} \cup \{\{G\}, \{H\}\}$ is a correct partition for T.

Proof. Order the (disjoint) maximal SN-sets $S_1, \ldots S_k$, in such a way that if N is of type $2c$ and $\{G, H\}$ is a maximal SN-set, then $\{G, H\} = S_k$. Now we prove by induction on i that $\mathcal{C}_i := \{S_1, \ldots, S_i\} \cup \{\{l\} \mid l \in L \setminus (S_1 \cup \cdots \cup S_i)\}$ is a correct partition for T. In fact, we prove that a level-2 network N_i consistent with T exists with the same set of reticulation leaves as N such that Collapse(N_i) has leaf set $\mathcal{C}_i$ and is not of type $2c$ if N is not. Note that $\mathcal{C} = \mathcal{C}_k$ and if N is of type $2c$ and $S_k = \{G, H\}$ then $\mathcal{C}' = \mathcal{C}_{k-1}$. Lemma 4.13 deals with the induction basis $i = 1$.

To prove the induction step, let N_i be a level-2 network consistent with T with the same set of reticulation leaves as N, such that Collapse(N_i) has leaf set $\mathcal{C}_i$ and is not of type $2c$ if N is not. If S_j (for $j \leq i$) is the set of leaves below highest cut-arc a_j, then denote by $N_i(S_j)$ the subnetwork of N_i below a_j. Clearly $N_i(S_j)$ is consistent with $T|S_j$. Note that Collapse(N_i) is a simple level-≤ 2 network consistent with $T \nabla \mathcal{C}_i$. Moreover, S_{i+1} is a maximal SN-set of $T \nabla \mathcal{C}_i$ by Lemma 4.14. Therefore, Lemma 4.13 applied to $T \nabla \mathcal{C}_i$ and S_{i+1} and Collapse(N_i) yields the existence of a network N^C_{i+1} (on the same leaf set as Collapse(N_i), i.e. on $\mathcal{C}_i$) that is consistent with $T \nabla \mathcal{C}_i$ but has S_{i+1} below a highest cut-arc whereas the other leaf sets below highest cut arcs are singletons. Moreover, Lemma 4.13 claims that Collapse(N^C_{i+1}) is not of type $2c$ if Collapse(N_i) is not, and it gives a 1-1 correspondence between reticulation leaves in Collapse(N_i) and reticulation leaves in N^C_{i+1}. Replacing the singleton leaves $S_1, \ldots, S_i$ in N^C_{i+1} by the networks $N_i(S_1), \ldots, N_i(S_i)$ respectively, results in a level-2 network N_{i+1} consistent with T, by Lemma 4.12. This network N_{i+1} has the same set of reticulation leaves as N_i. The sets of leaves below highest cut arcs in N_{i+1} are given by $\mathcal{C}_{i+1}$. Moreover Collapse(N_{i+1}) $= N^C_{i+1}$ is not of type $2c$ if Collapse(N_i) is not. This completes the description of the induction step.

The induction step may be applied for a certain i if Lemma 4.13 is indeed applicable with triplet set $T \nabla \mathcal{C}_i$, SN-set S_{i+1} and network Collapse(N_i). This is the case if Collapse(N_i) is not of type $2c$ or if S_{i+1} does not consist of its two reticulation leaves.

If N is not of type $2c$ then neither is Collapse(N_i), so in this case the induction step applies to every i with $i + 1 \leq k$ and we conclude that $\mathcal{C}_k$ is a correct partition for T.

If N is of type $2c$ but S_{i+1} does not consist of its two reticulation leaves, then we use the extra argument that, by the induction hypothesis, N_i has the same set of reticulation leaves as N. This means that the maximal SN-set S_{i+1} of $T \nabla \mathcal{C}_i$ is not equal to the set of reticulation leaves of Collapse(N_i) and hence we may apply the induction step. Thus, if N is of type $2c$ and $S_k \neq \{G, H\}$, then we apply the induction step for $i + 1 \leq k$ and conclude that $\mathcal{C}_k$ is a correct partition for T. If N is of type $2c$ and $S_k = \{G, H\}$ then we apply the induction step for $i + 1 \leq k - 1$ and conclude that $\mathcal{C}_{k-1}$ is a correct partition for T. $\square$

The next step is to extend the above result to general (not necessarily simple) level-2 networks. The following theorem essentially describes how a correct partition can be constructed from the collection of maximal SN-sets of a given triplet set.

Theorem 4.8. *Let T be a dense triplet set consistent with some level-2 network N. Then there exists a level-2 network N' consistent with T such that at most one maximal SN-set of T equals the union of the sets of leaves below two highest cut-arcs of N' and each other maximal SN-set is equal to the set of leaves below just one highest cut-arc.*

Proof. Let N be a level-2 network consistent with T. Suppose that $a_1, \ldots, a_q$ are the highest cut arcs of N, and C_i is the set of leaves below a_i. Then $\mathcal{C} := \{C_1, \ldots, C_q\}$ is a correct partition for T. Denote the subnetwork of N below a_i by $N(C_i)$. First note that Collapse(N) is a simple level-$\leq$2 network on leaf set $\mathcal{C}$ consistent with $T\nabla\mathcal{C}$. By Lemma 4.15, there exists a level-2 network $N_{\mathcal{C}}$ on leaf set $\mathcal{C}$, consistent with $T\nabla\mathcal{C}$, and having all maximal SN-sets of $T\nabla\mathcal{C}$, with at most one exception, below highest cut arcs. The exception occurs if Collapse(N) is of type $2c$ and one of the maximal SN-sets of $T\nabla\mathcal{C}$ consists of its two reticulation leaves: in that case one maximal SN-set of $T\nabla\mathcal{C}$ is equal to the union of two (singleton) sets of leaves below highest cut-arcs in $N_{\mathcal{C}}$. Let N' be the result of replacing each leaf C_i in $N_{\mathcal{C}}$ by the subnetwork $N(C_i)$ of N. Then N' is consistent with T by Lemma 4.12.

Lemma 4.14 describes a bijection between the maximal SN-sets of $T\nabla\mathcal{C}$ and the maximal SN-sets of T. Since the maximal SN-sets of $T\nabla\mathcal{C}$, with one possible exception, occur below highest cut arcs of $N_{\mathcal{C}}$, the corresponding maximal SN-sets of T occur below highest cut-arcs of N', also with one possible exception. An exceptional SN-set of $T\nabla\mathcal{C}$ consists of two (reticulation) leaves below highest cut-arcs in $N_{\mathcal{C}}$, and hence the corresponding exceptional SN-set of T is the union of two sets of leaves below highest cut-arcs in N'. $\square$

Now suppose that an input triplet set T is consistent with some level-2 network. The above theorem implies that, after possibly splitting one maximal SN-set, we obtain a correct partition and the problem then essentially reduces to constructing a simple level-≤ 2 network for the induced triplet set. Given that there is at most one maximal SN-set that needs to be split into two subsets, we can simply try splitting each maximal SN-set of T in turn, and also consider the case where no maximal SN-set of T is split. There are only $O(n)$ maximal SN-sets. The following lemma tells us how to split the chosen maximal SN-set into two subsets.

Lemma 4.16. *Let T be a dense set of triplets and N' a network with the properties described in Theorem 4.8. Suppose T contains a maximal SN-set X which occurs as the union of the sets S_1 and S_2 of leaves below two highest cut-arcs. Then the collection of maximal SN-sets of $T|X$ is $\{S_1, S_2\}$.*

Proof. Create a network N^* by taking the two subnetworks on the leaf sets S_1 and S_2 in N', and make their roots children of a new root. It is easy to see that N^* is consistent with all the triplets in $T|X$. Both S_1 and S_2 appear below highest cut-arcs in N^* so the result now follows immediately from Lemma 4.6. □

This lemma shows that only maximal SN-sets that internally decompose into two maximal SN-sets need to be considered as possible candidates to be split, and furthermore that this split is completely determined by the internal decomposition of the SN-set. From the proof of Theorem 4.8 it also follows that if it is necessary to split a maximal SN-set X to obtain a correct partition, then the simple level-2 network we are looking for on this partition must be of type $2c$ and X must correspond to its two reticulation leaves. This fact has a positive consequence for the running time of our algorithm. It means that whenever we have to split an SN-set, we can find a simple level-2 network by Algorithm SL-2 much faster, since we already know the two reticulation leaves.

The general outline of our algorithm LEVEL2, which constructs level-2 networks from dense triplet sets, is as follows. First we compute the maximal SN-sets. If there are precisely two maximal SN-sets then we recursively create two level-2 networks for the two maximal SN-sets and connect their roots to a new root. Otherwise, we try splitting each maximal SN-set in turn and we try the case where no maximal SN-set is split. If $\mathcal{S}$ is the obtained set of SN-sets then we compute the induced set of triplets $T\nabla\mathcal{S}$ and try to construct a simple level-1 or -2 network N consistent with $T\nabla\mathcal{S}$ using algorithm SL-2. We recursively create level-2 networks for each SN-set in $\mathcal{S}$ and replace each leaf of N by the corresponding, recursively created, level-2 network. The whole algorithm is presented in pseudo code in Algorithm 3.

Theorem 4.9. *Algorithm LEVEL2 constructs, in $O(|T|^{\frac{8}{3}})$ time, a level-2 network consistent with a dense set of triplets T if and only if such a network exists.*

Proof. Correctness of the algorithm follows from the above. To analyse the running time, recall that the algorithm works by recursively constructing simple level-1 and -2 networks based on maximal SN-sets. If there are s maximal SN-sets the simple level-1 algorithm takes $O(s^3)$ time [Jan06a] and the simple level-2 algorithm takes $O(s^8)$ time by Lemma 4.11. If no solution is found, the algorithm loops over the s maximal SN-sets (line 11) and replaces one, say X, that has two children in the SN-tree, say S_1 and S_2, by these two children. This results in a collection of $s + 1$ SN-sets, and finding a simple level-2

Algorithm 3 LEVEL2

```
1: $N := \emptyset$
2: compute the set $SN$ of maximal SN-sets of $T$
3: **if** $|SN| = 2$ **then**
4:    $N$ consists of a root connected to two leaves: the elements of $SN$
5: **else**
6:    **if** $T\nabla SN$ is consistent with a simple level-1 network **then**
7:       let $N$ be such a network
8:    **else if** $T\nabla SN$ is consistent with a simple level-2 network **then**
9:       let $N$ be such a network
10:   **else**
11:      **for** $X \in SN$ **do**
12:         compute the set $SN'$ of maximal SN-sets of $T|X$
13:         **if** $|SN'| = 2$ **then**
14:            $\mathcal{S} := SN \setminus \{X\} \cup SN'$
15:            **if** $T\nabla\mathcal{S}$ is consistent with a simple level-2 network of type 2c
               where the elements of $SN'$ are the two reticulation leaves **then**
16:               let $N$ be such a network
17: **for** each leaf $V$ of $N$ **do**
18:    recursively create a level-2 network $N_V$ consistent with $T|V$
19: **if** $N \neq \emptyset$ and all $N_X \neq \emptyset$ **then**
20:    replace each leaf $V$ of $N$ by the recursively created $N_V$.
21:    **return** $N$
22: **else**
23:    **return** $\emptyset$
```

network with leaves corresponding to those SN-sets (line 15) now takes only $O(s^6)$, since we can adapt Algorithm SL-2 in such a way that it does not loop over all leaves and TBRs to remove, but simply chooses to remove S_1 and S_2. Indeed, the network we are looking for now is of type $2c$ and has reticulation leaves S_1 and S_2. Hence, constructing a single biconnected component with s outgoing arcs takes the algorithm at most $O(s^8)$ time.

Suppose N is the level-2 network we are constructing. Let $s_1, \ldots, s_m$ denote the numbers of arcs leaving its m nontrivial biconnected components. The network N contains $O(n)$ arcs by Lemma 4.5, implying that $s_1 + \ldots + s_m$ is $O(n)$. Because $s_1^8 + \ldots + s_m^8 \leq (s_1 + \ldots + s_m)^8$ the total time needed for constructing biconnected components in lines 6 - 16 is $O((s_1 + \ldots + s_m)^8)$ and hence $O(n^8)$. The computation of maximal SN-sets in line 2 takes $O(n^3)$ time [Jan06a] and is executed $O(n)$ times. All other computations can also be done in $O(n^3)$ time and are executed at most $O(n^2)$ times. We conclude that the total running time of Algorithm 3 is $O(n^8)$ which is equal to $O(|T|^{\frac{8}{3}})$ because T is dense. $\square$

4.6.2 Practical Experiments

The algorithm from the previous section has been implemented in Java and applied to experimental data. The implementation was made publicly available [LEV07]. The data of this application consists of sequences from different isolates of the yeast *Cryptococcus gattii*. This yeast is potentially dangerous and an ongoing outbreak on the Westcoast of Canada [Kid04], which started in 1999, has caused many infections and even some fatalities. We have constructed a phylogenetic network for these isolates as a tool to find the origin of this *C. gattii* outbreak. We have blinded the names of the isolates here since the biological part of the research has not yet reached a conclusion.

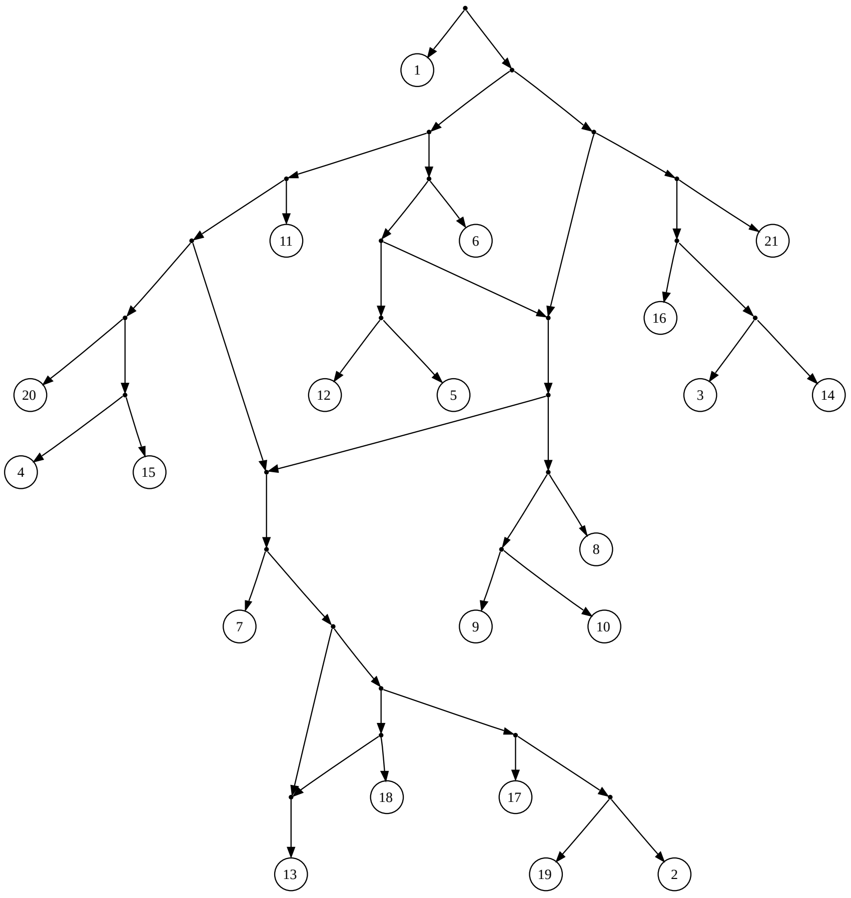


Figure 4.21: The network constructed by the algorithm LEVEL2 for the triplets based on the yeast data set.

Given the *C. gattii* sequences we have constructed a set of triplets as follows.

Firstly, all identical sequences are combined into a single *sequence type (ST)*. One of the sequence types (that is only distantly related to the others) is used as an outgroup and we have applied the Maximum Likelihood method of PHYML [Gui03] to each subset of four sequence types that includes the outgroup. Each output tree of PHYML gives us one input triplet for our algorithm LEVEL2. Running our algorithm for all STs tells us that there exists no level-2 network consistent with all triplets. Therefore, we have applied our algorithm to a set containing as many STs as possible (where certain important STs get priority over others) without destroying level-2-realisability. This set has been found by searching through all subsets. Given this subset and all triplets the execution of the algorithm LEVEL2 took 0.8 seconds on a Pentium IV, 3 GHz PC with 1GB memory. The resulting level-2 network is displayed in Figure 4.21. This network is consistent with all 1330 triplets that were generated over this set of taxa. The figure displays both a reticulate pattern and a dichotomous and tree-like structure. Our method is able to differentiate and visualise these.

The algorithm appears to work fast in practice. However, for data for which not all triplets can be found accurately or for which there are many reticulations that do not fit into a level-2 network, it might only be possible to find a phylogenetic network for a subset of the taxa. However, even in such cases, the constructed level-2 networks could give a better representation of reality than a tree or a level-1 network.

4.7 Minimising Reticulations

4.7.1 Constructing Level-1 Networks with a Minimum Number of Reticulations

This section proposes a dynamic programming algorithm for solving MinRL-1 on dense triplet sets:

Minimum Reticulation Level-1 Network on Dense Triplet Sets (MinRL-1D)

Input: A dense triplet set T.

Output: A level-1 network consistent with all triplets in T (if such a network exists) and containing a minimum number of reticulations over all such networks.

To describe the algorithm some more definitions are needed. We call a cycle containing the root a *highest cycle* and a reticulation in such a cycle a *highest reticulation*. The set $BHR(N)$ is used to denote the set of leaves in network N that is below a highest reticulation. If the root of N is not in a cycle then $BHR(N) = \emptyset$. Recall that an SN-set is *nontrivial* if it does not contain all leaves and that an SN-set S is *maximal under restriction* R if there is no nontrivial SN-set satisfying restriction R that is a strict superset of S.

Recall the following definition from Section 4.6.1. For a triplet set T and a partition $\mathcal{C} = \{C_1, \dots, C_q\}$ of $L(T)$, define the *induced* triplet set $T\nabla\mathcal{C}$ as the set of triplets $C_iC_j|C_k$ such that there exist $x \in C_i$, $y \in C_j$, $z \in C_k$ with $xy|z \in T$ and i, j and k all distinct.

The intuition behind the algorithm is as follows. Given a dense triplet set T, the algorithm considers all SN-sets of T from small to large and computes an optimal solution N_S for each SN-set S, based on optimal solutions for included SN-sets. The algorithm considers both the possibility that the root of an optimal network is contained in a cycle and the possibility that there are two cut-arcs leaving the root. In the latter case, there are two SN-sets S_1 and S_2 that are maximal under the restriction that they are a subset of S. Hence, if there are two such SN-sets, then the algorithm constructs a candidate for N_S by creating a new root connected to the roots of two optimal networks for S_1 and S_2.

The other possibility is that the root of an optimal network N_S is contained in some cycle. For this possibility the algorithm tries each SN-set as $BHR(N_S)$: the set of leaves below the highest reticulation in N_S. The sets of leaves below other highest cut-arcs can then be found using the property outlined in Lemma 4.17 below. Subsequently, an induced set of triplets is computed, where each set of leaves below a highest cut-arc is replaced by a single meta-leaf. A candidate network is constructed by computing a simple level-1 network (as in [Jan06a]) and replacing each meta-leaf S_i by an optimal network N_{S_i} for the corresponding subset of the leaves. An optimal network N_S can then be found by selecting a network with a minimum number of reticulations over all computed networks.

Our algorithm, called MARLON (Minimum Amount of Reticulation Level One Network), is described in Algorithm 4. For an SN-set S, we use $f(S)$ to denote the minimum number of reticulations in any level-1 network consistent with $T|S$. In addition, $g(S, S')$ denotes the minimum number of reticulations in any level-1 network N consistent with $T|S$ with $BHR(N) = S'$. The algorithm first computes the optimal number of reticulations. Then a network with this number of reticulations is constructed using backtracking.

To show that the algorithm indeed computes an optimal solution we need the following crucial property of optimal level-1 networks.

Lemma 4.17. *If there exists a solution N to* MinRL-*1*D*, then there also exists an optimal solution N', where the sets of leaves below highest cut-arcs equal either (i) $BHR(N')$ and the SN-sets that are maximal under the restriction that they do not contain $BHR(N')$, or (ii) the maximal SN-sets (if $BHR(N') = \emptyset$).*

Proof. If $BHR(N) = \emptyset$ then there are two highest cut-arcs and the sets below them are the maximal SN-sets. Otherwise, the root of N is part of a cycle. We prove the following.

Algorithm 4 MARLON (Minimum Amount of Reticulation Level One Network)

```
1: compute the set $SN$ of SN-sets of $T$
2: for $i = 1 \dots n$ do
3:   for each $S$ in $SN$ of cardinality $i$ do
4:     for each $S' \in SN$ with $S' \subset S$ do
5:       let $\mathcal{C}$ contain $S'$ and all SN-sets that are maximal under the restriction that they are a subset of $S$ and do not contain $S'$
6:       if $T \nabla \mathcal{C}$ is consistent with a simple level-1 network then
7:         $g(S, S') := 1 + \sum_{X \in \mathcal{C}} f(X)$
8:     if there are exactly two SN-sets $S_1, S_2 \in SN$ that are maximal under the restriction that they are a strict subset of $S$ then
9:       $g(S, \emptyset) := f(S_1) + f(S_2)$ ($\mathcal{C} := \{S_1, S_2\}$)
10:    $f(S) := \min g(S, S')$ over all computed values of $g(S, \cdot)$
11:    store the optimal $\mathcal{C}$ and the corresponding simple level-1 network
12: construct an optimal network by backtracking.
```

Claim 4.7. *Let S be an SN-set. Either S equals a (sub)set of the leaves below a highest cut-arc or there exists a directed path P ending in the highest reticulation or in one of its parents, such that S equals the set of leaves that are below a highest cut-arc with its tail on P.*

Proof. Assume that an SN-set S does not equal a (sub)set of the leaves below a highest cut-arc. It follows that S equals the union of sets of leaves below several highest cut-arcs. Indeed, if there are leaves $x, z \in S$ below distinct highest cut-arcs then for any leaf y below any of these cut-arcs holds that $xy|z \in T$ and hence that $y \in S$. Now observe that no two leaves in S have the root as their lowest common ancestor, since this would imply that *all* leaves are in S. It now follows that there exists a directed path P on the highest cycle such that all leaves in S are below a highest cut-arc with its tail on P. Let P be a minimal such path. We now argue that all leaves below a highest cut-arc with tail on P are in S. If this would not be the case, then there would be leaves x, z, y below highest cut-arcs with tails respectively p_1, p_2, p_3 that are on P (in this order) with $x, y \in S$ and $z \notin S$. However, this would lead to a contradiction because then the triplet $xy|z$ is not consistent with N, whilst $yz|x$ and $xz|y$ are not in T since S is an SN-set. It remains to prove that P ends in either the highest reticulation or in one of its parents. Assume that this is not true, then there exists a vertex v on the interior of a path from the last vertex of P to the highest reticulation. Consider some leaf $z \notin S$ below the highest cut-arc with v as tail and some leaves $x, y \in S$ below distinct highest cut-arcs with tails on P. Then this again leads to a contradiction because $xy|z$ is not consistent with N. □

To prove the lemma, consider a maximal SN-set S that is not equal to the set of leaves below a single highest cut-arc. By the maximality of S, it cannot equal a strict subset of the leaves below a highest cut-arc. Thus, from the above claim it follows that there exists a path P such that S equals the set of leaves that are below a highest cut-arc with its tail on P. First suppose that P ends in a parent of the highest reticulation. In this case we can modify the network by putting S below a single cut-arc, without increasing the number of reticulations. To be precise, if p and p' are the first and last vertex of P respectively and r is the highest reticulation, then we subdivide the arc entering p by a new vertex v, add a new arc (v, r), remove the arc (p', r) and suppress p', since it now has indegree and outdegree both equal to one. It is not too difficult to see that the resulting network is still consistent with T.

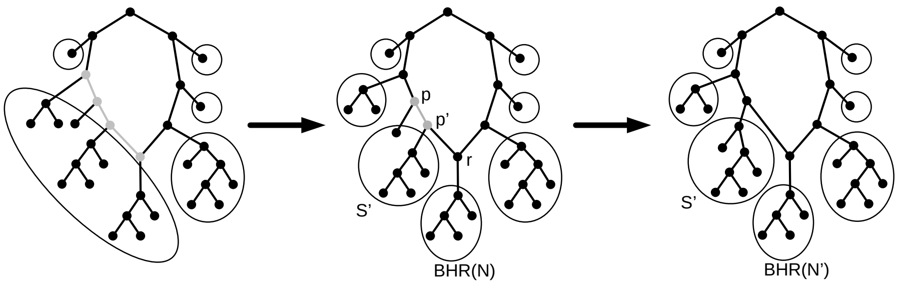


Figure 4.22: Visualisation of the proof of Lemma 4.17. From the maximal SN-sets (encircled in the network on the left) to the sets of leaves below highest cut-arcs (encircled in the network on the right). Remember that all arcs are directed downwards.

Now suppose that P ends in the highest reticulation. The sets of leaves below highest cut-arcs are all SN-sets (as is always the case, see page 77). One of these sets is equal to $BHR(N)$. Suppose that another such set X is *not* maximal under the restriction that it does not contain $BHR(N)$. Then X is a strict subset of a nontrivial SN-set S' that is maximal under the restriction that it does not contain $BHR(N)$. We apply Claim 4.7 to S'. Observe that S' cannot be equal to a (sub)set of the leaves below a highest cut-arc, since it is a strict superset of X. Thus, S' equals the set of leaves that are below a highest cut-arc with a tail on a path P' on the highest cycle. Moreover, since S' does not contain $BHR(N)$, P' does not end in the highest reticulation, but in one of its parents. Thus, the procedure from the previous paragraph can be used to put S' below a highest cut-arc.

The lemma now follows from the following. If there exists a solution to MinRL-1D, then there exists an optimal solution to MinRL-1D. After applying the modifications described above to this optimal solution, for each maximal SN-set S, the sets of leaves below highest cut-arcs in the resulting network N' are indeed equal to $BHR(N')$ and the SN-sets that are maximal under the restriction that they do not contain $BHR(N')$.

An example is given in Figure 4.22. In the network N on the left one maximal

SN-set equals the set of leaves below the grey path. In the middle is the same network, but now we encircled $BHR(N)$ and the SN-sets that are maximal under the restriction that they do not contain $BHR(N)$. There is still an SN-set (S') below several highest cut-arcs (with tails on the grey path). However, in this case the network can be modified by putting S' below a single cut-arc, without increasing the number of reticulations. This gives the network N' on the right, where the sets of leaves below highest cut-arcs are indeed equal to $BHR(N')$ and the SN-sets that are maximal under the restriction that they do not contain $BHR(N')$. $\square$

Theorem 4.10. *Given a dense set of triplets T, algorithm MARLON constructs a level-1 network that is consistent with T (if such a network exists) and has a minimum number of reticulations in $O(n^5)$ time.*

Proof. The proof is by induction on the size i of S. Suppose that N is an optimal level-1 network consistent with $T|S$, with the property described by Lemma 4.17. If $BHR(N) = \emptyset$ then the sets of leaves below highest cut-arcs are the two maximal SN-sets S_1 and S_2. In this case $f(S)$ can be computed by adding up the $f(S_1)$ and $f(S_2)$. Otherwise, it follows from Lemma 4.17 and the observation that $BHR(N)$ has to be an SN-set, that at some iteration the algorithm will consider the collection $\mathcal{C}$ of sets of leaves below the highest cut-arcs of N. In this case the number of reticulations can be computed by adding one to the sum of the values $f(X)$ over all $X \in \mathcal{C}$. This is because the network N consists of a (highest) cycle, connected to optimal networks for the different $X \in \mathcal{C}$. By induction, all values of $f(X)$ for $|X| < i$ have been computed correctly and correctness of the algorithm follows. The number of SN-sets is $O(n)$ because any two SN-sets are either disjoint or one is included in the other [Jan06b, Lemma 8]. The SN-sets can be found in $O(n^3)$ time by computing the SN-tree [Jan06a]. From this SN-tree it is possible to construct the SN-sets that are maximal under the restriction that they are contained in a specific SN-set and/or do not contain another specific SN-set, in $O(n)$ time. The triplet set $T\nabla\mathcal{C}$ can be computed and simple level-1 networks can be found in $O(n^3)$ time [Jan06a]. These computations are repeated $O(n^2)$ times: for all $S \in SN$ and all $S' \in SN$ with $S' \subset S$. Therefore, the total running time is $O(n^5)$. $\square$

MARLON has been implemented, tested and made publicly available [MAR08]. For example the network in Figure 4.23 with 80 leaves and 13 reticulations could be constructed by MARLON (from a computer generated triplet set) in less than six minutes on a Pentium IV 3 GHz PC with 1 GB of RAM.

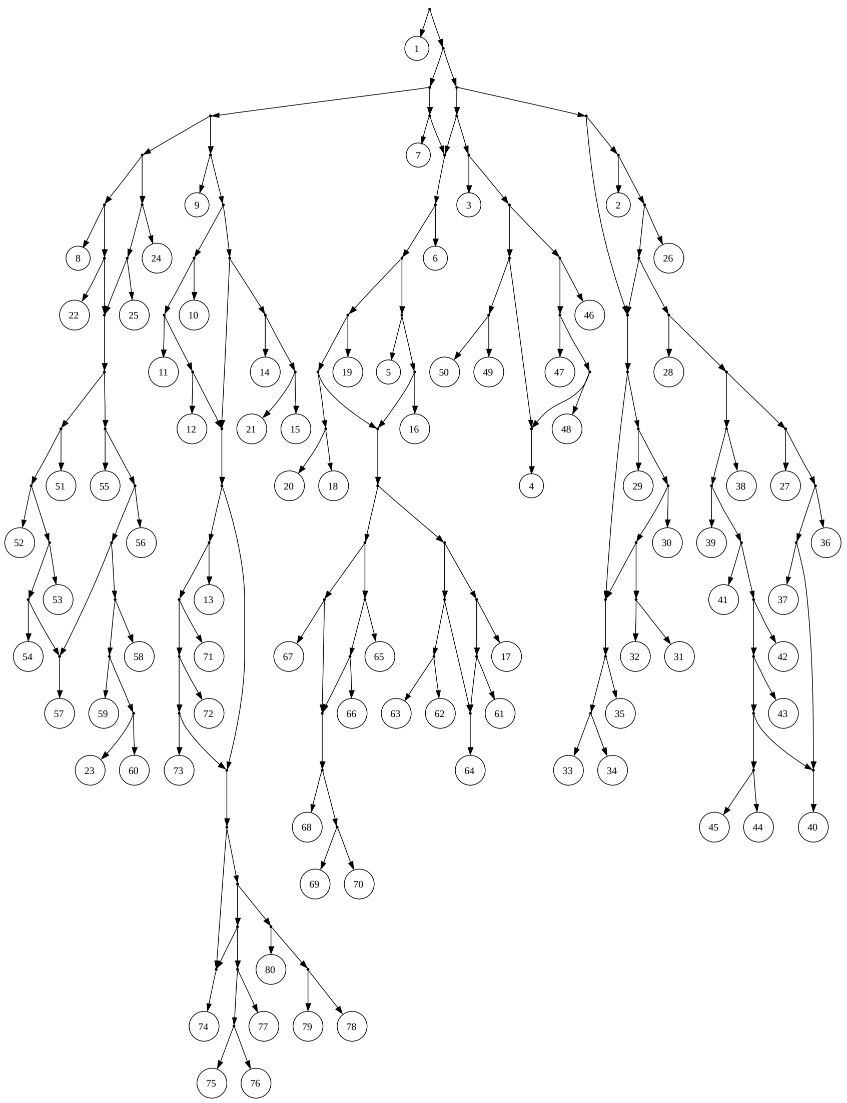


Figure 4.23: Example of a network constructed by MARLON.

4.7.2 Constructing a Level-2 Network with a Minimum Number of Reticulations

This section extends the approach from Section 4.7.1 to level-2 networks. We describe how one can find a level-2 network consistent with a dense input triplet set containing a minimum number of reticulations, or decide that such a network does not exist, in polynomial time.

The general structure of the algorithm is the same as in the level-1 case. We loop through all SN-sets S from small to large and compute an optimal solution N_S for that SN-set, based on previously computed optimal solutions for included SN-sets. For each SN-set we still consider, like in the level-1 case, the possibility that there are two cut-arcs leaving the root of N_S and the possibility that this root is in a biconnected component with one reticulation. However, now we also consider a third possibility, that the root of N_S is in a biconnected component containing two reticulations.

In the construction of biconnected components with two reticulations, we use the notion of "non-cycle-reachable"-arc, or ncr-arc for short, introduced in Section 4.4. An arc $a = (u, v)$ is an *ncr-arc* if u is not in a cycle and no ancestor of u is in a cycle. These ncr-arcs will be used to combine networks without increasing the network level. In addition, we use the notion *highest biconnected component* to denote the nontrivial biconnected component containing the root (if it exists).

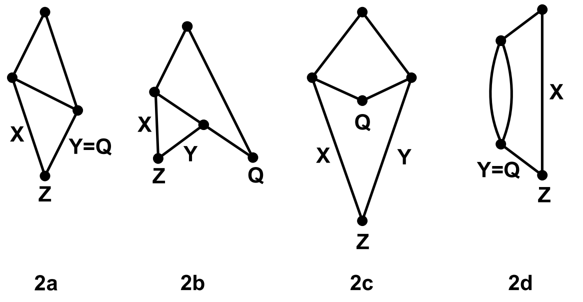


Figure 4.24: The four possible structures of a biconnected component containing two reticulations.

Our complete algorithm, called MARELET (Minimum Amount of REticulation LEvel Two network), is described in detail in Algorithm 5. To get an intuition of why the algorithm works, consider the four possible structures of a biconnected component containing two reticulations displayed in Figure 4.24. Let X, Y, Z and Q be the sets of leaves indicated in Figure 4.24 in the graph that displays the form of the highest biconnected component of N_S. Observe that after removing Z in each case X, Y and Q become sets of leaves below cut-arcs and hence SN-sets (w.r.t $T|(S \setminus Z)$). In cases 2a, 2b and 2c the highest biconnected component becomes a cycle, Q the set of leaves below the highest reticulation and X and Y sets of leaves below highest cut-arcs. We will first describe the approach for these cases and show later how a similar technique is possible for case 2d.

Our algorithm loops through all SN-sets that are a subset of S and will hence at some iteration consider the SN-set Z. The algorithm removes the set Z

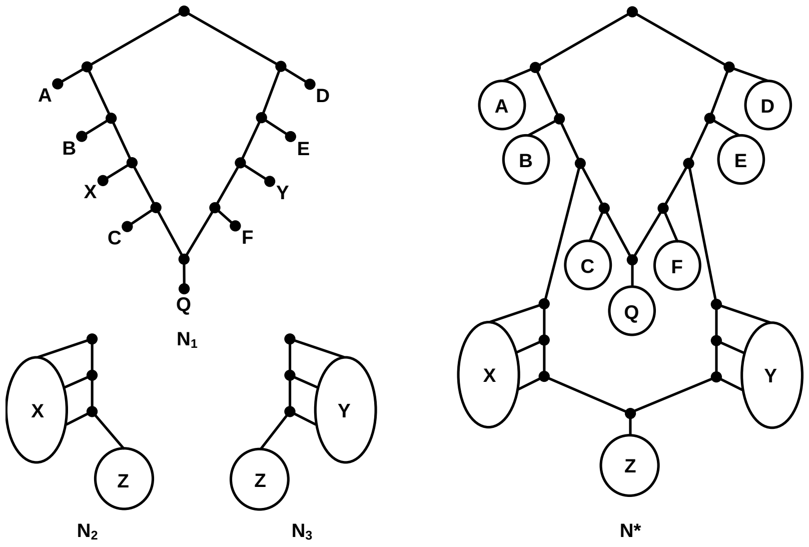


Figure 4.25: Example of the construction of network N^* from N_1, N_2 and N_3.

and computes the SN-sets of $T|(S \setminus Z)$. The sets of leaves below highest cut-arcs (in some optimal solution, if one exists) are now equal to X, Y, Q and the SN-sets that are maximal under the restriction that they do not contain X, Y or Q (by the same arguments as in the proof of Lemma 4.17). Therefore, the algorithm tries each possible SN-set for X, Y and Q and in one of these iterations it will correctly determine the sets of leaves below highest cut-arcs. Then the algorithm computes the induced set of triplets, where each set of leaves below a highest cut-arc is replaced by a single meta-leaf. All simple level-1 networks consistent with this induced set of triplets are obtained by the algorithm in [Jan06a]. Our algorithm loops through all these networks and does the following for each simple level-1 network N_1. Each meta-leaf V, not equal to X or Y, is replaced by an optimal network N_V, which has been computed in a previous iteration. To include leaves in Z, X and Y, we compute an optimal network N_2 consistent with $T|(X \cup Z)$ and an optimal network N_3 consistent with $T|(Y \cup Z)$ where in both networks Z is the set of leaves below an ncr-arc. Then we combine these three networks into a single network like in Figure 4.25. A new reticulation is created and Z becomes the set of leaves below this reticulation. Finally, we check for each constructed network whether it is consistent with $T|S$. As we will show, a network with the minimum number of reticulations over all constructed networks that are consistent with $T|S$ is an optimal solution N_S for this SN-set.

Now consider case 2d. Suppose we remove Z and replace X, Y (=Q) and each SN-set of $T|(S \setminus Z)$ that is maximal under the restriction that it does not contain X or Y by a single leaf. Then the resulting network consists of a path

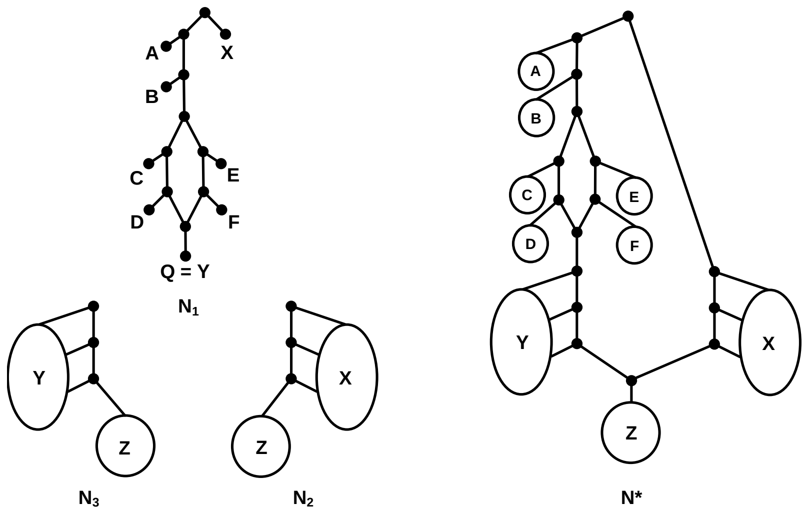


Figure 4.26: Example of the construction of network N^* from N_1, N_2 and N_3 in case 2d.

and a simple level-1 network below this path, in which X is a child of the root and Q the child of the reticulation and each vertex of the path has a leaf as child. Such a network can easily be constructed and subsequently one can use the same approach as in cases 2a, 2b and 2c. See Figure 4.26 for an example of the construction in case 2d.

Theorem 4.11. *Given a dense triplet set T, Algorithm MARELET constructs a level-2 network consistent with T (if such a network exists) that has a minimum number of reticulations in $O(n^9)$ time.*

Proof. Consider some SN-set S and assume that there exists an optimal solution N_S consistent with $T|S$. The proof is by induction on the size of S. The induction hypothesis is that the network N_U constructed by the algorithm is optimal for all SN-sets U with $|U| < |S|$. If the root of N_S is not in a cycle or in a biconnected component with one reticulation then the algorithm constructs an optimal solution by the proof of Theorem 4.10 and the induction hypothesis. Hence we assume from now on that the highest biconnected component of N_S contains two reticulations.

Consider the four graphs in Figure 4.24. These are the simple level-2 generators, defined in Section 4.2. Any biconnected component containing two reticulations is a subdivision of one of these graphs by Lemma 4.1. Recall that a *side* of a generator is either an arc or a vertex with indegree 2 and outdegree 0. Suppose that the highest biconnected component of a network N is a subdivision of a generator G. We say that a leaf x *is on side* S of G if there exists a cut-arc (u, v) in N such that u is on the subdivision of S (if S is an arc) or u is a reticulation (if S is a reticulation), and there is a directed path from v to x (possibly $v = x$). Furthermore, we identify the side S with the set of leaves that are on side S.

In each generator in Figure 4.24, X, Y, Z and Q are sides of the generator. Thus, if the highest biconnected component of N_S is a subdivision of a generator $G \in \{2a, 2b, 2c, 2d\}$ then we identify X (Y, Z, Q respectively) with the set of leaves in N_S on side X (Y, Z, Q respectively) of G. We assume for now that the highest biconnected component of N_S is not a subdivision of generator $2d$.

To find an optimal network consistent with $T|(X \cup Z)$ (or $T|(Y \cup Z)$) such that Z is below an ncr-arc we can use the following approach. In such a network there are two cut-arcs leaving the root. The sets of leaves below these cut-arcs are (as always) SN-sets and Z is fully contained in one of them. Thus, if it is possible to construct a network of the desired type then there are two maximal SN-sets. If one of them contains Z as a strict subset then we create a network for this set recursively. For maximal SN-sets that are equal to or disjoint from Z, we use the optimal networks computed in earlier iterations. Finally, we create a new root and connect it to the roots of the two networks for the two maximal SN-sets.

Given a network N' and a set of leaves L' below a cut-arc (u, v), we denote by $N' \setminus L'$ the network obtained by removing v and all vertices reachable from v from N' and tidying up the resulting graph.

Claim 4.8. *There exists an optimal solution N'_S to* MinRL-*2*D*, such that the sets of leaves below highest cut-arcs of $N'_S \setminus Z$ are X, Y, Q and the SN-sets of $T|(S \setminus Z)$ that are maximal under the restriction that they do not contain X, Y or Q.*

Proof. The highest biconnected component of $N_S \setminus Z$ contains just one reticulation and the same arguments can be used as in the proof of Lemma 4.17 to transform N_S into N'_S. □

Let N'_S be a network with the property described in the claim above and $\mathcal{C}$ the collection of sets of leaves below highest cut-arcs of $N'_S \setminus Z$. At some iteration the algorithm will consider this set $\mathcal{C}$. Let T' equal $T|(S \setminus Z)$. If we replace in $N'_S \setminus Z$ each set of leaves below a highest cut-arc by a single leaf, then we obtain a simple level-1 network consistent with $T' \nabla \mathcal{C}$ (under the assumption that the highest biconnected component of N'_S is not a subdivision of generator $2d$). The algorithm considers all networks of these types, so in some iteration it will consider the right one. Let N^* be the network constructed by the algorithm in this iteration. It remains to prove that N^* (i) is consistent with $T|S$, (ii) contains a minimum number of reticulations and (iii) is a level-2 network.

Algorithm 5 MARELET (Minimum Amount of REticulation LEvel Two network)

```
1: - compute the set $SN$ of SN-sets of $T$
2: for $i = 1 \ldots n$ do
3:     for each $S$ in $SN$ of cardinality $i$ do
4:         for each $S' \in SN$ with $S' \subset S$ do
5:             - let $\mathcal{C}$ contain $S'$ and all SN-sets that are maximal under the restriction that they are a subset of $S$ and do not contain $S'$
6:             if $T \nabla \mathcal{C}$ is consistent with a simple level-1 network $N_1$ then
7:                 - construct $N^*$ from $N_1$ by replacing each leaf $V$ by an optimal network $N_V$ constructed in a previous iteration
8:                 - $g(S, S')$ is the number of reticulations in $N^*$
9:         if there are exactly two SN-sets $S_1, S_2 \in SN$ that are maximal under the restriction that they are a strict subset of $S$ then
10:            - $N^*$ consists of a root connected to the roots of optimal networks $N_{S_1}$ and $N_{S_2}$ that have been constructed in previous iterations
11:            - $g(S, \emptyset)$ is the number of reticulations in $N^*$
12:        for each $Z \in SN$ with $Z \subset S$ do
13:            - $T' := T|(S \setminus Z)$
14:            - compute the set $SN'$ of SN-sets of $T'$
15:            for each $X, Y, Q \in SN'$ do
16:                - $\mathcal{C}$ is the collection consisting of $X, Y, Q$ and all SN-sets in $SN'$ that are maximal under the restriction that they do not include $X, Y$ or $Q$
17:                - construct an optimal network $N_2$ consistent with $T|(X \cup Z)$ such that $Z$ is the set of leaves below an ncr-arc $(u, v)$
18:                - construct an optimal network $N_3$ consistent with $T|(Y \cup Z)$ such that $Z$ is the set of leaves below an ncr-arc $(u', v')$
19:                - construct the set $\mathcal{N}$ of all simple level-1 networks consistent with $T' \nabla \mathcal{C}$
20:                - add to $\mathcal{N}$ all networks consistent with $T' \nabla \mathcal{C}$ that consist of a path ending in a simple level-1 network, with $X$ a child of the root, $Q$ the child of the reticulation; and with a leaf below each internal vertex of the path
21:                for each network $N_1 \in \mathcal{N}$ do
22:                    - construct $N^*$ from $N_1$ by doing the following: replace $X$ by $N_2$, $Y$ by $N_3$ and each other leaf $V$ by an optimal network $N_V$ constructed in a previous iteration, then subdivide $(u, v)$ into $(u, w)$ and $(w, v)$, delete everything below $u'$ and add an arc $(u', w)$
23:                    if $N^*$ is consistent with $T|S$ then
24:                        - $h(S, X, Y, Z, Q)$ is the number of reticulations in $N^*$
25:        - $f(S)$ is the minimum of all computed values of $g(S, S')$ and $h(S, X, Y, Z, Q)$
26:        - store network $N_S$, which is a network $N^*$ attaining the minimum number $f(S)$ of reticulations
```

To prove that N^* is consistent with $T|S$, consider any triplet $xy|z \in T|S$. First suppose x, y and z are all in Z or all in the same set of $\mathcal{C} \setminus \{X, Y\}$. Then x, y and z are elements of some SN-set S' with $|S'| < |S|$. Triplet $xy|z$ is consistent with the subnetwork $N_{S'}$ by the induction hypothesis and hence with N^*.

Now suppose that x, y and z are all in $X \cup Z$ (or all in $Y \cup Z$). Consider the construction of the network N_2 consistent with $X \cup Z$ such that Z is below an ncr-arc. First suppose that at some level of the recursive construction of N_2 there are two maximal SN-sets, each containing leaves from $\{x, y, z\}$. Then it follows that x and y are in one maximal SN-set and z in the other one, by the definition of SN-set, and hence that $xy|z$ is consistent with N_2 (as x and y are on one side of the root of some subnetwork of N_2 and z is on the other side). Otherwise, x, y and z are all in some subnetwork N_U with $|U| < |S|$ and $xy|z$ is consistent with this subnetwork (by the induction hypothesis) and hence with N^*.

Now consider any other triplet $xy|z \in T|S$, which thus contains leaves that are below at least two different highest cut-arcs. Observe that the highest biconnected components of N^* and N' are identical; the only differences between these networks occur in the subnetworks below highest cut-arcs. Therefore $xy|z$ is consistent with N^* since it is consistent with N'.

To show that N^* contains a minimum number of reticulations consider any set S' of leaves below a highest cut-arc $a = (u, v)$ of N^*. The subnetwork $N_{S'}$ rooted at v contains a minimum number of reticulations by the induction hypothesis. Hence N^* contains at most as many reticulations as N', which is an optimal solution.

To see that N^* is a level-2 network, note that in the networks N_2 and N_3 constructed by the algorithm, Z is the set of leaves below an ncr-arc. This implies that none of the potential reticulations in these networks end up in the highest biconnected component of N^*. Therefore, this biconnected component contains exactly two reticulations. All other biconnected components of N^* also contain at most two reticulations by the induction hypothesis. We thus conclude that N^* is a level-2 network.

For the proof of correctness it remains to consider the case that the highest biconnected component of N_S is a subdivision of generator $2d$. Let $\mathcal{C}$ be the collection containing X, $Y (= Q)$ and the SN-sets of $T|(S \setminus Z)$ that are maximal under the restriction that they do not contain X or Y. Similar to Claim 4.8, there exists an optimal solution N'_S such that the elements of $\mathcal{C}$ are all sets below (not necessarily highest) cut-arcs in $N'_S \setminus Z$. If we replace in $N'_S \setminus Z$ each element of $\mathcal{C}$ by a single leaf then we obtain a path and a simple level-1 network below this path, in which X is a child of the root and Q the child of the reticulation and each vertex of the path has a leaf as child. All such networks can easily be constructed by deleting Q, constructing a tree and trying to connect Q to it in all possible ways. These networks are constructed in line 20 of the algorithm. The same arguments as before can be used to show that the constructed network is also in this case an optimal solution to MinRL-2D.

To conclude the proof we analyse the running time of the algorithm. The number of SN-sets is $O(n)$ and hence there are $O(n)$ choices for each of S, X, Y, Z and Q. For each combination of S, X, Y, Z and Q there will be $O(n)$ networks N_1 constructed and for each of them it takes $O(n^3)$ time to check if the resulting network N^* is consistent with $T|S$ (in line 23). Hence the overall time complexity is $O(n^9)$. □

4.7.3 Simulations

To test the relevance of the constructed networks we applied MARLON to simulated data. The main advantage of using simulated data is that it enables us to compare output networks with the "real" network. We repeated the following experiment for various level-1 networks, which we in turn assumed to be the "real" network. Given such an input level-1 network, we used the program Seq-Gen [Ram97] to simulate sequences that could have evolved according to that network, if we see the network as a recombinant phylogeny. We assumed that the first block (e.g. a gene) of each sequence evolved according to a certain phylogenetic tree, the second block evolved according to a different tree and that the level-1 network contains a subdivision of each of these trees.

The sequences have been generated as follows. For each simulation, our input to Seq-Gen consisted of two trees T_1 and T_2. For each reticulation of the level-1 network, T_1 uses just one of the incoming arcs and T_2 uses the other one. This makes sure that each arc of the network is used by at least one of the two trees. Seq-Gen was used with the F81 model of nucleotide substitution to generate sequences of 4000 base pairs each of which the first 2000 base pairs evolved according to T_1 and the last 2000 base pairs evolved according to T_2.

From these simulated sequences we computed a set of triplets as follows. We assume that for one sequence it is known that it is only distantly related to the others. This is called the *outgroup* sequence. In each level-1 network that we used as input network, the root has a leaf as child. This leaf corresponds to the outgroup sequence. For each combination of three sequences, plus the outgroup sequence, we computed a phylogenetic tree using the maximum likelihood method PHYML [Gui03]. Each output tree of PHYML can be rooted by using that the outgroup must be a child of the root. This leads to a dense triplet set, which we used as input to MARLON.

All simulations gave similar results. Here we describe the results for one specific "real" level-1 network, displayed in Figure 4.27. We obtained the simulated triplet set T^* based on this network by the procedure described above. For this triplet set MARLON constructed the output network in Figure 4.28. The constructed network is very similar to the input network (which we assumed to be the "real" network). Both networks have four reticulations and also the branching structure is almost identical. The only differences are all of the following type. The output network contains some subnetworks rooted below a parent of a reticulation, where the input network has the corresponding leaves scattered along a path ending in the parent of the reticulation. For example in Figure 4.27 the leaves 37, 38, 39, 40 are below a path on a cycle consisting of three vertices. However, in the output network in Figure 4.28 these leaves are below a single vertex on the cycle.

For comparison, T-REX has been applied to directly construct a phylogenetic network from the same simulated sequences. It was used with the command “construct a reticulogram” with the Jukes-Cantor model of evolution. Also the number of reticulations in the input network was given to T-REX; it was specifically asked to construct a network with that number of reticulations (otherwise it would create far more reticulations than MARLON). For the input network from Figure 4.27 the output of T-REX is displayed in Figure 4.29. This test has been repeated three times and the networks have been compared to the input network by calculating the μ-distance [Car07]. This distance can be used because all level-1 networks as well as all networks generated by T-REX in these simulations were “tree-child” phylogenetic networks [Car07]. Our algorithm performs significantly better than T-REX, with an average μ-distance of 14.0 to the input network, whilst T-REX has an average distance of 39.7 to the input network. This comparison might not be completely fair since the input network is a level-1 network, the kind of network that MARLON is specifically designed for. On the other hand, T-REX was supplied with prior information about the number of reticulations it had to create. In any case, it is encouraging that all networks constructed by MARLON were almost identical to the input networks, except for some small differences of a well-defined type.

We conclude that MARLON correctly constructs level-1 networks and works very fast. For simulated data the produced networks are very close to the “real” networks used to generate the simulated sequences. When using real data we expect the amount of incorrect triplets to be larger and hence the results possibly less impressive. In addition, real data sets will not always originate from a level-1 network, in which case MARLON might not be able to compute a solution. This problem can partly be solved by implementing the method presented in Section 4.7.2, which constructs level-2 networks. However, the main conclusion to be drawn from the experiments is that, if the data is good enough (and originates from a level-1 network), our method is indeed able to produce good estimates of evolutionary histories. This for example shows that, when a set of triplets is computed from sequence data, sufficient information is retained to be able to reconstruct the phylogenetic network accurately. In addition, MARLON provides a very fast method to combine these triplets into a phylogenetic network.

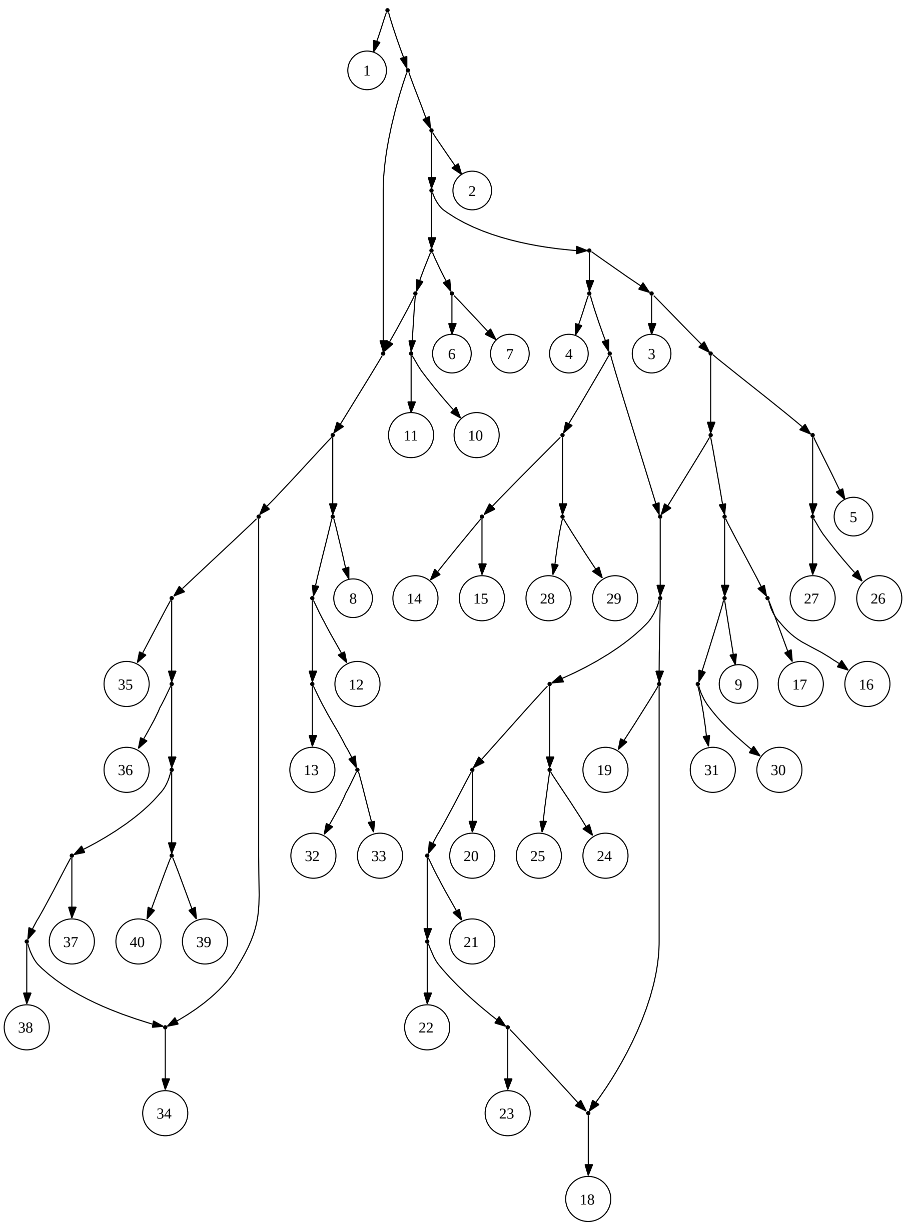


Figure 4.27: The level-1 network on which the simulated triplet set T^* is based.

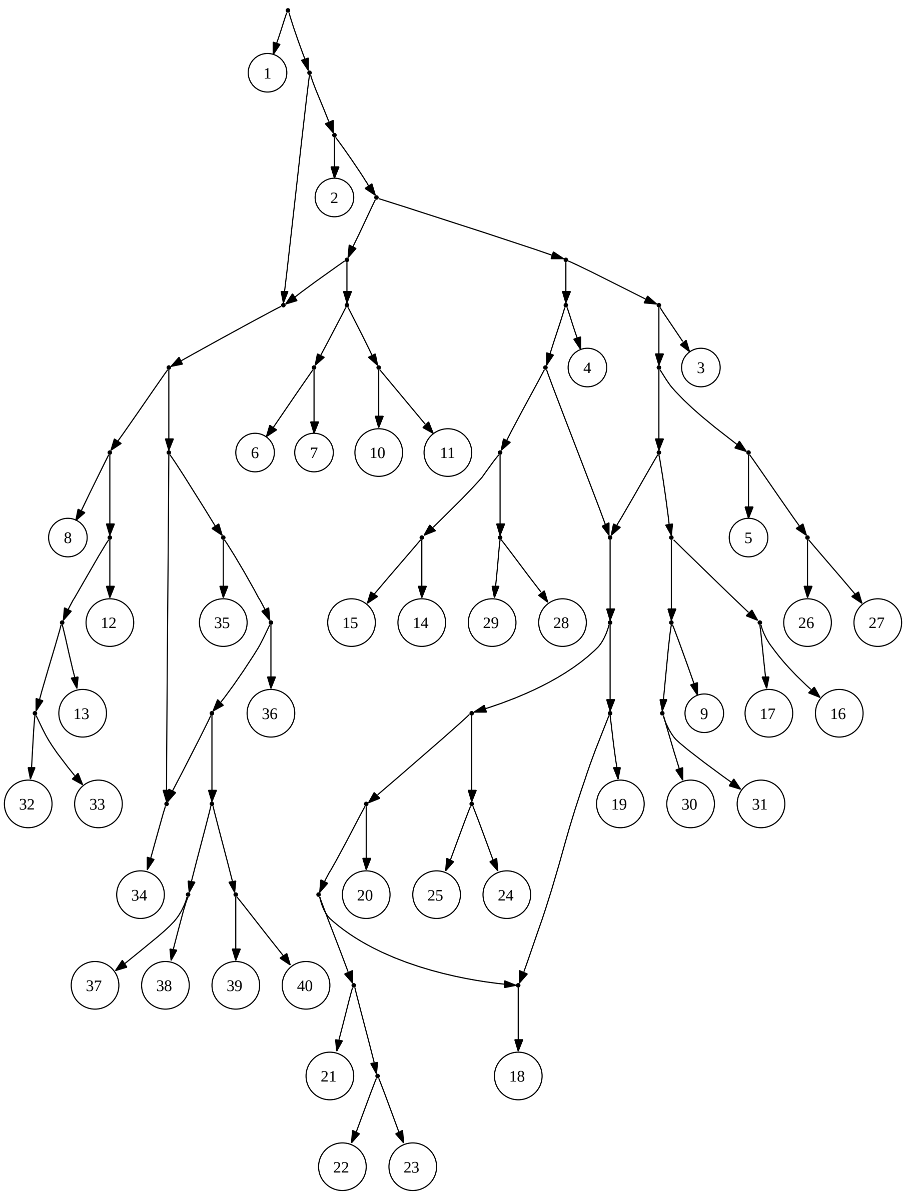


Figure 4.28: The network constructed by MARLON for the simulated triplet set T^*.

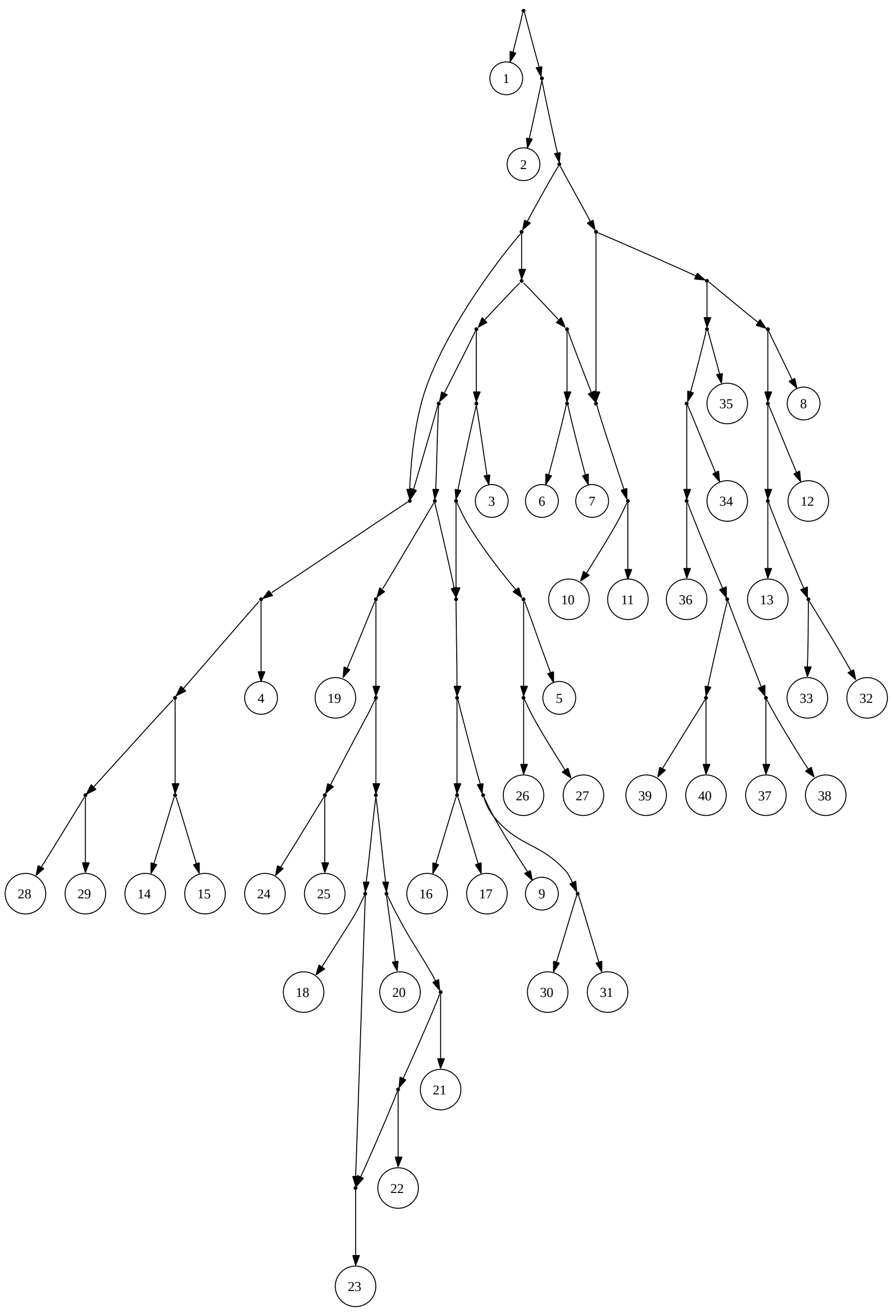


Figure 4.29: The network constructed by TREX for the simulated sequences.

4.8 Constructing Networks Consistent with Precisely the Input Triplet Set

In this section the problem is considered which, given a triplet set T, asks for a level-k network N that is consistent with precisely those triplets in T (if such a network exists). Such a network is said to *reflect* T. It will be shown that this problem is polynomial-time solvable for each fixed k. In addition, the described algorithm even finds a solution that minimises both the level and number of reticulations used. This algorithm thus solves the following problem.

MINIMUM REFLECTIVE LEVEL-k NETWORK (REFL-k)

Input: A triplet set T.

Output: A level-k network N that reflects T (if such a network exists) and, ranging over all such networks, minimises both the level and the number of reticulations used.

Recall that we use $T(N)$ to denote the set of all triplets consistent with a network N. Thus, a network N reflects a triplet set T if and only if $T(N) = T$. A triplet set T is called *reflective* if there exists a network N that reflects T. Note that, $T(N) = T(N')$ is possible for distinct networks $N \neq N'$. There are, for example, several distinct simple level-2 networks that reflect the triplet set $\{xy|z, xz|y, zy|x\}$. As a corollary of Theorem 4.7 we obtain the following.

Corollary 4.1. *For fixed k it is possible to generate, given a triplet set T, all simple level-k networks N that reflect T in time $O(n^{3k+3})$.*

Proof. The algorithm SL-k on page 98 (or, for that matter, the simple level-1 algorithm from [Jan06a]) can easily be adapted for this purpose: we change in line 43 "network consistent with T" to "network that reflects T". The running time is unchanged because, whether we are checking consistency or reflection, the implementation of [Byr08, Lemma 2] implicitly generates $T(N')$. □

The remainder of this section shows how general reflective networks can be built by recursively constructing simple reflective networks. This will be based on a crucial property of the SN-sets of simple networks, outlined in Lemma 4.18.

For a triplet $xy|z$ and a network N, we define an *embedding* of $xy|z$ in N as any set of four internally vertex disjoint paths $(q \to x, q \to y, p \to q, p \to z)$ with $p \neq q$. We say that the vertex p is the *summit* of the embedding. Clearly, $xy|z$ is consistent with N if and only if there is at least one embedding of $xy|z$ in N.

Lemma 4.18. *Let N be any simple network. Then all the nontrivial SN-sets of $T(N)$ are singletons.*

Proof. We prove the lemma by contradiction. Assume thus that there is some SN-set S of $T(N)$ such that $1 < |S| < |L(N)|$.

Let r be the root of N. An *in-out root embedding* of a triplet $xz|y$ with $x, z \in S$ and $y \notin S$ is an embedding of $xz|y$ that has r as its summit. We begin by proving that there exist $x, z \in S$ and $y \notin S$ such that $xz|y$ has an in-out root embedding in N. Suppose that this is not true. For all $x, z \in S$ and $y \notin S$, the triplet $xz|y$ is in $T(N)$ because $T(N)$ is dense and S is an SN-set. Consider a triplet embedding $(q \rightarrow x, q \rightarrow z, p \rightarrow q, p \rightarrow y)$ with $p \neq r$ that minimises the length of a shortest directed path in N from r to p, amongst all embeddings of triplets $xz|y$ with $x, z \in S$ and $y \notin S$. Let P be any shortest directed path from r to p. Now, consider any maximal directed path Q that leaves the path P (at least one such a path exists since the root has outdegree 2). First observe that Q does not intersect with the path $p \rightarrow y$ or $p \rightarrow q$, because this would contradict the minimality of the length of P. In addition, Q does not intersect the path $q \rightarrow x$ ($q \rightarrow z$) because this would mean $y \in S$. We conclude that such a path Q either terminates at a leaf l, or re-intersects with the path P. It does not terminate at a leaf $l \notin S$ because then we obtain an embedding of $xz|l$ that has a summit closer to the root than p, contradicting the minimality of P. It does also not end at a leaf $l \in S$, because then we have that $y \in S$. We conclude that all paths outgoing from P rejoin with P in a strict ancestor of p. It follows that the last arc on the path P is a cut-arc. But this is a contradiction since N does not contain any nontrivial cut-arcs. We conclude that there exists at least one in-out root embedding in N.

Let $(q \rightarrow x, q \rightarrow z, r \rightarrow q, r \rightarrow y)$ be any in-out root embedding of a triplet $xy|z$ with $x, z \in S$ and $y \notin S$. We observe that the path $r \rightarrow y$ must contain at least one internal vertex, by the absence of nontrivial cut-arcs (Lemma 4.3). Also, at least one of $q \rightarrow x$ and $q \rightarrow y$ must contain an internal vertex, because otherwise the arc entering q would be a nontrivial cut-arc.

We now argue that there exists a *twist cover* of the path $r \rightarrow q$. This is defined as a non-empty collection $\mathcal{C}$ of undirected paths (undirected in the sense that not all arcs need to have the same orientation) where (i) all paths in $\mathcal{C}$ are arc-disjoint from the in-out root embedding, (ii) exactly one path starts at an internal vertex s of $q \rightarrow z$ or $q \rightarrow x$, (iii) exactly one path ends at an internal vertex t of $r \rightarrow y$, (iv) all other start and endpoints of the paths in $\mathcal{C}$ lie on $r \rightarrow q$ and (v) for every vertex v of the path $r \rightarrow q$ (including r and q), there is at least one path in $\mathcal{C}$ whose start- and endpoint are respectively a descendant and an ancestor of v. Property (v) is crucial because it says (informally) that every vertex on $r \rightarrow q$ is "covered" by some path that begins and ends on either side of it and is arc-disjoint from the embedding. We define a *partial* twist cover as a collection of undirected paths that satisfies all properties of a twist cover except property (v). Formally, we say that a vertex v is *covered* by a (partial) twist cover $\mathcal{C}$ if $\mathcal{C}$ contains a path X such that v lies on a directed path on the embedding from the endpoint of X to the startpoint of X. Thus, partial twist covers leave at least one vertex on $r \rightarrow q$ uncovered. The length

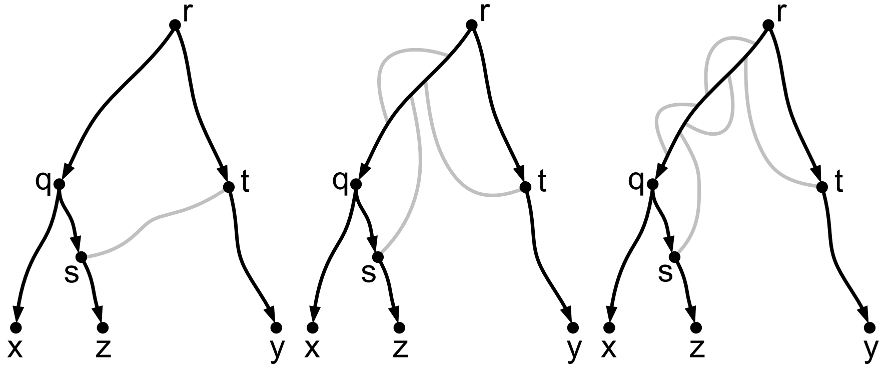


Figure 4.30: Several examples of twist covers (the grey, undirected paths) from the proof of Lemma 4.18. Note that these examples exhibit the regular, interleaved structure associated with minimum-length twist covers.

of a (partial) twist cover $\mathcal{C}$ is defined to be the sum over all paths $X \in \mathcal{C}$ of the number of arcs in X.

Suppose for contradiction that a twist cover does not exist. From the fact that neither the removal of q nor the removal of r is allowed to disconnect N, it follows that at least one partial twist cover exists. Let $\mathcal{C}$ be a partial twist cover with a minimum number of uncovered vertices. Let d be the uncovered vertex that is closest to q. If we removed d we would, by definition, disconnect the union of the paths in $\mathcal{C}$ with the in-out root embedding into a left part G and a right part H. Deletion of the vertex d does not, however, disconnect N, so there must be some path P not in $\mathcal{C}$ that begins somewhere in G and ends somewhere in H. If P has its startpoint on a path $X \in \mathcal{C}$ (where X is in G) and/or an endpoint on a path $Y \in \mathcal{C}$ (where Y is in H), then these paths can be "merged" into a new path that strictly increases the number of vertices covered. The merging occurs as follows. We take the union of the arcs in P with those in X and/or Y and discard superfluous arcs until we obtain a path that covers a strict superset of the union of the vertices covered by X and/or Y. (In particular, the fact that P begins in G and ends in H means that the vertex d becomes covered.) In this way we obtain a new partial twist-cover with fewer uncovered vertices, contradiction. If P has both its startpoint and endpoint on vertices of $r \rightarrow q$ that are not on paths in $\mathcal{C}$, then P can be added to the set $\mathcal{C}$ and this extends the number of covered vertices, contradiction. If P begins and/or ends elsewhere on the embedding, then P can be added to $\mathcal{C}$ which again increases the number of vertices covered, contradiction. (If P begins on $q \rightarrow z$ or $q \rightarrow x$, then it becomes the new property-(ii) path and the old property-(ii) path should be discarded. Symmetrically, if P ends on $r \rightarrow y$, then it becomes the new property-(iii) path and the old property-(iii) path should be discarded.)

We conclude that for every in-out root embedding there exists a twist cover,

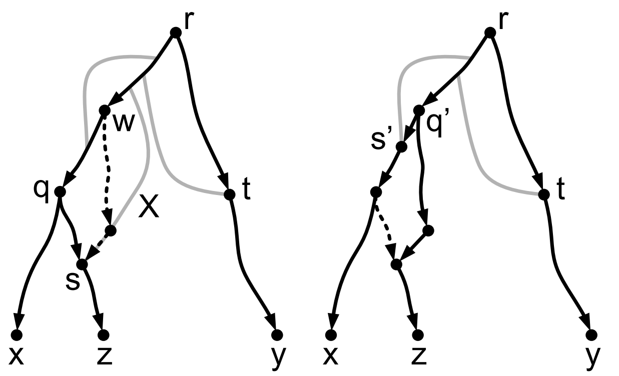


Figure 4.31: The case in the proof of Lemma 4.18 where the arc incident to s is incoming.

and in particular a minimum length twist cover.

We observe that a minimum-length twist cover $\mathcal{C}$ has a highly regular, interleaved structure. To be precise, each path $X \in \mathcal{C}$ not containing s nor t covers exactly one startpoint of a path $Y \in \mathcal{C}$ and one endpoint of a path $Z \in \mathcal{C}$ with $Y \neq Z$. Indeed, if X would cover startpoints of two paths $Y \in \mathcal{C}$ and $Y' \in \mathcal{C}$, then one of Y and Y' could be deleted to obtain a smaller twist cover. Similarly, if X would cover endpoints of two paths $Z \in \mathcal{C}$ and $Z' \in \mathcal{C}$ or if X would cover both the start- and endpoint of a single path, then one path could be deleted from the twist cover. Finally, if X covers no startpoint (no endpoint), then X can be removed from $\mathcal{C}$ since the startpoint (endpoint) of X is covered by some path in $\mathcal{C}$ and this path thus covers all vertices that X covers. For similar reasons, any two paths in a minimum-length twist cover are vertex- and arc-disjoint. In Figure 4.30 we show several examples of twist covers exhibiting this regular structure (although it should be noted that minimum-length twist covers can contain arbitrarily many paths).

Let $\mathcal{C}$ be a twist cover of minimum length ranging over all in-out root embeddings of triplets $xz|y$ with $x, z \in S$ and $y \notin S$. Let $\mathcal{E}$ denote the corresponding in-out root embedding. The high-level idea is to show that we can always find, by "walking" along the paths in $\mathcal{C}$, a new in-out root embedding and twist cover that is shorter than $\mathcal{C}$, yielding a contradiction. Let X be the path in $\mathcal{C}$ that begins at s, and consider the arc a on this path incident to s.

The first case is that this arc a is directed away from s. If X would be a directed path to $r \rightarrow q$, it would be part of a directed cycle, contradicting the acyclicity of the network. If X would be a directed path to t, this would imply that $yz|x \in T(N)$, contradicting $y \notin S$. We conclude that X is not a directed path. Continuing along X we will thus eventually traverse an arc backwards. Let a' be the first such arc. Let v be the head of this arc a'. Since v (which is a reticulation) has outdegree 1, there exists a directed path Q leaving v which eventually reaches a leaf m. If Q intersects with $r \rightarrow y$, then we have that $yz|x \in T(N)$, contradicting $y \notin S$. If Q intersects with $q \to x$, then we obtain a new in-out root embedding of $xz|y$ (with s the "q" vertex of the new embedding and the part of X up to v together with a part of Q as the new $q \to x$ path). Moreover, we obtain a new twist cover for that embedding (using only a part of X) that is shorter than $\mathcal{C}$ because at least arc a is removed from the twist cover, contradiction. If Q intersects with $q \to z$, then we obtain a new in-out root embedding of $xz|y$ (using the part of X up to v and part of Q as an alternative route to z) and a new twist cover for that embedding (using only a part of X) that is shorter than $\mathcal{C}$, contradiction. If Q does not intersect with the embedding at all, then $m \in S$ (because the triplet $mz|x$ is consistent with the network). But then we have an in-out root embedding ($s \to m$,$s \to z$,$r \to s$,$r \to y$) of the triplet $zm|y$ with twist cover (again using only a part of X) shorter than $\mathcal{C}$, contradiction.

The second case (see Figure 4.31) is when the first arc a of X is entering s. There exists some directed path R from r to s that uses a (since any vertex in any network is reachable from the root). The fact that r is part of the embedding $\mathcal{E}$ but a is not means that at some point R departs from $\mathcal{E}$. Let w be the last vertex that R has in common with $\mathcal{E}$. If w is on the path $r \to y$, then it follows that $yz|x \in T(N)$, contradicting $y \notin S$. If w is on the path $q \to x$, then this leads to a new in-out root embedding ($w \to x$,$w \to z$,$r \to w$,$r \to y$) with shorter twist cover (by removing from X the part of R after w that overlaps with X), a contradiction. If w is on the path $q \to z$, then this also leads to a new in-out root embedding (using the part of R after w, as an alternative route to z) with a shorter twist cover (again by removing from X the part of R after w that overlaps with X), a contradiction. If w lies on the path $r \to q$, we create a new in-out root embedding in which w becomes the "q" vertex of the new embedding (denoted q' in the figure) and the part of R after w becomes, together with arc a and the path from s to z over the embedding, the new $q \to z$ path. To see that we also obtain a new twist cover, note principally that paths in $\mathcal{C}$ that covered w become legitimate candidates for property-(ii) paths in the new twist cover; in the figure s' denotes the beginning of the property-(ii) path in the new cover. (Such a path can however partly overlap with R. In this case we only use the part after the last point of intersection with R.) It might be possible to discard some paths from the twist cover (in Figure 4.31, path X is discarded from the twist cover). Even if no paths from $\mathcal{C}$ are discarded, we still get a twist cover at least one arc smaller than $\mathcal{C}$, because (in particular) the first arc of X is no longer needed in $\mathcal{C}$. In any case, contradiction. □

Corollary 4.2. *Let N be a simple network. Then any network N' reflecting $T(N)$ is simple.*

Proof. If N' is not simple, then it contains a cut-arc below which at least two leaves can be found, by Lemma 4.3. Consider the set A of leaves below this cut-arc. This is an SN-set since triplets of the form $xy|z$ with $x, z \in A$

and $y \notin A$ are not in $T(N') = T(N)$. This is a contradiction because all the nontrivial SN-sets of T are singletons. □

Let T be a reflective set of triplets and N a network that reflects T. Observe that this implies that T is dense and thus that for any two SN-sets of T holds that either they are disjoint or one is contained in the other. Let $a_1, \ldots, a_q$ be the highest cut-arcs of N, let $S_1, \ldots, S_q$ be the sets of leaves below these highest cut-arcs and let $\mathcal{C}_N$ denote $\{S_1, \ldots, S_q\}$. As in Section 4.6.1, $Collapse(N)$ denotes the result of replacing everything reachable from a_i by a single leaf S_i, for $i = 1, \ldots, q$. The following observation will be critical in the proof of Lemma 4.19, which shows correctness of our algorithm MINPITS, displayed in Algorithm 6.

Observation 4.6. *(1) $Collapse(N)$ is a simple network reflecting $T\nabla\mathcal{C}_N$ and (2) $S_1, \ldots, S_q$ are the maximal SN-sets of T.*

Proof. First consider triplets $XY|Z \in T\nabla\mathcal{C}_N$. For each such triplet there exists at least one triplet $xy|z \in T$ with $x \in X, y \in Y$ and $z \in Z$. Since N is consistent with $xy|z$ it follows that $Collapse(N)$ is consistent with $XY|Z$. To show that only triplets in $T\nabla\mathcal{C}_N$ are consistent wit $Collapse(N)$, consider a triplet $XY|Z$ consistent with $Collapse(N)$. In this case all triplets $xy|z$ with $x \in X$, $y \in Y$ and $z \in Z$ are consistent with N. Since N reflects T this implies that $xy|z \in T$ and hence that $XY|Z \in T\nabla\mathcal{C}_N$. By construction, $Collapse(N)$ contains no nontrivial cut-arcs and is thus a simple network reflecting $T\nabla\mathcal{C}_N$. For (2), first observe that each set S_i is an SN-set of T since the set of leaves below a cut-arc is always an SN-set. Assume (for contradiction) that S_i is not maximal. Then there exists a maximal SN-set S of T that is a strict superset of S_i. Any maximal SN-set can be written as the union of sets of leaves below highest cut-arcs, by Lemma 4.6. Suppose (without loss of generality) that $S = S_1 \cup \ldots \cup S_r$ with $1 < r < q$ and $1 \leq i \leq r$. It follows that $\{S_1, \ldots, S_r\}$ is an SN-set of $T\nabla\mathcal{C}_N$, since the existence of a triplet $XY|Z \in T\nabla\mathcal{C}_N$ with $X, Z \subset S$, $Y \not\subset S$ would imply the existence of a triplet $xy|z \in T$ with $x, z \in S$ and $y \notin S$, while S is an SN-set. This is a contradiction since by Lemma 4.18 all SN-sets of $T\nabla\mathcal{C}_N$ are singletons. □

The following lemma shows the correctness of Algorithm 6. Recall that, given a set of triplets T, each leaf V of a network N' reflecting $T\nabla SN$ is a subset $V \subset L(T)$.

Lemma 4.19. *Let T be a reflective set of triplets and let SN be the set of maximal SN-sets of T. Let N' be a simple network of minimum level that reflects $T\nabla SN$. Then, for each leaf V of N', $T|V$ is reflective. Furthermore, replacing each leaf V of N' by a network that reflects $T|V$ and which (ranging over all networks that reflect $T|V$) simultaneously minimises both level and* *number of reticulations, yields a network N that reflects T and which simultaneously minimises both level and number of reticulations (ranging over all networks that reflect T).*

Proof. Let N^0 be any network that reflects T. The sets of leaves below highest cut-arcs of N^0 are the maximal SN-sets of T by Observation 4.6. It follows that N and N^0 have the same sets of leaves below highest cut-arcs: namely $\mathcal{C}_N = \mathcal{C}_{N^0} = SN$. Thus $T\nabla SN = T\nabla\mathcal{C}_{N^0} = T\nabla\mathcal{C}_N$. Note also that for any maximal SN-set V the set of triplets $T|V$ is reflective because the subnetwork of N^0 below the highest cut-arc corresponding to V reflects $T|V$.

To show that N reflects T, consider a triplet $xy|z$. First assume that x, y and z are all in the same maximal SN-set V, i.e. below the same highest cut-arc a of N. Then is $xy|z$ consistent with N if and only if $xy|z \in T$, since the subnetwork of N below the cut-arc a reflects $T|V$.

Now consider a triplet $xy|z$ with two leaves in the same maximal SN-set and the third leaf in a different maximal SN-set. Note that, by the construction of N, the maximal SN-sets of T correspond to the sets of leaves below highest cut-arcs in N. Thus, $xy|z$ is consistent with N if and only if x and y are below the same highest cut-arc and z is below a different one. On the other hand, by the definition of SN-set (and using that T is dense), $xy|z$ is in T if and only if x and y are in the same maximal SN-set and z in a different one. Consequently, $xy|z$ is consistent with N if and only if $xy|z \in T$.

Finally, consider triplets $xy|z$ where x, y and z are all in different maximal SN-sets X, Y and Z respectively. First suppose $xy|z \in T$. Then it follows that $XY|Z \in T\nabla SN$ and hence that $XY|Z$ is consistent with N'. From this it follows that $xy|z$ is consistent with N, since an embedding of $XY|Z$ in N' can easily be extended to an embedding of $xy|z$ in N. To show the other direction, assume that $xy|z$ is consistent with N. Then $XY|Z$ is consistent with N' and hence $XY|Z \in T\nabla SN = T\nabla\mathcal{C}_{N^0}$. From the fact that N^0 reflects T it follows by Observation 4.6(1) that $Collapse(N^0)$ reflects $T\nabla\mathcal{C}_{N^0}$. It follows that $XY|Z$ is consistent with $Collapse(N^0)$. An embedding of $XY|Z$ in $Collapse(N^0)$ can be extended to one of $xy|z$ in N^0, so $xy|z \in T$.

It follows that for any $x' \in X$, $y' \in Y$ and $z' \in Z$ the triplet $x'y'|z'$ is consistent with N^0, implying that $x'y'|z' \in T$. This thus means that also $xy|z \in T$.

It is left to show that N is optimal, i.e. that it has a minimum number of reticulations and a minimum level over all networks that reflect T. Recall that any network reflecting T has the maximal SN-sets of T as its sets of leaves below highest cut-arcs. Given that the subnetworks of N below its highest cut-arcs are optimal it follows that N is optimal if and only if N' is optimal. Finally, N' is optimal since it has minimum level and simple networks with minimum level also contain a minimum number of reticulations. □

Algorithm 6 MINPITS (MINimum network consistent with Precisely the Input Triplet Set)

```
 1: $N := \emptyset$
 2: compute the set $SN$ of maximal SN-sets of $T$
 3: if $|SN| = 2$ then
 4:    $N$ consists of a root connected to two leaves: the elements of $SN$
 5: else
 6:    if there exists a simple level-$\ell$ network for some $\ell \leq k$ that reflects
       $T \nabla SN$ then
 7:       let $N$ be such a network of minimum level
 8:    else
 9:       $N := \emptyset$
10: for each leaf $V$ of $N$ do
11:    recursively create a level-$k$ network $N_V$ of minimal level (and which uses
       a minimum number of reticulations) that reflects $T|V$
12: if $N \neq \emptyset$ and all $N_V \neq \emptyset$ then
13:    replace each leaf $V$ of $N$ by the recursively created $N_V$.
14:    return $N$
15: else
16:    return $\emptyset$
```

Theorem 4.12. *Given a set of triplets T, Algorithm MINPITS solves* RefL-k *in time $O(|T|^{k+1})$, for any fixed k.*

Proof. For $k = 0$ we can simply use the algorithm of Aho et al., which (with an advanced implementation [Hen99]) can be implemented to run in time $O(n^3)$ (with $n = |L(T)|$), which is $O(|T|)$. For $k \geq 1$ we use algorithm MINPITS. Correctness of the algorithm follows from Lemma 4.19. It remains to analyse the running time. A simple level-k network (that reflects the input) can be found (if it exists) in time $O(n^{3k+3})$ using algorithm SL-k. (To find the simple network of minimum level we execute in order SL-1, SL-2, ..., SL-k until we find such a network. This adds a multiplicative factor of k to the running time but this is absorbed by the $O(.)$ notation for fixed k.) Therefore, lines 6 and 7 of MINPITS take $O(|SN|^{3k+3})$ time. At every level of the recursion, the computation of the maximal SN-sets takes $O(n^3)$ time [Jan06b], and computation of $T \nabla SN$ can also clearly be done in time $O(n^3)$. The critical observation is that (by Observation 4.6) every SN-set in T appears exactly once as a leaf inside an execution of SL-k. Let s_i denote the number of leaves in execution i of SL-k. The overall running time is thus of the form $O(\sum_i (n^3 + s_i^{3k+3}))$ where $\sum s_i$ is equal to the total number of SN-sets in T. Noting that $\sum_i s_i^{3k+3} \leq (\sum_i s_i)^{3k+3}$, and that there are at most $O(n)$ SN-sets in T, we obtain for $k \geq 1$ an overall running time of $O(n^{3k+3})$, which is $O(|T|^{k+1})$ because T is dense. Note that for $k \in \{1, 2\}$ we can actually do slightly better by using the faster simple level-1 [Jan06a] and simple level-2 algorithms (see Lemma 4.11). This yields for $k = 1, 2$ overall running times of $O(|T|)$ and $O(|T|^{\frac{8}{3}})$ respectively. $\square$

4.9 Open Problems

The most important question arising from this chapter is whether constructing level-k phylogenetic networks from dense triplet sets (i.e. CL-k) is still polynomial-time solvable for $k \geq 3$. Section 4.5 already showed how simple level-k networks can be constructed for all k. However, constructing general level-k networks seems more difficult since it is not clear whether Lemma 4.13 (and hence Theorem 4.8) can be generalised to levels higher than two. The fact that there are already 65 simple level-3 generators [Kel08] suggests that even for level-3 one would either need a computer proof, or a proof that does not explicitly use the structures of the different generators. Our conjecture is that the problem is polynomial-time solvable for each fixed k, but becomes NP-hard if k is part of the input (or, equivalently, if one aims to minimise k).

Another interesting problem is the complexity of constructing level-k networks with a minimum number of reticulations. Also this problem is still open in the dense case for $k \geq 3$. It would be fascinating to find out if the complexity of this problem remains equal to the complexity of CL-k, for *all* k. It is also not yet known whether it is NP-hard to construct *any* network with a minimum number of reticulations consistent with a dense triplet set, without restrictions on the level.

If, as conjectured above, it is not possible to construct level-k networks in time polynomial in both the number of leaves n and the level k, then an FPT (fixed parameter tractable) algorithm would be interesting, i.e. an exact algorithm with running time $O(f(k) \cdot p(n))$ for some function f and a polynomial p. For the more restricted version of the problem, RefL-k (see Section 4.8), we know that it can be solved in time polynomial in n, for fixed k. However, also for this problem it is open whether an algorithm with running time $O(f(k) \cdot p(n))$ exists. Even the complexity of a fundamental question regarding reflectivity "Given a triplet set T, does there exist a level-k network reflecting T for *any* k?" is still open. Another interesting question is whether it is possible to define level-k closure-operations that extend a triplet set T to a triplet set T', such that each level-k network consistent with T is also consistent with T'.

From a theoretical point of view it is interesting to characterise the triplet sets that uniquely define a phylogenetic network. This could also be of practical interest since showing uniqueness might strengthen the confidence that a given solution indeed corresponds to the "real" network. A small step into this direction is taken in Section 4.3.2, where *a* unique level-k network is given for each k. This network can be built from a certain simple level-k generator by hanging a leaf from each side. It would be tempting to conjecture that hanging a leaf from each side of a different generator also leads to a network that is uniquely defined by the triplets consistent with it (for $k \geq 2$). However, computer calculations have shown that there exist simple level-3 generators for which the network obtained in such a way is *not* uniquely defined by the set of consistent triplets. These uniqueness questions turn out to be fairly complicated.

For all the algorithms presented in this chapter, it would be interesting to find out whether the running times can be improved. The level-1 algorithm by Jansson et al. [Jan06a] has an optimal running time in the sense that it runs in time $O(n^3)$, while the size of the input is also $O(n^3)$. This is not true for the $O(n^8)$ algorithm for level-2 in Section 4.6, nor for the $O(n^5)$ and $O(n^9)$ algorithms presented in Section 4.7. It might in principle be possible to improve the running times of these algorithms, which would improve their applicability. A structural improvement of the running time of the simple level-k algorithm (Section 4.5) would also directly help to improve the running times of both the LEVEL2 (Section 4.6) and the MINPITS (Section 4.8) algorithm.

The main drawback of algorithms that require full consistency with a dense triplet set (Sections 4.5 - 4.8, [Jan06a; Jan06b]) is that they are unreliable when not all triplets can be derived correctly. The logical response to this difficulty is to look at the non-dense case and/or the variant where the number (or total weight) of consistent triplets is maximised. However, both these variants are NP-hard (Section 4.3). To deal with this NP-hardness, we have suggested an exponential-time exact algorithm for the general MaxCL-1 problem (Section 4.4), but its running time needs to be improved before this algorithm becomes practical. In this light, it would also be interesting to find out whether improvement is possible of the $O(3^n(n^2+m))$ algorithm [Wu04] for constructing trees consistent with a maximum number of input triplets. Extending the exact approach to level two and higher is also still open.

Regarding approximability, there are lots of (potential) opportunities to either construct better approximation algorithms, or to provide evidence that such algorithms do not exist. For the problem MaxCL-k, approximation algorithms exist with ratios $\frac{1}{3}$ for level-0 [Gąs99], 0.48 for level-1 and 0.61 for level-2 [Byr08]. It is not clear whether these ratios can be improved since the strongest negative result shows only APX-hardness for level-0 [Byr08]. This rules out a *Polynomial Time Approximation Scheme* for MaxCL-0 unless P = NP, but ratios better than $\frac{1}{3}$, 0.48 and 0.61 respectively could still be possible. Other interesting open problems include extending the APX-hardness to level-1 and higher and constructing approximation algorithms (with any ratio) for level greater than two.

Finally, it is important to use the ideas and insights developed in this chapter to design efficient practical heuristics for constructing phylogenetic networks with high confidence. Such heuristics need to be tested thoroughly on both simulated and practical data to show how well they are able to construct the real evolutionary history. One practical heuristic has already been designed under the name SIMPLISTIC (SIMPle network heurISTIC) [SIM08]. This algorithm is a heuristic in the sense that it is not guaranteed to find a minimum level network. However, the program can be shown to possess the following desirable properties: (1) the program runs in polynomial time if the maximum level k is fixed; (2) if the triplets are consistent with a tree, a level-1 or a level-2 network then the program can find such a tree, level-1 or level-2 network respectively; (3) if there is a network consistent with precisely the input triplets, then the program can find such a network, which minimises both the level and the total number of reticulations; (4) if there exists a simple level-k network consistent with the input triplets, then the program can find all such networks and (5) the program always finds some network consistent with all input triplets. Evaluating and optimising this program based on simulated and practical data is still work in progress. However, preliminary results are promising and further research in this direction would be very useful.

Summary

Computational biology is the thriving research area where computer science and mathematics are applied to computational problems arising from biology. Since the discovery of DNA and the ability to "read" these DNA-sequences, an enormous demand has arisen for analysing and utilising this genetic data. The mathematical nature of many of the emerging problems has triggered mathematicians to formulate biological problems in a mathematical way and to use mathematics to tackle these problems. This development has been of great importance for mathematics as it introduced a whole new and interesting application area. This thesis considers problems from different areas of computational biology and describes new mathematical approaches leading to algorithms that solve these problems.

The first part of the thesis considers various problems arising from deriving accurate genetic information (haplotypes) from ambiguous, fragmented and/or incomplete data. We explore the borderline between hardness and tractability. In essence, a problem is called *tractable* if an efficient algorithm is possible, in contrast to *hard* problems for which the existence of such algorithms is considered (extremely) unlikely. This tractability border is essential in the construction of efficient algorithms that are guaranteed to return an optimal solution. This thesis explores various topics and tries to identify where hardness stops and tractability begins. It is shown for many problems that they remain hard even in very restricted cases. In addition, it is shown how imposing other restrictions can make these problems tractable. Efficient optimal algorithms are given that solve these restricted versions. Finally, the *approximability* of the hard problems is explored. For various problems it is shown that it is even hard to find solutions that are arbitrarily close to the optimum. However, efficient algorithms are proposed that compute solutions that are guaranteed to be within a bounded ratio from the optimum.

The second part of this thesis studies how evolutionary histories can be reconstructed from biological data of currently living organisms. Traditionally, such a history is described by a tree-shape and many accurate and efficient methods have been designed to construct such trees. However, it is well known that evolution is not always tree-like and that more general models are nec-

essary. Recent years have seen more and more attention devoted to *phylogenetic networks*, which can describe much more complicated evolutionary scenarios. From a mathematical point of view however, phylogenetic networks pose formidable challenges. This thesis first shows that various phylogenetic network problems are indeed computationally intractable in their most general form. To deal with this intractability, restrictions are identified that make these problems easier. Efficient algorithms are proposed that can reconstruct phylogenetic networks in several such restricted cases. These algorithms have been applied to both simulated and practical data. Finally, it is also shown how one of the more general, hard problems can be solved. An algorithm is given that is not as efficient as is usually hoped for, but does always return an optimal solution.

In summary, this thesis presents numerous new algorithms for problems from different areas of computational biology, using different mathematical approaches. Theoretical algorithms are given, showing insights that could be essential for dealing with these problems. Moreover, practical algorithms are presented, which have been tested and applied to biological data. In addition, boundaries are identified that show in which cases it is useful to search for which types of algorithms. Finally, many interesting mathematical questions are described that originate from biology.

Samenvatting

Computational biology is het snel groeiende onderzoeksgebied waarin wiskunde en informatica worden toegepast op problemen die voortkomen uit de biologie. Sinds de ontdekking van DNA en de mogelijkheid om het DNA van individuen experimenteel te bepalen, is er een gigantische interesse ontstaan in het analyseren en gebruiken van deze genetische data. De wiskundige aard van de hieruit voortvloeiende problemen heeft wiskundigen aangezet om biologische problemen op een wiskundige manier te formuleren en wiskunde te gebruiken om deze problemen aan te pakken. Deze ontwikkeling is heel belangrijk voor de wiskunde omdat deze heeft gezorgd voor een geheel nieuw en interessant toepassingsgebied. Dit proefschrift beschouwt problemen uit verschillende gebieden binnen de computational biology en onderzoekt hoe met nieuwe wiskundige technieken algoritmes ontwikkeld kunnen worden die deze problemen oplossen.

Het eerste deel van dit proefschrift beschouwt problemen voortkomend uit het afleiden van nauwkeurige genetische informatie (haplotypes) uit gefragmenteerde, onvolledige en/of dubbelzinnige data. Er worden verschillende problemen uit dit gebied beschouwd er wordt in kaart gebracht onder welke restricties deze problemen efficient oplosbaar zijn. In essentie wordt een probleem *makkelijk* genoemd als er een efficient algoritme bestaat dat dit probleem op kan lossen. Dit staat in tegenstelling tot *moeilijke* problemen, waarvoor het bestaan van efficiente algoritmes extreem onwaarschijnlijk wordt geacht. Deze grenzen van de moeilijkheid van problemen zijn essentieel in onderzoek naar efficiente algoritmes die gegarandeerd een optimale oplossing vinden. Dit proefschrift bestudeert verschillende onderwerpen en probeert te bepalen waar de moeilijkheid van deze problemen ophoudt en ze makkelijk worden. Voor een groot aantal problemen wordt aangetoond dat ze zelfs in hele specifieke gevallen nog steeds moeilijk blijven. Daarnaast worden ook restricties aangegeven die deze problemen makkelijk maken. Efficiente, optimale algoritmes worden gegeven die deze speciale gevallen van de problemen oplossen. Voor de moeilijke versies van de problemen wordt ook onderzocht of ze wel te *benaderen* zijn. Voor verscheidene problemen wordt bewezen dat het zelfs moeilijk is om een benadering te vinden die willekeurig dicht bij het optimum ligt. Anderzijds worden ook efficiente benaderingsalgoritmes gegeven die oplossingen vinden die gegarandeerd binnen een beperkte afstand van het optimum liggen.

Het tweede deel van dit proefschrift bestudeert hoe de evolutionaire geschiedenis van een groep organismen kan worden afgeleid uit hun genetische data. Traditioneel wordt een dergelijke geschiedenis weergegeven in een boomvormige figuur en tal van nauwkeurige en efficiente methoden zijn ontwikkeld die een dergelijke boom kunnen reconstrueren. Desalniettemin is het al lange tijd bekend dat evolutie niet altijd de vorm van een boom volgt en dat dus algemenere modellen nodig zijn. In recente jaren wordt er daarom steeds meer aandacht besteed aan *fylogenetische netwerken*, die veel ingewikkeldere evolutionaire geschiedenissen kunnen beschrijven. Voor wiskundigen zorgen deze fylogenetische netwerken echter voor een gigantische uitdaging. Dit proefschrift laat eerst zien dat verscheidene problemen met betrekking tot fylogenetische netwerken inderdaad moeilijk zijn in hun meest algemene vorm. Om deze problemen desalniettemin op te kunnen lossen worden verschillende restricties voorgesteld die deze problemen makkelijker maken. Efficiente algoritmes worden beschreven die fylogenetische netwerken kunnen reconstrueren in verschillende van deze speciale gevallen. Deze algoritmes zijn bovendien toegepast op zowel gesimuleerde als experimentele data. Tenslotte wordt ook aangegeven hoe één van de algemene, moeilijke problemen alsnog kan worden aangepakt. Voor dit probleem wordt een algoritme gegeven dat minder efficient is dan normaal gesproken gewenst is, maar dat in ieder geval altijd een optimale oplossing vindt.

Kortom, dit proefschrift presenteert een groot aantal nieuwe algoritmes die problemen oplossen uit verschillende gebieden binnen de computational biology en die een groot aantal verschillende wiskundige methoden gebruiken. Er worden theoretische algoritmes gegeven die nieuw inzicht verschaffen dat essentieel zou kunnen zijn in verder onderzoek naar deze problemen. Bovendien worden praktische algoritmes gepresenteerd die getest zijn en toegepast op biologische data. Daarnaast worden grenzen beschreven die aangeven in welke gevallen het nuttig is om naar welk type algoritme te zoeken. Tenslotte zijn ook talloze interessante wiskundige problemen geformuleerd die uit de biologie naar voren komen.

Curriculum Vitae

Leo van Iersel was born in Hellevoetsluis, The Netherlands, on the 29th of August 1981. He moved to Enschede in 1999 to study Applied Mathematics at the Universiteit Twente and obtained his ingenieursdiploma, equivalent to a Master's degree, in 2004. After that, he moved to Eindhoven to pursue his PhD degree in computational biology at the Technische Universiteit Eindhoven. He will defend his PhD thesis on the 29th of January 2009. After that, he will work as a postdoctoral researcher at the University of Canterbury in Christchurch, New Zealand, where he will study mathematical approaches to phylogenetics.

References

[Aho81] A. V. Aho, Y. Sagiv, T. G. Szymanski and J. D. Ullman. Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM Journal on Computing*, vol. 10(3):pp. 405–421, 1981. 4, 16, 67, 90

[Ali97] P. Alimonti and V. Kann. Hardness of approximating problems on cubic graphs. In *Italian Conference on Algorithms and Complexity (CIAC)*, pp. 288–298. 1997. 21, 30, 41

[Alo99] N. Alon and B. Sudakov. On two segmentation problems. *Journal of Algorithms*, vol. 33:pp. 173–184, 1999. 24

[Alo06] N. Alon. Ranking tournaments. *SIAM Journal on Discrete Mathematics*, vol. 20(1):pp. 137–142, 2006. 87

[Aus99] G. Ausiello, M. Protasi, A. Marchetti-Spaccamela, G. Gambosi, P. Crescenzi and V. Kann. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer-Verlag, 1999. 2, 3

[Ave44] O. T. Avery, C. M. MacLeod and M. McCarty. Studies on the chemical nature of the substance inducing transformation of pneumococcal types. *Journal of Experimental Medicine*, vol. 79(2):pp. 137–158, 1944. 1

[Baf96] V. Bafna and P. Pevzner. Genome Rearrangements and Sorting by Reversals. *SIAM Journal on Computing*, vol. 25(2):pp. 272–289, 1996. 5

[Baf04] V. Bafna, D. Gusfield, S. Hannenhalli and S. Yooseph. A note on efficient computation of haplotypes via perfect phylogeny. *Journal of Computational Biology*, vol. 11(5):pp. 858–866, 2004. 10, 39, 41

[Baf05] V. Bafna, S. Istrail, G. Lancia and R. Rizzi. Polynomial and APX-hard cases of the individual haplotyping problem. *Theoretical Computer Science*, vol. 335(1):pp. 109–125, 2005. 28, 36

[Bar04] M. Baroni, C. Semple and M. Steel. A framework for representing reticulate evolution. *Annals of Combinatorics*, vol. 8:pp. 391–408, 2004. 66

[Bar05] M. Baroni, S. Grünewald, V. Moulton and C. Semple. Bounding the number of hybridisation events for a consistent evolutionary history. *Mathematical Biology*, vol. 51:pp. 171–182, 2005. 15, 69

[Bau92] B. R. Baum. Combining trees as a way of combining data sets for phylogenetic inference. *Taxon*, vol. 41:pp. 3–10, 1992. 68

[Ber99] P. Berman and M. Karpinski. On some tighter inapproximability results (extended abstract). In *International Colloquium on Automata, Languages and Programming (ICALP)*, vol. 1644 of *Lecture Notes in Computer Science*, pp. 200–209. 1999. 21

[Ber04] A. Bergeron, J. Mixtacki and J. Stoye. Reversal distance without hurdles and fortresses. In *Combinatorial Pattern Matching (CPM)*, vol. 3109 of *Lecture Notes in Computer Science*, pp. 388–399. 2004. 5

[Bla93] J. R. S. Blair and B. Peyton. An introduction to chordal graphs and clique trees. In *Graph theory and sparse matrix computation*, pp. 1–29. Springer, 1993. 44

[Bon03] P. Bonizzoni, G. D. Vedova, R. Dondi and J. Li. The haplotyping problem: An overview of computational models and solutions. *Journal of Computer Science and Technology*, vol. 18(6):pp. 675–688, 2003. 2, 17, 39

[Bor07a] M. Bordewich, S. Linz, K. S. John and C. Semple. A reduction algorithm for computing the hybridization number of two trees. *Evolutionary Bioinformatics*, vol. 3:pp. 86–98, 2007. 69

[Bor07b] M. Bordewich and C. Semple. Computing the minimum number of hybridization events for a consistent evolutionary history. *Discrete Applied Mathematics*, vol. 155(8):pp. 914–928, 2007. 15, 69

[Bro06] D. G. Brown and I. M. Harrower. Integer programming approaches to haplotype inference by pure parsimony. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 3(2):pp. 141–154, 2006. 38, 40

[Bry97] D. Bryant. *Building Trees, Hunting for Trees, and Comparing Trees: Theory and Methods in Phylogenetic Analysis.* Ph.D. thesis, University of Canterbury, Christchurch, New Zealand, 1997. 15, 16, 67, 68, 87

[Byr08] J. Byrka, P. Gawrychowski, K. T. Huber and S. M. Kelk. Worst-case optimal approximation algorithms for maximizing triplet consistency within phylogenetic networks, 2008. ArXiv:0710.3258v3 [q-bio.PE]. 67, 100, 134, 143

[Car07] G. Cardona, F. Rossello and G. Valiente. Comparison of tree-child phylogenetic networks, 2007. ArXiv:0708.3499v1 [q-bio.PE]. 130

[Cha07] P. Charbit, S. Thomassé and A. Yeo. The minimum feedback arc set problem is NP-hard for tournaments. *Combinatorics, Probability and Computing*, vol. 16(1):pp. 1–4, 2007. 87

[Cho05] C. Choy, J. Jansson, K. Sadakane and W.-K. Sung. Computing the maximum agreement of phylogenetic networks. *Theoretical Computer Science*, vol. 335(1):pp. 93–107, 2005. 66

[Cil05] R. Cilibrasi, L. J. J. van Iersel, S. M. Kelk and J. Tromp. On the complexity of several haplotyping problems. In *Algorithms in Bioinformatics (WABI)*, vol. 3692 of *Lecture Notes in Computer Science*, pp. 128–139. 2005. 9, 11

[Cil07] R. Cilibrasi, L. J. J. van Iersel, S. M. Kelk and J. Tromp. The complexity of the single individual SNP haplotyping problem. *Algorithmica*, vol. 49(1):pp. 13–36, 2007. 9

[Clo01] P. Clote and R. Backofen. *Computational Molecular Biology: An Introduction*. John Wiley & Sons, 2001. 1

[Din06] Z. Ding, V. Filkov and D. Gusfield. A linear-time algorithm for the perfect phylogeny haplotyping (PPH) problem. *Journal of Computational Biology*, vol. 13(2):pp. 522–533, 2006. 38, 58, 61

[Dri04] P. Drineas, A. Frieze, R. Kannan, S. Vempala and V. Vinay. Clustering large graphs via the singular value decomposition. *Machine Learning*, vol. 56(1):pp. 9–33, 2004. 24

[Gam08] P. Gambette. Who's who in phylogenetic networks, 2008. `http://www.lirmm.fr/~gambette/PhylogeneticNetworks/`. 66

[Gar79] M. R. Garey and D. S. Johnson. *Computers and intractability*. W. H. Freeman and Co., 1979. A guide to the theory of NP-completeness, A Series of Books in the Mathematical Sciences. 2, 3, 85

[Gąs99] L. Gąsieniec, J. Jansson, A. Lingas and A. Östlin. On the complexity of constructing evolutionary trees. *Journal of Combinatorial Optimization*, vol. 3(2-3):pp. 183–197, 1999. 67, 143

[Gat79] W. Gates and C. Papadimitriou. Bounds for sorting by prefix reversal. *Discrete Mathematics*, vol. 27:pp. 47–57, 1979. 5

[Gav77] F. Gavril. Testing for equality between maximum matching and minimum node covering. *Information Processing Letters*, vol. 6:pp. 199–202, 1977. 48

[Gre04] H. J. Greenberg, W. E. Hart and G. Lancia. Opportunities for combinatorial optimisation in computational biology. *INFORMS Journal on Computing*, vol. 16(3):pp. 211–231, 2004. 7, 17

[Gui03] S. Guindon and O. Gascuel. A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood. *Systematic Biology*, vol. 52(5):pp. 696–704, 2003. 67, 117, 129

[Gus91] D. Gusfield. Efficient algorithms for inferring evolutionary history. *Networks*, vol. 21:pp. 19–28, 1991. 38

[Gus97] D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, 1997. 38

[Gus03] D. Gusfield. Haplotype inference by pure parsimony. In *Combinatorial Pattern Matching (CPM)*, vol. 2676 of *Lecture Notes in Computer Science*, pp. 144–155. 2003. 9, 10, 38

[Gus04] D. Gusfield, S. Eddhu and C. Langley. Optimal, efficient reconstruction of phylogenetic networks with constrained recombination. *Journal of Bioinformatics and Computational Biology*, vol. 2:pp. 173–213, 2004. 66, 69

[Gus07] D. Gusfield, D. Hickerson and S. Eddhu. An efficiently computed lower bound on the number of recombinations in phylognetic networks: Theory and empirical study. *Discrete Applied Mathematics*, vol. 155(6-7):pp. 806–830, 2007. 69

[Hal03] B. V. Halldorsson, V. Bafna, N. Edwards, R. Lippert, S. Yooseph and S. Istrail. A survey of computational methods for determining haplotypes. In *RECOMB Satellite on Computational Methods for SNPs and Haplotype Inference*, vol. 2983 of *Lecture Notes in Bioinformatics*, pp. 26–47. 2003. 17, 38

[Hei90] J. Hein. Reconstructing evolution of sequences subject to recombination using parsimony. *Mathematical Biosciences*, vol. 98:pp. 185–200, 1990. 65, 69

[Hen99] M. Henzinger, V. King and T. Warnow. Constructing a tree from homeomorphic subtrees, with applications to computational evolutionary biology. *Algorithmica*, vol. 24(1):p. 113, 1999. 98, 141

[Hoo01] H. Hoogeveen, P. Schuurman and G. J. Woeginger. Non-approximability results for scheduling problems with minsum criteria. *INFORMS Journal on Computing*, vol. 13(2):pp. 157–168, 2001. 21

[Hop73] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matching in bipartite graphs. *SIAM Journal on Computing*, vol. 2:pp. 225–231, 1973. 48

[Hua05] Y.-T. Huang, K.-M. Chao and T. Chen. An approximation algorithm for haplotype inference by maximum parsimony. *Journal of Computational Biology*, vol. 12(10):pp. 1261–1274, 2005. 39

[Hur07a] C. A. J. Hurkens, L. J. J. van Iersel, J. C. M. Keijsper, S. M. Kelk, L. Stougie and J. Tromp. Prefix reversals on binary and ternary strings. In *Algebraic Biology (AB)*, vol. 4545 of *Lecture Notes in Computer Science*, pp. 292–306. 2007. iii, 5

[Hur07b] C. A. J. Hurkens, L. J. J. van Iersel, J. C. M. Keijsper, S. M. Kelk, L. Stougie and J. Tromp. Prefix reversals on binary and ternary strings. *SIAM Journal on Discrete Mathematics*, vol. 21(3):pp. 592–611, 2007. iii, 5

[Hus06] D. H. Huson and D. Bryant. Application of phylogenetic networks in evolutionary studies. *Molecular Biology and Evolution*, vol. 23(2):pp. 254–267, 2006. 66

[Huy05] T. Huynh, J. Jansson, N. Nguyen and W.-K. Sung. Constructing a smallest refining galled phylogenetic network. In *Research in Computational Molecular Biology (RECOMB)*, vol. 3500 of *Lecture Notes in Bioinformatics*, pp. 265–280. 2005. 69

[Ier06] L. J. J. van Iersel, J. C. M. Keijsper, S. M. Kelk and L. Stougie. Beaches of islands of tractability: Algorithms for parsimony and minimum perfect phylogeny haplotyping problems. In *Algorithms in Bioinformatics (WABI)*, vol. 4175 of *Lecture Notes in Computer Science*, pp. 80–91. 2006. 11

[Ier08a] L. J. J. van Iersel, J. C. M. Keijsper, S. M. Kelk and L. Stougie. Shorelines of islands of tractability: Algorithms for parsimony and minimum perfect phylogeny haplotyping problems. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 5(2):pp. 301–312, 2008. 11

[Ier08b] L. J. J. van Iersel, J. C. M. Keijsper, S. M. Kelk, L. Stougie, F. Hagen and T. Boekhout. Constructing level-2 phylogenetic networks from triplets. In *Research in Computational Molecular Biology (RECOMB)*, vol. 4955 of *Lecture Notes in Bioinformatics*, pp. 464–476. 2008. 16

[Ier08c] L. J. J. van Iersel and S. M. Kelk. Constructing the simplest possible phylogenetic network from triplets. In *International Symposium on Algorithms and Computation (ISAAC)*, vol. 5369 of *Lecture Notes in Computer Science*, pp. 472–483. 2008. 16

[Ier08d] L. J. J. van Iersel, S. M. Kelk and M. Mnich. Uniqueness, intractability and exact algorithms: Reflections on level-k phylogenetic networks, 2008. ArXiv:0712.2932v3 [q-bio.PE]. 16

[Int01] International Human Genome Sequencing Consortium. Initial sequencing and analysis of the human genome. *Nature*, vol. 409:pp. 860–921, 2001. 1

[Jan01] J. Jansson. On the complexity of inferring rooted evolutionary trees. In *Brazilian Symposium on Graphs, Algorithms and Combinatorics*, vol. 7 of *Electronic Notes in Discrete Mathematics*, p. 4. 2001. 15, 16, 67, 68, 87

[Jan06a] J. Jansson, N. B. Nguyen and W.-K. Sung. Algorithms for combining rooted triplets into a galled hylogenetic network. *SIAM Journal on Computing*, vol. 35(5):pp. 1098–1121, 2006. 15, 16, 66, 67, 68, 69, 76, 85, 87, 90, 98, 103, 114, 115, 118, 121, 124, 134, 142, 143

[Jan06b] J. Jansson and W.-K. Sung. Inferring a level-1 phylogenetic network from a dense set of rooted triplets. *Theoretical Computer Science*, vol. 363(1):pp. 60–68, 2006. 15, 16, 76, 79, 84, 90, 100, 101, 121, 141, 143

[Jia04] Y. Jiao, J. Xu and M. Li. On the k-closest substring and k-consensus pattern problems. In *Combinatorial Pattern Matching (CPM)*, vol. 3109 of *Lecture Notes in Computer Science*, pp. 130–144. 2004. 24, 36

[Kel08] S. M. Kelk. `http://homepages.cwi.nl/~kelk/lev3gen/`, 2008. 72, 142

[Kid04] S. Kidd, F. Hagen, R. Tscharke, M. Huynh, K. Bartlett, M. Fyfe, L. MacDougall, T. Boekhout, K. Kwon-Chung and W. Meyer. A rare genotype of Cryptococcus gattii caused the Cryptococcosis outbreak on Vancouver Island (British Columbia, Canada). *Proceedings of the National Academy of Sciences of the United States of America*, vol. 101:pp. 17258–17263, 2004. 15, 116

[Kle98a] J. M. Kleinberg, C. H. Papadimitriou and P. Raghavan. A microeconomic view of data mining. *Data Mining and Knowledge Discovery*, vol. 2(4):pp. 311–324, 1998. 24

[Kle98b] J. M. Kleinberg, C. H. Papadimitriou and P. Raghavan. Segmentation problems. In *Symposium on Theory of Computing (STOC)*, pp. 473–482. 1998. 18, 24

[Kle04] J. M. Kleinberg, C. H. Papadimitriou and P. Raghavan. Segmentation problems. *Journal of the ACM (JACM)*, vol. 51(2):pp. 263–280, 2004. 24

[Lan01] G. Lancia, V. Bafna, S. Istrail, R. Lippert and R. Schwartz. SNPs problems, complexity and algorithms. In *European Symposium on Algorithms (ESA)*, vol. 2161 of *Lecture Notes in Computer Science*, pp. 182–193. 2001. 8, 17, 18, 28, 36

[Lan04] G. Lancia, M. Pinotti and R. Rizzi. Haplotyping populations by pure parsimony: Complexity of exact and approximation algorithms. *INFORMS Journal on Computing*, vol. 16(4):pp. 348–359, 2004. 10, 38, 39

[Lan06] G. Lancia and R. Rizzi. A polynomial case of the parsimony haplotyping problem. *Operations Research Letters*, vol. 34(3):pp. 289–295, 2006. 38, 39, 46, 49, 50

[LEV07] LEVEL2: A fast method for constructing level-2 phylogenetic networks from dense sets of rooted triplets, 2007. `http://homepages.cwi.nl/~kelk/level2triplets.html`. 116

[Lew00] B. Lewin. *Genes VII.* Oxford University Press, 2000. 1

[Mak06] V. Makarenkov, D. Kevorkov and P. Legendre. *Phylogenetic Network Reconstruction Approaches*, vol. 6 of *International Elsevier Series: Bioinformatics*, pp. 61–97. 2006. 66

[MAR08] MARLON: Constructing level one phylogenetic networks with a minimum amount of reticulation, 2008. `http://homepages.cwi.nl/~kelk/marlon.html`. 121

[Mor04] B. Moret, L. Nakhleh, T. Warnow, C. Linder, A. Tholse, A. Padolina, J. Sun and R. Timme. Phylogenetic networks: Modeling, reconstructibility, and accuracy. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 1(1):pp. 13–23, 2004. 66

[Mor05] D. Morrison. Networks in phylogenetic analysis: New tools for population biology. *International Journal for Parasitology*, vol. 35(5):pp. 567–582, 2005. 65, 66

[Nak05] L. Nakhleh, T. Warnow, C. R. Linder and K. S. John. Reconstructing Reticulate Evolution in Species-Theory and Practice. *Journal of Computational Biology*, vol. 12(6):pp. 796–811, 2005. 66

[Nei00] M. Nei and S. Kumar. *Molecular Evolution and Phylogenetics*. Oxford University Press, 2000. 2

[Ost02] R. Ostrovsky and Y. Rabani. Polynomial-time approximation schemes for geometric min-sum median clustering. *Journal of the ACM*, vol. 49(2):pp. 139–156, 2002. 24

[Pag02] R. D. M. Page. Modified mincut supertrees. In *Algorithms in Bioinformatics (WABI 2002)*, vol. 2452 of *Lecture Notes in Computer Science*, pp. 537–551. 2002. 68

[Pan04] A. Panconesi and M. Sozio. Fast hare: A fast heuristic for single individual SNP haplotype reconstruction. In *Algorithms in Bioinformatics (WABI)*, vol. 3240 of *Lecture Notes in Computer Science*, pp. 266–277. 2004. 17, 24

[Pap91] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, vol. 43:pp. 425–440, 1991. 20, 21, 41

[Pap94] C. H. Papadimitriou. Computational complexity. 1994. 3

[Pap05] Personal communication with Christos H. Papadimitriou, 2005. 24

[Pev00] P. Pevzner. *Computational Molecular Biology: An Algorithmic Approach*. MIT Press, 2000. 1

[Rag92] M. A. Ragan. Matrix representation in reconstructing phylogenetic relationships among the eukaryotes. *Biosystems*, vol. 28:pp. 47–55, 1992. 68

[Ram97] A. Rambaut and N. Grassly. Seq-gen: An application for the monte carlo simulation of DNA sequence evolution along phylogenetic trees. *Computer Applications in the Biosciences*, vol. 13:pp. 235–238, 1997. 129

[Ros76] D. J. Rose, R. E. Tarjan and G. S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on Computing*, vol. 5:pp. 266–283, 1976. 46

[San98] M. J. Sanderson, A. Purvis and C. Henze. Phylogenetic supertrees: Assembling the trees of life. *Trends in Ecology and Evolution*, vol. 13:pp. 105–109, 1998. 68

[Sem00] C. Semple and M. Steel. A supertree method for rooted trees. *Discrete Applied Mathematics*, vol. 105(1-3):pp. 147–158, 2000. 68

[Sha06] R. Sharan, B. V. Halldórsson and S. Istrail. Islands of tractability for parsimony haplotyping. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 3(3):pp. 303–311, 2006. 10, 11, 38, 39, 40, 41, 42, 51, 56, 60

[SIM08] SIMPLISTIC: Simple network heuristic, 2008. `http://homepages.cwi.nl/~kelk/simplistic.html`. 144

[Sni06] S. Snir and S. Rao. Using max cut to enhance rooted trees consistency. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 3(4):pp. 323–333, 2006. 68

[Son04] Y. Song and J. Hein. On the minimum number of recombination events in the evolutionary history of DNA sequences. *Journal of Mathematical Biology*, vol. 48:p. 160186, 2004. 15, 69

[Son05a] Y. Song, Y. Wu and D. Gusfield. Efficient computation of close lower and upper bounds on the minimum number of recombinations in biological sequence evolution. *Bioinformatics*, vol. 21 (Suppl. 1):pp. i413 – i422, 2005. 69

[Son05b] Y. S. Song, Y. Wu and D. Gusfield. Algorithms for imperfect phylogeny haplotyping (IPPH) with single haploplasy or recombination event. In *Algorithms in Bioinformatics (WABI)*, vol. 3692 of *Lectures Notes in Bioinformatics*, pp. 152–164. 2005. 38

[Ste92] M. Steel. The complexity of reconstructing trees from qualitative characters and subtrees. *Journal of Classification*, vol. 9(1):pp. 91–116, 1992. 5

[Swo98] D. L. Swofford. Paup*: phylogenetic analysis using parsimony (and other methods), 1998. 67

[Ven01] J. C. Venter et al. The sequence of the human genome. *Science*, vol. 291:pp. 1304–1351, 2001. 1

[Vij06] R. Vijayasatya and A. Mukherjee. An optimal algorithm for perfect phylogeny haplotyping. *Journal of Computational Biology*, vol. 13(4):pp. 897–928, 2006. 38, 39, 63

[Wan01] L. Wang, K. Zhang and L. Zhang. Perfect phylogenetic networks with recombination. *Journal of Computational Biology*, vol. 8(1):pp. 69–78, 2001. 69

[Wat53] J. D. Watson and F. H. C. Crick. Molecular structure of nucleic acids. *Nature*, vol. 171:p. 737738, 1953. 1

[Wat95] M. S. Waterman. *Introduction to Computational Biology: Maps, Sequences and Genomes.* Chapman & Hall/CRC, 1995. 1

[Wu04] B. Y. Wu. Constructing the maximum consensus tree from rooted triples. *Journal of Combinatorial Optimization*, vol. 8(1):pp. 29–39, 2004. 15, 16, 67, 68, 87, 90, 143

[Zha06] X.-S. Zhang, R.-S. Wang, L.-Y. Wu and L. Chen. Models and algorithms for haplotyping problem. *Current Bioinformatics*, vol. 1:pp. 105–114, 2006. 39

Index