Università degli Studi di Milano-Bicocca
Facoltà di Scienze Matematiche, Fisiche e Naturali
Dottorato di Ricerca in Informatica – Ciclo XXII

Combinatorial Problems in Studies of Genetic Variations: Haplotyping and Transcript Analysis

Tesi di Dottorato di
Yuri Pirola

Supervisore:
Prof. Paola Bonizzoni

Anno Accademico 2008–2009

Contents

1 Introduction

Computer Science provides powerful tools to biologists to deeply investigate and understand the basic functioning of living organisms. Conversely, Biology challenges computer scientists to design efficient algorithmic solutions to complex problems where the size of the data to process is growing at exponential rates. In particular, increasing attention is devoted to the study of genetic variations between individuals of a population as they are a crucial medium to map different observable characteristics (phenotypic traits) of the individuals to the underlying genes and biological processes. This kind of studies benefits from the presence of a significant amount of data, since associations between genetic differences and phenotypic traits can be more accurately identified over large population data. Unfortunately, observing and obtaining directly the genetic data of interest is a long and costly operation, especially for large populations. Less informative sources of data, instead, are available at a fraction of cost. Computational methods are then called to recover the original data of interest from their less informative observations.

The design of efficient combinatorial algorithms to infer relevant data for genetic variation studies starting from less informative observations is the main aim of the work of this thesis. We considered two different kinds of data, *haplotypes* and *transcripts*, and in the two parts of the thesis we addressed the main computational problems that arise from the need of extracting relevant genetic information from each of them.

In the first part of the thesis, haplotypic data have been considered. In this case, biologists are interested in obtaining the genetic sequences inherited from each parent (*haplotypes*) for each member of a population. However, only the "conflation" of the two haplotypes of each individual (called *genotype*) is routinely collected. Haplotype Inference (HI) is the computational activity of recovering haplotypes from the genotypes of a population according to a particular genetic model of evolution and inheritance. Clearly, different genetic models and representations of the genotypic data determine the formalization of different computational HI problems.

In this thesis we formalized and studied two new problems of HI: the pure parsimony xor haplotyping (PPXH) problem and the minimum-event haplotype configuration problem (MinEHC).

The PPXH problem is the problem of inferring haplotypes under the pure parsimony criterion from a particular representation of genotypic data called xor-

genotypes. Both the pure parsimony criterion and the xor genotype representation are known concepts in the HI literature, but they have been studied separately and under different assumptions (the pure parsimony principle has been applied on the regular genotypic data, while the xor-genotypes have been previously studied on the perfect phylogeny model). In this work, we introduced a graph representation of the PPXH solutions from which we have been able to derive exact algorithms for restricted instances and an approximation algorithm. The graph representation has also inspired a heuristic solution strategy, whose validity has been experimentally evaluated on synthetic and real datasets. This work has revealed important connections between the PPXH problem and classic graph-theoretic problems: designing new algorithmic solutions for the PPXH problem could effectively leverage our ability to tackle other combinatorial problems.

The MinEHC problem is a HI problem where the family relationships between population's members are known and represented by a pedigree. Therefore, the HI process must recover a haplotype configuration for the pedigree's members that is consistent with the genotypic data and laws of inheritance. However, genetic variation events can alter the genetic data during the transmission from parent to children and the computational problem asks for the haplotype configuration consistent with the genotypic data that requires the minimum number of variation events. In this work we studied the problem under the assumption that the two most frequent kinds of variation events, recombinations and mutations, can occur, thus extending previous works where only one of them (or none) was allowed. By exploring the computational complexity of MinEHC (and related restriction) we have been able to show that even simple instances are computationally hard to solve or to approximate. In absence of variation events, the Haplotype Inference problem on pedigrees is a basic constraint satisfaction problem. In this work we further explored such approach and we modelled variation events as "errors" that have to be corrected in order to obtain the solution. The formalization of this idea led to a combinatorial reduction to a coding theory problem, and a polynomial-time heuristic algorithm for MinEHC. An experimental evaluation of the designed heuristic under various simulated scenarios has revealed extremely good performances, both in term of accuracy and running times. The heuristic we designed is able to gracefully include prior knowledge about the pedigree or the genotype structure: how to fully exploit this ability is a relevant challenge. Accommodating missing data is still an open issue, but we believe that our reduction, again, could play a crucial role in achieving this important objective.

In the second part of the thesis, a computational problem involving transcripts has been studied. Transcripts are sequences produced from the information contained in a region of DNA and they are the basis in several key processes of the cell, such as protein synthesis and gene expression. To fully understand how genetic vari-

ations impact on the expression of phenotypic traits, a better understanding of the hidden structure of the genome is needed. Important insights of such structure can be provided by the alignment of the transcripts, which represent the functional part of the DNA, to the genomic sequence. As a consequence, the computational problem of aligning transcript sequences against a reference DNA sequence (transcript alignment problem) naturally arises. Computational approaches for this problem have to face the following hurdles: (*i*) transcripts are the concatenation of several (spliced) portions of DNA and not a one-to-one copy of a long DNA region, (*ii*) several different transcripts are produced from the same DNA region, and (*iii*) only small transcript fragments (ESTs) are usually available, while the full-length transcripts (the complete sequences) are hardly observable. Moreover, two other characteristics of the data have to be considered: ESTs may contain several sequencing errors and the reference genomic sequence may present long repeated regions. All those elements introduce ambiguity in the alignment of a transcript, and the determination of an accurate solution for the problem is still a challenging task. Traditional basic approaches try to compute high-quality transcript alignments in order to predict the genomic structure. Clearly, errors in the transcript alignments could heavily affect the prediction quality.

In this work, instead, we present a new formulation of the transcript alignment problem that works in the opposite sense. The basic idea of our formulation is to consider the transcript alignment problem as the problem of choosing a (simple) genomic structure that can explain the alignments of a set of transcripts. In this formulation, the inherent ambiguity of a single transcript alignment can be managed by exploiting the high redundancy of currently available EST databases. In the thesis, we formalized two main combinatorial problems and we proposed two algorithmic solutions for them that combined together address the problem underlying the above mentioned formulation. In the first problem that we formalized, called maximal embedding problem, we exploited the interesting combinatorial property that all maximal substrings of a pattern P and a text T can be detected in linear time. The aim of the maximal embedding problem is finding representing particular sequences of maximal substrings of P and T that can correspond to transcript alignments. The second combinatorial problem that we formalized has been called minimum factorization agreement (MFA) problem. The instance of the MFA problem is an ordered set F of factors and a set S of colored subsequences of F. Then, the MFA problem asks for the minimum cardinality subset F' of F such that, for each color c, a sequence in S colored with c is a subsequence of F'.

Our algorithm for the maximal embedding problem and the MFA problem, applied to EST and genomic data, provide:

- an efficient algorithm to compute all the possible (meaningful) alignments of a given transcript against a reference genomic sequence;

- an algorithm, that among a set of all possible alignments of a set of transcripts, extract a single alignment for each transcript. The resulting alignments all agree with the same genomic structure.

We also conducted an experimental evaluation of our strategy on a significant set of genes, and promising preliminary results have emerged even in absence of specific biological criteria.

The thesis is structured as follows. In **Chapter 2** we review some basic notions of computational complexity theory, approximation, graph theory and linear algebra which the subsequent original results are based on.

The computational problems related to the first kind of data, haplotypes, are studied starting from **Chapter 3**, where the principal existent approaches of HI are presented along with the related terminology.

The problem of inferring haplotypes from xor-genotypes under the pure parsimony principle is defined and investigated in **Chapter 4**. Part of the results of this chapter have been presented in [16].

Chapter 5 presents the results concerning the problem of inferring haplotypes on pedigrees with recombination and mutation events. In particular we provide the computational complexity analysis of the problem and we illustrate an accurate and efficient heuristic algorithm. A manuscript regarding these results is in preparation [63].

The second part of the thesis starts from **Chapter 6**, where the problem of computing all the possible alignments of a transcript against a genomic sequence is addressed. We present an efficient algorithm for the problem based on the determination of maximal common substrings between the transcript sequence and the genomic sequence. The results of this work have been submitted to an international conference [17].

The last chapter, **Chapter 7**, studies the problem of choosing an "agreement" genomic structure that can explain a transcript alignment for each transcript of a set. A simple, but effective, heuristic algorithm is proposed and, together with the previous algorithm, it represents a complete methodology for inferring the structure of a genomic region given a set of transcripts. An experimental evaluation of the method has been performed on a real dataset. The agreement problem and the heuristic solution has been presented in [15].

2 Preliminaries

This chapter is devoted to the formal definition of several prerequisite notions that will be used through the rest of the thesis. In particular, in the presentation of our results we will use some basic concepts of three main areas: *computational complexity and approximation*, *linear algebra* (on $\mathbb{Z}_2^n$), and *graph theory*. For a detailed presentation of the three areas, we refer the interested readers to a monograph such as [6, 33, 47, 87, 93].

2.1 Computational Complexity and Approximation

Computational Complexity Theory is the study and the classification of computational problems based on the computational resources needed to solve them. Here we present few basic definitions and results used in the rest of the thesis.

The most basic "objects" in computational complexity theory are problems. A *problem* is a (mathematical) relation $P \subseteq \mathcal{I} \times S$ between a set $\mathcal{I}$, called set of *problem instances*, and a set $\mathcal{S}$, called set of *problem solutions*. When the solution set of a problem P is represented by the binary set $\{\text{YES}, \text{NO}\}$, we say that P is a *decision problem*. In some cases, the solutions of a problem P are ranked according to a quality measure $c : \mathcal{S} \to \mathbb{R}$. An *optimization problem* is a triplet (P, c, obj) where $P \subseteq \mathcal{I} \times \mathcal{S}$ is a problem, $c : \mathcal{S} \to \mathbb{R}$ is a quality measure of solutions, and $obj \in \{\text{MIN}, \text{MAX}\}$ is the search criterion. Aim of the optimization problem is to find, for each instance $i \in \mathcal{I}$ a solution $s^* \in \mathcal{S}$ (called *optimal solution*) such that $(i, s^*) \in P$ and $c(s^*) = \min_{s \in \mathcal{S}}\{c(s) \mid (i, s) \in P\}$ if $obj = \text{MIN}$, or $c(s^*) = \max_{s \in \mathcal{S}}\{c(s) \mid (i, s) \in P\}$ otherwise. An optimization problem is called *minimization problem* if $obj = \text{MIN}$, or *maximization problem* otherwise. In the context of optimization problems, we say that a solution $s \in \mathcal{S}$ is a *feasible solution* for the instance $i \in \mathcal{I}$ iff $(i, s) \in P$. The quality measure $c(s)$ of a feasible solution is often called *cost* of the solution, and the cost of an optimal solution for an instance i is often called *optimum (cost)* of the instance i and it is generally denoted with $c^*(i)$.

An *algorithm* $\mathcal{A}$ is a description of a procedure for solving a problem in a finite-number of steps by a (simple) computational device. There are several parameters to evaluate the efficiency of an algorithm. The most used parameter is the time that the algorithm uses to solve the problem. Time is usually measured as number of elementary steps required to compute the solution. Clearly, different instances

require different times and a concise representation of all of them would be overwhelming. To solve this issue, the worst-case time complexity of the algorithm is usually evaluated. Given an algorithm $\mathcal{A}$ for the problem $P \subseteq \mathcal{I} \times \mathcal{S}$, the *worst-case time complexity* of $\mathcal{A}$ is a function $f : \mathbb{N} \to \mathbb{R}$ which specifies the maximum time $f(n)$ that $\mathcal{A}$ requires to solve an instance $i \in \mathcal{I}$ of size at most n (under a reasonable encoding scheme of the instances). The worst-case time complexity function is almost always expressed in the *big-O notation*, which represents an asymptotic upper-bound of the time complexity function when the size of the instance grows to infinity. More formally $O(f(n))$ is the set of functions $g(n)$ such that, for two positive constants n_o and c, $g(n) \leq f(n)$ for all $n > n_o$. For simplicity, since we are always interested in the worst-case time complexity of an algorithm, we will refer to it as the time complexity of the algorithm, omitting the specifier "worst-case". Moreover, we say that an algorithm is a *polynomial-time* algorithm if its time complexity is a function $f(n) \in O(n^k)$ for some constant k, and that is a *linear-time* algorithm if its time complexity is bounded by a linear function of the size (i.e. $O(n)$), and an *exponential-time* algorithm if its time complexity is a function $f(n) \in O(a^{n^{k_1}} \cdot n^{k_2})$ for some constants a, k_1, and k_2.

In some cases, we could be interested in relaxing the goal of an optimization problem by looking for a feasible solution s "sufficiently close" to an optimal solution s^*, where the distance between the solutions are generally expressed in term of performance ratio, $\frac{c(s)}{c(s^*)}$. An algorithm $\mathcal{A}$ for an optimization problem (P, c, obj) is an *approximation algorithm* if and only if, for every instance i it computes a feasible solution s. An approximation algorithm is a $f(n)$-*approximation algorithm* if and only if, for every instance i of size n, it computes a feasible solution s such that $\frac{c(s)}{c^*(i)} \geq f(n)$ if the problem is a maximization problem, or $\frac{c(s)}{c^*(i)} \leq f(n)$ if the problem is a minimization problem.

A computational complexity class is a collection of problems that requires related amounts of computational resources to solve them. In particular we are interested in two classes of decision problems: P and NP. P is the class of decision problems that are *solved* by a polynomial-time algorithm. NP is the class of decision problems that can be *verified* by polynomial-time algorithm. A *verifier* is an algorithm that recognizes instances i of P such that $(i, \textsc{Yes}) \in P$ in polynomial-time based on an addition input $c(i)$ called *certificate*.

We are also interested in two classes of optimization problems: APX and PTAS. APX is composed by optimization problems that have a k-approximation algorithm where k is a constant. An optimization problem P is in PTAS if and only if there exists a family of $(1 + \varepsilon)$-approximation polynomial-time algorithms for P for any constant $\varepsilon > 0$ if P is a minimization problem or any $\varepsilon < 0$ if P is a maximization problem. Such family of approximation algorithms is called *approximation scheme*.

The *time complexity lower bound* of a problem is defined as the best (=lowest) time complexity of an algorithm that solves the problem. Proving the time complexity lower bound of a problem is a hard task, because it must consider all the possible algorithms that solve the problem, known or not yet known. Therefore, in computational complexity theory, the (time) complexity of a problem is compared with the (time) complexity of other problems using the concept of *reduction* among problems. Generally speaking, a reduction from a problem P_1 to a problem P_2 provides a method to solve P_1 by using a solution of P_2. As a consequence, if P_1 is reducible to P_2, then P_2 is at least as hard as P_1. Different kinds of reductions exist, but we are interested in two of them: *Karp-reductions* and *L-reductions*. The first kind is defined between decision problems, while the second one is defined between optimization problems.

Definition 2.1 (Karp-reduction [6])**.** A decision problem P_1 is *Karp-reducible* to a decision problem P_2 if there exists an algorithm $\mathcal{R}$ that, given an instance i_1 of P_1, it computes an instance i_2 of P_2 such that $(i_1, \text{YES}) \in P_1$ if and only if $(i_2, \text{YES}) \in P_2$. The reduction is said to be a polynomial-time reduction if $\mathcal{R}$ is a polynomial-time algorithm.

Definition 2.2 (L-reduction [6])**.** A minimization (maximization) problem P_1 is *L-reducible* to a minimization (maximization) problem P_2 it there exists two functions f and g and two positive constants β and γ such that for any instance i_1 of P_1:

- $f(i_1)$ is an instance of P_2 that can be computed in polynomial-time;
- if i_1 has a non-empty set of feasible solutions, then instance $f(i_1)$ has a non-empty set of feasible solutions;
- for any feasible solution s_2 of $f(i_1)$, it is possible to compute in polynomial-time a feasible solution $g(i_1, s_2)$ of i_1;
- $c^*(f(i_1)) \leq \beta \cdot c^*(i_1)$;
- for any feasible solution s_2 of $f(i_1)$, $|c^*(i_1) - c(g(i_1, s_2))| \leq \gamma \cdot |c^*(f(i_1)) - c(s_2)|$.

Based on the concept of Karp-reduction, we say that a problem P is NP-hard if and only if any problem $P' \in \mathsf{NP}$ is Karp-reducible to P in polynomial-time. A decision problem P is NP-complete if $P \in \mathsf{NP}$ and P is NP-hard. If one polynomial-time algorithm for a NP-hard problem exists, then $\mathsf{P} = \mathsf{NP}$. However, in many years, nobody was able to design a polynomial-time algorithm for a NP-hard problem, thus it is widely believed that the two classes P and NP are separated, i.e. there exists a problem $P \in \mathsf{NP} \setminus \mathsf{P}$.

A problem P is APX-hard if the existence of an approximation scheme for P would imply the existence of an approximation scheme for every APX-complete

problem P' such that $P' \in \mathsf{APX}$. It is possible to prove that if $\mathsf{P} \neq \mathsf{NP}$ then a APX-hard problem P does not belong to PTAS. Moreover, it is also possible to prove that if a problem P_1 is L-reducible to a problem P_2, then $P_2 \in \mathsf{APX}$ (respectively, $P_2 \in \mathsf{PTAS}$) implies $P_1 \in \mathsf{APX}$ (respectively, $P_1 \in \mathsf{PTAS}$).

Finally, we are interested in a last class of decision problems called FPT (described in [36]). A pair (P, k), where P is a decision problem and k is a parameter, belongs to FPT if there exists a *fixed-parameter algorithm* for the pair, i.e. an algorithm that solves P in time $O(f(k) \cdot p(n))$ where f is an arbitrary function of k, and p is a polynomial function of the size of the instance n. The class FPT has been introduced to characterize computationally hard problems which have an "efficient" algorithm if the value of the parameter is small.

2.2 Vector Spaces and Matrices over the Finite Field $\mathbb{Z}_2$

Several results presented in this thesis are based on mathematical structures, called *vector spaces*, defined over the two element set $\mathbb{Z}_2$. In this section, we formalize the relevant concepts, we state the basic properties of such structures and we highlight some connections with matrices.

In order to define the concept of vector space, we first have to define the concepts of abelian group and field.

Definition 2.3 (Abelian group)**.** Let G be a set and $+$ a binary operation on G (i.e. a function $+ : G \times G \to G$). Then $(G, +)$ is an *abelian* (or *commutative*) *group* if and only if:

- $+$ is associative, $\quad \forall a, b, c \in G, (a + b) + c = a + (b + c)$;
- $+$ has an identity element, $\quad \exists 0 \in G : \forall a \in G, a + 0 = 0 + a = 0$;
- $+$ has the inverse element, $\quad \forall a \in G \ \exists b \in G : a + b = b + a = 0$;
- $+$ is commutative, $\quad \forall a, b \in G a + b = b + a$.

Definition 2.4 (Field)**.** Let F be a set, and let $+$ and $\cdot$ be two binary operations on F. Then $(F, +, \cdot)$ is a *field* if and only if:

- $(F, +)$ is an abelian group;
- $(F \setminus \{0\}, \cdot)$, where 0 is the identity element of $+$, is an abelian group;
- $\cdot$ is distributive over $+$, $\quad \forall a, b, c \in F, a \cdot (b + c) = a \cdot b + a \cdot c$.

We can now define the concept of *vector space*.

Definition 2.5. Let $(F, +, \cdot)$ be a field with 0 the identity element for $+$ on F and 1 the identity element for $\cdot$ on $F \setminus \{0\}$. Let$(V, +)$ be an abelian group, and $\cdot$ a function from $F \times V$ to V (external operation). Then $(V, F, \cdot)$ is a *vector space* over the field F if and only if:

- $\forall a, b \in F$, $\forall v \in V$, $(a \cdot b) \cdot v = a \cdot (b \cdot v)$;
- $\forall v \in V$, $1 \cdot v = v$;
- $\forall a \in F$, $\forall u, v \in V$, $a \cdot (u + v) = a \cdot u + a \cdot v$;
- $\forall a, b \in F$, $\forall v \in V$, $(a + b) \cdot v = a \cdot v + b \cdot v$.

Given a vector space $(V, F, \cdot)$, we call vector any element of V and scalar any element of F.

Let $\oplus$ be the binary operation on $\mathbb{Z}_2 = \{0, 1\}$ defined as $0 \oplus 0 = 1 \oplus 1 = 0$ and $0 \oplus 1 = 1 \oplus 0 = 1$, and let $\cdot$ be the binary operation on $\mathbb{Z}_2$ defined as $0 \cdot 0 = 0 \cdot 1 = 1 \cdot 0 = 0$ and $1 \cdot 1 = 1$. It is easy to see that $(\mathbb{Z}_2, \oplus, \cdot)$ is a field.

Now let us consider the set $\mathbb{Z}_2^n$ composed by all the ordered n-tuple over $\mathbb{Z}_2$. Let $v \in \mathbb{Z}_2^n$ and denote with $v[i]$ the i-th element of the tuple v. Define the external operation $\cdot : \mathbb{Z}_2 \times \mathbb{Z}_2^n \to \mathbb{Z}_2^n$ as the function that associates with $(a, v) \in \mathbb{Z}_2 \times \mathbb{Z}_2^n$ the tuple $u \in \mathbb{Z}_2^n$ such that $u[i] = a \cdot v[i]$ for all $i = 1 \dots n$. It is easy to see that $(\mathbb{Z}_2^n, \mathbb{Z}_2, \cdot)$ is a vector space. We call vectors of such a vector space binary vectors.

From now on, we denote with V the vector space $(V, F, \cdot)$ if the field F and the external operation $\cdot$ are clear from the context.

A vector space V over the field K is a subspace of a vector space U over K if V is defined over the same vector space operations of U and $V \subseteq U$.

Given a set of vectors $B = \{v_1, \dots, v_k\} \subseteq V$, a linear combination of B is an expression $\alpha_1 \cdot v_1 + \dots + \alpha_k \cdot v_k$ where $\alpha_1, \dots, \alpha_k$ are scalars. A subset B of vectors is *linearly dependent* if there exists $b \in B$ such that b can be obtained as a linear combination of $B \setminus \{b\}$. Otherwise we say that B is *linearly independent*. The set of the linear combinations of a set of vectors B is a subspace of V and is denoted with $V(B)$. A basis of the vector space V is a minimal-cardinality set B of vectors such that $V(B) = V$. All the bases of a vector space V are linearly independent and have the same cardinality (called *dimension* of V and denoted with $\dim V$).

The following properties hold also for generic vector spaces, but, for simplicity, let us focus on the vector space $\mathbb{Z}_2^n$. Denote with $\mathbf{0}$ a binary vector composed only by zeroes. Let M be a $n \times m$ binary matrix (i.e. a matrix whose entries belong to $\mathbb{Z}_2$) and let $\cdot$ be the usual matrix dot product. We denote with $M[i, j]$ the entry at row i and column j. The transpose of M is a $m \times n$ binary matrix M^T such that $M^T[i, j] = M[j, i]$, for all $i = 1 \dots m$ and $j = 1 \dots n$. The matrix can be seen as a collation of n row vectors of $\mathbb{Z}_2^m$ (its rows) or as a collation of m column vectors of $\mathbb{Z}_2^n$ (its columns). The rank of M (denoted with $\text{rank}(M)$) is the maximum number of

linearly independent row vectors or, alternatively, the maximum number of linearly independent column vectors.

Matrix M determines two vector spaces:

1. the *column space* (or *image*), defined as $\text{im}(M) = \{M \cdot x \mid x \in \mathbb{Z}_2^m\}$;
2. the *kernel*, defined as $\ker(M) = \{y \in \mathbb{Z}_2^m \mid M \cdot y = \mathbf{0}\}$.

The rank-nullity theorem states that $\text{rank}(M) + \dim \ker(M) = m$.

A matrix M is in *row echelon form* if (i) all rows that have at least a 1-entry (non-zero rows) are above any row that has only 0-entries (all-zero row), and (ii) the first 1-entry in a row is strictly to the right of the first 1-entry of the row above it. A matrix M is in *reduced row echelon form* if it is in row echelon form and the first 1-entry of a row is the only 1-entry of its column. The first 1-entry of each row in a matrix M in row echelon form is called pivot.

To each set B of vectors of $\mathbb{Z}_2^n$ we associate the $n \times |B|$ binary matrix M_B whose column vectors are exactly the vectors in B. Almost all the basic problems on the vector space $\mathbb{Z}_2^n$ can be solved using the Gauss elimination algorithm. In particular, given a $n \times m$ binary matrix M, the Gauss elimination algorithm computes in time $O(\min(n, m)nm)$ the row echelon form of M. A simple extension of the Gauss elimination algorithm is the Gauss-Jordan elimination algorithm, which computes the reduced row echelon form of M. In particular we will be interested in the following applications of the Gauss elimination algorithm:

- computing the rank of a matrix M, that is equal to the number of non-zero rows in the row echelon form of M;
- deciding if a set of vectors $B \subseteq \mathbb{Z}_2^n$ is linearly dependent (by comparing the rank of M_B with n, if it is smaller the set is linearly dependent);
- finding a basis of $\ker M$;
- finding the linear combination of a basis equal to a given vector of the space.

2.3 Graph Theory

Informally, graphs are structures that represent relationships between pairs of objects. More formally, a graph G is a pair (V, E) where V is the set of its vertices (vertex set) and E is the set of its edges (edge set). An edge e of E is an unordered pair of vertices (u, v), with $u, v \in V$. Notice that, since the pair is unordered, (u, v) and (v, u) are the same edge. Vertices are sometimes called nodes and edges are sometimes called arcs. The vertex set of a graph G is denoted with $V(G)$ and the

edge set with $E(G)$. Unless stated otherwise, we implicitly assume that the graph is simple, i.e. $(v, v) \notin E(G)$.

An edge $e = (v, u)$ is incident to a vertex x if $x = v$ or $y = u$. If there exists an edge $e = (u, v)$ we say that vertices u and v are adjacent, or that e is an edge between u and v, or that u and v are the endpoints of e. The set of edges incident to a vertex v is denoted with $E(v)$. The cardinality of $E(v)$ is the degree of v. If the degree of every vertex is k, then the graph is k-regular. A 3-regular graph is called cubic graph. If the degree of a vertex is 0, then the vertex is isolated. The minimum (maximum) degree of a graph is the minimum (maximum) degree of one of its vertices.

A graph $G' = (V', E')$ is a subgraph of a graph $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. Given $V' \subseteq V(G)$, the induced subgraph $G(V')$ is the graph $(V', \{(u, v) \in E(G) \mid u, v \in V'\})$.

A sequence $P = \langle v_1, v_2, v_3, \ldots, v_k, v_{k+1} \rangle$ is a path of length k of a graph $G = (V, E)$ if $(v_i, v_{i+1}) \in E$ for each $i = 1 \ldots k$. An edge (u, v) is contained in a path $P = \langle v_1, \ldots, v_{k+1} \rangle$ if $u = v_i$ and $v = v_{i+1}$ for some $i = 1 \ldots k$. We say that a path connects v and u if $v = v_1$ and $u = v_{k+1}$. A path is simple if it does not contain repeated vertices. A path of length at least 3 that connects v to v is a cycle. A graph G is acyclic if no paths of G are cycles.

We say that a graph is connected if for each pair of vertices there exists a path that connects them, otherwise we say that it is disconnected. A subgraph $G' = (V', E')$ of $G = (V, E)$ is a connected component of G if G' is connected and no edges in $E \setminus E'$ are incident to a vertex of V'. A connected acyclic graph is a tree, while a disconnected acyclic graph is a forest. Given a graph $G = (V, E)$ and a set $V' \subseteq V$, the graph $G \setminus V'$ is the subgraph of G obtained by removing the vertices V' from V and every edge $e \in E$ incident to a vertex of V'. A separator of a connected graph G is a subset V' of $V(G)$ such that $G \setminus V'$ is disconnected. A graph is k-connected if there exists a separator of cardinality k but not a separator of cardinality $k - 1$. A graph G is bipartite if there exists a bipartition $\{V_1, V_2\}$ of $V(G)$ such that each edge of G has an endpoint in V_1 and an endpoint in V_2. Given a graph G and a bipartition $\{V_1, V_2\}$ of $V(G)$, the set C of edges of G that have an endpoint in V_1 and an endpoint in V_2 is called a cut of G. Given a cut C of G, the edges in C are called crossing edges and edges in $E(G) \setminus C$ are called non-crossing edges.

A spanning tree T of a connected graph G is an acyclic connected subgraph of G such that $V(T) = V(G)$. A spanning forest F of a (disconnected) graph G is a subgraph of G such that, for each connected component C of G, a connected component T of F is a spanning tree of C.

A graph G is vertex-labelled if there exists a function $\lambda_v : V(G) \to L_v$ that associates each vertex v with a label $\lambda_v(v)$. A graph G is edge-labelled if there exists a function $\lambda_e : E(G) \to L_e$ that associates each edge e with a label $\lambda_e(e)$.

A directed graph is a pair $G = (V, E)$ where V is set of vertices, and E a set of ordered pairs of vertices. In a directed graph $G = (V, E)$, edge (u, v) is different from (v, u) and loops are admitted, i.e. (u, u) may belong to E. An oriented graph is a graph $G = (V, E)$ together with a function $d : E \to V$ (called edge-direction function) such that $d(u, v) = u$ or $d(u, v) = v$. In other words, an oriented graph is a (undirected) graph in which every edge is associated with a direction. A directed path P of a directed graph $G = (V, E)$ is a sequence of vertices $\langle x_1, \ldots, x_{k+1} \rangle$ such that $(x_i, x_{i+1}) \in E$ for each $i = 1 \ldots k$. An oriented path P of an oriented graph $G = (V, E)$ with edge-direction function c is a sequence $\langle x_1, \ldots, x_{k+1} \rangle$ such that $(x_i, x_{i+1}) \in E$ and $c(x_i, x_{i+1}) = x_{i+1}$ for each $i = 1 \ldots k$. A cycle C in a directed (oriented, resp.) graph is a directed (oriented, resp.) path where the first vertex is equal to the last vertex. The set of incoming edges in a vertex v^* of a directed graph $G = (V, E)$ is the set $E_I(v^*) = \{(v^*, u) \in E\}$. Similarly, the set of incoming edges in a vertex v^* of an oriented graph $G = (V, E)$ with edge-direction function c is the set $E_I(v^*) = \{(v^*, u) \in E \mid c(v^*, u) = u\}$. The set of outgoing edges of a vertex v^* in a directed (or oriented) graph is $E_O = E(v^*) \setminus E_I(v^*)$, where $E(v^*)$ is the set of edges incident to v^* defined as in the undirected case. The indegree (outdegree, resp.) of a vertex v in a directed (or oriented) graph is the cardinality of the set $E_I(v^*)$ ($E_O(v^*)$, resp.).

Part I

Haplotype Inference Problems

3 Haplotype Inference Problems

The Haplotype Inference problem is the computational problem of distinguishing (inferring) the genetic material that each individual in a population has inherited from each parent (*haplotype*) starting from a less informative source of information (*genotype*). The computational problem is mainly motivated by cost considerations: Haplotype data have been proven useful for several genetic studies but obtaining them directly from the individuals of a population is costly and time-consuming. Obtaining genotypes, instead, is much cheaper and faster.

The inference process is guided by a model of genetic evolution and inheritance. Unfortunately an "universal" model that can guide the inference process in every situation and on every dataset is not known. Therefore several different models have been proposed, and for each model a different computational problem arises.

This chapter introduces the concepts and terminology related to Haplotype Inference problems (Section 3.1) and reviews the most important approaches for HI appeared in literature. In particular we classify such approaches with respect to the information that are available about the population. Therefore we consider and review approaches for populations of unrelated individuals (Section 3.2) and approaches for populations in which family relationships between individuals are present (Section 3.3).

3.1 Introduction

The genome of almost all organisms is organized in macromolecules of DNA called *chromosomes*. Along each chromosome there are some positions, called *loci*, where "particularly significant" features are located. The exact meaning of "particularly significant" depends on the context of the problem we are dealing with. The state of the feature in a given locus is called *allele*, and, based on the number of different states a feature exhibits in a population, loci are classified as biallelic (if only two different states are possible) or multi-allelic (if more than two different states may appear). Since the set of different alleles of a given locus is known in advance, for convenience each allele is encoded by a numeric identifier. For example, given a biallelic locus l, the major allele (i.e. the allele that appears more frequently in the population) is represented by the value 0, while the minor allele (i.e. the allele that appears more rarely in the population) is represented by the value 1. The sequence

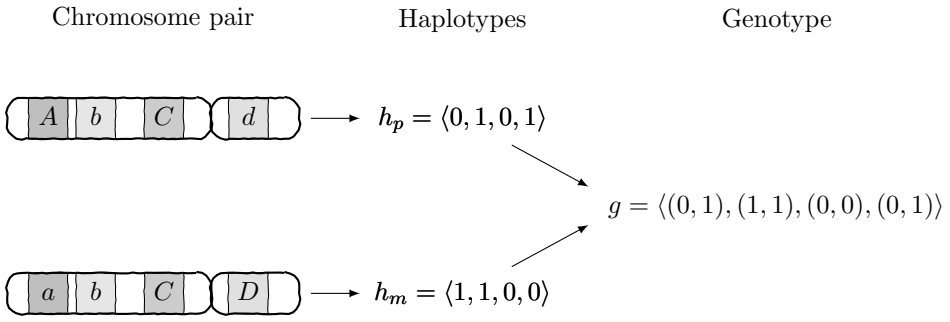


Figure 3.1: Example of haplotypes and genotype of an individual.

of the alleles (or of their identifiers) that appear at a set of loci on a chromosome of an individual is called *haplotype* of the individual.

The most evolved organisms have two copies of almost all chromosomes. One copy is inherited from one parent, and one copy is inherited from the other parent. The two copies, also defined as the two *homologous* chromosomes, determine the same set of biological traits and are usually almost identical. Therefore, the same locus is present in both the copies and each individual possesses two alleles for every locus or, in other words, each individual has two (possibly distinct) haplotypes. The sequence of unordered pairs of the alleles that an individual possesses at the loci of a pair of homologous chromosomes is called *genotype* of the individual. The genotype of an individual can be considered as the conflation of its haplotypes: For example if the two haplotypes of an individual are $h_p = \langle 0, 1, 0, 1 \rangle$ and $h_m = \langle 1, 1, 0, 0 \rangle$, then its genotype is $g = \langle (0, 1), (1, 1), (0, 0), (0, 1) \rangle$. A locus l is *homozygous* in a genotype g if the pair of alleles at locus l, denoted by $g[l]$, is composed by equal values, otherwise is *heterozygous*. In the previous example, illustrated in Figure 3.1, the first and the fourth loci are heterozygous while the second and the third ones are homozygous.

A set of features that is usually considered in genetic studies (such as linkage analysis, gene mapping, and association studies) is mainly composed by Single Nucleotide Polymorphisms (SNPs), i.e. loci where different bases appear among the members of a population. Clearly, in this case, the allele at each SNP locus is determined by the base that appears in that position. Moreover, since for almost all SNPs only two bases are present in a population, SNP loci can be considered as biallelic. When the set of loci that are considered is composed only by biallelic loci, the genotype is usually represented by a sequence on the alphabet $\{0, 1, 2\}$, where symbol 0 stands for the pair of alleles $(0, 0)$, symbol 1 stands for $(1, 1)$, and symbol 2 stands for $(0, 1)$. This mapping is a widely-adopted convention and, thus, it will be adopted in this thesis. However other alternatives exists: For example some

authors replace the symbol 2 of our notation with the symbol ? (or $*$), while other authors map the symbol 1 to the heterozygous pair $(0, 1)$ and the symbol 2 to the homozygous pair $(1, 1)$. In particular, the last mapping is mainly adopted by works that use ILP formulations because it simplifies the specification of constraints.

The determination of the haplotypes of each individual in a given population is a time-consuming and costly operation, while determining their genotype is far more quick and cheaper. Therefore, large-scale studies (i.e. studies that consider a vast population and/or a large number of loci) usually prefer (due to cost considerations) to determine the genotypes instead of haplotypes, even if haplotype data has been shown to be more informative than genotypes.

Computational methods have been proposed to infer haplotypes starting from the genotypes of a population. Their aim is to recover computationally the information provided by the haplotypes without incurring in the increased costs of determining them by biological assays. While this problem, called *haplotype inference* or *haplotyping* or *phasing*, is an easy task for homozygous loci, it becomes hard on heterozygous loci. Indeed, the allele of the two haplotypes on a homozygous locus is trivially equal to the only allele of the genotype on the same locus. Instead, without any prior information or assumption, it is not possible to resolve the ambiguities that heterozygous loci pose. In fact, if a genotype g is heterozygous at two biallelic loci, then two distinct and equally-probable pairs of haplotypes may have generated g: the first one is composed by the haplotypes $\langle 0, 0\rangle$ and $\langle 1, 1\rangle$, while the other one is composed by the haplotypes $\langle 0, 1\rangle$ and $\langle 1, 0\rangle$. We say that a pair of haplotypes h_1 and h_2 *resolves* a genotype g if g is the conflation of the two haplotypes. To guide the choice of the "right" set of haplotypes, we need a *genetic model* which specifies how the haplotypes have been evolved and how have been inherited by the individuals.

At the most abstract level, the computational problem of inferring the haplotypes of the individuals of a population starting from their genotypes can be defined as follows.

Problem 1. Haplotype Inference (HI).
Input: The set $G = \{g_1, \ldots, g_n\}$ of genotypes of the individuals of a population, and a genetic model M.
Output: A set $H = \{h_1, \ldots, h_m\}$ of haplotypes that "satisfy" the genetic model M and such that for each genotype g_i there exists a pair of haplotypes h_1^i and h_2^i of H that resolves g.

Clearly the genetic model is an element that heavily influences the quality of the results and deeply determines the characteristics of the computational problem. Therefore the Haplotype Inference problem can be regarded as a *family* of closely-related computational problems which have a unique aim: to infer the "best" set of

haplotypes which resolves a given set of genotypes according to the genetic model that has been assumed in the specific HI problem.

In the literature no model can claim to be the most suitable for all instances and datasets: often a quality improvement of the results achieved by a model implies a consistent increase in the computational resources required and/or the presence of additional assumptions. As a consequence, several models and several algorithmic have been proposed to tackle the Haplotype Inference problem.

Several works, such as [14, 46, 56–58, 70, 79, 90], have extensively reviewed the state-of-the-art in Haplotype Inference. Here we present an overview of the principal models and approaches and we refer to one of the works above for a more in-depth presentation.

The approaches toward the solution of the Haplotype Inference problem can be classified in two different categories: *statistical methods* and *combinatorial methods*. Moreover, a second orthogonal classification is based on the information that we have about the population: *population-based methods* and *pedigree-based methods*. In population-based methods, the individuals are assumed to be unrelated, i.e. no family relationships are present, while pedigree-based methods receive as input data also a description of the parental relationships among population members. In the following we briefly describe the principal methods proposed in literature in each of the resulting four categories.

3.2 Population-based Methods

Population-based methods deal with populations in which members do not have kinship relationships or such relationships are not known. They heavily depend on the reference genetic model: if the actual genotype data depart from such assumptions, the accuracy and the overall quality of the results may substantially decrease or, in some cases, especially on combinatorial methods, a solution cannot be found at all. The following two sections describe the most important statistical and combinatorial approaches for Haplotype Inference on unstructured populations.

3.2.1 Population-based Statistical Methods

Among statistical methods, one of the prominent approaches is represented by the *Maximum Likelihood* formulation [39]. In this case the solution of the problem is a set of haplotypes composed by the haplotypes that maximize the likelihood of observing the given set of genotypes. The maximization of the likelihood is performed by the Expectation Maximization (EM) algorithm [32] that computes the haplotype frequencies that maximize the likelihood of observing the genotypes. From haplotype frequencies it is then possible to reconstruct a possible solution set of haplotypes by picking, for each genotype, the pair of haplotypes with maximum

frequency that resolve the genotype. Since the EM algorithm does not guarantee to reach a global optimum, the authors suggest to try several set of initial parameters (haplotype frequencies) and to pick the final solution which maximizes the likelihood function. In the ML approach, the genetic model is implicitly stated in the assumptions of the formulation. Indeed the ML approach assumes that the population satisfies the Hardy-Weinberg principle and that random mating between the individuals has occurred. Therefore, the reliability of the results is influenced by deviations from Hardy-Weinberg equilibrium and from random mating (i.e. the presence of assortative mating or inbreeding in animals).

A second remarkable exponent in the class of statistical approaches is *PHASE* [104, 105]. This method is based on a Bayesian approach which starts with an initial guess of the haplotype set and iteratively updates the individuals' haplotypes based on the other haplotypes. The update step is performed using a Gibbs sampling technique and the method is guaranteed of converging to the desired posteriori distribution after a "sufficiently large" number of step. The genetic model underlying this approach represents the key component from which the conditional distribution used in the Gibbs sampler is calculated. In fact, the authors assume that the genetic model is approximately coalescent (i.e. the evolution history of the haplotypes has a "tree-like" structure).

A third notable statistical methods is called *partition-ligation (PL)* [89]. In this case a Bayesian approach similar to PHASE is employed on small blocks that partition the original genotypes instead of considering the whole genotypes. The results of each pair of contiguous blocks are then combined until a complete solution is found. On such small block, whose boundaries are chose based on a measure of Linkage Disequilibrium, the computation of the conditional distribution of the Gibbs sampler is not coalescent-based as in PHASE. Nevertheless, also PL assumes a particular genetic model: in fact, since it partitions genotypes in small contiguous blocks, it assumes a "block-like" structure of haplotypes, assumption that has received substantial empirical support.

3.2.2 Population-based Combinatorial Methods

Combinatorial methods try to overcome the main limitation against a broad applicability of statistical approaches: the excessive amount of computational resources they require. Statistical methods potentially investigate a number of haplotypes that is exponential in the number of loci considered and/or in the size of the population. In combinatorial methods, instead, the genetic model is translated in a combinatorial formulation and then (hopefully) efficient algorithms are devised based on the properties of the formulation.

The first combinatorial method has been proposed by Clark [29]. This algorithm was based on the iterative "resolution" of unsolved genotypes by combining

a previously-computed haplotype with a new one. Clark's algorithm requires an initial set of unambiguous genotypes (i.e. genotypes with at most one heterozygous locus) from which the initial set of haplotypes is computed. In case every genotype is ambiguous, the initial haplotype set cannot be computed and Clark's method is not applicable. The underlying genetic model of Clark's algorithm (and a partial justification of its soundness) is empirically based on the infinite-site model with random mating. In the infinite-site model each locus is mutated at most once during the haplotype evolution, and since random mating is assumed, the haplotypes of the individuals that have been considered should be the most frequent of the whole population. Therefore there should exist a subset of haplotypes which resolves most of the genotypes and from which the other haplotypes can be recovered.

Two extensions of Clark's algorithm have been proposed by Gusfield *et al.*: namely *maximum resolution* [53] and a *consensus approach* to reconcile the results obtained by multiple runs of Clark's algorithm [92].

The second important combinatorial formulation of the Haplotype Inference problem is represented by the *Perfect Phylogeny Haplotyping (PPH)* problem [54]. In this case, the problem asks for a set of haplotypes which evolution history is compatible with an explicit genetic model: the coalescent model combined with the infinite-site assumption. We recall that in the coalescent model, the haplotype evolution can be represented by an oriented acyclic graph and that under the infinite-site assumption each locus has mutated at most once. An important positive characteristic of such problem is that solutions can be efficiently computed: indeed [13], [34], and [97] had proposed three linear-time algorithms for solving the problem.

The Perfect Phylogeny model requires that no recombinations have occurred during the evolution of haplotypes. A recombination is a variation event where a new sequence is created by concatenating a prefix of an original sequence with the remaining suffix of another original sequence. Requiring the absence of recombinations had limited the direct applicability of the PPH method to long genotypes, where some recombinations have likely occurred. Some other approaches [59, 103] relax such strict requirement and are applicable when some recombination events are present.[1]

The Perfect Phylogeny model has been also used on a different kind of genotype data called *xor-genotypes*. These genotypes are produced by the technique known as Denaturing High-Performance Liquid Chromatography (DHPLC) [117] which is able to distinguish between homozygous and heterozygous loci but it cannot determine ("call") the allele that is present in homozygous loci. The *Xor*

[1] The approach of Halperin and Eskin [59] is not entirely combinatorial: it uses a PPH method to infer haplotypes for small blocks that are then combined in long range haplotypes via a ML approach. Since a PPH method is its inner routine, we (arbitrarily) chose to list it in the combinatorial methods.

Perfect Phylogeny Haplotyping (XPPH) problem has been proposed and solved by Barzuza *et al.* [8, 9]. The basic idea of their almost linear-time solution algorithm is the reduction of the XPPH problem to the *Graph Realization (GR)* problem. A Graph Realization of a family of sets of labels is a labelled tree in which all the given sets of labels induce a path. They show that the Graph Realization of the set of xor-genotypes is precisely the evolution history of the individuals' haplotypes and, thus, they solve the problem via one of the existent algorithms for GR [11, 42, 108]. Despite the inherent loss of information due to the absence of the allele of homozygous loci, they also show that adding the full genotypes of three (carefully selected) individuals suffices to recover a unique set of haplotypes if the graph realization is unique.

The last major approach for HI is based on the *parsimony* principle. In this case, the HI problem is regarded as an optimization problem (called *Haplotype Inference by Pure Parsimony, HIPP*) where one wants to minimize the number of distinct haplotypes needed to solve the given genotypes. The genetic model is implicit: if the rate of variation events is low and individuals come from a restricted number of ancestors, the number of their distinct haplotypes should be small compared to the number of possible haplotypes. Unfortunately, the HIPP problem is NP-hard (the proof, along with the formulation conception, is attributed to Earl Hubbel in [57]) and APX-hard [71]. To tackle the computational intractability of the problem several heuristic [61, 82], ILP-based [20, 55, 73], or approximation [71] algorithms have been proposed. Moreover restricted cases of the problem with polynomial time solutions have been studied [72, 109].

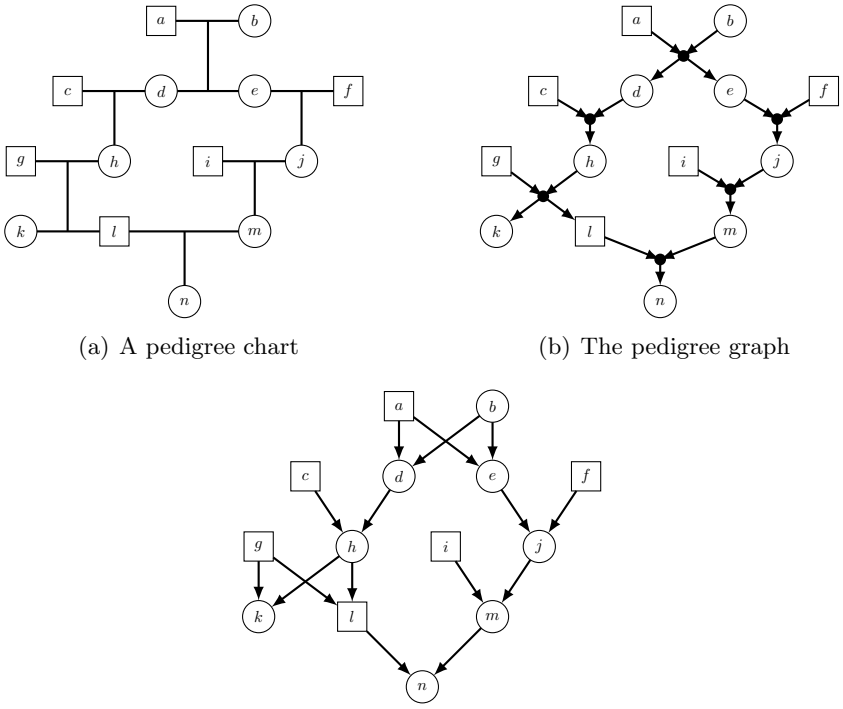
3.3 Pedigree-based Methods

Parental relationships (or, more in general, kinship) provide an invaluable source of information in the Haplotype Inference problem. Indeed, assuming *Mendelian Inheritance law* and in absence of genetic variation events, each offspring receives one haplotype from the mother and one from the father. Therefore, the resolution of the offspring haplotypes is constrained by the resolution of parent haplotypes, and vice versa.

In the following we will present the related terminology, and we will briefly introduce the most important statistical and combinatorial methods for HI on structured populations.

3.3.1 Terminology

Parental relationships are represented by a structure called *pedigree chart*. An example of pedigree chart is depicted in Figure 3.2(a). In such a representation, each individual is represented either by a square (if it is male) or a circle (if it is



(a) A pedigree chart

(b) The pedigree graph

(c) The simple pedigree graph

Figure 3.2: Example of representations of the same pedigree.

female), and an edge connects parents to their children. Conventionally, edges are oriented from top to bottom. In the example, individuals a and b are the parents of both d and e.

A pedigree chart provides a clear representation of the parental relationships but when dealing with computational methods a more formal (equivalent) representation is used: the *pedigree graph*.

Definition 3.1 ([76])**.** A *pedigree graph* (or *marriage node graph* [23]) is a connected oriented acyclic graph $G = (V, E)$, where:

- $V = M \cup F \cup N$, M and F are the sets of male and female nodes, N is the set of mating nodes;
- edges connect either an individual node to a mating node, or a mating node to an individual node;
- the indegree of an individual node is at most one (edge coming from a mating node);

- the indegree of a mating node is 2 (one edge coming from a male node representing the father and one coming from a female node representing the mother);
- the outdegree of a mating node must be greater than zero.

Figure 3.2(b) represents the pedigree graph associated to the pedigree chart of Figure 3.2(a). Mating nodes are represented by small black circles, while male and female nodes are represented by square and circle vertices, respectively.

Sometimes the pedigree graph, although formal, is cumbersome. Therefore a lighter representation has often been used. Unfortunately also such a representation has been called pedigree graph in the literature (see, for example, [114]), and to avoid potential confusion we refer to it as *simple pedigree graph*.

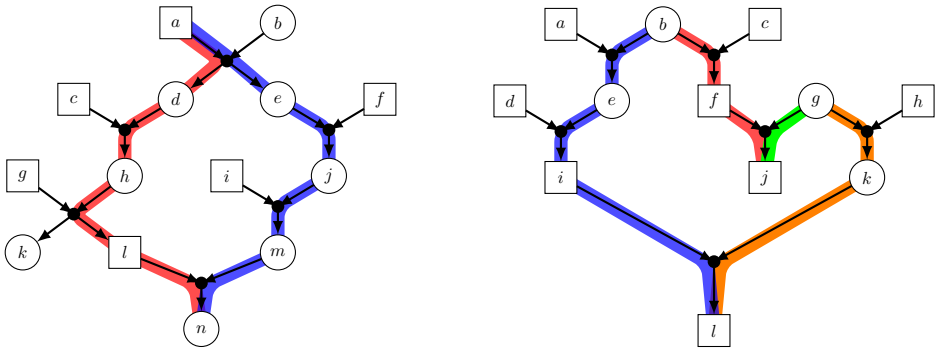
A *simple pedigree graph* can be obtained from a pedigree graph by removing mating nodes and connecting directly parents to their children. An example of simple pedigree graph is depicted in Figure 3.2(c).

The pedigree chart, the pedigree graph, and the simple pedigree graph are all representations of the relationships among individuals. When we are interested in the family relationships and not in the peculiar characteristics of each representation, we will use the general term of *pedigree*.

The triplet composed by an individual and its parents is called *trio*, and it is conventionally denoted by a triplet (f, c, m) where f is the father, c is the child, and m the mother. The set composed by parents and their children is a *nuclear family*. If parents of an individual are not included in the pedigree, such individual is called *founder*. In the example depicted in Figure 3.2, (a, d, b) is a trio, $\{a, b, d, e\}$ is a nuclear family, and $\{a, b, c, f, g, i\}$ is the set of founders of the pedigree.

If an individual and one of its descendants are connected by two distinct oriented paths in the pedigree graph (or in a simple pedigree graph), then the pedigree has a *mating loop* [26, 78, 84, 114]. For example, the pedigree graph of Figure 3.3(a) has one mating loop which involves individuals a and n connected by two paths $\langle a, e, j, m, n\rangle$ and $\langle a, d, h, l, n\rangle$ (plus some mating nodes). We want to remark that the presence of a cycle in a *simple pedigree graph* (if we ignore edge directions) does not imply the presence of a mating loop. Indeed each nuclear family with more than one offspring forms a cycle (if edge directions are ignored) in the simple pedigree graph but not in the pedigree graph.

The definition of mating loop that we presented is the strictest possible: some authors, instead, define a mating loop as a cycle in the *pedigree graph* if edge directions are ignored [18, 27, 35]. The difference is subtle but nevertheless present: Suppose to have the following four trios: (a, b, c), (e, d, c), (e, f, g), and (a, h, g). If we depict the pedigree graph, we will found that founders and mating nodes induce a cycle in the pedigree graph (if we ignore the edge direction). Thus, in the second definition, such a cycle is a mating loop. However, there does not exist two



(a) A non-tree pedigree

(b) A tree pedigree

Figure 3.3: Examples of non-tree and tree pedigrees. Oriented paths that compose a (undirected) cycle have been highlighted.

distinct paths which connect the same pair of individuals, thus there are no mating loops according to the first (stricter) definition. The previous example seems to be a limit case that cannot be encountered in practice: however there exists other plausible multi-generational pedigrees in which mating loops are present in the broader definition but not in the stricter one (see, for example, Figure 3.3(b)). The difference, although small, had lead to some ambiguities in the literature. For example, the same algorithm is presented in [26] and [27]: in the first version the broader definition is assumed, while in the second version the stricter one is considered. However, adopting the stricter definition in the second version of the work [27] seems invalidate the proof of their Lemma 10. We define *tree pedigree* a pedigree which does not contains mating loops. A tree pedigree is a *restricted-tree pedigree* if it does not contain mating loops according to the broader definition. For example, the tree pedigree of Figure 3.3(a) is a tree pedigree but not a restricted-tree pedigree.

Since we are interested in Haplotype Inference, a genotype is associated to each individual of the pedigree, leading to the *genotyped pedigree*. Similarly, a *haplotyped pedigree* is a pedigree in which every individual has associated a pair of haplotypes. The genotype associated with an individual i in a genotyped pedigree is denoted with g_i, while the two haplotypes associated to i in a haplotyped pedigree are denoted with h_i^1 and h_i^2. Given a haplotyped pedigree P_h and a genotyped pedigree P_g of the same pedigree graph P, the haplotyped pedigree is said to be *consistent* with the genotyped pedigree if for each individual i the pair of haplotypes h_i^1 and h_i^2 of P_h resolves the genotype g_i of P_g. Given a locus of a genotype, the *parental source (PS)* of the alleles at such locus is the indication of which parents the two alleles come from. Clearly, the PS is meaningful only for heterozygous loci, since

the two alleles at homozygous loci are indistinguishable. Conventionally the PS information is 0 if the allele with the minimum identification number has been inherited from the father, 1 otherwise. Since also parents are diploid, an additional information is sometimes useful, the *grandparental source (GS)*, which specifies the grandparent from which the allele has been inherited. A *haplotype configuration* is the assignment of parental source to each allele of the individuals' genotypes. Obviously a haplotype configuration of a genotyped pedigree P_g induces a haplotyped pedigree P_h consistent with P_g, and vice versa.

A second level of consistency, *Mendelian consistency*, could be required. A haplotype configuration of a trio (f, c, m) is said to be *Mendelian consistent* at locus l if the pair of alleles $(a', a'') = g_c[l]$ is composed by one allele presents in the paternal genotype at locus l ($g_f[l]$) and by one allele presents in the maternal genotype at locus l ($g_m[l]$).

During the inheritance of (half) genome from one parent to the child, *genetic variations* may arise. In other words, the haplotype of the offspring can be different from both haplotypes of the parent. Two kinds of genetic variation events are usually studied: *recombinations* and *mutations*. Let h be the haplotype of an individual c inherited from its parent p, and let $h' = h_p^1$ and $h'' = h_p^2$ be the haplotypes of p. A *recombination* (or crossover) at locus l between p and c is the event in which haplotype h has been generated by concatenating a prefix $h'[1..l-1]$ of h' with the remaining suffix $h''[l..n]$ of the other haplotype h''. Locus l is called starting position (or, simply, position) of the recombination. Multiple recombinations are also possible: in this case h is the concatenation of alternating blocks of h' and h''. For example, two recombinations in position l_1 and l_2, respectively, produce the haplotype $h := h'[1..l_1 - 1]h''[l_1..l_2 - 1]h'[l_2..n]$. A *mutation* is an event which transforms an allele of parent haplotype to distinct allele in child haplotype. More formally, suppose that c inherits haplotype h' from p. A mutation at locus l between p and c is the event which replaces in h the allele $a' = h'[l]$ with a different allele $a = h[l]$. Recombinations are believed to be one of the most common (although quite rare) form of variation events in diploid organisms. Moreover, recombinations events usually start at positions located in particular regions, called *recombination hotspots* (see [88, 106, 107] for humans), thus a synthetic estimation of the recombination rate would be meaningless. Instead, mutations are believed to be much less frequent than recombinations. As a consequence, an accurate estimation of the mutation rate has been an elusive aim: a recent study [121] had reported a mutation rate of $3 \cdot 10^{-8}$ mutations/nucleotide/generation.

The general haplotype inference problem on pedigree data asks for a haplotyped pedigree (or, equivalently, a haplotype configuration) consistent with a given genotyped pedigree. Also in this case, the choice of the "best" haplotype configuration among the various possibilities is guided by a genetic model underlying each

method. In the following we will overview the most important methods which have appeared so far in the literature.

3.3.2 Pedigree-based Statistical Methods

The first breakthrough in HI over pedigree data is probably due to Lander and Green [74] and later improved by Kruglyak *et al.* [69]. In their work, they proposed a method to calculate the most likely haplotype configuration of a given genotyped pedigree, based on a Expectation Maximization algorithm. The crucial aspect of their approach is the use of Hidden Markov Chains, where the l-th hidden state is the vector of the grandparental sources (called *inheritance vector*) of all individuals at locus l, the transition between state l and $l+1$ is regulated by the recombination frequency θ_l between the two consecutive loci, and the observable state is the set of genotypes (at locus l). The computational complexity of Lander-Green algorithm is linear in the number of loci but exponential in the number of non-founder individuals, thus it is only applicable for moderately long genotypes and small pedigrees. Before the Lander-Green algorithm, the Maximum Likelihood estimation was performed via the Elson-Stewart algorithm [38], which computes the probability of all individuals' haplotypes conditional on the parental haplotypes, the individual's genotype, and the descendants' genotypes. Since all possible haplotypes are considered, the Elson-Stewart algorithm requires exponential time in the genotype length, thus it is only applicable for moderately large pedigrees and short genotypes.

Most of the subsequent approaches were essentially aimed to the reduction of the computational resources that are required by Lander-Green and Elson-Stewart algorithms. For example, Allegro2 [50] is a tool developed by Gudbjartsson *et al.* which employs multiterminal binary decision diagrams to compactly store probability distributions, in order to reduce both time and memory requirements. A second tool that implements and improves the Lander-Green algorithm is Merlin [1] by Abecasis *et al.*. This software achieves a performance gain by representing the inheritance vectors (gene flow, in their notation) as sparse binary trees and avoiding unnecessary computations based on symmetry considerations. Time-complexity analysis of both tools would be uninformative because the running time heavily depends on pedigree structure and the presence/absence of the genotypes of some individuals. A recent survey [46] reports that the original Lander-Green algorithm [69], Allegro2, and Merlin are applicable on pedigrees of moderate size (up to 40 individuals), but Allegro2 and Merlin can manage genotypes considerably longer (thousands of loci) than the Lander-Green algorithm.

The spacing (on chromosomes) between genotype loci is getting smaller and smaller with the rapid development of genotyping technologies and platforms. For example, Phase II of one of the most important human genotyping project, HapMap, has produced genotypes where consecutive loci (SNPs) are approximately

separated by 1 kilobase [107]. On such data, the main assumption of Lander-Green-based methods is not met. Indeed, such methods require that loci are in *Linkage Equilibrium (LE)*, or, in other words, that the frequencies of population haplotypes are the product of the frequencies of the alleles that compose them. SNP haplotypes (i.e. haplotypes whose loci are SNPs) are composed by consecutive blocks of alleles with high *Linkage Disequilibrium (LD)* inside the block and with Linkage Equilibrium between alleles of different blocks. The difference between the assumption and the characteristics of SNP genotype data can lead to misleading results as reported in [99]. Abecasis and Wigginton [2] have proposed an additional improvement of Merlin that admits LD among loci. Basically, alleles with high LD are considered as a single block-locus and the original algorithm is applied on the block-loci instead of the single loci. Haplotype reconstruction within block-loci is then performed separately.

To cope with large pedigrees, sampling-based or hybrid approaches have also been proposed (such as SimWalk2 [101] and PhyloPed [67]).

For sake of completeness, we want to remark that some methods for unstructured population presented in the previous section such as PHASE [104] and Halperin and Eskin method [59], have been also adapted to handle populations composed by unrelated trios.

3.3.3 Pedigree-based Combinatorial Methods

The advance of high-density SNP genotyping technologies had highlighted the limits of pedigree-based statistical approaches. Indeed, as previously noted, the two most basic algorithms, namely Lander-Green and Elson-Stewart, cannot directly handle an inherent characteristic of SNP genotypes: Linkage Disequilibrium among loci. Moreover, since genotyping costs are getting lower and lower, large-scale (on both population size and genotype length) studies are becoming common. For example, Kong *et al.* [68] conducted a study on approximately 30,000 individuals and 300,000 loci. Unfortunately the computational requirements of statistical-based methods on such large instances are prohibitive and even a sampling-based approach such as SimWalk2 does not appear suitable.

Recombination events inside blocks with high LD are much less frequent than between blocks [88]. Therefore, if we consider SNP genotypes that span a limited number of blocks with high LD, we expect that the real haplotype configuration induces a small number of recombination events in the whole pedigree. This observation leads to a natural combinatorial formulation of the HI problem on pedigrees: the *Minimum-Recombinant Haplotype Configuration* problem.

Problem 2. Minimum-Recombinant Haplotype Configuration (MinRHC).
Input: A genotyped pedigree P.
Output: A haplotype configuration for P (if it exists) such that the number of induced recombinations is minimum.

If genotypes are composed by allele pairs of tightly linked loci, we can assume that no recombinations are present, thus an interesting MinRHC problem restriction arises: the *Zero-Recombinant Haplotype Configuration* problem.

Problem 3. Zero-Recombinant Haplotype Configuration (ZRHC).
Input: A genotyped pedigree P.
Output: A haplotype configuration for P (if it exists) such that no recombinations are present.

The Zero-Recombinant Haplotype Configuration problem was first formulated by Wijsman [112] and solved by a 20-rule approach. O'Connell [91] proposed an hybrid combinatorial-statistical two-step approach: a first "recoding" step and, then, a EM step. Since recombination events are not admitted, each haplotype of an individual can be considered as a complex single allele. In the first step multi-locus genotypes are replaced with a multi-allelic single-locus genotypes. After that, consistent haplotype configurations are enumerated and haplotype frequencies are determined via a EM algorithm. A third approach [123] combines and extends the ideas of the previous two methods and achieves a better efficiency. However, such approaches may require exponential-time in the worst-case.

A first polynomial-time solution for ZRHC has been proposed by Li and Jiang [76, 77]. Their work is based on the formulation of the problem as a system of linear equations on the field $\mathbb{Z}_2$, and all solutions (i.e. all consistent haplotype configurations) can be efficiently computed via the Gauss elimination algorithm in $O(m^3n^3)$ time, where m and n are the number of loci and the size of the pedigree, respectively. An improvement of this algorithm has been proposed by Xiao *et al.* [114, 115]. They exploit the sparsity of the system of linear equations (each equation involves, at most, 3 variables) and the structure of the simple pedigree graph, to reduce the system to an equivalent set of $O(mn)$ equations over $O(n)$ variables, which are solved in $O(mn^3)$ time via Gaussian elimination. A further improvement employs low-stretch spanning trees [37] on subgraphs of the simple pedigree graph in order to achieve a final $O(mn^2 + n^3 \log^2 n \log\log n)$ running time on general pedigrees.

Human pedigrees of moderate size are often *tree pedigrees* (i.e. they do not contain mating loops). Therefore, a second kind of restriction has been considered: the ZRHC problem on tree pedigrees. We note in passing that the previous algorithm [114, 115] works in $O(mn^2+n^3)$-time on restricted-tree pedigrees. Previously, an approach by Chan *et al.* [26, 27] achieved a better time bound of $O(mn)$ for a

single haplotype configuration (if it exists), while Xiao's algorithm outputs all the solutions. Since Chan's algorithm cannot find all the consistent haplotype configurations, Liu and Jiang [85], and Li and Li [80] proposed a two algorithms to find all the solutions in $O(mn^2)$ and $O(\alpha(n,m)nm)$ time, respectively. All the previous approaches for ZRHC problem work assuming that the pedigree is a restricted-tree pedigree (Chan *et al.* claim in [27] that their algorithm works also on "general" tree pedigrees, but the correctness of this affirmation is debatable). Therefore, to the best of our knowledge, designing an optimal algorithm for ZRHC on tree pedigrees is still an open problem.

Real genotypes may have missing data, i.e. loci where the pair of alleles is not known. Despite ZRHC is polynomial-time solvable, the ZRHC problem with missing data is NP-hard [83, 84].

The Minimum-Recombinant Haplotype Configuration is a much more realistic formulation since the no-recombinant assumption only holds for genotypes over tightly linked loci. Unfortunately, Li and Jiang [76, 77], and Liu *et al.* [83, 84] have shown that the computational problem is also much harder that ZRHC. In fact, they prove that (*i*) MinRHC is NP-hard and (*ii*) MinRHC cannot be approximated within any constant under the Unique Games Conjecture [65]. These results hold also on very simple genotyped pedigrees, namely *binary-tree pedigrees* (i.e. pedigrees where each individual has at most one mate and one offspring) or *2-locus pedigrees* (i.e. pedigrees where genotypes are defined over two loci).

To cope the computational intractability of MinRHC, several algorithms have been proposed. Qian and Beckmann [31] designed an exponential-time heuristic that computes a haplotype configuration minimizing the number of recombinants in each nuclear family. Li and Jiang [76, 77] proposed an efficient (polynomial-time) heuristic that reconstructs long parts of haplotypes (called blocks) without inducing recombinations and then fills the remaining gaps between blocks. It has been reported as much more efficient than the previous method but slightly less accurate [76]. In a subsequent work, Li and Jiang [78] formulated the MinRHC problem as an Integer Linear Program. This ILP formulation can also accommodate missing data, it can take into account different likelihoods of recombinations between different pairs of loci, and it can enumerate all the optimal solutions (i.e. those with the fewest number of recombinants). Besides the ILP formulation, other exact algorithms include two dynamic-programming approaches [35]: one is exponential in the number of loci (and works only for tree pedigree) while the second one is exponential in the size of pedigree (and works also for general pedigrees). The ZRHC algorithm described in [76] has been extended to deal with recombinations in two other directions: bounding the maximum number of recombination events and finding a partition of the genotype loci in maximal non-recombinants regions. In the first case, if the number of recombination events is bounded by a small constant k, there exists an algorithm [116] that finds a solution in $O(mn \log^{k+1} n)$ time with

high probability, $1 - O\left(k^2 \frac{\log^2 n}{mn} + \frac{1}{n^2}\right)$. In the second case, a $O(n^3 m^3)$ algorithm that partitions the genotype in the minimum number of non-recombinant regions has been proposed in [22].

Mutations are widely believed to be less frequent than recombinations. However, as the pedigrees become larger and genotypes become denser, the incidence of mutation events could become noticeable. Wang and Jiang [110] have recently proposed a natural formulation of the HI problem that considers this kind of variation events as a minimization problem: finding the haplotype configuration consistent with a genotyped pedigree that requires the minimum number of mutations (Minimum-Mutations Haplotype Configuration, MinMHC). In this work, they propose a solution based on a iterative refinement of an ILP program until a consistent haplotype configuration is found. In particular, since the computational complexity of a ILP program depends on the number of constraints, the algorithm starts with a reduced set of constraints, and then new consistency constraints are added until the ILP solver is able to find a haplotype configuration consistent with the input pedigree.

4 Pure Parsimony Xor Haplotyping

Xor-genotypes have been recently introduced as a new (and low-cost) representation of genetic data for Haplotype Inference [8, 9]. In their work, Barzuza *et al.* assume the perfect phylogeny model on populations of unrelated individuals. To the best of our knowledge, the computational problem arising from the other combinatorial principle for HI on unstructured populations, *pure parsimony*, has never been directly studied on this kind of genetic representation. In this chapter we want to "fill the gap" and study this new computational problem of HI.

The chapter is structured as follows. Section 4.1 formulates the computational problem and formalizes the principal concepts. Section 4.2 introduces the basic combinatorial properties of two equivalent representations of the solutions. Based on such properties, two polynomial-time algorithms for restricted cases and a fixed-parameter algorithm for the general case are illustrated in Section 4.3 and Section 4.4, respectively. Furthermore, in Section 4.5 we show that the problem has a polynomial-time l-approximation, where l is the maximum number of heterozygous loci in a xor-genotype. Finally, in Section 4.6 we propose a heuristic and we perform an experimental analysis showing that it scales to real-world large instances taken from the HapMap project.

4.1 The Computational Problem

In this section, we formulate the computational problem of Haplotype Inference on xor-genotypes for unrelated individuals assuming the pure parsimony criterion. The richest (and fast-growing) source of genotype variation data is represented by SNPs, thus we assume that the set of genotype loci are all bi-allelic.

Since xor-genotypes are basically different from "regular" genotypes, some concepts that have been informally presented in Section 3.1 assume a slightly different meaning in the context of xor-genotypes. Therefore, before formulating the computational problem, we will formally define the concepts we need.

Let Σ be a set of m loci (also called characters or sites). Since xor-genotypes distinguish heterozygous loci from homozygous loci without specifying the homozygous allele, a *xor-genotype* (or simply a *genotype*) x_i is the subset of Σ composed by loci that are heterozygous in individual i. We explicitly forbid the presence of empty xor-genotypes (i.e. individuals that are homozygous at all loci). Since the allele at homozygous loci is not specified by xor-genotypes, such individuals

are completely uninformative: every pair of identical haplotypes would resolve the genotype. We represent a *haplotype* as a (possibly empty) subset of Σ composed by loci that have the minor allele. Given two distinct haplotypes h_1, h_2, then the pair (h_1, h_2) *resolves* the xor-genotype x iff $x = h_1 \oplus h_2$, where $\oplus$ is defined as the classical symmetric difference of h_1 and h_2, i.e. the set of characters that are present in exactly one of h_1 and h_2. (The symmetric difference operator on sets is closely related to the *exclusive disjunction (xor)* in logic, hence the name of xor-genotypes.) A set H of haplotypes resolves a set X of xor-genotypes if for each genotype $x \in X$, there exists a pair of haplotypes in H that resolves x.

We are now able to formally introduce the problem that has been studied in this chapter.

Problem 4. Pure Parsimony Xor Haplotyping (PPXH).
Input: A set X of xor-genotypes of unrelated individuals over the set of loci Σ.
Output: A minimum-cardinality set H of haplotypes that resolves X.

As presented in Section 3.2.2, xor-genotypes have been studied under the perfect phylogeny model [8, 9], while the HI problem under the pure parsimony principle has been deeply investigated only on "regular" genotypes. To the best of our knowledge, the PPXH problem has only been mentioned in a work by Brown and Harrower [20], which provides an ILP formulation for the problem. This formulation, however, seems impractical because the linear-relaxation of the program (i.e. the elimination of the integrality constraints) has a trivial solution (of cardinality 1) equally distant from every other feasible solution. As a consequence, the resolution process essentially reduces to an enumeration of feasible solutions. In fact, as explained in Section 4.6.1, using this formulation we were not able to solve the problem even on the smallest instances that we considered in our experimentation.

The lack of efficient algorithms for PPXH motivates our work. We investigate the PPXH problem mainly by devising exact solutions of the problem by either considering fixed-parameter tractability or polynomial time algorithms for some restricted instances of the problem. We introduce a new graph representation of xor-genotypes and haplotypes, called *xor-graph*, that is crucial in the study of the PPXH problem. Indeed most of the results that we will present rely on combinatorial properties of xor-graphs.

Initially we will show that the PPXH problem is equivalent to the problem of building a xor-graph with the fewest possible vertices. We then design two polynomial time algorithms for restricted instances of the PPXH problem. Subsequently we describe a fixed-parameter algorithm of $O(mn + 2^{k^2}km)$ time complexity, for k the size of the optimum solution. Moreover we provide an l-approximation algorithm, where l is the maximum number of occurrences of a character in the set of input genotypes. Since the previous results are mainly of theoretical interest, we finally propose an efficient and effective heuristic for the general problem and an

experimental analysis on real and artificial datasets. The main goal of our experimental analysis is to show that our heuristic is effective on a large class of instances of various sizes where other methods, such as the ILP formulation proposed by Brown and Harrower [20] or the adaption of the ILP formulation of Gusfield [55] to the PPXH problem, are not applicable.

4.2 Basic Properties

A fundamental idea used in this work is a graph representation of a feasible solution. More precisely, given a set X of xor-genotypes, the *xor-graph* associated with a set H of haplotypes resolving X is the graph $\mathcal{G} = (H, E)$ where edges of $\mathcal{G}$ are labeled by a bijective function $\lambda : E \to X$ such that, for each edge $e = (h_i, h_j)$, $\lambda(e) = h_i \oplus h_j$. The labeling λ is generalized to a set S by defining $\lambda(S) = \{\lambda(s) \mid s \in S\}$. We call *optimal xor-graph for* X, a xor-graph associated with an optimal solution for X (that is a xor-graph with the minimum number of vertices).

In this section we state some basic combinatorial properties of xor-graphs that will be used to prove the other main results. Among all possible haplotypes, we identify a distinguished haplotype, called *null* haplotype and denoted by h_0, which corresponds to the empty set. Since the operation $\oplus$ is associative and commutative, by a slight abuse of language, given a family $F = \{s_1, \ldots, s_n\}$ of subsets of a set Σ we denote by $\oplus(F)$ the expression $s_1 \oplus s_2 \oplus \cdots \oplus s_n$.

The cycles of a xor-graph satisfy the following property.

Lemma 4.1. *Let X be a set of xor-genotypes, let $\mathcal{G}$ be a xor-graph associated with a set of haplotypes resolving X, and let C be the edge set of a cycle of $\mathcal{G}$. Then $\oplus(\lambda(C))$ is equal to the empty set.*

Proof. By definition of cycle, C consists of a set $\{(h_1, h_2), (h_2, h_3), \ldots, (h_n, h_{n+1})\}$, with $h_1 = h_{n+1}$. By definition of xor-graph, $\oplus(\lambda(C)) = \oplus_{i=1}^{n} (h_i \oplus h_{i+1})$. By the associativity and commutativity of $\oplus$, $\oplus(\lambda(C)) = (h_1 \oplus h_{n+1}) \oplus (h_2 \oplus h_2) \oplus \ldots \oplus (h_n \oplus h_n)$. Since $h_1 = h_{n+1}$ and $h \oplus h = \varnothing$ for all h, we obtain $\oplus(\lambda(C)) = \varnothing$. □

The above property of cycles of a xor-graph is sufficient to construct a set of haplotypes resolving a set of genotypes from a xor-graph. Let X be an instance of PPXH and let $\mathcal{G} = (V, E)$ be a graph whose edges are biunivocally labeled by a function $\lambda : E \to X$ such that $\oplus(\lambda(C)) = \varnothing$ for each cycle C of the graph. Then it is immediate to compute a feasible solution H from $\mathcal{G}$ where $|H| \leq |V|$. More precisely, we associate a haplotype with each vertex of $\mathcal{G}$ as follows. Associate the null haplotype h_0 with any vertex in each connected component of $\mathcal{G}$. Perform a depth-first visit of each connected component of $\mathcal{G}$, starting from the vertex associated with h_0. When visiting a new vertex v of $\mathcal{G}$ there must exist an edge

$e = (v, w)$ so that the haplotype w_h has been previously assigned to w. Then associate the haplotype $w_h \oplus \lambda(e)$ with v.

It is not hard to verify that our construction guarantees that H is actually a feasible solution of X, that is for each edge $e = (v, w)$ of $\mathcal{G}$, $v_h \oplus w_h = \lambda(e)$, where v_h and w_h are respectively the haplotypes associated with v and w. It is trivial to notice that the property holds for all edges that are part of the spanning forest computed by the depth-first search. Therefore we can restrict our attention to edges $e = (v, w)$ that are not in such spanning forest. Since v and w are in the same connected component of $\mathcal{G}$ the spanning tree T of the connected component contains both v and w. Let x be the least common ancestor of v and w in T. By construction the two paths of T, both starting from x and ending one in v and the other in w are edge disjoint. Let us denote by P_v and P_w respectively the edges of the paths ending in v and w, and let x_h be the haplotype associated with x. Now we want to prove that $v_h \oplus w_h = \lambda(e)$. It is immediate to verify that $v_h = \bigoplus(\lambda(P_v)) \oplus x_h$ and $w_h = \bigoplus(\lambda(P_w)) \oplus x_h$. Since the edges in $P_v \cup P_w \cup \{e\}$ form a simple cycle of $\mathcal{G}$, by Lemma 4.1 we can conclude $\lambda(e) = \bigoplus(\lambda(P_v)) \oplus \bigoplus(\lambda(P_w))$, completing the proof.

The following results justify our attention to connected xor-graphs and their cuts.

Lemma 4.2. *Let X be a set of xor-genotypes and let $\mathcal{G}$ be a xor-graph associated with a set H of haplotypes resolving X. Let α be any character of Σ. Then the set A of edges of $\mathcal{G}$ whose label contains α is a cut of $\mathcal{G}$.*

Proof. Let H_α be the subset of H containing the character α, and let $\bar{H}_\alpha = H \setminus H_\alpha$. Let E' be the edges of $\mathcal{G}$ with an endpoint in H_α and one in $\bar{H}_\alpha$ (clearly E' is a cut of $\mathcal{G}$.) Notice that E' is exactly the set of edges connecting a haplotype containing α and a haplotype not containing α, therefore $E' = A$. □

Lemma 4.3. *Let X be a set of xor-genotypes, and let $\mathcal{G} = (H, E)$ be a disconnected xor-graph for X. Then $\mathcal{G}$ is not an optimal xor-graph of X*

Proof. Since $\mathcal{G}$ has at least two connected components C_1 and C_2, we denote with a_1, a_2 two vertices from C_1 and C_2 respectively. Construct the set H' from H by replacing each haplotype $h \in C_1$ by $h \oplus a_1$ and each haplotype $h \in C_2$ by $h \oplus a_2$. Since C_1 and C_2 are not connected, the set of genotypes resolved by H' is equal to that of H.

But both a_1 and a_2 are replaced by the null haplotype in H', therefore $|H'|$ is strictly smaller than $|H|$. □

Instances and solutions of the PPXH problem can be represented by binary matrices. More precisely, we can have a *genotype matrix* associated with a set of xor-genotypes and a *haplotype matrix* associated with a set of haplotypes. In both matrices each column is uniquely identified by a character in Σ, while the rows

	a	b	c	d	e
g_1	1	1	0	0	0
g_2	1	1	1	0	0
g_3	0	1	1	0	0
g_4	0	0	1	1	1
g_5	1	0	0	0	0
g_6	0	0	0	0	1
g_7	1	0	1	0	1

	a	b	c	d	e
h_1	0	0	0	0	0
h_2	0	0	1	1	0
h_3	1	1	1	1	0
h_4	1	0	1	0	0
h_5	1	0	0	1	0
h_6	0	0	0	1	0
h_7	0	0	0	0	1

Figure 4.1: Example of genotype (left) and haplotype (right) matrices.

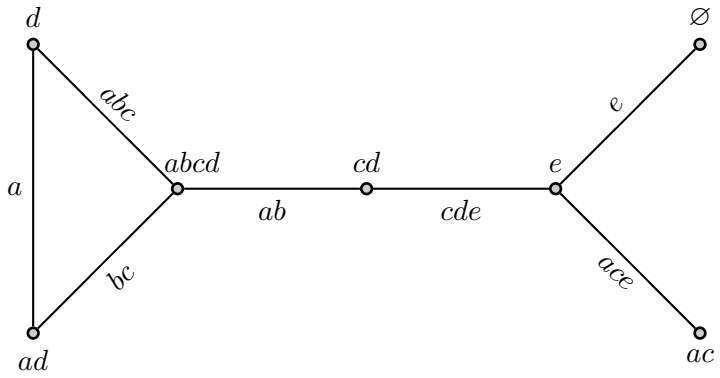


Figure 4.2: Xor-graph representing the set of haplotypes in Figure 4.1.

of a genotype matrix (respectively haplotype matrix) correspond to the genotypes (resp. haplotypes).

For example let Σ be the set $\{a, b, c, d, e\}$ and let X be the set of xor-genotypes $\{\{a, b\}, \{a, b, c\}, \{b, c\}, \{c, d, e\}, \{a\}, \{e\}, \{a, c, e\}\}$. A possible, albeit suboptimal, set of haplotypes resolving X is $\{\varnothing, \{c, d\}, \{a, b, c, d\}, \{a, c\}, \{a, d\}, \{d\}, \{e\}\}$. The matricial representation of both sets is in Figure 4.1, while the associated xor-graph is represented in Figure 4.2.

Given an ordering of the character set (that is $\Sigma = \langle \sigma_1, \ldots, \sigma_{|\Sigma|} \rangle$), the entry in the i-th row and j-th column of a genotype matrix (respectively, haplotype matrix) is 1 if σ_j belongs to the i-th genotype (respectively, i-th haplotype) and is equal to 0 otherwise. In the following we identify rows of a genotype (or haplotype) matrix with the corresponding genotypes (or haplotypes). Given a matrix M, we denote by $M[\cdot, A]$ (by $M[B, \cdot]$, respectively) the submatrix of M induced by the set A of columns (by the set B of rows, respectively).

Given a genotype or haplotype matrix M over Σ, we will say that a subset Σ_1 of Σ is a *linearly dependent* set of characters (or, simply, a *dependent* set of characters) in matrix M if there exists a non-empty subset Σ_2 of Σ_1 such that, for each row i, $\oplus_{\sigma\in\Sigma_2} M[i,\sigma] = 0$. Otherwise it is called *linearly independent* (or simply *independent*).

The rationale for introducing the notion of linearly independent characters is computational efficiency; in fact we have a very efficient algorithm for selecting a maximal subset of linearly independent characters, and any solution on such subset actually allows us to solve the original instance, as stated in the following lemma.

Lemma 4.4. *Let X be a xor-genotype matrix and H be a haplotype matrix over the same character set Σ. Let Σ_1 be a maximal independent subset of Σ in X. Then, H resolves X if and only if $H[\cdot,\Sigma_1]$ resolves $X[\cdot,\Sigma_1]$.*

Proof. The *only-if* part is obviously true because $H[\cdot,\Sigma_1]$ and $X[\cdot,\Sigma_1]$ are two submatrices of H and X respectively. The *if* part can be proved by constructing a feasible solution H for X from the smaller solution $H[\cdot,\Sigma_1]$ for $X[\cdot,\Sigma_1]$ (for simplicity we will refer to the two submatrices respectively as H' and X'). For each character $\alpha \in \Sigma \setminus \Sigma_1$, since $\Sigma_1 \cup \{\alpha\}$ is dependent, there exists a non-empty subset Σ_α of Σ_1 such that, for each genotype x, $X[x,\alpha] = \oplus_{\sigma\in\Sigma_\alpha} X'[x,\sigma]$. Set the entry $H[x,\alpha]$ to $\oplus_{\sigma\in\Sigma_\alpha} H'[x,\sigma]$.

We claim that H resolves X. Since H' resolves X', it suffices to prove that for each character $\alpha \in \Sigma\setminus\Sigma_1$, $H[h_1,\alpha]\oplus H[h_2,\alpha] = X[x,\alpha]$, for some pair of haplotypes h_1, h_2. We already know that for each genotype x' of X', there is a pair (h'_1, h'_2) of haplotypes in H' that resolves x'. Notice that $X[x,\alpha] = \oplus_{\sigma\in\Sigma_\alpha} X'[x,\sigma]$ since Σ_1 is a maximal subset of independent characters of X. Since H' resolves X', $\oplus_{\sigma\in\Sigma_\alpha} X'[x,\sigma] = \oplus_{\sigma\in\Sigma_\alpha}(H'[h_1,\sigma] \oplus H'[h_2,\sigma])$. Moreover, by the associativity of $\oplus$, $\oplus_{\sigma\in\Sigma_\alpha}(H'[h_1,\sigma] \oplus H'[h_2,\sigma]) = (\oplus_{\sigma\in\Sigma_\alpha} H'[h_1,\sigma]) \oplus (\oplus_{\sigma\in\Sigma_\alpha} H'[h_2,\sigma])$. Finally, by our construction of the columns of H corresponding to characters in $\Sigma \setminus \Sigma_1$, $(\oplus_{\sigma\in\Sigma_\alpha} H'[h_1,\sigma]) \oplus (\oplus_{\sigma\in\Sigma_\alpha} H'[h_2,\sigma]) = H[h_1,\alpha] \oplus H[h_2,\alpha]$, hence completing the proof. □

Notice that, given a $n \times m$ xor-genotype matrix X, a maximal subset of independent characters in X can be extracted by applying the Gauss elimination algorithm on X in $O(\min(n,m)nm)$ time. Observe that the proof of Lemma 4.4 is constructive and shows how to compute efficiently a solution H for X from a solution $H[\cdot,\Sigma_1]$ for $X[\cdot,\Sigma_1]$.

We can introduce another simplification of the instance which can be performed efficiently. It affects the construction of the xor-graph and allows an efficient reconstruction of an optimal xor-graph for the general instance, given a xor-graph for the reduced or simplified instance.

Lemma 4.5. *Let X be an instance of PPXH, and let α be a character of X such that there exists exactly one genotype $x \in X$ with $\alpha \in x$. Then there exists an optimal xor-graph $\mathcal{G}$ for X such that there is a vertex v of $\mathcal{G}$ with exactly one edge e incident on v and $\lambda(e) = x$.*

Proof. Let $\mathcal{G}$ be an optimal xor-graph $\mathcal{G}$ for X. Since α appears in only one genotype in X, there is exactly one edge e of $\mathcal{G}$ such that $\alpha \in \lambda(e)$. By Lemma 4.2 removing e from $\mathcal{G}$ results in a bipartition $\{H_\alpha, \bar{H_\alpha}\}$ where H_α consists of the haplotypes containing α. Let $v \in H_\alpha$ and $w \in \bar{H_\alpha}$ be the two endpoints of e, and let D be the set of vertices of H_α adjacent to v. Change each haplotype in $h \in H_\alpha \setminus \{v\}$ to $h \oplus v \oplus w$, obtaining a new xor-graph $\mathcal{G}_1$.

By construction, $\mathcal{G}_1$ has set of edges $E_1 = E \setminus \{(v, d) \mid d \in D\} \cup \{(w, d) \mid d \in D\}$. Indeed, let $e = (v, d)$ be any edge of $\mathcal{G}$ connecting v with a vertex $d \in D$; in $\mathcal{G}_1$ there is an edge $f = (w, d)$ such that $\lambda(e) = \lambda(f)$. It is immediate to notice that $\mathcal{G}$ and $\mathcal{G}_1$ have the same number of vertices, therefore $\mathcal{G}_1$ is optimal and satisfies the statement of the lemma. □

Also the proof of Lemma 4.5 is constructive and can be exploited directly in an algorithm to simplify the instance of the problem. More precisely, the removal of a genotype and a character as stated by Lemma 4.5 can be repeated until all characters appear in at least two genotypes (or we obtain the special case of an instance containing only one genotype; in such case the optimal solution is trivially made by two haplotypes). Following the same idea, if the set of characters is linearly dependent, we can extract a maximal subset of linearly independent characters. Moreover the executions of the two reductions can be intertwined until none of those reductions can be performed.

An instance X of PPXH is called *reduced* if (i) X consists of only one genotype, or the two following conditions are satisfied: $(ii\ a)$ the set of characters of X are independent and $(ii\ b)$ each character appears in at least two genotypes. Lemma 4.4 and Lemma 4.5 justify the fact that we will assume in the rest of the chapter that all instances are reduced, as the reduction process can be performed efficiently, and we can easily compute a solution of the original instance given a solution of a reduced instance (see Algorithm 1 for a formal description of the reduction process).

The reduction process leads us to an important lower bound on the size of the optimum.

Lemma 4.6. *Let X be a reduced genotype matrix having n rows and m columns. Then any haplotype matrix H resolving X has at least $m + 1$ rows.*

Proof. Let $\mathcal{G}$ be a xor-graph for X. By Lemma 4.2, each character α induces a cut in graph $\mathcal{G}$. Each cut can be represented as n-bit binary vector c_α in which each element $c_\alpha[i]$ is equal to 1 if and only if the genotype x_i belongs to the cut.

Algorithm 1: The reduction step

Data: a xor-genotype matrix X
Result: a reduced xor-genotype matrix associated with the input matrix X

```
repeat
    C ← subset of linearly independent columns of X obtained by the Gauss
    elimination algorithm;
    D ← X \ C;
    A ← set of symbols appearing in exactly one genotype in X;
    Remove from X all columns in D and all rows in A;
until D and A are both empty ;
return X;
```

Clearly, such vector is precisely the column vector corresponding to character α of matrix X. Thus, since the characters are independent, also the family of the cuts (represented as binary vectors) induced by the set of characters is linearly independent. By Theorem 1.9.6 of [33] all connected graphs with m independent cuts have at least $m + 1$ vertices. □

As a consequence of Lemma 4.6, in a *reduced* xor-genotype matrix, the number of rows is greater than or equal to the number of columns. In fact, in any matrix, the number of linearly independent columns is equal to the number of linearly independent rows and, clearly, is bounded by the minimum between the number of columns and the number of rows.

The process of reducing a xor-genotype matrix by restricting ourselves to a maximal subset of independent characters is an application of the *kernelization* technique for designing a fixed-parameter algorithm [36]. The technique consists of reducing the original instance to a new instance whose size depends only on the parameter (in our case the size of the optimal solution). The size of the reduced xor-genotype matrix is clearly bounded by a polynomial function of the optimum k since at most $O(k^2)$ distinct genotypes can be generated by k distinct haplotypes and, by the previous consideration, the number of columns is less than the number of rows. As a result, the number of entries of a reduced xor-genotype matrix is bounded by $O(k^4)$.

4.3 Algorithms for Restricted Instances

In this section we investigate two restrictions of the PPXH problem obtained by bounding the number of characters that can appear in each genotype and the num-

ber of genotypes where a character can occur. Those restrictions are summarized by the following formulation.

Problem 5. Constrained Pure Parsimony Xor Haplotyping (PPXH(p, q)). The instance consists of a set X of xor-genotypes, where each xor-genotype $x \in X$ contains at most p characters, and each character appears in at most q xor-genotypes. The goal is to compute a minimum cardinality set H of haplotypes that resolves X. We use the symbol ∞ when one of parameters p or q is unbounded.

More precisely we will present efficient algorithms for the case when each character is contained in at most two xor-genotypes (PPXH$(\infty, 2)$) and the case that each genotype consists of at most two characters (PPXH$(2, \infty)$).

4.3.1 A Polynomial Time Algorithm for PPXH$(\infty, 2)$

The structure of the cycles in a xor-graph characterizes the solutions for the PPXH$(\infty, 2)$ problem as stated in the following Lemma.

Lemma 4.7. *Let X be a reduced instance of PPXH$(\infty, 2)$, let $\mathcal{G}$ be an optimal xor-graph for X, and let e be an edge of $\mathcal{G}$. Then e belongs to exactly one simple cycle of $\mathcal{G}$.*

Proof. Assume to the contrary that an edge e belongs to two cycles C_1 and C_2. Notice that the three sets $C_1 \setminus C_2$, $C_2 \setminus C_1$, $C_1 \cap C_2$ are pairwise disjoint and not empty. Let d be any element of $\lambda(C_1 \cap C_2)$. By Lemma 4.1, $\oplus(\lambda(C_1 \setminus C_2)) = \oplus(\lambda(C_2 \setminus C_1)) = \oplus(\lambda(C_1 \cap C_2))$. Consequently there exist three distinct edges $e_1 \in C_1 \setminus C_2$, $e_2 \in C_2 \setminus C_1$, $e_3 \in C_1 \cap C_2$ such that $\lambda(e_1)$, $\lambda(e_2)$, $\lambda(e_3)$ all contain d, which contradicts the fact that there are only two genotypes containing d. By the first part of the proof, we have now to prove that e belongs to at least a cycle of $\mathcal{G}$.

Assume to the contrary that X is a smallest counterexample, that is no such optimal xor-graph exists for X, while such a graph exists for all reduced instances with fewer genotypes, and let $\mathcal{G}$ be any optimal xor-graph for X. Since there is an edge that does not belong to any cycle of $\mathcal{G}$, there is a character a such that both edges e_1 and e_2 containing a do not belong to any cycle. Notice that two such edges must exist, since the instance is reduced. Let us denote by α, β and γ respectively the sets $\lambda(e_1) \cap \lambda(e_2)$, $\lambda(e_1) \setminus \lambda(e_2)$, $\lambda(e_2) \setminus \lambda(e_1)$.

Compute a new reduced instance X_1 from X by removing the xor-genotypes e_1 and e_2, and adding a new genotype $x_c = \beta \cup \gamma = e_1 \oplus e_2$. Clearly X_1 is a reduced instance of PPXH$(\infty, 2)$ smaller than X, therefore X_1 admits an optimal graph $\mathcal{G}_1$ where all edges are in some cycle. Let us consider the unique cycle C of $\mathcal{G}_1$ containing the edge $e = (u, v)$, with $\lambda(e) = x_c$. Now, compute a xor-graph $\mathcal{G}_2$ for instance X, by replacing in $\mathcal{G}_1$ the edge e with a path made of two edges $e'_1 = (u, w)$, $e'_2 = (w, v)$, where w is a new vertex, $\lambda(e'_1) = e_1$, and $\lambda(e'_2) = e_2$. The graph $\mathcal{G}_2$ is a

xor-graph of X as $x_c = e_1 \oplus e_2$. Clearly the newly obtained graph $\mathcal{G}_2$ is a xor-graph for X satisfying the requirements of the lemma and $\mathcal{G}_2$ contains one more vertex than $\mathcal{G}_1$.

We have to prove that $\mathcal{G}_2$ is optimal, therefore assume that $\mathcal{G}_2$ is not optimal and let $\mathcal{G}_*$ be an optimal xor-graph for X, that is $\mathcal{G}_*$ has no more vertices than $\mathcal{G}_1$. It is immediate to notice that contracting each of e_1 and e_2 into single vertices result in a xor-graph that is a solution of X_1 with fewer vertices than $\mathcal{G}_1$, hence violating the optimality of $\mathcal{G}_1$. □

Since the optimal xor-graph is connected (Lemma 4.3) and consists of a set of edge-disjoint cycles (Lemma 4.7), the size of the optimum solution is equal to $|X| + 1 - |\mathcal{C}|$, for $|X|$ the number of genotypes or edges of the graph and $|\mathcal{C}|$ the number of simple cycles of the graph, since any set of $|\mathcal{C}|$ simple cycles on a graph with at most $|X| - |\mathcal{C}|$ must share at least an edge.

Algorithm 2 solves the PPXH$(\infty, 2)$ problem by computing the set $\mathcal{C}$ of all simple cycles of an optimal xor-graph. In fact Lemma 4.7 allows us to introduce a binary relation R between genotypes, where two genotypes are related if and only if they share a common character. By Lemma 4.7 any two genotypes (or edges of the xor-graph) that are related must also belong to the same simple cycle. It is immediate to notice that the partition of the edges of the xor-graph into simple cycles is equal to the most refined partition of edges such that any two edges sharing a common character belong to the same set of such partition. In fact Algorithm 2 computes exactly the closure of R.

4.3.2 A Polynomial Time Algorithm for PPXH$(2, \infty)$

For simplicity's sake we will assume that the instance of the problem is a genotype matrix X and the desired output is a haplotype matrix H.

Since in both matrices the columns are indexed by characters, we will denote by $X[\cdot, \sigma]$ (respectively $H[\cdot, \sigma]$) the column of X (resp. H) indexed by the character σ. The algorithm is based on Lemma 4.4 and Lemma 4.6.

In fact we will first compute a largest set Σ_1 of independent characters in X. Moreover for each character $\alpha \in \Sigma \setminus \Sigma_1$ we determine the subset Σ_α of Σ_1 such that $X[\cdot, \alpha] = \oplus_{\sigma \in \Sigma_\alpha} X[\cdot, \sigma]$. Notice that this step can be carried out by a simple application of the Gauss elimination algorithm.

Let X' be the submatrix $X[\cdot, \Sigma_1]$. An optimal solution of the instance X' is a matrix H' containing $|\Sigma_1| + 1$ rows. More precisely the i-th row of H', for $1 \leq i \leq |\Sigma_1|$, consists of all zeroes, except for the i-th column (where it contains 1). The last row contains only zeroes.

Clearly H' resolves X'. In fact it is immediate to notice that each row of X' contains at most two 1s, as the same property holds for X, therefore for each row

Algorithm 2: Algorithm for PPXH$(\infty, 2)$

Data: a set X of reduced xor-genotypes
Result: an optimal solution H for X

```
1  C ← ∅;
2  while X ≠ ∅ do
3      x ← any element of X;
4      C ← {x};
5      repeat
6          D ← the set of genotypes in X \ C sharing at least one element with
             some genotypes in C;
7          C ← C ∪ D;
8      until D = ∅ ;
9      Add C to C;
10     Remove all genotypes in C from X;
11 end
12 foreach C ∈ C do
13     Transform C into a cycle;
14 end
15 Build a graph H from C, so that all cycles in C share a common vertex;
16 return H
```

r of X' there are two rows of H' resolving r. The optimality of such solution is a direct consequence of Lemma 4.6. Clearly H' is not a feasible solution of the original instance X, but such a feasible solution H can be easily computed from H' by adding, for each character $\alpha \in \Sigma \setminus \Sigma_1$, a column equal to $\oplus_{\sigma \in \Sigma_\alpha} H'[\cdot, \sigma]$ (where Σ_α satisfies $X[x, \alpha] = \oplus_{\sigma \in \Sigma_\alpha} X'[x, \sigma]$ for each genotype x). The matrix H is a feasible solution as shown in the proof of Lemma 4.4.

4.4 Fixed-Parameter Tractability of PPXH

As observed at the end of Section 4.2, the reduction of an instance of the unrestricted PPXH problem lead us to a fixed-parameter algorithm (where the parameter is the optimum). Moreover we can observe that there exists another fixed-parameter algorithm for the unrestricted PPXH problem without using the reduction of input instance. Let H be a set of haplotypes and let X be a set of genotypes resolved by H. (In the following we will denote $|X|$ by n.) Since H can resolve at most $\binom{|H|}{2}$ genotypes, $n \geq |H| \geq \sqrt{2n}$. In other words, if k is the size of the minimum-cardinality set of haplotypes resolving X, $n \in O(k^2)$.

The number of the possible graphs with at most $n+1$ vertices and exactly n edges is no more than $2^{2n\log_2(n+1)} = (n+1)^{2n}$ which, by our previous observation, is $O(k^{4k^2})$, i.e. a function dependent only on k. The time needed to check if one of such graphs is a xor-graph for H is clearly polynomial in n and thus we can immediately derive a fixed-parameter algorithm to find an optimal xor-graph for X.

The time complexity of the algorithm is well beyond what is deemed acceptable in practice, therefore we propose a more efficient algorithm that is based on the matrix representation of genotypes and haplotypes.

In the following we will assume that the genotype matrix X is reduced, and that X has n rows and m independent columns, and that we are looking for a haplotype matrix H with at most k distinct rows that resolves X. The basic idea of our algorithm is to enumerate all possible haplotype matrices.

In the naïve approach, testing if a haplotype matrix resolves a given genotype matrix requires $O(k^2nm)$ time because each pair of haplotypes has to be considered and then each resulting genotype has to be searched in the genotype matrix. Our strategy, instead, is to enumerate all the haplotype matrices by changing only one haplotype each time, in such a way that only $k-1$ new pairs of haplotypes must be considered when testing if H resolves the set X.

We use Gray codes [98] to visit all the haplotype matrices in such a way that each pair of consecutive matrices differs by a single bit and, thus, by a single haplotype. More precisely, we enumerate all $k \times m$ matrices by generating all km-long bit vectors. Indeed, the bits from position $(i-1)m+1$ to position im in a km-long vector give the i-th row of the matrix (for $1 \leq i \leq k$). The fastest known algorithm for computing the next vector of a Gray code requires constant time for each invocation [10].

Observe that the naïve algorithm requires $O(nm)$ time to test if there is a genotype in matrix X resolved by a pair of haplotypes. By representing the set of the row vectors of matrix X as a *binary trie* [41], the time required to get the index of the row containing a m-long binary vector is reduced to $O(m)$.

The details of the fixed-parameter algorithm are given in Algorithm 3, where we also use some additional data structures: the array *ResolvedByHowMany* which associates with each genotype the number of pairs of haplotypes resolving such genotype, and *ListResolvedGenotypes* which associates with each haplotype h a list of the relevant pairs of haplotypes in which h is involved. In fact, the elements of the lists in *ListResolvedGenotypes* are triples (h_1, h_2, x) where (h_1, h_2) is a pair of haplotypes resolving x.

Notice that the outermost foreach loop (lines 7–26) iterates 2^{km} times (which, by Lemma 4.6, is at most $O(2^{k^2})$), while the for loop at lines 15–21 iterates k times. Each iteration of the latter loop consists of a lookup in a trie (which can be done in $O(m)$ time) and updating in constant time some arrays and lists. Since each list can

Algorithm 3: A fixed-parameter algorithm for PPXH

Data: A genotype matrix X defined over a set of m independent characters, and an integer k.
Result: A set H of at most k haplotypes resolving X if it exists, *No* otherwise.

if $\binom{k}{2} < n$ *or* $k < m$ **then return** *No*;
if $k > n$ **then return** $H \cup \{h_0\}$;
Build a trie T that stores the xor-genotypes contained in X;
Let *ListResolvedGenotypes* be an array of k initially empty lists;
ResolvedByHowMany $\leftarrow (0, 0, \ldots, 0)$;
TotalResolvedG $\leftarrow 0$;
foreach *binary matrix* H *in Gray code* **do**
    if H *is the matrix containing only zeros* **then** continue to the next matrix;
    ChangedRow $\leftarrow$ index of the row changed from the previous iteration;
    `/* Update state of xor-genotypes resolved by changed haplotype */`
    foreach *entry* (h_1, h_2, x) *of* *ListResolvedGenotypes*[*ChangedRow*] **do**
        Remove (h_1, h_2, x) from *ListResolvedGenotypes*$[h_1]$ and *ListResolvedGenotypes*$[h_2]$;
        ResolvedByHowMany$[x] \leftarrow$ *ResolvedByHowMany*$[x] - 1$;
        if *ResolvedByHowMany*$[x] = 0$ **then** *TotalResolvedG* $\leftarrow$ *TotalResolvedG* $- 1$;
    end
    `/* Look for genotypes resolved by the new haplotype */`
    for $r \leftarrow 1$ **to** k **do**
        if l *is the index returned by the lookup of the vector* $H[r, \cdot] \oplus H[$*ChangedRow*$, \cdot]$ *in* T **then**
            if *ResolvedByHowMany*$[l] = 0$ **then**
            TotalResolvedG $\leftarrow$ *TotalResolvedG* $+ 1$;
            ResolvedByHowMany$[l] \leftarrow$ *ResolvedByHowMany*$[l] + 1$;
            Add $(r,$ *ChangedRow*$, l)$ to *ListResolvedGenotypes*$[r]$ and to *ListResolvedGenotypes*[*ChangedRow*];
        end
    end
    if *TotalResolvedG* $= n$ **then**
        `/* all genotypes are resolved */`
        Remove from H all duplicate rows;
        return H;
    end
end
return *No*;

Algorithm 4: The approximation algorithm

Data: A set X of xor-genotypes over a set of characters Σ.
Result: A set H of haplotypes which resolves X.

```
1 H ← {h_0};
2 while Σ ≠ ∅ do
3     α ← any character in Σ;
4     foreach x ∈ X s.t. α ∈ x do
5         add to H the genotype x;
6     end
7     Remove from X all genotypes that contain the character α;
8     Remove α from Σ;
9 end
```

contain at most k elements, the time required for each iteration of the outermost loop is $O(km)$, resulting in an overall $O(nm + 2^{k^2}km)$ time complexity.

Finally, we point out that the time complexity is likely to make our fixed-parameter algorithm infeasible for moderate values of k, therefore it is mainly of theoretical interest.

4.5 An Approximation Algorithm

We present a simple approximation algorithm, detailed as Algorithm 4, which guarantees for a reduced instance X of PPXH an approximation factor l, where l is the maximum number of xor-genotypes where each character appears.

Initially the set H of haplotypes computed by the algorithm contains only the null haplotype. While the set of genotypes is not empty, pick a character α that appears in at least a genotype, move to H all genotypes containing α, and remove from X all genotypes that are solved by a pair of haplotypes in H. Clearly the final set of haplotypes H solves the set of genotypes X.

The proposed algorithm returns a solution of size at most l times larger than the optimum which, by Lemma 4.6, is at least $|\Sigma| + 1$. Our algorithm starts with a partial solution H containing only the null haplotype, and at each iteration adds at most l haplotypes to the solution H, as l is the maximum number of genotypes containing any character. Since there can be at most $|\Sigma|$ steps, $|H| \leq l|\Sigma| + 1$.

Clearly the approximation ratio is at most $(l|\Sigma| + 1)/(|\Sigma| + 1) \leq l$, completing the proof.

4.6 Solving PPXH by a Heuristic Method

In this section we propose a heuristic algorithm to build a near optimal xor-graph for an input matrix X of genotypes. Observe that an optimal xor-graph for X is a graph having the minimum-cardinality vertex set and where each edge is uniquely labeled by a genotype X. By Lemma 4.1, a cycle of the xor-graph consists of a subset X' of the input genotypes such that $\oplus X' = \varnothing$. Consequently we will call a subset X' with $\oplus X' = \varnothing$ a *candidate cycle*.

The basic idea that guides our heuristic is first to select a subset of the candidate cycles of X and then to build a labeled graph (a xor-graph) where the selected candidate cycles are actual cycles. The procedure successively iterates over the genotypes that are not yet successfully realized in the xor-graph.

A related problem is the one called *Graph Realization (GR)* [108], which consists of building a graph given its fundamental cycles. We recall that the set $\mathcal{C}$ of fundamental cycles of a graph G with respect to a fixed spanning tree T of G, is defined as $\mathcal{C} = \{\textit{the unique cycle of } T \cup \{e\} \mid e \in E(G) \setminus E(T)\}$ (see e.g. [33], pag. 26). More precisely, the Graph Realization problem can be formally stated as follows [108]. Given two disjoint sets T and C, the input of the GR problem is a family F of subsets of $T \cup C$ such that (i) for each set F_i of the family F, $F_i \cap C = \{c_i\}$, and (ii) for each pair of subset F_i and F_j of F, $F_i \cap F_j \cap C = \varnothing$. The GR problem consists of finding a labeled graph $G = (V, E)$ (if such a graph exists) which *realizes* F, that is there is a bijection between the set T and a spanning tree of G, and the elements of each set F_i label exactly the edge set of a (simple) cycle of G.

In the case that we have selected a set of candidate cycles which are fundamental cycles of a graph G, an immediate application of any algorithm solving the GR problem (two almost linear time algorithms exist [11, 42]), gives a xor-graph resolving all those candidate cycles.

A simple, albeit not practical, exponential-time algorithm to compute the optimal xor-graph G consists of guessing (i.e. trying all possibilities) the partition $\{W_1, \ldots, W_l\}$ of the genotypes corresponding to the biconnected components of G. Then compute the GR on each W_i to obtain a set of haplotypes Z_i. All sets Z_i are finally merged together by picking any vertex of Z_i and making such vertex the unique articulation point of the xor-graph.

We have been inspired by the algorithms for GR [11, 42]) and by our previous observations to develop our heuristic. We denote by $G(F)$ a graph realization G of a family of sets F.

The heuristic procedure transforms a $r \times c$ genotype matrix X into an instance of GR as described in the following two main steps. In the first step, the set T is defined as a maximal subset $T = \{x_{j_1}, \ldots, x_{j_c}\}$ of linearly independent input genotypes of X. This means that any other input genotype x_i can be expressed as a linear

Algorithm 5: The heuristic **Heu**(X)

Data: a xor-genotype reduced matrix X
Result: a haplotype matrix H that resolves X

$r \leftarrow$ the number of rows of X;
$c \leftarrow$ the number of columns of X;
if $r = c$ **then**
    return *the set consisting of* h_0 *and the canonical haplotypes of* X;
end
$R \leftarrow$ output of Gaussian elimination on X^T;
$T = \{x_1, \ldots, x_c\} \leftarrow$ the independent xor-genotypes labeling the first c columns of R, and $C = \{x_{c+1}, \ldots x_r\} \leftarrow$ the set of the remaining genotypes;
$F \leftarrow \varnothing$;
for $x_i \in C$ **do**
    $Z(x_i) \leftarrow \{x_i\} \cup \{x_j \in T \mid$ the element in row j and column i of R is equal to $1\}$;
    if $F \cup \{Z(x_i)\}$ *admits Graph Realization* **then**
        $F \leftarrow F \cup \{Z(x_i)\}$;
    end
end
Let $G(F)$ be the Graph Realization of F;
$H \leftarrow$ the set of vertices of $G(F)$;
$v \leftarrow$ a random element of H;
Each $h \in H$ becomes $h \oplus v$; `// Now` v `is the null haplotype`
Remove from X the genotypes that label any edge of $G(F)$;
`/* The instance` X `is reduced before the subroutine` ***Heu*** `is recursively called, and the general solution is then obtained as described in the proof of Lemma 4.4 */`
return $H \cup Heu(X)$;

combination $\alpha_{i,1}x_{j_1} \oplus \ldots \oplus \alpha_{i,c}x_{j_c}$ of the genotypes in T. Then $C = \{c_1, \ldots c_{r-c}\}$ is defined as consisting of the set of genotypes not in T. In a second step, the family of subsets of $T \cup C$ giving an instance of the GR is built by building sets F_i such that $F_i = \{c_i\} \cup \{x_{j_l} \in T \mid \alpha_{i,l} = 1\}$. Informally, F_i consists of c_i and the unique set $P_i \subseteq T$, such that $\oplus P_i = \{c_i\}$. An immediate consequence of our definition is that $\oplus F_i = \varnothing$, therefore F_i is, by definition, a candidate cycle.

Computing the set T from X is simply a matter of running the Gauss elimination algorithm on X^T (that is the transpose matrix of X). The family F can be easily inferred by computing the coefficients $\alpha_{i,1}, \ldots, \alpha_{i,c}$ for all $c_i \in C$, where the unknowns $\alpha_1^i \ldots \alpha_r^i$ are the coefficients of the linear combination and the binary matrix M is a matrix whose columns are the xor-genotypes in I.

Clearly, the Gauss elimination procedure applied on the matrix X^T results in a matrix R whose first r columns form the identity matrix while the other columns are the vectors of the linear combination coefficients.

We have a final hurdle, that is to handle the case where the GR does not exist for the family F. Once the family F is identified, the heuristics computes a maximal subfamily F' of F, so that there exists a GR from F'. Now, let us detail the construction of the family F giving an instance of GR. The heuristic starts defining F' as an empty family and iteratively adding to F' a candidate cycle F_i if and only if the resulting family admits a Graph Realization. Clearly, this approach ends with a maximal subset of candidate cycles that admits a Graph Realization. The two steps of the heuristic procedure are then recursively iterated on the set of xor-genotypes of X that do not label an edge of the computed Graph Realization. The details of the procedure are presented in Algorithm 5.

Let n and m be, respectively, the number of xor-genotypes and sites. The time complexity of the heuristic is determined by the time complexity of the Gauss elimination algorithm, which requires $O(nm^2)$ time because it is called on matrix X^T (which we suppose being reduced), and of the Graph Realization algorithm, whose best time complexity is $O(\alpha(n,m)nm)$, where α is the inverse Ackermann function. Notice that the Graph Realization algorithm is repeated at most n times in order to compute a maximal subfamily F', hence subfamily F' is computed in $O(\alpha(n,m)n^2m)$ time. Finally, there is at least one xor-genotype of X that labels an edge of the Graph Realization, hence the total number of iterations is at most n, that, combined with $m \leq n$ because the matrix is reduced, leads to an overall time complexity $O(\alpha(n,m)n^3m)$.

4.6.1 Experimental Results

We have implemented our heuristic as a C program using the software `GREAL` [7] as a routine to solve the Graph Realization problem. The `GREAL` program implements the algorithm of Gavril and Tamari [49] even if its time complexity is $O(nm^2)$ (opposed to the $O(\alpha(n,m)nm)$ time complexity of the best known algorithm), since it is still effective for our purposes.

The experimental analysis of our heuristic is composed of two parts. In the first part we have applied the algorithm on synthetic instances to evaluate the quality of the results in terms of cardinality of the solutions and running time. The main goal of this experimentation is to show the applicability of our heuristic to large instances, much larger than the ones that could be previosly attacked (to the best of our knowledge, the only known algorithms are based on Integer Linear Programming [20], [55]).

In the second part we have assessed the applicability of the heuristic to some real-world large instances.

Synthetic Data

Each synthetic instance has been created starting from a set of initial haplotypes and then each xor-genotype has been generated as combination of two haplotypes randomly selected from the initial set. Notice that such process does not guarantee that every haplotype is selected to form a genotype.

We have used two different methods to generate the set of initial haplotypes: (a) pure random generation, and (b) generation under the neutral model. The first strategy, pure random generation, selects uniformly sets of h distinct haplotypes from the set of all binary haplotypes of length m. The second strategy, generation under the neutral model, uses the standard Hudson's simulator `ms` [62] to generate a sample of h haplotypes assuming the neutral model of genetic variation. In this case, the sample of haplotypes can contain repeated elements. Using two different methods to generate the set of initial haplotypes allows us to verify if the behavior of the heuristic is influenced by the choice of the initial haplotypes.

The evaluation criteria, in both cases, were (a) the number of distinct haplotypes computed by our method, and (b) its running time. In particular, we have considered as main indicator of the quality of the solutions the ratio (r) between the number of distinct haplotypes of the computed solution and the number of distinct initial haplotypes selected to generate a genotype of the instance. We notice that r is only a proxy for the actual approximation ratio (that is the ratio between the number of distinct computed haplotypes and the size of a optimal solution) achieved by the algorithm, as the number of the selected haplotypes represents only an upper bound of the optimum, thus the ratio r might be strictly less than 1.

Since the outcome of our heuristic can be influenced by the order of the input genotypes, for each instance we have run the algorithms on ten random permutations of the genotypes, and we have retained only the smallest set of computed haplotypes. The running time refers to the total time required by the heuristic on the 10 permutations of genotypes and has been measured on a standard PC with 1GB of memory with CentOS Linux 5.

The pure random generation strategy is characterized by three parameters, namely the number of input genotypes (n), the number of haplotypes (h), and the number of characters (m). We have considered 4 different values of the parameter n (100, 200, 300, 400), and we have computed the values of h and m from n: in fact those values are $n/4$, $n/3$, and $2n/3$. The maximum size of the test instances (400 genotypes and 233 characters) has been chosen in such a way that repeated tests on several instances of the same size would be feasible on a normal computer. In fact, as discussed below, on average the heuristic required roughly an hour on the largest instances, therefore any further increase of the instance size would have made the experimentation impractical.

Table 4.1 reports the average size of the solutions computed by our heuristic, its average running time, and the average ratio r on 10 random instances generated for each choice of the parameters n, h, and m.

The second strategy, generation under the neutral model, is characterized by the three parameters n, m, and ρ, where n is the number of genotypes, m is the number of characters, and ρ is the crossover (or recombination) rate of the Hudson's program. The size of the initial sample of haplotypes has been set equal to the number n of genotypes. Since the sample can contain several copies of the same haplotype, the number of distinct haplotypes randomly selected to form a genotype has been significantly lower than the number of genotypes for almost all of the generated instances.

We considered 30 instances for each choice of the parameters (n, m, ρ) with $n \in \{50, 75, 100\}$, $m \in \{50, 75, 100\}$, and $\rho \in \{0, 8, 16, 24\}$. As for the previous dataset, Table 4.2 reports the average size of the solution computed by our heuristic, its average running time, and the average ratio r.

On both datasets the heuristic produces comparable results. In particular, the average ratio is never larger than 1.57, while quite often it is close to 1. In other words, it can often reconstruct a solution of size similar to the number of the haplotypes used to generate the instance and, in the worst case, the computed solution is at most 1.57 larger than the set of initial haplotypes. The ability of computing a good approximation seems affected by two combined factors: the number of independent characters of the genotype matrix and the number of initial haplotypes. Indeed in both tables we can observe that the smaller the number of independent characters compared to the number of initial haplotypes, the worse is the computed solution. Conversely good solutions are computed by the heuristic when the number of independent characters is close to the number of initial haplotypes.

Lemma 4.6 offers a possible explanation to such regular behavior of our heuristic. In fact, let H be the set of initial haplotypes of an instance X and suppose that they are defined on a set Σ of independent characters such that $|H| = |\Sigma| + 1$ (i.e. H is also a solution that meets the lower bound of Lemma 4.6). Then, the set T computed during step 7 of the heuristic algorithm contains exactly $|\Sigma|$ independent xor-genotypes. As a consequence, the set C computed in the same step admits a Graph Realization and, thus, the heuristic solves optimally the instance X. Although this is not the general case, our intuition suggests that, when the number of independent characters is close to the number of initial haplotypes, the selection of the set T is constrained and the Graph Realization of the maximal subset of C computed by the heuristic is similar to the xor-graph associated with the initial haplotypes. Conversely, if the number of independent characters is significantly lower than the number of initial haplotypes, there are a lot of degrees of freedom in the choice of the set T, thus the output of the Graph Realization step can vary greatly from the xor-graph of the initial haplotypes.

Table 4.1: Results on instances generated using the pure random strategy. For each choice of the first three columns, 10 random instances were generated. The column *average independent characters* reports the average number of independent character in the genotype matrix, while column *average initial haplotypes* reports the average number of distinct haplotypes selected to generate each instance. The last two columns report, respectively, the average size of the solution computed by our heuristic and the average ratio r.

number of genotypes n	number of generated haplotypes h	number of characters m	average independent characters	average initial haplotypes	average result size	average ratio r
100	25	25	23.70	25.00	25.90	1.04
		33	24.00	25.00	25.00	1
		66	24.00	25.00	25.00	1
	33	25	25.00	32.90	51.60	1.57
		33	31.50	32.80	33.20	1.01
		66	32.00	33.00	33.00	1
	66	25	25.00	63.00	87.30	1.39
		33	33.00	63.30	87.20	1.38
		66	62.70	63.90	63.80	1
200	50	50	48.70	50.00	50.90	1.02
		66	49.00	50.00	50.00	1
		133	49.00	50.00	50.00	1
	66	50	50.00	65.80	96.20	1.46
		66	64.50	65.90	66.20	1
		133	64.80	65.80	65.80	1
	133	50	50.00	126.90	185.80	1.46
		66	66.00	126.20	186.10	1.47
		133	126.60	128.10	127.70	1
300	75	75	73.70	75.00	75.80	1.01
		100	74.00	75.00	75.00	1
		200	73.90	74.90	74.90	1
	100	75	75.00	99.80	149.80	1.50
		100	98.60	99.90	100.00	1
		200	98.80	99.80	99.80	1
	200	75	75.00	190.80	285.90	1.50
		100	100.00	191.10	284.60	1.49
		200	188.20	190.30	189.20	0.99
400	100	100	98.50	100.00	100.90	1.01
		133	98.90	99.90	99.90	1
		266	98.90	99.90	99.90	1
	133	100	100.00	132.80	194.90	1.47
		133	131.40	132.80	133.00	1
		266	131.80	132.80	132.80	1
	266	100	100.00	254.70	385.30	1.51
		133	133.00	253.60	384.40	1.52
		266	251.90	253.50	252.90	1

Table 4.2: Results on instances generated using the neutral model. For each choice of the first three columns, 30 random instances were generated. The column *average independent characters* reports the average number of independent character in the genotype matrix, while column *average initial haplotypes* reports the average number of distinct haplotypes selected to generate each instance. The last two columns report, respectively, the average size of the solution computed by our heuristic and the average ratio r.

number of genotypes n	number of characters m	recombination rate ρ	average independent characters	average initial haplotypes	average result size	average ratio r
50	50	0	18.5	19.5	19.5	1
		8	20.13	21.73	22.03	1.02
		16	22.27	24.77	25.77	1.04
		24	20.63	24.17	25.23	1.05
	75	0	22.1	23.13	23.1	1
		8	24.63	26	26.27	1.01
		16	25.6	27.3	27.37	1.01
		24	25.3	27.63	28.1	1.02
	100	0	25.07	26.13	26.07	1
		8	26.7	27.93	27.8	1
		16	28.27	29.87	29.73	1
		24	28.5	30.5	30.2	0.99
75	50	0	21.77	22.77	22.77	1
		8	23.1	25.37	26.63	1.05
		16	24.97	29.77	34.4	1.16
		24	25.1	31.4	38.33	1.23
	75	0	26.17	27.2	27.17	1
		8	29.9	31.5	31.77	1.01
		16	30.93	34.63	37.1	1.07
		24	31.23	35.83	38.83	1.08
	100	0	29.5	30.5	30.5	1
		8	32.87	34.23	34.13	1
		16	33.67	36.1	36.77	1.02
		24	36.2	39.73	41.17	1.04
100	50	0	24.33	25.33	25.33	1
		8	27	30.67	36.6	1.2
		16	27.53	32.8	41.8	1.28
		24	27.93	36.1	49	1.36
	75	0	27.83	28.83	28.83	1
		8	32.2	34.4	36.07	1.05
		16	34.23	38.33	43	1.12
		24	35.23	42.07	50.43	1.2
	100	0	34.5	35.5	35.5	1
		8	37.37	39.13	40	1.02
		16	39.87	43.27	45.63	1.06
		24	38.6	45.13	52.87	1.17

The time required by the heuristic to compute a solution to the pure-random synthetic instances varies between circa 25 seconds on instances with 100 genotypes and 70 minutes on instances with 400 genotypes. All the instances generated using the neutral model, instead, have been solved in less than 1 minute. We also observe that instances where the heuristic fails to find a good solution have been solved considerably faster than the ones where the heuristic computes a good approximation. However, a more careful analysis suggests that such fluctuations are due to the different amount of I/O operations needed to communicate with the `GREAL` software that we use to solve the Graph Realization problem.

Finally we tried to compare our heuristic method with the ILP formulation proposed by Brown and Harrower [20]. In the paper, they formulate the PPXH problem as a polynomial-size integer linear program and they introduce cuts and modification of the objective function that should help finding the optimal solution. However, the `GLPK` solver [86], using the basic formulation as well as the augmented formulations, was not able to find a feasible solution even for the smallest instances of our experimentation (50 genotypes and 50 characters) within the maximum time of 24 hours.

Also a comparison with the ILP formulation proposed by Gusfield [55] has turned out to be infeasible. The approach proposed in [55] would require to compare all "haplotype pairs" in order to trim the resulting ILP. Unfortunately, for our smaller instance the number of such "haplotype pairs" is more than $8 \cdot 10^9$ (the same number for our larger instance is more than 10^{14}) hence making *de facto* impossible to compute this ILP formulation.

Real Data

To validate the feasibility of applying our heuristic on real data, we have produced some instances from the Phase I dataset of the HapMap project [106] (release 2005-06_16c.1). A set of xor-genotypes were produced from the data for each population in the dataset (discarding non biallelic sites and non autosomal chromosomes). Those instances vary from 44 genotypes and 184604 sites to 90 genotypes and 91812 sites. On average, an instance contains 67 genotypes and 46906 sites. On all those instances our heuristics has never required more than 2 seconds on the same PC used in the experimental part over synthetic instances, clearly establishing that the heuristic can be successfully used on real-world large instances.

5 Haplotype Inference on Pedigrees with Recombinations and Mutations

Pedigrees have been shown as a valuable kind of data to improve the accuracy of haplotype inference (HI) methods, since Mendelian inheritance restricts the set of possible configurations and provides an effective tool to compute the likelihood of the solutions. In order to overcome the limitations of classic statistical-based haplotyping methods, a combinatorial formulation of the HI problem on pedigrees has been proposed in the literature: the Minimum-Recombinant Haplotype Configuration (MinRHC) problem. In this formulation, a set of haplotypes is inferred from a pedigree whose individuals are labelled by a set of genotypes, by assuming that only one kind of genetic variation events has occurred, namely the recombinations.

In this chapter we formulate a new problem, called Minimum-Event Haplotype configuration (MinEHC), that extend the formulation of MinRHC in order to accommodate also a second kind of variation events: *mutations*. In particular we study the computational hardness of MinEHC and of some closely related problems and we propose an efficient heuristic algorithm for it. As a by-product, the same heuristic can be also used to solve the original MinRHC problem. Moreover we show that our heuristic can easily integrate additional knowledge about the input genotypes, such as the presence of *recombination hotspots* and a different rate of recombinations and mutations. Finally we present an extensive experimental evaluation of our approach under several simulated scenarios.

5.1 Motivations

The advance of high-throughput and high-density SNP genotyping technologies, combined with a consistent reduction of genotyping costs had led to a great abundance of genotypic data. Since haplotypes substantially increase the power of genetic variation studies, accurate and efficient computational prediction of haplotypes from genotypes is highly desirable. Mendelian inheritance laws, which model the transmission of genetic material between parents and children, have been effectively used to improve the accuracy of haplotyping methods. The increasing density and length of SNP genotypes challenge classic statistical-based methods (such as Lander-Green and Elson-Stewart methods) for two reasons: on

such data, statistical-based methods require an infeasible amount of computational resources, and they do not take directly in account the presence of Linkage Disequilibrium among loci. Combinatorial formulations have been proposed to overcome such limitations. Among them, the most popular formulation is represented by the Minimum-Recombinant Haplotype Configuration (MinRHC) problem. The aim of this formulation is the computation of a haplotype configuration which is consistent with a input genotyped pedigree and which induces the minimum number of recombination events. The formulation naturally arises since recombinations are the most common form of variation events. However, with the progressive increase of the size of genetic variation studies (both on genotype length and on pedigree size), the incidence of the other kinds of variation events (such as *mutations*) will inevitably become noticeable.

This observation motivates the work presented in this chapter, where the HI problem on pedigrees with recombinations and mutations, called Minimum-Event Haplotype Configuration (MinEHC), is studied.

The main contribution of this work is an efficient and effective heuristic algorithm for MinEHC. Our algorithm is based on a L-reduction of MinEHC to a central problem of coding theory: the Nearest Codeword Problem (NCP) [6, probl. MS3]. Even if NCP is hard to approximate [5], there exists several heuristics that approximate well NCP in practice. Our idea is to transform the instance of MinEHC to an instance of NCP, to solve the instance of NCP with a custom-tailored version of a standard heuristic for NCP, and, finally, to reconstruct a solution of the original instance of MinEHC from the solution of NCP. The L-reduction guarantees that the transformation of the instance and the reconstruction of the solution are performed in polynomial-time. Moreover, the reconstruction process preserves the solution cost.

For sake of completeness, we also prove that MinEHC and some related problems are computationally hard (namely, APX-hard), justifying our heuristic approach.

In the remaining, we implicitly assume that genotypes are biallelic to simplify the exposition. Nonetheless all the results we present can be easily extended to the multi-allelic case.

The chapter is structured as follows. In Section 5.2, we formalize the Minimum-Events Haplotype Configuration problem, and we define the related basic terminology. In Section 5.3, we prove that MinEHC is APX-hard and that the problem remains APX-hard even if we admit only mutation events. On the positive side, in Section 5.4, we present the L-reduction from MinEHC to NCP and we illustrate our heuristic algorithm for MinEHC. Finally, an experimental evaluation of our algorithm is presented and discussed in Section 5.5.

5.2 The Computational Problem

Haplotype Inference problems on structured populations have been introduced in Section 3.3. In this section we recall the most important concepts and we formalize the computational problems that will be studied in the rest of the chapter.

A *simple pedigree graph* is an oriented acyclic graph $P = (V, E)$ such that (i) vertices correspond to population members and are distinguished in *male* and *female* (i.e. $V = M \cup F$, with M and F disjoint), and (ii) each vertex has indegree 0 or 2 and, in the latter case, one edge comes from a male node and one comes from a female node. For each edge $(p, c) \in E$, we say that p is a parent of c and c is an offspring (or child) of p. More precisely we say that p is the father (mother, resp.) of c if p is male (female, resp.). Since simple pedigree graphs are the only medium that we use to represent family relationships in this chapter, for sake of simplicity we will call them *pedigree graphs*, omitting the specifier "simple".

A *trio* is a triplet (f, c, m) where f is the father and m is the mother of c. Individual f and individual m are said to be *mates*. A *nuclear family* is a set $\{f, m, c_1, \ldots, c_k\}$ such that (f, c_i, m) is a trio for all $i = 1 \ldots k$. Every pair of individuals c_i and c_j that have the same parents are *siblings*. Given an oriented path from an individual a to an individual d, we say that a is an ancestor of d and d is descendant of a. A pedigree graph contains a *mating loop* if there exists two nodes a and d such that they are connected by two distinct paths. (Notice that this definition is completely equivalent to the strictest definition of mating loops of Section 3.3.) A pedigree graph is a *tree pedigree* if it does not contain mating loops. A pedigree graph is a *binary tree pedigree* if each individual has at most one offspring. (This also implies that each individual has at most one mate and that the pedigree graph is a tree pedigree.)

Let Σ be an ordered set $\langle l_1, \ldots, l_m \rangle$ of m loci, and let c be an individual of the population P. Then, a *haplotype* h_c of individual c is an m-dimensional vector over the set $\{0, 1\}$, where the i-th element (denoted with $h_c[i]$) is 0 if individual c presents the major allele at locus l_c, and 1 otherwise. Instead, the *genotype* g_c of individual c is an m-dimensional vector over the set $\{0, 1, 2\}$, where the i-th element (denoted with $g_c[i]$) represents the pair of alleles that individual c possesses at locus l_i. We follow the convention of encoding pair $(0, 0)$ with 0, $(1, 1)$ with 1, and $(0, 1)$ with 2.

A *genotyped* (*haplotyped*, respectively) *pedigree* is a pedigree such that every individual has associated a genotype (an ordered pair of haplotypes, respectively). We denote with g_c the genotype associated with an individual c of a genotyped pedigree and we denote with $\langle h_c^0, h_c^1 \rangle$ the haplotypes associated with an individual c of a haplotyped pedigree. Moreover, we say that h_c^0 is the paternal haplotype of c and h_c^1 is the maternal haplotype of c. A genotyped pedigree is a *m-locus pedigree* if its members are associated with a genotype defined on a set of m loci. A haplotyped

pedigree P_h is *consistent* with a genotyped pedigree P_g of the same population if for each individual c, the genotype g_c is resolved by the pair of haplotypes $\langle h_c^0, h_c^1 \rangle$. The *grandparental source vector* of a non-founder individual c w.r.t. one of its parents p, is a m-long binary vector $s_{p,c}$ defined as follows. Let l_i be a locus of Σ. If p is the father (mother, respectively) of c, then $s_{p,c}[i] = 0$ if the allele of the paternal (maternal, resp.) haplotype of c at locus l_i has been inherited from the paternal haplotype of p, otherwise $s_{p,c}[i] = 1$ if the allele has been inherited from the maternal haplotype of p. Given a population P, a (consistent) *haplotype configuration* of a genotyped pedigree P_g of P is a pair (P_h, S) where P_h is a (consistent) haplotyped pedigree of P and S is an assignment of two grandparental source vectors to each individual of P.

In general terms, the Haplotype Inference problem on pedigrees asks for a haplotype configuration consistent with a given genotyped pedigree. As explained in Section 3.3, the choice of the "best" haplotype configuration is guided by a reference genetic model. In this chapter we are interested in a parsimony-based approach. In particular we want to minimize the number of genetic variation events that are induced in the resulting haplotyped pedigree. Two kinds of variation events will be considered, *recombinations* and *mutations*, defined as follows. Let (P_h, S) be a consistent haplotype configuration of a genotyped pedigree P_g. The haplotype configuration induces (or contains) a *recombination* at locus l_i between an individual c and one of its parents p if $s_{p,c}[i] \neq s_{p,c}[i+1]$. The haplotype configuration induces (or contains) a *mutation* at locus l_i between c and its parent p if $h_c^j[i] \neq h_p^s[i]$ where $s = s_{p,c}[i]$ and $j = 0$ ($j = 1$, resp.) if p is the father (mother, resp.) of c.

We are now able to formally define the computational problem that we are interested in.

Problem 6. Minimum-Event Haplotype Configuration (MinEHC).
Input: A genotyped pedigree P_g of a population P.
Output: A haplotype configuration (P_h, S) consistent with P_g that induces the minimum number of variation events.

The Minimum-Event Haplotype Configuration problem is a generalization of two problems proposed in literature: the Minimum-Recombinant Haplotype Configuration (MinRHC) problem (see, for example, [76]), and the Minimum-Mutation Haplotype Configuration (MinMHC) problem [110].

Problem 7. Minimum-Recombinant Haplotype Configuration (MinRHC).
Input: A genotyped pedigree P_g of a population P.
Output: A haplotype configuration (P_h, S) consistent with P_g that does not contain mutations and that induces the minimum number of recombinations.

Problem 8. Minimum-Mutation Haplotype Configuration (MinMHC).
Input: A genotyped pedigree P_g of a population P.
Output: A haplotype configuration (P_h, S) consistent with P_g that does not contain recombinations and that induces the minimum number of mutations.

We want to remark that, differently from [110], we are not constraining the number of mutations that can occur at the same locus (in different individuals).

Some restrictions of the MinEHC and MinMHC problem will be used in the following. In particular we define *binary-tree*-MinEHC the MinEHC problem on binary tree pedigrees, and *2-locus*-MinEHC the MinEHC problem on genotyped pedigree where the genotypes are defined on 2 loci. The restrictions *binary-tree*-MinMHC and *2-locus*-MinMHC are defined in the same way.

Notice that the MinEHC problem requires the explicit computation of both a haplotyped pedigree and a family of grandparental source vectors. This is needed since one of them is not sufficient for a not-ambiguous reconstruction of the variation events induced by the solution. If we allow some ambiguity, the grandparental source vectors are not strictly required, since it is possible to reconstruct in polynomial-time a minimum-cardinality set of events from the haplotyped pedigree alone.

5.3 Computational Complexity

The computational complexity and the approximation hardness of various cases of the Minimum-Recombinant Haplotype Configuration (MinRHC) problem have been extensively studied by Liu *et al.* [83, 84]. They essentially present three results: (*i*) MinRHC is NP-hard even for simple pedigrees (namely, for binary-tree pedigrees), (*ii*) MinRHC with missing data (i.e. loci where the genotype of a individual is not known) cannot be approximated, and (*iii*) MinRHC is APX-hard even for simple instances (2-locus pedigrees and tree pedigrees).

In this section we extend the work of Liu *et al.* by studying the computational complexity and the (in)approximability properties of the Minimum-Mutation Haplotype Configuration problem and the Minimum-Event Haplotype Configuration problem. In particular we are interested in designing reductions that use simple instances of MinEHC and MinMHC in order to highlight that the problems are hard to solve (and approximate) even on tight restrictions. We succeed in our aim: In fact we prove (Sect. 5.3.1) that MinMHC on binary-tree pedigrees is APX-hard and, based on the work of Liu *et al.*, we also show (Sect. 5.3.2) that 2-locus-MinMHC and 2-locus-MinEHC are APX-hard. These results obviously imply the APX-hardness of MinEHC and MinMHC on general pedigrees.

The simple concepts of computational complexity that we use in the rest of the chapter have been recalled in Section 2.1.

5.3.1 binary-tree-MinMHC is APX-hard

Here we present an L-reduction from the Minimum Edge-Bipartization problem on cubic graphs (Min Edge-Bipartization-R3) to the MinMHC problem. As a consequence we obtain that MinMHC is APX-hard.

Before describing the reduction, we define the Min Edge-Bipartization problem.

Problem 9 (optimization version of GT25 [47])**.** Min Edge-Bipartization.
Input: An unoriented graph $G = (V, E)$.
Output: A minimum subset E' of E such that $G' = (V, E \setminus E')$ is bipartite.

The Min Edge-Bipartization problem can be equivalently formulated as the problem of finding a cut C of the graph which minimizes the number of non-crossing edges. In fact G' is bipartite if and only if $E \setminus E'$ is a cut of graph G and, clearly $| E' |$ is minimum if and only if $E \setminus E'$ is a maximum cardinality cut. In this case, the problem is known as MinUncut.

Min Edge-Bipartization is NP-hard even if we restrict the input graph to be cubic (a graph which the degree of each vertex is exactly 3) and triangle-free (a graph which does not contain cycles of length 3) [122].

Min Edge-Bipartization is closely related to another well-known NP-hard problem: MaxCut.

Problem 10 (ND14 [6])**.** Maximum Cut (MaxCut).
Input: An unoriented graph $G = (V, E)$.
Output: A bipartition $\{V_1, V_2\}$ of V such that the cardinality of the cut (i.e. the number of edges with one endpoint in V_1 and one endpoint in V_2) is maximum.

MaxCut is APX-hard even on cubic graphs (MaxCut-R3) [3, 94], and, from that, we also obtain the APX-hardness of Min Edge-Bipartization on cubic graphs (Min Edge-Bipartization-R3). This result can be easily obtained by an L-reduction from MaxCut-R3 (as in [28]) or, directly, by observing that a PTAS for Min Edge-Bipartization-R3 would imply a PTAS for MaxCut-R3 which is not possible under the assumption P $\neq$ NP. We provide such proof for sake of exposition completeness.

Lemma 5.1. Min Edge-Bipartization-R3 *is* APX*-hard.*

Proof. Denote with $c(G)$ (and $u(G)$, resp.) the size of a cut (the cardinality of a Min Edge-Bipartization-R3 solution, resp.) of a cubic graph G with n vertices and m edges. Denote with $c^*(G)$ and $u^*(G)$ the size of an optimal solution of the two problems on graph G. Moreover, notice that we can easily compute a cut C that bipartizes the vertex set of G in $\{V_1, V_2\}$ such that at least $\frac{2}{3}$ of the edges cross C. In fact, if a vertex has two adjacent vertices in the same set of the partition, we can

move it in the other element of the partition and the number of edges which cross the new cut is increased. With the same procedure we can easily derive a subset of at most $\frac{m}{3}$ edges whose removal bipartizes G. In other words we can compute in polynomial-time a cut s.t. $c(G) \geq \frac{2}{3}m$ and a Min Edge-Bipartization-R3 solution s.t. $u(G) \leq \frac{1}{3}m$. Now suppose to have a k-approximation algorithm for Min Edge-Bipartization-R3, and thus $u(G) \leq ku^*(G)$. Using such algorithm we can compute a solution for MaxCut-R3 such that $\frac{c(G)}{c^*(G)} = \frac{m-u(G)}{c^*(G)}$. By combining the previous inequalities we obtain:

$$\frac{c(G)}{c^*(G)} \geq \frac{m - \frac{k}{3}m}{\frac{2}{3}m} = \frac{3-k}{2}$$

Since MaxCut-R3 is APX-hard, there exists a $\varepsilon > 0$ such that approximating MaxCut-R3 within $1-\varepsilon$ is NP-hard. Thus, assuming $\mathsf{P} \neq \mathsf{NP}$, we have:

$$\frac{3-k}{2} \leq \frac{c(G)}{c^*(G)} \leq 1 - \varepsilon$$

which implies $k \geq 1 + 2 \cdot \varepsilon$ or, in other words, that Min Edge-Bipartization-R3 is APX-hard. □

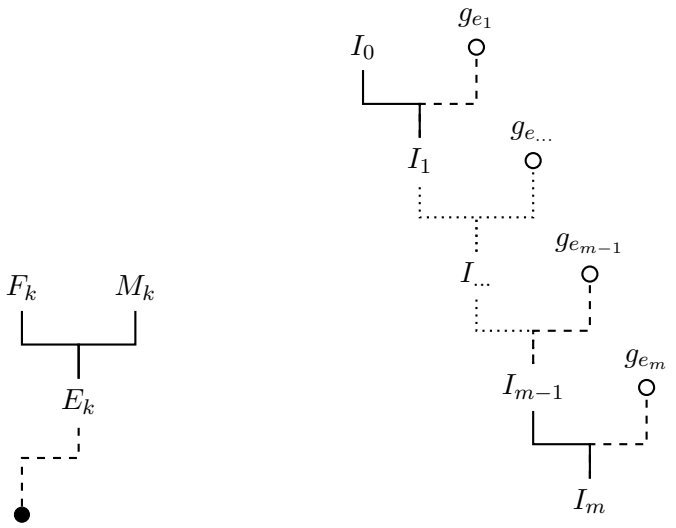
In the following we will start by describing the gadget used in the reduction from Min Edge-Bipartization-R3 to MinMHC and then we will formally prove that the reduction is an L-reduction.

Description of the gadget

Let $G = (V, E)$ be a cubic graph, and let n and m be the cardinality of V and E, respectively. Define the set of loci as the set $L := \{v_i \mid v_i \in V\}$. First of all we need a set G_E of gadgets to encode the edge set. An edge $e_k = (v_i, v_j)$ of G is represented by a trio $g_{e_k} = (F_k, E_k, M_k)$ as depicted in Figure 5.1(a).

The genotype of individual F_k is set to 1 in locus v_i, 0 in locus v_j, and 2 in all the other positions. The genotype of individual M_k, instead, is set to 0 in locus v_i, 1 in locus v_j, and 2 in all the other positions. Finally, individual E_k is heterozygous in every locus. For simplicity, in the remainder we say that a locus i is 0-homozygous (1-homozygous, respectively) in a given individual if the genotype of the individual at locus i is 0 (1, respectively).

The edge-gadgets g_{e_k} are connected to a sort of "backbone" that represents the entire graph. The "backbone" is composed by $m+1$ individuals I_k with $k = 0 \dots m$ such that I_{k-1} and E_k are the parents of I_k for all $1 \leq k \leq m$. (By convention we assume that I_k are male nodes, while E_k are female.) Finally we have to specify the genotypes of individuals I_k. For all $k = 0 \dots m$, individuals I_k are heterozygous



(a) The part g_{e_k} of the gadget that represents an edge $e_k = (v_i, v_j)$

(b) The complete gadget

Figure 5.1: The gadget used in the reduction.

at every locus. Figure 5.1(b) illustrates the complete gadget. Given a cubic graph G, we indicate with P_G the complete gadget (i.e. the pedigree graph) that encodes G.

The L-reduction

The reduction from Min Edge-Bipartization-R3 to MinMHC is composed by the following two lemmas.

Lemma 5.2. *Let $G = (V, E)$ be a cubic graph and let $E' \subseteq E$ be a solution of* Min Edge-Bipartization-R3 *on G. Then, a haplotype configuration H for P_G with at most $|E'|$ mutations can be computed in polynomial time.*

Lemma 5.3. *Let $G = (V, E)$ be a cubic graph and let H be a haplotype configuration for P_G that requires l mutations. Then, a solution E' of* Min Edge-Bipartization-R3 *on G such that $|E'| \leq l$ can be computed in polynomial time.*

Clearly the following corollary can be easily derived from the previous lemmas and from the APX-hardness of Min Edge-Bipartization-R3.

Corollary 5.4. *There exists an L-reduction with $\beta = 1$ and $\gamma = 1$ from* Min Edge-Bipartization-R3 *to* MinMHC. *In other words,* MinMHC *is* APX*-hard.*

Lemma 5.2 can be easily proved by constructing a haplotype configuration with at most $|E'|$ mutations based on the given solution. The second lemma (Lemma 5.3), instead, is a little bit trickier than the first one, and requires the transformation of the given haplotype configuration to another haplotype configuration where the mutations are located in specific haplotypes. After that, a Min Edge-Bipartization-R3 solution of the given cardinality can be easily reconstructed. We start by proving two key properties about the gadget.

Property 5.5. *In a haplotype configuration of the whole pedigree, each trio $g_{e_k} = (F_k, E_k, M_k)$ does not contain mutations, or otherwise the haplotype configuration can be transformed in a new one that contains no mutations in the trio.*

Proof. Suppose that the gadget g_{e_k} represents edge $e_k = (v_i, v_j)$. Notice that F_k and M_k are founders of the pedigree and they are heterozygous at every locus but v_i and v_j. Given a haplotype configuration that contains (at least) a mutation in trio g_{e_k}, we can show that such a mutation can be avoided or "moved" in a different place.

Two cases must be studied separately: either (*1*) the mutation does not involve locus v_i or v_j, or (*2*) the mutation involves one of such loci.

In the first case, let v be the locus where the haplotype inherited from parent P_k has been mutated. Then change the phase of individual P_k at locus v. Clearly this change does not violate the genotype consistency of the individual and does not violate the Mendelian inheritance since (*i*) P_k is a founder (thus its haplotypes are not inherited from other individuals), and (*ii*) it does not have other children apart E_k (thus its haplotypes are not transmitted to other individuals). This change can be repeated until the haplotype configuration of the trio does not contain mutations in any loci but v_i or v_j.

In the second case, suppose, without loss of generality, that v_i is the locus where the mutation has occurred. Clearly, since both parents of E_k are homozygous at locus v_i, we cannot proceed as in the previous case. Instead we show how the mutation can be "pushed down" from individual E_k to individual I_k of the "backbone" (or, in some case, it can be completely removed). By genotype consistency, notice that if a mutation has occurred in locus v_i between individual F_k and E_k, then a mutation in locus v_i has occurred also between individual M_k and E_k and vice versa. In fact, suppose that the mutation from F_k to E_k has transformed allele 0 to allele 1, then individual E_k has inherited allele 0 from M_k. By construction, if locus v_i is 0-homozygous in F_k, then the same locus is 1-homozygous in M_k, which implies the presence of a mutation in v_i between M_k and E_k. Now, we claim that if a mutation has occurred in v_i between E_k and its children I_k, then all three mutations can be removed. Let a be the allele of the maternal haplotype of I_k at locus v_i. Since a mutation has occurred between E_k and I_k, individual E_k has allele $1-a$ at locus v_i. As previously said, both haplotypes of E_k have the allele at locus v_i

mutated with respect to the parental haplotype. Thus the haplotype of F_k or M_k that has been inherited by I_k has allele a at locus v_i. Removing the three mutations does not affect genotype consistency and, furthermore also Mendelian consistency is preserved because I_k has no siblings and its haplotypes do not change. Finally we have to prove that if locus v_i has not been mutated between individual E_k and individual I_k, then the mutations between E_k and its parents can be replaced with a single mutation between E_k and I_k. In this case, E_k and I_k share the same allele at locus v_i (on the haplotype that I_k has inherited from E_k). However such allele is not the same of the allele inherited from the grandparent of I_k because of the mutations that have occurred between E_k and its parents. Thus a single mutation on locus v_i between E_k and I_k does not change the haplotypes of I_k if the mutations between E_k and its parents are removed. Also in this case, genotype and Mendelian consistency are preserved since I_k is the only child of E_k.

In all cases, we were able to compute (in polynomial time) a new haplotype configuration without adding new mutations and such that no mutations occur between two individual of any trio (F_k, E_k, M_k). □

A haplotype configuration is *basic* if it does not induce mutations in any trio (F_k, E_k, M_k). From the previous property we can easily derive the following statement.

Property 5.6. *In a basic haplotype configuration, for all individuals E_k of the gadget which represents edge (v_i, v_j), the alleles at loci v_i and v_j differ.*

Proof. The property is a direct consequence of Property 5.5 and the fact that F_k and M_k have different alleles at loci v_i and v_j. □

We are now ready to prove the first lemma that composes the reduction.

Proof of Lemma 5.2. First of all, we can assume that for each vertex v, the set E' contains at most one edge incident to v. Otherwise we can shrink the original solution E' by replacing the edges of E' incident to v with the other edges (if they exist) incident to v. Clearly this procedure can be performed in polynomial time and it does not increment the cardinality of the solution E'.

Let $\{V_1, V_2\}$ be a bipartition of V induced by the removal of E'. Set the paternal haplotype of every individual I_k such that every locus v has allele 0 if $v \in V_1$, otherwise set it to allele 1. Set the maternal haplotype as the bit-wise complement of the paternal haplotype. Finally set the haplotypes of each trio (F_k, E_k, M_k) in such a way that the haplotype configuration is basic (see Property 5.5).

We claim that the haplotype configuration constructed so far has exactly one mutation for each edge $e_k = (v_i, v_j)$ of the graph that belongs to E'. In fact, no matter of which haplotypes individual E_k inherit from his parents, a single mutation

occurs in the transmission of a haplotype from E_k to I_k. By Property 5.6, the alleles at loci v_i and v_j are different in individual E_k, while they are equal in individual I_k because e_k belongs to E'. Thus, exactly one mutation is contained in the haplotype configuration for each edge $e_k \in E'$. Instead, the alleles of the loci that represent the endpoints of an edge $e_{k'}$ which does not belong to E' are different, so no mutations from $E_{k'}$ to $I_{k'}$ occur, which concludes the proof. □

The basic idea which the proof of Lemma 5.3 relies on is that the haplotype of individual I_0 naturally encodes a bipartition of the vertex set. By removing the edges that do not cross the bipartition, we then obtain a solution E' for Min Edge-Bipartization-R3. We claim that the haplotype configuration can be transformed to another haplotype configuration which induces mutations only on the maternal haplotype of individuals I_k, and that each mutation of this haplotype configuration corresponds to an edge of E'.

Proof of Lemma 5.3. We can assume that the haplotype configuration H does not contain mutations in a trio (F_k, E_k, M_k), otherwise we can eliminate such mutations as discussed in the proof of Property 5.5. Moreover, given an edge $e_k = (v_i, v_j)$, a mutation of the maternal haplotype of individual I_k must involve locus v_i or locus v_j, otherwise we can remove it by changing the haplotypes of the trio g_{e_k} because the mother and the maternal grandparents of I_k are heterozygous at all loci but v_i and v_j. Mutations of the paternal haplotype of I_k that do not involve locus v_i or v_j can also be removed by changing the maternal haplotype and proceeding as previously described. In addition, if both loci v_i and v_j are mutated in the maternal haplotype of I_k, then such mutations can be avoided. We can also remove two mutations of loci v_i and v_j in the paternal haplotype of I_k by changing the maternal haplotype at such loci, and then applying the previous procedure. (Notice that mutations are not always removed in a strict sense: the mutations that we "remove" at I_k have been actually moved on I_{k+1}.) Repeat the previous mutation removal steps until a case applies. Let H' be the resulting haplotype configuration and let l' be the number of mutations of H'. (Clearly $l' \leq l$.)

Construct a bipartition $\{V_1, V_2\}$ of the vertex set V as follows: $v_i \in V_1$ if the locus v_i of the paternal haplotype of I_0 contains allele 0, otherwise $v_i \in V_2$. Let us show that bipartition $\{V_1, V_2\}$ induces a Min Edge-Bipartization-R3 solution E' such that $|\, E' \,| = l'$.

A mutation can occur (i) in a paternal haplotype of I_k, or (ii) in a maternal haplotype of I_k. (Suppose that $e_k = (v_i, v_j)$.) By construction of H', either v_i or v_j has been mutated (suppose w.l.o.g. v_i), and the other loci has been inherited without mutations. In case (i), moreover, we can assume that there exists at least an edge e_t incident to v_i with $t < k$ (otherwise we can remove the mutation by changing allele in the paternal haplotype of I_o at locus v_i and propagating). We

can also assume that the alleles at loci v_i and v_j are different in I_k and equal in I_{k-1}, otherwise a mutation on locus v_j of the maternal haplotype of I_k is required and we can easily remove both of them. Let us first consider the case in which two edges $e_{t'}$ and $e_{t''}$ incident to v_i have index less than k (i.e. $t' < k$ and $t'' < k$). In this case, the mutation can be moved (but not removed) to the maternal haplotype and we can change accordingly the haplotypes of every descendant of I_k without creating new mutations. Indeed, no edge $e_t = (v_i, v')$ with $t > k$ exists, so the procedure described at the beginning of the proof can be applied (on locus v_i) on every descendant of I_k. If the mutation occurs at individual I_k and there exists two edges $e_{t'} = (v_i, v')$ and $e_{t''} = (v_i, v'')$ such that $t' < k < t''$, then set the paternal allele of locus v_i in $I_{t'}$ equals to the paternal allele of same locus in I_k and propagate such change to descendants and ancestors. Clearly, this operation moves the mutation on the maternal haplotype of $I_{t'}$ and it does not create any other mutations.

In case (*ii*) the alleles of the maternal (and thus of the paternal) haplotype of I_k at loci v_i and v_j coincide. Assuming that the paternal haplotype of I_k at loci v_i and v_j is equal to (or is the complement of) the paternal haplotype of I_0, vertices v_i and v_j belong to the same set of the bipartition $\{V_1, V_2\}$, so edge $e_k \in E'$.

In this proof we have shown how to move mutations in order to put them on the maternal haplotypes of individuals I_k. The analysis of case (*ii*) has shown that a mutation on a maternal haplotype is in correspondence with an edge of E'. Moreover, it is easy to see that for each edge $e_k \in E \setminus E'$, individual I_k does not contain mutations. Since the procedure that moves and removes mutations does not increase the number of them, we have $|E'| = l' \leq l$ that concludes the proof. □

Final remark

It is interesting to notice that, in case the input graph of Min Edge-Bipartization-R3 is already bipartite, the optimum solution is the empty set. Then, there exists a haplotype configuration of the genotyped pedigree that we use to encode the graph that does not contain mutations. The problem to decide if a pedigree can be haplotyped without mutations is polynomially solvable (crf. [114] and others). This does not contrast with the complexity of the Min Edge-Bipartization-R3 problem: in fact also recognizing bipartite graphs can be performed in polynomial time (by a simple visit of the graph).

5.3.2 2-locus-MinEHC and 2-locus-MinMHC are APX-hard

In this section we show that a previous result of Liu *et al.* [84] can be extended to prove the APX-hardness of MinEHC on a 2-locus pedigree (i.e. a pedigree in which the genotypes of the individuals have 2 loci). We also point out that the same

arguments can be used to show the APX-hardness of 2-locus-MinMHC, completing the results presented in the previous section.

We achieve the APX-hardness of 2-locus-MinEHC by observing that the effects of a recombination on a 2-locus pedigree cannot be distinguished from the effects of a mutation on the same point.

Lemma 5.7. *Let H be a haplotype configuration on a 2-locus pedigree P. If it is present a recombination between individual I and individual J starting at locus l, then the recombination can be replaced by a mutation between I and J on locus l.*

Proof. First notice that a recombination starting at the first locus is not interesting: inheriting the other haplotype would have had the same effect. Thus we can assume that all the recombinations start at the second locus. Moreover, individual I has to be heterozygous at locus 2 because otherwise the recombination would not have any effect. Let $h_j = \langle a_1, a_2 \rangle$ be the haplotype inherited by J from its parent I. Since individual I is heterozygous at locus 2 and since there are only two possible alleles (denoted 0 and 1), one haplotype of individual I is $h_i = \langle a_1, 1 - a_2 \rangle$. Clearly a mutation between I and J would replace allele $1 - a_2$ with a_2, having the same effect of the recombination. □

Lemma 5.7 permits to derive the following results.

Corollary 5.8. MinEHC *on a 2-locus pedigree is* APX*-hard.*

Corollary 5.9. MinMHC *on a 2-locus pedigree is* APX*-hard.*

Proof of Corollary 5.8 and Corollary 5.9. Liu *et al.* proved that there exists an L-reduction from Min Edge-Bipartization (called MinUncut) to MinRHC on a 2-locus pedigree (Lemma 9 of [84]). They rely on a pedigree where each individual is either a founder or a son of a founder. All founders are heterozygous at every locus, thus also the converse of Lemma 5.7 holds. In fact it is possible to prove that a mutation at locus l from an individual that is heterozygous at locus l can be replaced by a recombination starting at l. Therefore the same gadget used in [84] can be applied to prove the existence of an L-reduction from Min Edge-Bipartization to MinEHC or to MinMHC. As a consequence, by the APX-hardness of Min Edge-Bipartization [48], we obtain the APX-hardness of MinEHC and of MinMHC on 2-locus pedigrees. □

5.4 A Heuristic Algorithm for MinEHC

The presentation of the heuristic algorithm that we propose is divided in three parts. First, we illustrate a system of linear equations over the field $\mathbb{Z}_2$ that represents the set of haplotype configurations that are consistent with the input genotyped pedigree. This is an extension of the system of linear equations proposed

by Xiao *et al.* [114, 115] for the problem of computing a haplotype configuration without recombinations and mutations (ZRHC). In the extended system that we propose, recombinations and mutations are explicitly modeled as variables of the equations, and the goal of the problem is to find a solution of the system that minimizes the number of non-zero recombination and mutation variables. In the second part, we prove an L-reduction from MinEHC to Nearest Codeword Problem (NCP) by splitting the system in two parts: one part contains only variables needed for the haplotype reconstruction while the other one contains only recombination and mutation variables. The second part of the system is, directly, an instance of NCP. Finally, we present a tailored version of a well-known heuristic algorithm for NCP. Using this version, we can guarantee that a feasible solution for NCP (and hence for MinEHC) is found. The experimental evaluation of the quality of the solutions found by our heuristic approach is deferred to the next section.

5.4.1 A System of Linear Equations for MinEHC

In this part, we first illustrate the linear system over $\mathbb{Z}_2$ proposed in [114, 115] for the ZRHC problem, and then we describe how it can be extended to accommodate recombinations and mutations events. We refer to Section 2.2 for a basic review of the concepts related to vector spaces and matrices over $\mathbb{Z}_2$ that will be used in the rest of this section. We remark that, for improved clarity, we denoted with the symbol + the addition over $\mathbb{Z}_2$ instead of using $\oplus$.

A Linear System for ZRHC [115]

Notice that computing the paternal haplotypes of all individuals is sufficient to fully describe the haplotyped pedigree because the maternal haplotype can be reconstructed from the paternal haplotype and the genotype of the individual. Therefore, we introduce a variable $h_i[l]$ for each individual i and locus l which represents the allele presents at locus l of the paternal haplotype of i. Secondly, we need to represent the grandparental source. Let i be an individual and p one of its parents. Since no recombinations were admitted in the original linear system, the grandparental source is denoted by a single variable $s_{p,i}$. Variable $s_{p,i}$ is equal to 0 if i has inherited from p the paternal haplotype of p, or 1 otherwise. To express concisely the linear equations we need two additional sets of constants: the w- and the d-constants. For each locus l and each individual i, constant $w_i[l]$ is equal to 0 if i is homozygous at locus l, and 1 otherwise. For each locus l and each pair of individuals p and i such that p is a parent of i, constant $d_{p,i}[l]$ is equal to 0 if p is the father of i and it is equal to $w_i[l]$ is p is the mother of i. Finally, since the paternal haplotype (and hence the maternal haplotype) is known at homozygous loci, we set $h_i[l] = g_i[l]$ for every individual i and locus l such that $g_i[l] \neq 2$.

A case-by-case analysis shows that any solution of the following linear system over $\mathbb{Z}_2$ is a zero-recombinant haplotype configuration consistent with the genotyped pedigree (and vice versa), as formalized by the following lemma.

Lemma 5.10 ([115])**.** *Let P_g be a genotyped pedigree. Then a solution of the following linear system represents a haplotype configuration consistent with P_g that does not induce recombinations nor mutations. Moreover, a haplotype configuration consistent with P_g that does not induce recombinations nor mutations is represented by a solution of the linear system.*

The linear system is composed as follows.

For all loci l and individuals i:

$$\begin{cases} h_p[l] + s_{p,i} \cdot w_p[l] = h_i[l] + d_{p,i}[l] & \text{for each parent } p \text{ of } i \\ h_i[l] = g_i[l] & \text{if } g_i[l] \neq 2 \\ w_i[l] = 0 & \text{if } g_i[l] \neq 2 \\ w_i[l] = 1 & \text{if } g_i[l] = 2 \\ d_{p,i}[l] = 0 & \text{if } p \text{ is the father of } i \\ d_{p,i}[l] = w_i[l] & \text{if } p \text{ is the mother of } i \end{cases}$$

Notice that, if the pedigree has n members and the genotypes are defined over a set of m loci, then we have nm h-variables, at most $2n$ s-variables, and at most $2nm$ equations.

A Linear System for MinEHC

We now show how the previous linear system can be modified for representing all the haplotype configurations (admitting recombinations and mutations) consistent with the genotyped pedigree. In other words, our aim is to characterize the solution space of MinEHC as a linear subspace of $\mathbb{Z}_2^{O(nm)}$. In the following we denote with i a generic individual and with p one of its parents.

To accommodate recombinations we introduce a set of δ-variables defined as follows. For each locus l, variable $\delta_{p,i}[l]$ is equal to 1 if a recombination has occurred at locus l between p and i, 0 otherwise. The grandparental source vector of a consistent haplotype configuration can be expressed as a (linear) function of a s-variable and a subset of δ-variables. In particular, by induction on l, it is easy to prove that the grandparental source of i w.r.t. p at locus l, $s_{p,i}[l]$, is equal to $s_{p,i} + \sum_{j=1}^{l} \delta_{p,i}[j]$. (All operations are intended on the $\mathbb{Z}_2$ field.) Denote with $\Delta_{p,i}[l]$ the sum $\sum_{j=1}^{l} \delta_{p,i}[j]$. By replacing $s_{p,i}$ with $(s_{p,i} + \Delta_{p,i}[l])$ in the first equation of the system in Lemma 5.10, we obtain a linear system that represents all the haplotype configurations consistent with the genotyped pedigree and that allow recombination events. The correctness of this claim is a direct consequence of Lemma 5.10 and the previous arguments.

Mutations are point events that replace the allele inherited from the parent with the other allele. Therefore it suffices to add a term in the first equation of the original system in order to model mutation events. We denote this term with $\mu_{p,i}[l]$ and we set $\mu_{p,i}[l] = 1$ if a mutation at locus l between p and i has occurred, and $\mu_{p,i}[l] = 0$ otherwise.

From the previous observations, we derive the following lemma.

Lemma 5.11. *Let P_g be a genotyped pedigree. Then each solution of the system:*

For all loci l and individuals i:

$$\begin{cases} h_p[l] + (s_{p,i} + \Delta_{p,i}[l]) \cdot w_p[l] = h_i[l] + d_{p,i}[l] + \mu_{p,i}[l] & \text{for each parent } p \text{ of } i \\ h_i[l] = g_i[l] & \text{if } g_i[l] \neq 2 \\ w_i[l] = 0 & \text{if } g_i[l] \neq 2 \\ w_i[l] = 1 & \text{if } g_i[l] = 2 \\ d_{p,i}[l] = 0 & \text{if } p \text{ is the father of } i \\ d_{p,i}[l] = w_i[l] & \text{if } p \text{ is the mother of } i \end{cases}$$

represents a haplotype configuration consistent with P_g that admits recombination and mutation events. Conversely, a haplotype configuration consistent with P_g that admits recombination and mutation events is represented by a solution of the linear system.

By construction, a haplotype configuration that induces k variation events is represented by a solution of the linear system in which exactly k δ- and μ-variables are non-zero.

5.4.2 Reducing MinEHC to NCP

In this second part we prove that there exists an L-reduction from MinEHC to NCP. First we formally introduce the Nearest Codeword Problem, then we describe the L-reduction.

The Nearest Codeword Problem arises in the field of coding theory and asks for the codeword y of a binary linear code $\mathcal{C}$ that minimizes the Hamming distance with the received message $\tilde{y}$. More formally, the Nearest Codeword Problem is defined as follows. (We are using a non-standard but completely equivalent formulation that is more suitable for the following reduction.)

Problem 11 (equiv. to MS3 in [6])**.** Nearest Codeword Problem (NCP).
Input: A $r \times n$ matrix H over $\mathbb{Z}_2$, and a column vector $q \in \mathbb{Z}_2^r$.
Output: A column vector $e \in \mathbb{Z}_2^n$ with the minimum number of non-zero entries such that $H \cdot e = q$.

For decoding purposes, the input vector q is computed from the received message $\tilde{y} \in \mathbb{Z}_2^n$ as $q = H \cdot \tilde{y}$. The transmitted codeword y is then recovered by adding e to $\tilde{y}$.

In the following we denote with M^T the transpose of a matrix M and with $\|x\|$ the number of non-zero entries of a vector x. The quantity $\|x\|$ is called weight of x.

Description of the gadget

The basic idea of our reduction is to split the linear system of Lemma 5.11 in two linear systems: one containing only δ- and μ-variables, and the other one containing only h- and s-variables. Notice that all the equations of the linear system but the first can be eliminated by a simple substitution process because, given a genotyped pedigree, $w_i[l]$ and $d_{p,i}[l]$ (and $h_i[l]$ for homozygous loci) assume a constant (predetermined) value.

By simple algebraic manipulations, we can write the linear system as the following matricial equation:

$$A_{h,s} \cdot x_{h,s} + A_{\delta,\mu} \cdot x_{\delta,\mu} = b \tag{5.1}$$

where:

- $x_{h,s}$ is the column vector of the h- and s-variables;
- $x_{\delta,\mu}$ is the column vector of the δ- and μ-variables;
- $A_{h,s}$ is the $n \times m_1$ matrix of the coefficients of the h- and s-variables;
- $A_{\delta,\mu}$ is the $n \times m_2$ matrix of the coefficients of the δ- and μ-variables;
- b is a column vector composed by constant entries.

Denote with k the rank of the matrix $A_{h,s}$, and suppose, w.l.o.g., that the first k rows of $A_{h,s}$ are linearly independent. Now, build the instance of NCP associated to an instance of MinEHC as follows. Let $B := \{v_1, \ldots, v_r \mid v_i \in \mathbb{Z}_2^n\}$ be a basis of the vector space $\ker(A_{h,s}^T) := \{y \in \mathbb{Z}_2^n \mid A_{h,s}^T \cdot y = \mathbf{0}\}$, where $\mathbf{0}$ denotes the all-zero column vector. Collate vectors v_i to form a $r \times n$ matrix V such that the i-th row is equal to v_i^T. Then, the instance I' of NCP associated with an instance $I = (A_{h,s}, A_{\delta,\mu}, x_{h,s}, x_{\delta,\mu}, b)$ of MinEHC is the pair $I' = (H, q)$ with $H := VA_{\delta,\mu}$ and $q := Vb$. Clearly the transformation of I in I' can be computed in polynomial-time via Gaussian elimination (to compute V) and two matrix multiplications (to compute H and q).

The L-reduction

We accomplish the proof that MinEHC reduces to NCP in the following two lemmas: we first show (Lemma 5.12) how to reconstruct in polynomial-time a solution of a MinEHC instance given a solution for the associated NCP instance, and then we show how to compute (in polynomial-time) a solution for an instance I' of NCP associated with an instance I of MinEHC given a solution for I. Both the transformations preserve the solution's costs, hence the reduction is an L-reduction with parameters $\beta = \gamma = 1$.

Lemma 5.12. *Let $I = (A_{h,s}, A_{\delta,\mu}, x_{h,s}, x_{\delta,\mu}, b)$ be an instance of* MinEHC *and $I' = (H, \tilde{y})$ the* NCP *instance associated with I. Then, given a solution e of* NCP *on I', it is possible to compute in polynomial-time a haplotype configuration $(\widehat{x}_{h,s}, \widehat{x}_{\delta,\mu})$ of I that induces $\|e\|$ variation events.*

Proof. Let n be the number of rows of $A_{h,s}$ (or $A_{\delta,\mu}$), m_1 be the number of columns of $A_{h,s}$, and m_2 be the number of columns of $A_{\delta,\mu}$. We want to prove that $x_{\delta,\mu} := e$ is a partial solution of the linear system $A_{h,s}x_{h,s} + A_{\delta,\mu}x_{\delta,\mu} = b$, or, in other words, that the linear system $A_{h,s}x_{h,s} = b + A_{\delta,\mu}e$ is consistent (i.e. that admits at least one solution). For concision, denote with M the matrix $A_{h,s}$ and with t the vector $b + A_{\delta,\mu}e$. A well-known result in linear algebra (see, for example, Sect. 2.3 of [87]) states that the linear system $Mx_{h,s} = t$ is consistent iff the rank of matrix M is equal to the rank of the augmented matrix $(M|t)$ (i.e. the matrix obtained from M by adding one new column equals to the vector t). Since the rank of matrix M is the maximum number of its independent rows, to achieve the consistency of the system $Mx_{h,s} = t$ we have to prove that a subset of rows of M is linearly dependent iff the same subset of rows of $(M|t)$ is linearly dependent. In other words we have to prove that, for each row vector $d \in \mathbb{Z}_2^n$ such that $d \cdot M = \mathbf{0}^T$ we have that $d \cdot (M|t) = \mathbf{0}^T$ and vice versa.

(⇒-part) By construction, vector d^T is precisely an element of the subspace $\ker(A_{h,s}^T)$. Recall that the construction of the instance I' uses a $r \times n$ matrix V whose rows form a basis of $\ker(A_{h,s}^T)$. Therefore there exists a row vector $\alpha_d \in \mathbb{Z}_2^r$ such that $\alpha_d \cdot V = d$. Since $d \cdot M = \mathbf{0}^T$, we have $\alpha_d \cdot V \cdot M = \mathbf{0}^T$. Let us consider the row vector $z := d \cdot (M|t)$. We want to show that $z = \mathbf{0}^T$. Since $(M|t)$ is the augmented matrix of M, the first m_1 elements of z are equal to $d \cdot M$, and hence are zeros. The last element, instead, is equal to $d \cdot t = d \cdot (b + A_{\delta,\mu} \cdot e) = \alpha_d \cdot V \cdot b + \alpha_d \cdot V \cdot A_{\delta,\mu} \cdot e$. Since e is a solution of I', we have that $V \cdot A_{\delta,\mu} \cdot e = V \cdot b$, thus $d \cdot t = \alpha_d \cdot V \cdot b + \alpha_d \cdot V \cdot b = 0$, and hence $d \cdot (M|t) = \mathbf{0}^T$, completing this part of the proof.

(⇐-part) If $d \cdot (M|t) = \mathbf{0}^T$, we have to prove that $d \cdot M = \mathbf{0}^T$. As noted before, the first m_1 elements of vector $d \cdot (M|t)$ are equal to vector $d \cdot M$, and since $d \cdot (M|t) = \mathbf{0}^T$, we have $d \cdot M = \mathbf{0}^T$, completing the proof.

As a consequence, the rank of M is equal to the rank of $(M|t)$, implying that there exists a vector $\widehat{x}_{h,s}$ such that $A_{h,s}\widehat{x}_{h,s} + A_{\delta,\mu}e = b$. In other words, the pair of vectors $(\widehat{x}_{h,s}, \widehat{x}_{\delta,\mu})$ with $\widehat{x}_{\delta,\mu} = e$ is a consistent haplotype configuration, and induces exactly $\|e\|$ variation events. □

Now we prove that a solution of the NCP instance I' associated with a MinEHC instance I can be computed in polynomial-time starting from a solution of I and preserving the cost.

Lemma 5.13. *Let $S = (\widehat{x}_{h,s}, \widehat{x}_{\delta,\mu})$ be a solution of* MinEHC *on the instance $I = (A_{h,s}, A_{\delta,\mu}, x_{h,s}, x_{\delta,\mu}, b)$ and let $I' = (H, q)$ the* NCP *instance associated with I. Then, vector $e := \widehat{x}_{\delta,\mu}$ is a solution of* NCP *on I'.*

Proof. Since S is a solution for I, we have $A_{h,s} \cdot \widehat{x}_{h,s} + A_{\delta,\mu} \cdot \widehat{x}_{\delta,\mu} = b$. Recall that the construction of the instance I' uses a matrix V whose rows form a basis of $\ker(A_{h,s}^T)$. Multiplying each side of the previous equation by V, we obtain $V \cdot A_{h,s} \cdot \widehat{x}_{h,s} + V \cdot A_{\delta,\mu} \cdot \widehat{x}_{\delta,\mu} = V \cdot b$. By construction each row v_i^T of V, is an element of $\ker(A_{h,s}^T)$, thus $A_{h,s}^T \cdot v_i = (v_i^T \cdot A_{h,s})^T = \mathbf{0}$. Therefore $V \cdot A_{h,s}$ is a all-zero matrix and, as a consequence, $V \cdot A_{h,s} \cdot \widehat{x}_{h,s} + V \cdot A_{\delta,\mu} \cdot \widehat{x}_{\delta,\mu} = V \cdot A_{\delta,\mu} \cdot \widehat{x}_{\delta,\mu} = V \cdot b$. Since $H := VA_{\delta,\mu}$ and $q := V \cdot b$, we can conclude that $e := \widehat{x}_{\delta,\mu}$ is a solution of NCP on the instance I'. Clearly, the number of variation events induced by the haplotype configuration S is equal to $\|\widehat{x}_{\delta,\mu}\| = \|e\|$. □

The following corollary easily derives from Lemma 5.12 and Lemma 5.13.

Corollary 5.14. MinEHC *is L-reducible to* NCP *with parameters $\beta = \gamma = 1$.*

An immediate (positive) result is that MinEHC is $O(l/\log l)$-approximable (where l is the number of δ- and μ-variables), since (i) there exists a polynomial-time $O(l/\log l)$-approximation algorithm for NCP [4], and (ii) the L-reduction preserves the solutions' costs. This result, however, is mainly of theoretical interest because the approximation bound is too much large. Indeed, on real data, only a small number of genetic events are expected to occur, much smaller than the number l of all possible events. Unfortunately, no (significantly) better approximation algorithms are known and, moreover, if $\mathsf{P} \neq \mathsf{NP}$ the NCP problem cannot be approximated within any constant factor (i.e. NCP $\notin \mathsf{APX}$) [5].

5.4.3 The Heuristic Algorithm

In this section, we present an efficient heuristic algorithm that solves the MinEHC problem. In addition, this heuristic can be also used to solve the MinRHC and the MinMHC problem by restricting the kind of variation events that are allowed.

The algorithm is based on the L-reduction from MinEHC to NCP that has been presented in the previous section. As discussed above, there does not exist algorithms that can guarantee a good (i.e. constant) approximation ratio unless $\mathsf{P} = \mathsf{NP}$. Nevertheless, an effective and efficient heuristic is known since the work of Gallager in 1963 [44]: the *sum-product* (SP) algorithm (independently proposed in a different field as *belief-propagation* algorithm [95]). We will briefly and informally explains how the SP algorithm is used as a decoder of a linear code. Suppose that a source want to send a k-bit message x to a destination. To do that, it encodes x in a codeword y computed as $y = x \cdot G$ for a suitable $k \times n$ matrix G. The destination receives a different message $\tilde{y}$ because the channel added some errors, thus $\tilde{y} = y+e$. The aim at the destination side is to recover the original codeword y (and hence the original message x) from $\tilde{y}$. From matrix G (called generator matrix), it is possible to compute a check matrix H such that $H \cdot y = \mathbf{0}$ iff y is a codeword of the code (i.e. if there exists x such that $y = x \cdot G$). Since $y = \tilde{y} + e$, it suffices to reconstruct the error vector e to recover the original codeword y. The SP algorithm, in this context, computes for each bit $e[i]$ an approximation of the likelihood that $e[i]$ is 1 based on the vector $q = H \cdot \tilde{y}$ and on the matrix H. The check matrix H represents a set of (parity) constraints, while vector q, called syndrome, represents the constraints that are not satisfied by the received message $\tilde{y}$.

Our idea is to consider the variation events (recombinations and mutations) as the "errors" that we have to reconstruct and, once the "errors" (variation events) have been determined, it is easy to reconstruct the haplotyped pedigree (by the Gauss elimination algorithm). The L-reduction of Corollary 5.14 formalizes this idea: the check matrix H and the syndrome q are obtained from the genotyped pedigree (represented by the matrices $A_{h,s}$ and $A_{\delta,\mu}$) as illustrated in the previous section. The likelihoods computed by SP algorithm on this instance represents the likelihoods that each δ- or μ-variable is equal to 1. In other words, for each possible variation event, it computes the likelihood that the event has occurred on the pedigree.

The heuristic approach that we propose iteratively adds the most likely variation event (as computed by the SP algorithm) to a set E of *imputed* variation events until a haplotype configuration that induces exactly the imputed events can be found. Given a set of variation events E, the reconstruction of the haplotype configuration that induces E can be performed efficiently. Indeed it suffices to solve the linear system of Lemma 5.11 with the δ- and μ-variables assigned to 1 if the corresponding event (the mutation or the recombination they represent) belongs to E, or 0 otherwise.

The details of the approach are given in Algorithm 6. In particular, lines 1–2, compute the NCP instance $I' = (H, q)$ associated with the genotyped pedigree P_g. Initially (line 3), no variation events are imputed, thus $E = \varnothing$ and N contains all the variables that represent a variation event (δ- and μ-variables). Since the

Algorithm 6: The heuristic algorithm for MinEHC

Data: A genotyped pedigree P_g.
Result: A haplotype configuration consistent with P_g.

1 Let $I = (A_{h,s}, A_{\delta,\mu}, x_{h,s}, x_{\delta,\mu}, b)$ be system of Lemma 5.11 (in the form of Eq. 5.1) associated with P_g;
2 Computes the NCP instance $I' = (H, q)$ associated with I;
3 Let $N = \{e \mid e \text{ is a } \delta\text{- or } \mu\text{-variable}\}$ and $E = \varnothing$;
/* Each column of H is associated with a variable of the set N since $H = V \cdot A_{\delta,\mu}$ for some matrix V. */
4 **while** $q \neq \boldsymbol{0}$ **do**
5 $L \leftarrow$ the likelihood of each variable in N computed by the SP algorithm on H and q;
6 Let e^* be a variable of N s.t. $L[e^*]$ is maximum;
7 **foreach** *row r such that* $H[r, e^*] = 1$ **do**
8 Change the value of $q[r]$ to $1 - q[r]$;
9 **end**
10 Remove the column associated with e^* from H;
11 Move e^* from N to E;
12 **end**
13 Set $x_{\delta,\mu}[e] = 1$ if $e \in E$, $x_{\delta,\mu}[e] = 0$ otherwise;
14 Solve the system $A_{h,s} \cdot x_{h,s} = b + A_{\delta,\mu} \cdot x_{\delta,\mu}$ in the variables $x_{h,s}$;
15 **return** $(x_{h,s}, x_{\delta,\mu})$;

check matrix H is computed as $H = V \cdot A_{\delta,\mu}$ for some matrix V, matrix H has the same number of columns as $A_{\delta,\mu}$ which, in turn, are associated to a δ- or μ-variable. Therefore, we associate each column i of H with the δ- or μ-variable that is associated with the i-th column of $A_{\delta,\mu}$. By a small abuse of notation, we identify each column i of H with the associated variable.

The haplotype configuration is computed in two steps: first (lines 4–12) the set of variation events E that permits to reconstruct the haplotype configuration is computed, and then (lines 13–14), the haplotype configuration is recovered using the imputed events E.

The first part iteratively computes the set of variation events. Using the SP algorithm, it computes an event e^* that most likely is induced in a haplotype configuration consistent with the pedigree (lines 5–6). If more than one event have the maximum likelihood, one of them is chosen at random. Once e^* has been determined, the corresponding δ- or μ-variable is set to 1, thus we need to change the r-th element of the syndrome q for each row r of H where e^* appears (i.e. $H[r_i, e^*] = 1$) (lines 7–9). Finally, since we update the syndrome q, the column

of H associated with the event e^* can be removed, and the event can be moved from the set of possible events N to the set of imputed events E. By construction, the L-reduction guarantees that $\|q\|$ (i.e. the weight of q) is equal to the number of variation events needed to recover the haplotype configuration. As a consequence, no more variation events are needed if $q = \mathbf{0}$, and the cycle of the first part ends.

The second part (lines 13–14) reconstructs the haplotype configuration consistent with the input genotyped pedigree by solving the linear system of Lemma 5.11 using, as a partial solution, the set E of imputed events.

To improve clarity of the presentation we omitted to describe one step in Algorithm 6. To guarantee that the algorithm finds a haplotype configuration consistent with the genotyped pedigree, we have to check, at each iteration, if matrix H and syndrome q imply the presence or the absence of some variation events in the haplotype configuration. We call such events *determined events*. Determined events can be easily recognized by the Gauss elimination algorithm. Indeed, they corresponds to the variables that do not depend to free variables in the solution of the linear system $H \cdot x = s$. (Recall that the NCP problem Therefore by applying the Gauss elimination algorithm on $(H|s)$ and removing the determined events at the beginning of each iteration, we can guarantee that the algorithm finds a haplotype configuration consistent with the genotyped pedigree (if such a configuration exists).

One important remark is in order. The sum-product algorithm (used in line 5) receives as an additional input, for each variation event e, the prior probability that e has occurred. Although we did not deeply investigate such possibility, this characteristic could be extremely useful to model *recombination hotspots* (by increasing the prior probability of recombination events in regions where recombinations occurs more frequently), to differentiate the rate of recombinations and mutations (i.e. by increasing the prior probability of a recombination event with respect to a mutation event), and/or to model additional knowledge about the input genotypes. This peculiar characteristic could mix the pure combinatorial formulation of the problem presented here with some elements of pure statistical-based methods.

The time complexity of the heuristic is determined by several parameters. Let n be the number of individuals in the genotyped pedigree and m the length of their genotype. Matrix $A_{h,s}$ has at most $2nm$ rows (one for each locus of each haplotype) and at most $nm + 2n$ columns (one for each locus of each paternal haplotype plus 2 s-variables for each individual). Similarly, matrix $A_{\delta,\mu}$ has at most $2nm$ rows (actually the same number of rows of $A_{h,s}$) and at most $4nm$ columns (one for each possible variation events, $2nm$ for the mutations and $2nm$ for the recombinations). The NCP instance I' is calculated by the Gauss elimination algorithm on $A_{h,s}^T$ (that requires $O(n^3m^3)$ time) and two matrix multiplications (that require $O(n^3m^3)$ time). The check-matrix H has $O(nm)$ rows and at most $4nm$ columns (one for each

variation event). Therefore the reduction from the pedigree to the NCP instance is computed in $O(n^3m^3)$ time. The time required by each iteration is bounded by $O(n^3m^3)$ since the check of the existence of predetermined events (by Gaussian elimination) requires $O(n^3m^3)$ time, the SP algorithm requires linear-time in the number of one-entries of matrix H, and the other operations that update matrix H and vector q can be accomplished in $O(n^2m^2)$ time. The resolution of the final linear system can be performed in $O(n^3m^3)$ time by the Gauss elimination algorithm. Let k be the number of events that are imputed, then the overall time complexity of the heuristic is $O(kn^3m^3)$.

5.5 Experimental Results

Our approach has been experimentally analyzed under several simulated scenarios to evaluate the quality of the results that produce and its running time. We mainly judge the quality of the results in term of number of events that are induced in the computed haplotype configuration, since our primary aim is the design of a heuristic that solves the MinEHC problem. We chose to not compare our results with the results obtained by statistical-based methods because they are based on different genetic model thus, as pointed out in [45, 102], the ML solution may be suboptimal in the MinEHC sense (i.e. it may induce a number of recombinants greater than the minimum). Moreover, even the minimum size of an instance we considered is not practical for many statistical-based methods.

The experimental analysis is divided in 2 parts. In the first part we applied our heuristic to various randomly generated MinEHC instances and we evaluated performances and accuracy (in term of number of events and number of haplotype loci not correctly reconstructed). In the second part, we adapted our approach in order to allow only recombination events and we compared it with PedPhase v2.1 [78] (see also Section 3.3.3), an ILP-based combinatorial approach that solves exactly the MinRHC problem. The primary aim of this part is to compare the number of recombinant induced by a heuristic solution and the minimum number of recombinant as computed by the exact method. Moreover we also compared the running times required by the two methods.

In synthesis, the analysis has revealed that our approach is able to handle moderately large instances (40 individuals and 100 loci) in at most few minutes on a standard PC. Even if it cannot guarantee a worst-case approximation factor, our heuristic was able to determine extremely good solutions in almost all cases. Moreover the preliminary comparison with PedPhase v2.1 has revealed that in 599 cases over 600, our approach was able to recover an optimal solution 4–5 times faster than PedPhase.

In the following sections we present the two parts of the experimental analysis.

5.5.1 Solving MinEHC

Our experimentation involved randomly generated instances (on both the structure of the pedigree and the set of genotypes) under several choices of 4 parameters: population size (n), genotype length (m), recombination probability (θ_r), and mutation probability (μ_r). For each choice of the parameters, 6 different random pedigree graphs on n individuals have been generated, and for each pedigree graph 5 different haplotype configurations over a set of m loci has been generated (a total of 30 haplotype configurations for each choice of the parameters). During the random generation of the haplotypes, we applied a variation event at each locus with probability θ_r for recombinations and μ_r for mutations. Then the genotypes has been obtained as conflation of the generated haplotypes. Finally we removed from the haplotype configuration all those events whose removal did not changed any genotype. We changed the generated haplotypes according with the removal of those events. An example of such removed events is a mutation between a founder and its only child in a locus where the founder is heterozygous: clearly in this case the mutation cannot be distinguished from the case in which the alleles of the paternal and the maternal haplotype are exchanged. (It is the same basic principle that we employed in the L-reduction from Min Edge-Bipartization-R3 to MinMHC, see, for example, Property 5.5.)

We generated two kinds of pedigrees: tree pedigrees and pedigrees with mating loops. In both cases, we modeled the presence of nuclear families with several children and the presence of some individuals with more than one mate.

For each instance, we ran our heuristic 10 times and we picked the solution with the minimum number of induced events. At each execution we changed the prior probability of the sum-product algorithm linearly in a range $[p_{min}, p_{max}]$. We chose (arbitrarily) $p_{min} = 0.001$ and $p_{max} = 0.125$ for recombinations and $p_{min} = 0.00075$ and $p_{max} = 0.100$ for mutations. In this way, we empirically modelled an higher recombination rate compared with the mutation rate, as observed in several studies in humans [88, 106, 107, 121]. We considered as running time of the heuristic on each instance the sum of the running time of all the 10 executions.

We evaluated the quality of the results with 3 measures: *precision*, *phase error*, and *approximation ratio*. Precision is defined as the ratio between the number of events that have been correctly predicted and the number of events that have been predicted. An event is considered correctly predicted if it is contained in the generated haplotype configuration and in the haplotype configuration that has been reconstructed by our heuristic approach. Phase error is the ratio between the number of heterozygous loci whose phase has not been correctly predicted and the total number of heterozygous loci. The phase of locus l in individual c is considered correctly predicted if the allele of the paternal haplotype at locus l in the generated haplotype configuration coincides with the allele of the paternal

Table 5.1: Summary of the results obtained increasing the size (n) of the population. The other parameters are: $m = 40$, $\theta_r = 0.02$, $\mu_r = 0.004$.

	Tree pedigrees			General pedigrees			
Population size $n =$	40	60	100	40	60	100	Mean
Avg. no. of heterozygous loci	813	1206	2001	796	1194	1988	1333
Avg. no. of generated events	22.0	30.4	55.2	25.5	35.8	63.6	38.7
Avg. no. of predicted events	21.3	29.5	53.2	24.5	34.9	61.8	37.5
Avg. precision	0.787	0.744	0.768	0.778	0.812	0.809	0.783
Avg. phase error	0.027	0.029	0.028	0.022	0.024	0.024	0.026
Avg. approximation ratio	0.968	0.975	0.965	0.963	0.975	0.972	0.970
Avg. time (s)	36	73	265	62	118	460	169

haplotype at locus l in the haplotype configuration that has been computed by our heuristic. The approximation ratio, instead, is the ratio between the number of events in our solution and the number of events in the generated haplotype configuration. Approximation ratio can be less than 1.0 because the generated haplotype configuration could be not-optimal. In these cases, our approach has found a better (in term of number of variation events) haplotype configuration than the generated one. Finally we also evaluated the total running time required by the heuristic.

We chose a base set of values for the parameters n, m, θ_r, and μ_r and we conducted three series of tests. In each series, we modified the value of one of these parameters: in the first we modified the population size n, in the second we modified the genotype length m, and in the third we modified the two event probabilities θ_r and μ_r. The base values were: population size $n = 40$, genotype length $m = 40$, recombination probability $\theta_r = 0.02$, and mutation probability $\mu_r = 0.004$.

In the first series of tests, we varied the population size n and we analysed the cases $n = 40$, $n = 60$, and $n = 100$ on both tree pedigrees and "general" pedigrees (i.e. pedigree with mating loops). Table 5.1 summarizes the results. In particular, each column contains the results of each case, with the last one presenting the mean computed on all the case. In the first case, we report a characteristic of the instance, the average number of heterozygous loci in the 30 instances that have been considered. In the second row, we report the average number of events contained in the generated haplotype configuration, while the third row reports the average number of events that are contained in our solution. As for the first row (and the subsequent ones), the average is computed on the 30 instances that have been generated. The three rows that follow report the average precision, the average phase error, and the average approximation ratio as defined above. Finally, we reported the running time of the heuristic (in seconds). In all cases, we obtained an average approximation ratio always smaller than 1.0. This means that the heuristic

Table 5.2: Summary of the results obtained increasing the length (m) of the genotypes. The other parameters are: $n = 40$, $\theta_r = 0.02$, $\mu_r = 0.004$.

	Tree pedigrees			General pedigrees			
Genotype length $m =$	40	60	100	40	60	100	Mean
Avg. no. of heterozygous loci	806	1207	2009	800	1196	2032	1342
Avg. no. of generated events	24.0	34.7	53.2	26.4	38.0	61.0	39.6
Avg. no. of predicted events	23.1	33.0	51.2	25.7	36.9	59.6	38.3
Avg. precision	0.732	0.683	0.750	0.797	0.819	0.804	0.764
Avg. phase error	0.035	0.057	0.042	0.026	0.026	0.044	0.039
Avg. approximation ratio	0.964	0.956	0.964	0.975	0.972	0.976	0.968
Avg. time (s)	41	95	247	76	148	485	182

has been able to find a haplotype configuration that induces less variation events than the generated one. Clearly this fact does not imply that the heuristic was able to find the optimal solution. However this result increases our confidence in the soundness of the approach. The values of the quality measures (precision, phase error and approximation ratio) appear to be similar in all cases, with a small difference between tree pedigrees and general pedigrees in favor of the last ones. We think that this fact can be easily explained since the presence of mating loops increases the number of constraints that the haplotype configuration must satisfy. Our criterion for considering correct an event was quite stringent since sometimes is not possible to univocally determine the position of a given event based on the genotype data. In fact, precision is roughly between 75% and 80%. Instead, phase error is usually small (2%–3%). If we interpret these values together, we can conclude that, since phase error is small, the incorrectly predicted events were located "near" (in term of genotype position and kinship) to the generated ones, otherwise the error rate would be higher. In all cases, the algorithm never required more than 6 minutes. Notice that the running time for tree pedigrees is considerably smaller than the running time for general pedigrees.

In the second series of test, we varied the genotype length m and we considered the following cases: $m = 40$, $m = 60$, and $m = 100$. Table 5.2 summarizes the results that we obtained using the same structure of the previous table. The results are similar to the ones obtained in the previous series, confirming the considerations that we made.

In the third series of tests, we varied the probability of recombinations and mutations. The following pairs of values (θ_r, μ_r) have been considered: $(0.02, 0.004)$, $(0.04, 0.01)$, and $(0.10, 0.02)$. The results are summarized in Table 5.3. In this case, the quality of the results decreases with the increase of the number of generated events. The worst results are obtained when recombination and mutation probabilities are maximum ($\theta_r = 0.1, \mu_r = 0.02$). However we have to make an important

Table 5.3: Summary of the results obtained increasing the mutation and recombination rates (θ_r and μ_r). The other parameters are: $n = 40$, $m = 40$.

	Tree pedigrees			General pedigrees			
Recombination prob. θ_r =	0.02	0.04	0.10	0.02	0.04	0.10	
Mutation probability μ_r =	0.004	0.01	0.02	0.004	0.01	0.02	Mean
Avg. no. of heterozygous loci	798	807	799	804	804	796	801
Avg. no. of generated events	24.5	48.8	111.5	24.9	48.9	121.8	63.4
Avg. no. of predicted events	23.8	45.7	94.8	24.0	45.8	105.3	56.6
Avg. precision	0.756	0.707	0.556	0.784	0.746	0.555	0.684
Avg. phase error	0.035	0.061	0.114	0.020	0.053	0.099	0.064
Avg. approximation ratio	0.973	0.937	0.848	0.963	0.939	0.866	0.921
Avg. time (s)	45	74	164	63	86	248	113

observation: the heuristic reconstructed a solution with much less events than the generated haplotype configuration. This fact implies that the "real" haplotype configuration deviates significantly from the parsimony principle that is assumed in the MinEHC formulation. Our heuristic tries to reconstruct a haplotype configuration with the minimum number of events, and computes, in this case, a haplotype configuration that better suits the assumptions. Moreover, we remark that this case is a limit case, since such an high number of variation events should not occur in the moderately small genotyped pedigree that we considered (40 individuals and 40 loci).

Finally, we point out that, even if the average approximation factor is always less than 1.0, in 5 cases (over 540) the heuristic computed a solution which contains a few more events than the generated haplotype configuration.

5.5.2 Solving MinRHC

In order to empirically assess the ability of computing the haplotype configuration that induces the minimum number of events, we compared our approach with a popular combinatorial exact method: PedPhase v2.1 [78]. Since PedPhase solves the MinRHC problem (with missing data) but not the MinEHC problem, we modified our approach by removing the μ-variables from the linear system of Lemma 5.11 and Eq. 5.1.

We considered 600 genotyped pedigrees with different genotype length and number of recombinations. In particular, the genotyped pedigrees have been randomly generated from a real pedigree graph with 52 individuals [81, 110]. The first two columns of Table 5.4 reports the values of the parameters that have been used in this part of the experimental analysis. For each choice of parameters' values (a row in the table), 100 random genotyped pedigrees have been generated. Table 5.4

Table 5.4: Comparison with PedPhase.

Genotype length	Number of generated recombinations	Avg. no. of computed recombinations		Avg. time (s)	
		PedPhase	Heuristic	PedPhase	Heuristic
50	10	8.87	8.87	6.41	1.21
60	10	8.82	8.82	7.69	1.47
80	15	13.41	13.41	16.46	3.31
80	20	18.02	18.04	17.18	4.2
90	20	18.07	18.07	23.61	5.22
95	15	13.48	13.48	22.44	4.61
Mean		13.45	13.45	15.63	3.34

reports in the third and fourth column the average number of recombinations that are induced by a PedPhase's solution and by a solution computed by our approach. The two columns coincides, but a more in-depth analysis has revealed that for one instance (over 600), our approach computed a sub-optimal haplotype configuration containing 2 m that induces 22 recombinations, while PedPhase computed a haplotype configuration that induces 20 recombinations. The average running time of the two methods on each instance is reported in the last two columns of the table. Our approach appears considerably faster than PedPhase and, in particular, our method has never required more than 8.83s while PedPhase in one case has required 151.49s to compute the solution (data not shown).

In conclusion, we have shown that our heuristic is both accurate and efficient. We mainly evaluated accuracy by comparing the number of the predicted events with the number of generated events or with the number of recombinations computed by an exact method. In almost all cases, our approach has been able to reconstruct a solution with fewer events than the generated haplotype configuration. The algorithm is reasonably fast and seems appropriate even for moderately large pedigrees (in some our tests it was able to handle a tree pedigree with 50 individuals on 1000 loci in approximately 2.5 hours of computation on a standard PC).

Part II

Alignment of Spliced Sequences

6 Spliced Alignments

This second part of the thesis is devoted to the investigation of a fundamental problem involving transcripts data which is the inference of the structure of a gene and its variants as a consequence of the *splicing* mechanism. This mechanism, that takes place during the synthesis of a protein, consists in the excision of the intronic (non-coding) regions of the premature mRNA (pre-mRNA) while the exonic regions are then reconnected to re-form a single continuous molecule, the mature mRNA, also called *transcript isoform* or full transcript. A complex regulatory system mediates the splicing process which, under different conditions, may produce *alternative* transcript isoforms starting from a single pre-mRNA molecule. Due to the unpredictable combination of exonic regions in the mature-mRNA, the reconstruction of isoforms and the determination of the gene structure underlying the various alternative transcript isoforms cannot be completely performed in vitro. Even the simpler task of validating in vitro putative transcript isoforms implies high experimental costs.

On the other side, in these last years, the amount of partial transcripts available in public databases has grown exponentially, mainly due to the increasing availability of EST (Expressed Sequence Tag) data. An EST is just a short fragment of cDNA (i.e. complementary DNA), produced in vitro by making a mirror copy of a mature-mRNA. Though transcripts derive from the gene structure, they do not provide directly the information on it and hence computational methods must be used to predict alternative splicing events from the analysis of such sequences.

These computational methods are based on the use of alignment tools either for the comparison of multiple ESTs from the same gene or for the single comparison of an EST sequence against the genomic sequence (or gene).

Despite the amount of software tools for transcript analysis available today, the computational investigation of the gene structure from transcript data still lacks of a complete understanding of the combinatorial data structures and underlying problems that could lead to an efficient solution of the general problem.

Motivated by the above fact, the main aim of this work of the thesis is the formalization of basic simple combinatorial problems underlying the task of gene structure prediction from transcript data analysis.

More precisely we formulate two combinatorial problems whose solutions can effectively provide a basic approach for the gene structure prediction problem. First we consider the problem of finding the possible alignments of a transcript sequence

against a reference genomic sequence. A typical alignment of an EST against a genomic sequence consists in the alternation of perfect matching regions and smaller regions where sequencing errors are located. Therefore we propose the *maximal embedding problem*, that asks for the set of particular sequences, called maximal embeddings, of common substrings between the transcript sequence and the genomic sequence. We achieve an efficient algorithmic solution for the problem by providing a compact graph-representation of the solutions that can be easily constructed from the two sequences and from which maximal embeddings can be easily enumerated. The study of the *maximal embedding problem*, along with the derivation of the algorithmic solution is the subject of the present chapter.

The second problem that we consider is the choice of a single alignment of a transcript when several different alignments exist. Instead of relying on empirical criteria or on manual curation, we formulate the *minimum factorization agreement* (MFA) problem that asks for the simplest (minimal) gene structure that "agrees" with at least one alignment for each transcript of a set of related transcripts. Even though the problem is computationally hard, a simple but effective preprocessing stage permits to greatly reduce the instance size on real data and, thereafter, an exact solution of the problem becomes feasible.

The definition and the study of the minimum factorization problem is deferred to the next chapter, where an experimental analysis of our approach on real data is also presented.

For basic notions about sequence alignment and comparison the interested reader can refer to [52].

6.1 Introduction

Alternative Splicing (AS), i.e. the production of alternative transcripts from the same gene, is the main mechanism responsible for the expansion of the transcriptome (the set of transcripts generated by the genome of one organism) in eukaryotes and it is also involved in the onset of several diseases [21].

A great extent of work has been performed to solve two basic problems on AS: characterizing the intron-exon structure of a gene and finding the set of different transcript isoforms that are produced from the same gene. Computational approaches to these crucial problems have been proven both effective and economical, while software tools implementing them are made available [25, 40, 60, 66, 75, 119]. The basic ingredients of most computational approaches are represented by the alignments of several short fragments, called ESTs, of a given transcript against the reference genomic sequence [19, 43, 51, 64, 118, 120]. Since ESTs are fragments of transcript isoforms, the alignment of such sequences on the genome must consider the effects of the splicing process, which has removed the intronic regions and has

joined the exonic parts. Thus, when considering ESTs, a particular kind of alignment arises: the spliced alignment. The spliced alignment problem requires to compute, given a spliced sequence (such as a transcript) S and a reference sequence T, a set $\{f_1, \ldots, f_k\}$ of strings such that $S = f_1 \cdots f_k$ and $T = pf_1i_1f_2i_2 \cdots f_{k-1}i_{k-1}f_ks$. Clearly, in the biological context, the spliced sequence S is an EST or a mRNA sequence, while the reference sequence T is the genomic sequence of the locus of the gene where the EST comes from.

Unfortunately, perfect matching between factors of the EST and factors of the genomic sequence can hardly be achieved on real data. In fact, EST data contain mismatches (deletion and insertions) against the genome because of sequencing errors and polymorphism. A more comprehensive formulation of the spliced alignment problem is then needed. In this formulation, given a spliced sequence S and a reference sequence T, we want to find two sets $F_S = \{f_1, \ldots, f_k\}$ and $F_T = \{f'_1, \ldots, f'_k\}$ of strings such that $S = f_1 \cdots f_k$, $T = pf'_1i_1 \cdots i_{k-1}f'_ks$, and for each i, the edit distance between f_i and f'_i is at most a small constant e (or an appropriate error function). The sequence of pairs (f_i, f'_i) is called *spliced alignment* of S on T, each factor f_i is called spliced *sequence factor* (or EST factor), and each f'_i is called *genomic factor*.

Such formulation, or a similar one, is commonly (and in some cases implicitly) used by several combinatorial methods for the alignment of spliced sequences (such as [111, 113]) since it captures the characteristics of the classical sequencing technologies and of the vast majority of biological sequences stored in public databases.

However, allowing an approximate alignment between factors makes the spliced alignment problem computationally harder, mainly when EST data and the genomic sequence are large. Moreover, more than one spliced alignment can exist for the same input data and thus integrating biological meaningfulness is a primary goal.

On the other side, finding efficient solutions to the spliced alignment problem, though crucial, is only a preliminary step of the two main problems on AS, that is predicting the intron-exon structure of a gene and all its full length isoforms. Indeed, the recent literature on the topic (see [12] for a survey) points out the need of designing algorithmic methods that allow to combine spliced alignments of multiple ESTs, as each EST sequence provides the information on a partial region of the whole gene. In this work we give a contribution in this direction by proposing a quite fast algorithmic approach to the spliced alignment problem that produces several alternative spliced alignments. In the next chapter we will study the subsequent problem of combining the alternative spliced alignments of a set of transcripts in order to predict the gene structure.

To develop an efficient algorithm for the spliced alignment problem, we exploit a fundamental property of its formulation: the edit distance between each pair of corresponding factors is small. Thus, there should be some common substrings

between the EST factors and the genomic factors whose total length is linear in the length of the factors and the error rate. For example, a typical (complete) exon is approximately 50bp long and a low-quality sequence has a typical error rate of 6%. On this case, in the worst scenario we can expect that the alignment of the expressed and genomic factors breaks up each factor in four pairs of 12bp substrings that match perfectly. Clearly, if the sequence of perfect matching pairs is known, it is quite easy to reconstruct a possible alignment of the factors. Based of these observations, we propose the problem of finding a particular common subsequence of a generic pattern P and a generic text T, that we call *embedding*, consisting of common substrings, or factors, between P and T from which a spliced alignment of the sequence P on T is recovered. We show that more than one embedding of P in T can be possible. A key result of this work is that we can compute efficiently each embedding from a graph having vertex set V consisting of all possible *maximal pairings* of P and T. A maximal pairing of P and T generalizes the notion of maximal pair of a sequence [52] and it provides the occurrence on P and T of a maximal common substring of P and T. The vertex set V can be computed in time linear in the size of P, T and the size k of the output, that is $k = |V|$. Edges of the graph are computed in time at most $O(k^2)$, leading to a very fast algorithm to compute spliced alignments of P on T.

The chapter is organized as follows. In Section 6.2 the problem of finding all the embeddings of a pattern in a text is formalized. A graph representation of the solutions (called Maximal Embedding Graph) is introduced in Section 6.3. The problem of finding all the embeddings of a pattern in a text is solved in Section 6.4 via a second representation of the solutions called Compact Maximal Embedding Graph whose basic properties and relationships with the Maximal Embedding Graph are presented in Section 6.4.1. The Compact Maximal Embedding Graph achieves a two-fold objective: (i) it is possible to efficiently reconstruct the Maximal Embedding Graph from it (as shown in Section 6.4.2 and 6.4.3), and (ii) it can be efficiently constructed (as shown in Section 6.4.4). In Section 6.5 we discuss an efficient algorithm that, given a Compact Maximal Embedding Graph, reconstructs a set of possible spliced alignments of a expressed sequence against a genomic sequence.

The experimental analysis of the algorithm is deferred to the next chapter, where the problem of determining a gene structure that agrees with a spliced alignment of each EST sequence is addressed and solved.

6.2 The Maximal Embedding Problem

According to the traditional notations, given a string $S = s_1 s_2 \cdots s_q$, we denote with $S[i, j]$ the substring $s_i s_{i+1} \cdots s_j$ and with $|S|$ its length.

The fundamental concept of our approach is represented by the concept of *pairing* (or *common pair* in [52]) of two strings. Given two strings, namely a pattern P and a text T, a *pairing* of P and T is a triplet (p, t, l) such that $P[p, p+l-1] = T[t, t+l-1]$. In other words, a pairing (p, t, l) represents a common substring x (or *factor*) of P and T of length l occurring in position p and t on P and T respectively. We call p and t the *starting position* on P and T respectively, $p+l$ and $t+l$ the *ending position* on P and T respectively, while l is the *length* of the pairing. On the remainder, for brevity we will omit the specification of the two strings P and T which a pairing refers to.

Let f be the common factor represented by a pairing $v = (p, t, l)$. Since every substring of f is a common factor too, for each δ_1 and δ_2 such that $0 < \delta_1 < l$ and $\delta_1 \leq \delta_2 \leq l - \delta_1$, the triplet $v' = (p + \delta_1, t + \delta_1, l - \delta_2)$ is a pairing.

This fact leads to the order relation $\preceq$ among pairings. Let $v_1 = (p_1, t_1, l_1)$ and $v_2 = (p_2, t_2, l_2)$ be two pairings, then $v_1 \preceq v_2$ if and only if $p_2 \leq p_1 < p_1+l_1 \leq p_2+l_2$ and $p_1 - p_2 = t_1 - t_2$. Then we say that v_1 is a *sub-pairing* of v_2, or v_2 *contains* v_1. Moreover, we say that v_1 is a *prefix-pairing* (*suffix-pairing*, resp.) of v_2 iff v_1 is contained in v_2 and the starting positions (the ending positions, resp.) of v_1 and v_2 on P and T are equal.

Based on the order relation $\preceq$ we can define the concepts related to maximality of pairings. A pairing v is *left-maximal* if and only if it is not a suffix-pairing of a distinct pairing v'. On the other hand, a pairing v is *right-maximal* if and only if it is not a prefix-pairing of a distinct pairing v'. Finally, a pairing is *maximal* if it is both left-maximal and right-maximal. Notice that an alternative definition of left-maximality and right-maximality is possible. Indeed, a pairing (p, t, l) is left-maximal (right-maximal) if the character at position $p - 1$ ($p + l$, resp.) on P is different from the character at position $t - 1$ ($t + l$, resp.) on T. In other words, a left-maximal (right-maximal) pairing cannot be "extended" to the left (right) since the character to the immediate left (right) of the occurrence of the common factor on P is different from the character to the immediate left (right) of the occurrence of the common factor on T.

The following example clarifies some of the concepts defined so far.

Example 6.1. Consider the strings $P = cbac$ and $T = ccabbaabbacc$. Then the triplets $v_1 = (2, 5, 2)$ and $v_2 = (2, 9, 2)$ are two pairings that represent the common factor ba. The first pairing, v_1, is maximal, while the second one, v_2, is not. In fact, pairing v_1 cannot be extended to the left since $P[1] = c \neq T[4] = b$, neither to the right since $P[4] = c \neq T[7] = a$. Instead, pairing v_2 can be extended to the right since $P[4] = c = T[11]$. Thus, pairing $v_3 = (2, 9, 3)$ contains v_2 and v_2 is a prefix-pairing of v_3.

In this work we are interested in the computation of a sequence of pairings of P and T that allows to decompose P into factors from which we can recover a spliced

alignment of P on T. This is formalized by the notion of *embedding* of P in T which consists of a sequence of pairings $\langle (p_1, t_1, l_1), \cdots, (p_n, t_n, l_n) \rangle$ such that $p_i + l_i \leq p_{i+1}$ and $t_i + l_i \leq t_{i+1}$ for each $1 \leq i < n$. An embedding ε is an (ℓ, g)-*embedding* if for each i, $l_i \geq \ell$ and $p_{i+1} - (p_i + l_i) \leq g$, for ℓ, g positive integers. The parameter g represents an upper bound on the number of consecutive mismatches that we admit between the pattern P and the text T in a spliced alignment, while ℓ is a lower bound on the number of consecutive matches.

Various notions of *maximality* have been used in Computational Biology to address the issue of reporting (possibly) meaningful common (or repetitive) structures of biological sequences without generating an excessive large output (see, i.e. , the classical problem of maximal local alignment [100]). For the same reason, we extend the order relation $\preceq$ between pairings to pairs of embeddings and, based on this, we derive the notion of maximal embeddings. Given two embeddings $\varepsilon = \langle v_1, \cdots, v_n \rangle$ and $\varepsilon' = \langle v'_1, \cdots, v'_m \rangle$, then ε is contained in ε' (in short $\varepsilon \subseteq \varepsilon'$) if and only if for each v_i in ε there exists a pairing v'_j in ε' such that $v_i \preceq v'_j$. Given the set $\mathcal{E}$ of the embeddings of P in T, we say that $\varepsilon \in \mathcal{E}$ is *maximal* iff there does not exist $\varepsilon' \in \mathcal{E}$ such that $\varepsilon \subset \varepsilon'$.

Since an embedding could provide the information for reconstructing a spliced alignment of a pattern P on a text T and since, as pointed out earlier, a notion of maximality is a common tool to filter out (probably) meaningless results, we propose the problem of finding all the maximal (ℓ, g)-embeddings of P in T, formalized as follows.

Problem 12. Maximal Embedding Problem.
Input: A pattern P, a text T, and two parameters ℓ and g.
Output: The set $\mathcal{E}_{max}$ of the maximal (ℓ, g)-embeddings of P in T.

For simplicity, in the following we refer to a maximal (ℓ, g)-embedding as a maximal embedding.

6.3 The Maximal Embedding Graph

A useful representation of the set of all maximal embeddings of a pattern P in a text T is provided by the Maximal Embedding Graph (in short MEG) defined as follows.

Definition 6.1 (MEG)**.** Given a pattern P and a text T, the Maximal Embedding Graph (MEG) of P and T is a directed graph $G = (V, E)$ where V is the set of pairings that appear in some maximal embedding of P in T, and E is the set of arcs from v_i to v_j such that v_i and v_j are two consecutive pairings in some maximal embedding.

Property 6.1 highlights a first important connection between paths of the MEG and embeddings. Such a connection will be further expanded in Corollary 6.5, where a bijection between maximal embeddings and maximal paths will be proved. We define maximal path a path that connects a source of the graph (i.e. a vertex without incoming arcs) to a sink (i.e. a vertex without outgoing arcs).

Property 6.1. *Let G be the MEG of a pattern P and a text T. Then, the sequence of vertices of a path $\mathcal{P}$ of G is the sequence of pairings of an embedding $\varepsilon(\mathcal{P})$ of P and T. Moreover, the sequence of pairings of a* maximal *embedding ε of P and T is a* maximal *path $\mathcal{P}(\varepsilon)$ of G.*

Proof. The first part is a consequence of the definitions of embedding and MEG (Def. 6.1). For the second part, notice that if v_1 or v_k were not a source or a sink respectively, then we could extend $\mathcal{P}(\varepsilon)$ by adding some other vertices. Thus, the extended path is a new embedding that strictly contains ε, contradicting its maximality. □

The previous property establishes one direction of the bijection we want to prove. Before presenting the other direction (Lemma 6.4), we need to prove the following intermediate results.

Lemma 6.2. *Let $v_i = (p_i, t_i, l_i)$ and $v_j = (p_j, t_j, l_j)$ be two pairings connected by an edge (v_i, v_j) of the MEG $G = (V, E)$ of P and T. Then there does not exist a pairing v s.t. $v_i \preceq v$ and $v_j \preceq v$.*

Proof. Assume to the contrary that there exists a vertex v s.t. $v_i \preceq v$ and $v_j \preceq v$ and $(v_i, v_j) \in E$. Clearly, the triplet $v' = (p_i, t_i, p_j + l_j - p_i)$ is a sub-pairing of v. Consider the sequence of pairings of an embedding ε containing consecutively v_1 and v_2, and replace these two pairings by v'. The resulting sequence ε' is trivially an embedding because v_1 is a prefix-pairing of v', and v_2 is a suffix-pairing of v''. Moreover, the embedding ε' strictly contains the embedding ε. Thus $\varepsilon \subset \varepsilon'$ and therefore there does not exist a maximal embedding containing v_1 and v_2, contradicting the existence of (v_1, v_2) in the MEG. □

Lemma 6.3. *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two maximal paths. Then the sequence of vertices of $\mathcal{P}_1$ does not form a subsequence of the vertices of $\mathcal{P}_2$.*

Proof. To the contrary, suppose that $\mathcal{P}_1$ is a subsequence of $\mathcal{P}_2$. Since $\mathcal{P}_1$ and $\mathcal{P}_2$ are maximal paths, then at least two consecutive vertices of $\mathcal{P}_1$ are not consecutive in $\mathcal{P}_2$. Let v_i and v_{i+1} be such vertices and let $\mathcal{P}'$ be the sub-path of $\mathcal{P}_2$ which connects v_i to v_{i+1}. By the definition of MEG, there exists a maximal embedding ε in which v_i and v_{i+1} appear consecutively. Replacing v_i and v_{i+1} in ε with $\mathcal{P}'$, we obtain an embedding ε' that strictly contains ε, contradicting its maximality. □

We are now ready to prove the other direction of the bijection between maximal paths and maximal embeddings.

Lemma 6.4. *Every maximal path $\mathcal{P}$ in the MEG is a maximal embedding $\varepsilon(\mathcal{P})$.*

Proof. By Property 6.1 the path $\mathcal{P}$ is an embedding $\varepsilon(\mathcal{P})$. We will show by absurd that $\varepsilon := \varepsilon(\mathcal{P})$ is maximal. Assume that ε is not maximal. Then there exists a path $\mathcal{P}'$ such that the embedding $\varepsilon' := \varepsilon(\mathcal{P}')$ strictly contains ε. First notice that, by Lemma 6.3, $\mathcal{P}$ does not form a subsequence of $\mathcal{P}'$. Therefore, there exists a pairing $v_i = (p_i, t_i, l_i) \in \varepsilon$ and a pairing $v' = (p', t', l') \in \varepsilon'$ s.t. $v_i \prec v'$. The following cases may arise.

- $p' = p_i < p_i + l_i < p' + l'$. By Lemma 6.2 we have $p' + l' \leq p_{i+1}$ and $t' + l' \leq t_{i+1}$. By construction of the MEG, there exists a maximal embedding ε'' which contains v_i and v_{i+1}. By replacing v_i with v' in ε'', we obtain an embedding that contains ε'', contradicting its maximality.

- $p' < p_i < p_i + l_i = p' + l'$. By Lemma 6.2 we have $p_{i-1} + l_{i-1} \leq p'$ and $t_{i-1} + l_{i-1} \leq t'$. By construction of the MEG, there exists a maximal embedding ε'' which contains v_{i-1} and v_i. By replacing v_i with v' in ε'', we obtain an embedding that contains ε'', contradicting its maximality.

- $p' < p_i < p_i + l_i < p' + l'$. There exists a sub-pairing v'' of v' having the same starting position of v_i on P and T. By replacing v' with v'', the proof continues as in the previous first case.

- If pairing v_i is a sink (in the first case) or v_i is a source (in the second case), then there exists a maximal embedding ε'' which contains v_i. By replacing v_i with v' in ε'', we obtain an embedding that strictly contains ε'', contradicting its maximality.

In every case we reach a contradiction, thus the embedding $\varepsilon(\mathcal{P})$ is maximal. □

Clearly the bijection easily derives from the definition of MEG, from Property 6.1, and from Lemma 6.4.

Corollary 6.5. *Let G be the MEG of a pattern P and a text T. Then every maximal embedding ε of P in T is a maximal path $\mathcal{P}(\varepsilon)$ of G and, conversely, every maximal path $\mathcal{P}$ of G is a maximal embedding $\varepsilon(\mathcal{P})$ of P in T.*

6.4 Solving the Maximal Embedding Problem

The MEG is a representation of the solutions of the Maximal Embedding problem, but an algorithm that efficiently constructs it from the pattern P and the text T

does not appear immediate. Therefore we propose the *Compact Maximal Embedding Graph (CMEG)*: a graph that can be efficiently built from the two sequences, and from which the whole MEG can be reconstructed.

In the first part of this section, we start from the definition of the CMEG and we explore the theoretical relationships between the MEG and the associated CMEG. In particular we will show a direct characterization of its edge set, characterization that will be used in the construction of the CMEG itself. Unfortunately the CMEG loses the simple bijection between maximal (ℓ, g)-embeddings and maximal paths of the MEG that derives from Corollary 6.5. However we will show that we can identify some particular paths of the CMEG that are related to maximal (ℓ, g)-embeddings. These paths are called *extended maximal paths* and connect two particular kinds of vertices: extended sources and extended sinks. Lemma 6.8 and Lemma 6.9 prove two properties that permit us to recognize such vertices and paths.

The second part of this section will clarify the relationship between a path of the CMEG and a set of embeddings. Indeed, Algorithm 7 is a procedure for computing a set of embeddings from a path of the CMEG. In particular we show that such procedure computes a set of maximal embeddings if the input path is an extended maximal path. Moreover, if we restrict our attention to all bounded-length extended maximal paths, Algorithm 7 provides a way to reconstruct all maximal (ℓ, g)-embeddings. Such strategy, though theoretically correct, is inefficient. Therefore, in the third part of this section we propose an algorithm that, given a CMEG $H = (V', E')$, reconstructs the MEG $G = (V, E)$ which H is associated to. Clearly, by virtue of Corollary 6.5, from the MEG G we can then easily enumerate the set (or a significant part) of all maximal embeddings. The time complexity of the MEG reconstruction procedure is linear in the size $|\,V\,| + |\,E\,|$ of the MEG plus the additional time required by a data structure that stores the intermediate results of the reconstruction. A careful choice of this data structure could achieve amortized linear time in the size of V.

Besides of being able to efficiently enumerate all maximal embeddings, the use of CMEG is also motivated by the fact that it can be efficiently constructed starting from the pattern P and the text T. The fourth part of this section will delineate the main ideas underlying an algorithm that builds the CMEG in linear time in the total length of sequences P and T and in quadratic time in the total number of maximal pairings V'. Since we are requiring that the length of a pairing is at least ℓ, the number of maximal pairings is expected to be low in practice and, thus, the time complexity of this algorithm is acceptable.

Summarizing, the four parts of this section provide an efficient two-step algorithm that solves the Maximal Embedding problem. Indeed, given the pattern P and the text T, we can first build the CMEG $H = (V', E')$ of P and T in time $O(|\,P\,| + |\,T\,| + |\,V'\,|^2)$, and then we can use the CMEG to reconstruct the MEG $G = (V, E)$

in amortized time $O(|V| + |E|)$. An optional third step could visit the MEG G to explicitly enumerate all the maximal embeddings.

6.4.1 The Compact Maximal Embedding Graph

Before introducing the formal definition of a CMEG, we need to state the following basic property of pairings:

Lemma 6.6. *For each pairing v of a pattern P and a text T, there exists exactly one maximal pairing that contains v.*

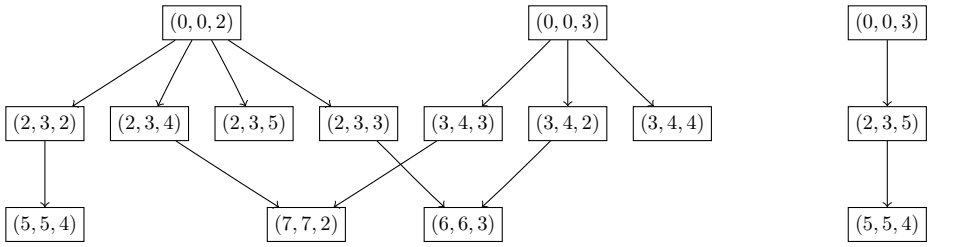
Proof. Let v be the pairing (p, t, l) and suppose, by absurd, that two non-comparable maximal pairings, $v_1 = (p_1, t_1, l_1)$ and $v_2 = (p_2, t_2, l_2)$, contain v. Since $v \preceq v_1$ and $v \preceq v_2$, then $p - p_1 = t - t_1$ and $p - p_2 = t - t_2$. Thus, by combining the previous two equalities, we obtain $p_2 - p_1 = t_2 - t_1$. Without loss of generality suppose that $p_1 < p_2$. Since v_1 and v_2 are not comparable, the ending positions of v_1 on P and T are strictly smaller than the ending positions of v_2 on P and T respectively. Moreover, since v is a sub-pairing of both v_1 and v_2, the ending positions of v_1 on P and T are strictly greater than the starting positions of v_2 on P and T. Consider the first characters on P and T to the right of the common factor represented by the pairing v_1. Their positions on P and T are $p_1 + l_1$ and $t_1 + l_1$ respectively. Since $p_2 - p_1 = t_2 - t_1 = \delta$, the expressions $p_1 + l_1$ and $t_1 + l_1$ can be rewritten as $p_2 - \delta + l_1$ and $t_2 - \delta + l_1$. The quantity $-\delta + l_1$ is a positive constant not greater than l_2. Thus $p_2 - \delta + l_1$ and $t_2 - \delta + l_1$ are internal positions of v_2 on P and T, which implies that $P[p_1 + l_1] = P[p_2 - \delta + l_1] = T[t_2 - \delta + l_1] = T[t_1 + l_2]$. In other words, $v_1' = (p_1, t_1, l_1 + 1)$ is a pairing which contains v_1, contradicting the maximality of v_1. □

The above property allows us to define a function φ from the set of vertices V of a MEG $G = (V, E)$ to the set of maximal pairings V' such that associates to pairing v the maximal pairing $\varphi(v)$ which contains v. The CMEG is then obtained by replacing each pairing v of the MEG with the maximal pairing $\varphi(v)$, and by connecting two maximal pairings v_1' and v_2' if a sub-pairing of v_1' is connected to a sub-pairing of v_2' in the MEG.

More formally we have the following definition.

Definition 6.2 (CMEG)**.** The *Compact Maximal Embedding Graph (CMEG)* associated to a MEG $G = (V, E)$ is the graph $H = (V', E')$ such that $V' = \{\varphi(v) \mid v \in V\}$ and $E' = \{((\varphi(v_1), \varphi(v_2)) \mid (v_1, v_2) \in E\}$.

Figure 6.1 depicts the MEG and the CMEG that represent the maximal (ℓ, g)-embeddings (with $\ell = 2$ and $g = 1$) of the string $P = aabcdddde$ in $T = aabbcddde$.



(a) The MEG (b) The CMEG

Figure 6.1: An example of a MEG and the associated CMEG for strings $P = aabcdddde$ and $T = aabbcddde$.

A crucial step in the direct construction of the CMEG (without resorting to the MEG) is the computation of its edge set starting from the set of maximal pairings. The following property gives a characterization of the edges of the CMEG based on the characteristics of the maximal pairings they connect.

Property 6.7. *Let $G = (V, E)$ be a MEG and let $H = (V', E')$ be the CMEG associated to G. Then (v_1, v_2) is an edge of H if and only if:*

- $p_1 + 2\ell \leq p_2 + l_2$ *and* $t_1 + 2\ell \leq t_2 + l_2$ *(ℓ-conditions)*
- $p_2 \leq p_1 + l_1 + g$ *and* $(p_2 - p_1) - (t_2 - t_1) < g$ *(g-conditions)*

where $v_1 = (p_1, t_1, l_1)$ and $v_2 = (p_2, t_2, l_2)$ are two maximal pairings.

Proof. Recall that we are only interested in maximal (ℓ, g)-embeddings. First, note that if $(v_1, v_2) \in E'$ then there exists an edge (v_3, v_4) in G such that $\varphi(v_3) = v_1$ and $\varphi(v_4) = v_2$, and, hence, $p_1 \leq p_3 < p_3 + l_3 \leq p_1 + l_1$ and $p_2 \leq p_4 < p_4 + l_4 \leq p_2 + l_2$, where $v_3 = (p_3, t_3, l_3)$ and $v_4 = (p_4, t_4, l_4)$.

By the definition of MEG (Def. 6.1) and the definition of (ℓ, g)-embedding, we have $p_1 \leq p_3 < p_3 + \ell \leq p_3 + l_3 \leq p_4 < p_4 + \ell \leq p_4 + l_4 \leq p_2 + l_2$, which proves the first ℓ-condition. Similar arguments on t_i and l_i for $i = \{1, 2, 3, 4\}$ show that also the second ℓ-condition holds.

The first g-condition is a direct consequence of the definition of MEG. It remains to prove the second g-condition. Let $\delta_1 = p_3 - p_1$ and $\delta_2 = (p_2 + l_2) - (p_4 + l_4)$. In other words, δ_1 and δ_2 are the differences between the starting positions of v_3 and v_1 and between the ending positions of v_2 and v_4, respectively. Consider now the sum $l_3 + l_4$. Clearly such sum cannot be greater than the minimum between $p_2 + l_2 - p_1 - \delta_1 - \delta_2$ and $t_2 + l_2 - t_1 - (\delta_1 + \delta_2)$. By the definition of MEG, we have that the difference δ_g between the starting position of v_4 on P and the ending position of v_3 on P is not greater than g. But δ_g is equal to $p_2 + l_2 - p_1 - (\delta_1 + \delta_2) -$

$(l_3 + l_4)$. By applying the upper-bound of the sum $l_3 + l_4$ stated above, we obtain $(p_2 + l_2 - p_1 - (\delta_1 + \delta_2)) - (t_2 + l_2 - t_1 - (\delta_1 + \delta_2)) = (p_2 - p_1) - (t_2 - t_1) \leq \delta_g \leq g$, that proves the second g-condition.

Clearly, given two maximal pairings that meet the ℓ- and g-conditions, it is easy to derive two sub-pairings that form an embedding, proving the converse (this process will be formalized by the concept of *2-pairing set* later on). The proof uses the same idea of the *if*-part and thus it is omitted. □

The strong relationship between maximal paths and maximal embeddings proved for MEGs is not longer true in CMEGs. Despite that, the following lemmas show that it is possible to characterize the subset of vertices of the CMEG that are images (under the function φ) of sources and sinks of the MEG. In the reconstruction of maximal embeddings from the CMEG, these kinds of vertices will play the same role of sources and sinks in MEGs.

Lemma 6.8. *Let G be the MEG of a pattern P and a text T, and let H be the CMEG associated to G. Then the pairing $v = (p, t, l)$ is a sink in G if and only if either (i) $v' := (p', t', l) := \varphi(v)$ is a sink in H, or (ii) for each successor $v'' = (p'', t'', l'')$ of v' in H, $p'' + l'' < p' + l' + \ell$ or $t'' + l'' < t' + l' + \ell$.*

Proof. Clearly if v' is a sink of H, then also v is a sink of G. Let us concentrate on condition (*ii*). Assume to the contrary that there exists a successor v'' of v' such that $p' + l' + \ell \leq p'' + l''$ and $t' + l' + \ell \leq t'' + l''$. We will show that, under this assumption, there exists at least a sub-pairing u of v'' such that (v, u) is an edge of G, contradicting the hypothesis that v is a sink of G. First, notice that $p + l = p' + l'$ and $t + l = t' + l'$ for the same arguments of the proof of Lemma 6.4. Moreover, observe that $p' + l' > p''$ and $t' + l' > t''$, otherwise (v, v'') would be an edge of G (by Property 6.7). Now consider the sub-pairing $u = (p_u, t_u, l_u)$ of v'' such that $p_u = p'' + \delta$, $t_u = t'' + \delta$, and $l_u = l'' - \delta$ where $\delta = \max(p' + l' - p'', t' + l' - t'')$. From the hypothesis, we can conclude that $l_u \geq \ell$ and, by construction and by Property 6.7, that $p + l = p' + l' \leq p_u \leq p' + l' + g = p + l + g$ and $t + l = t' + l' \leq t_u$. In other words, (v, u) is an edge of G, which concludes the proof. □

From the previous lemma we can easily derive a similar property for the sources of the MEG.

Lemma 6.9. *Let G be the MEG of a pattern P and a text T, and let H be the CMEG associated to G. Then the pairing $v = (p, t, l)$ is a source in G if and only if either (i) $v' = (p', t', l) = \varphi(v)$ is a source in H, or (ii) for each predecessor $v'' = (p'', t'', l'')$ of v' in H, $p'' + \ell > p'$ or $t'' + \ell > t'$.*

Proof. Let $P' = p_m \cdots p_1$ and $T' = t_n \cdots t_1$ be the reverse of the strings $P = p_1 \cdots p_m$ and $T = t_1 \cdots t_n$, respectively. The lemma derives from Lemma 6.8 on strings P' and T'. □

Given a CMEG, we say that a vertex is an *extended source* (*extended sink*, resp.) if it is the image of a source (sink, resp.) of the underlying MEG. A path of a CMEG is a *extended maximal path* if it starts from an extended source and ends in an extended sink.

The following section will show that from all the bounded-length extended maximal paths we can compute the set of all maximal embeddings.

6.4.2 Reconstruction of Embeddings from a Path

In this section we introduce a procedure that can be used for obtaining all maximal (ℓ, g)-embeddings from the Compact Maximal Embedding Graph of a pattern P and a text T. The primary concern of this section is to prove the correctness and the completeness of such procedure, and we defer to the following section the design of an algorithm that, based on these ideas, can *efficiently* solve the Maximal Embedding problem.

To this aim, we start by describing an algorithm (see Algorithm 7) for reconstructing a set of (ℓ, g)-embeddings from a path of the CMEG and we prove its correctness (Lemma 6.10). Moreover we will show that the same algorithm applied on the set of all bounded-length extended maximal paths computes the set of all maximal (ℓ, g)-embeddings (Lemma 6.12 and Lemma 6.13).

Before introducing Algorithm 7, we need to give the preliminary notion of *2-pairing set*. In the following, we say that two pairings $v_1 = (p_1, t_1, l_1)$ and $v_2 = (p_2, t_2, l_2)$ are *consecutive* on P (on T, resp.) if $p_1 + l_1 \leq p_2$ ($t_1 + l_1 \leq t_2$, resp.) or that are *overlapping* otherwise.

Definition 6.3 (2-pairing-set)**.** Given two pairings $v_1 = (p_1, t_1, l_1)$ and $v_2 = (p_2, t_2, l_2)$ such that $p_1 < p_2 + l_2$ and $t_1 < t_2 + l_2$, the *2-pairing set* of v_1 and v_2 is the set Q of all the pairs (v', v'') of consecutive pairings on P and T such that:

- pairing v' is a prefix-pairing of v_1 whose length is at least ℓ;
- pairing v'' is a suffix-pairing of v_2 whose length is at least ℓ;
- the sum of the lengths of v' and v'' is maximum.

Now we can describe the recursive algorithm **BuildEmbeddings** for reconstructing a set $\mathcal{E} = \{\varepsilon_1, \ldots, \varepsilon_m\}$ of (ℓ, g)-embeddings related to a path $\mathcal{P} = \langle v_1, \ldots, v_n \rangle$ in the CMEG of a pattern P and a text T. Subsequently we will prove that, if $\mathcal{P}$ is an extended maximal path, then the set $\mathcal{E}$ is composed by *maximal* (ℓ, g)-embeddings.

If the path $\mathcal{P}$ is composed by a single pairing v, then the procedure returns one embedding $\varepsilon = \langle v \rangle$ (Lines 1–2). The embedding ε is clearly an (ℓ, g)-embedding.

If the path $\mathcal{P}$ is composed by more than one node, then the procedure is recursively called on the subpath $\mathcal{P}' = \langle v_2, \ldots, v_m \rangle$ in order to obtain the set $\mathcal{E}'$ of all the

Algorithm 7: **BuildEmbeddings**$(G, \mathcal{P})$

Data: A CMEG graph H and a path $\mathcal{P} = \langle v_1, \ldots, v_m \rangle$ of H.

```
1  if P = ⟨v⟩ then
2      return {⟨v⟩} ;
3  E ← ∅ ;
4  E' ← BuildEmbeddings(H, ⟨v_2, ..., v_m⟩) ;
5  foreach ε' = ⟨v'_2, ..., v'_m⟩ ∈ E' do
6      if v_1 is link-compatible with v'_2 then
7          Q ← 2-pairing set of v_1 and v'_2 ;
8          Add ⟨v', v'', v'_3, ..., v'_m⟩ to E for each (v', v'') ∈ Q ;
9      end
10 end
```

Result: A set $\mathcal{E}$ of (ℓ, g)-embeddings related to $\mathcal{P}$.

(ℓ, g)-embeddings related to $\mathcal{P}'$ (Line 4). Then, for each embedding ε' returned by the recursive call, the *if* statement at Line 6 checks if the pairing v_1 of the path $\mathcal{P}$ is link-compatible with the first pairing v'_2 of the embedding ε'. We say that a pairing u_1 is *link-compatible* with a pairing u_2 if and only if the pair (u_1, u_2) meets the ℓ- and g-conditions of Property 6.7. The notion of link-compatibility is closely related to the notion of 2-pairing set. In fact, two pairings are link-compatible if and only if their 2-pairing set is not empty. The procedure of computing the 2-pairing set of two link-compatible pairings is straightforward and it is omitted.

For each embedding ε' returned by the recursive call, the *if* statement at Line 6 checks if the pairing v_1 of the path $\mathcal{P}$ is link-compatible with the first pairing v'_2 of the embedding ε'. Then for each pair (v', v'') that belongs to the 2-pairing set $\mathcal{Q}$, the embedding ε' is extended by replacing the pairing v'_2 with v'' and adding v' to the front (Lines 7–8). Let ε'' be the resulting sequence of pairings.

We can prove by induction that ε'' is an embedding. By hypothesis ε' is an embedding and so is the sequence $\langle v'', v'_3, \ldots, v'_m \rangle$ since v'' is a suffix-pairing of v'_2 whose length is at least ℓ (by definition of 2-pairing set). Adding v' to the front of $\langle v'', v'_3, \ldots, v'_m \rangle$ produces a sequence of pairings that satisfies the properties of an (ℓ, g)-embedding. Let $v' = (p', t', l')$ and $v'' = (p'', t'', l'')$. From the definition of 2-pairing set, we have $l' \geq \ell$, $p' + l' \leq p''$, and $t' + l' \leq t''$. We only have to prove that $p'' \leq p' + l' + g$ or, equivalently, that $p'' - p' - l' \leq g$. First notice that the procedure **BuildEmbeddings** maintains the following invariant (see Lemma 6.11): The starting positions on P and T of the first pairing of the path $\mathcal{P}$ are equal to the starting positions on P and T of every embedding returned by the procedure.

Three different cases must be examined in order to conclude the proof of $p'' - p' - l' \leq g$.

- Pairings v_1 and v'_2 are consecutive both on P and T. Then the 2-pairing set is composed by only one pair (v_1, v'_2) (since the pairs in the 2-pairing set must have maximum length). But v_1 and v'_2 are link-compatible, thus $p'' = p'_2 \leq p_1 + l_1 + g = p' + l' + g$.

- Pairings v_1 and v'_2 overlaps on P or T, and $t'_2 + l'_2 - t_1 \leq p'_2 + l'_2 - p_1$. Then the sum of the lengths of the pairings of a pair in the 2-pairing set $\mathcal{Q}$ is $l'+l'' = t'_2+l'_2-t_1 = t''+l''-t'$. The difference $\delta_g = p''-p'-l'$ can be rewritten as $p''+l''-p'-l'-l''$ that is equal to $p'_2+l'_2-p_1-t'_2-l'_2+t_1 = (p'_2-p_1)-(t'_2-t_1)$ and, from the second g-condition of Property 6.7, is upper-bounded by g.

- Pairings v_1 and v'_2 overlaps on P or T, and $p'_2 + l'_2 - p_1 \leq t'_2 + l'_2 - t_1$. Then the sum of the lengths of the pairings of a pair in the 2-pairing set $\mathcal{Q}$ is $l' + l'' = p'_2 + l'_2 - p_1 = p'' + l'' - p'$. The difference $\delta_g = p'' - p' - l'$ can be rewritten as $p'' + l'' - p' - l' - l''$ that is equal to $p'' + l'' - p' - p'' - l'' + p' = 0$ thus, clearly, not greater than g.

The following lemma derives from the previous arguments.

Lemma 6.10. *Algorithm* **BuildEmbeddings** *correctly computes a set $\mathcal{E}$ of (ℓ, g)-embeddings related to the input path $\mathcal{P}$ of a CMEG H in time $O(|\mathcal{E}| \cdot |\mathcal{P}|)$.*

Proof. The correctness of the algorithm is a consequence of the considerations presented above.

The time complexity of the algorithm is proportional to the number of the embeddings in $\mathcal{E}$ multiplied by the length m of the path $\mathcal{P}$ since every operation can be performed in constant time. □

The following lemma proves some properties of the embeddings computed by procedure **BuildEmbeddings** on a path $\mathcal{P}$. To simplify the exposition, we denote by $\mathcal{E}(\mathcal{P})$ the set of embeddings computed by **BuildEmbeddings** on the path $\mathcal{P}$.

Lemma 6.11. *Let $\varepsilon = \langle v'_1, \ldots, v'_m \rangle$ be an embedding of P in T that belongs to the set $\mathcal{E}(\mathcal{P})$ where $\mathcal{P} = \langle v_1, \ldots, v_m \rangle$. Then for each $1 \leq i \leq m$, we have $v'_i \preceq v_i$. Moreover v'_1 and v_1 have the same starting positions on both P and T, and v'_m and v_m have the same ending positions on both P and T.*

Proof. The properties can be easily proven by induction on the length of the path. First, let us consider the base case in which $\mathcal{P}$ is composed by one pairing v. The procedure returns a single embedding composed only by v, and the properties are trivially verified.

Now let us suppose that the lemma holds for a path of length n and let us prove it for $n + 1$. If the path has length $n + 1$, the embeddings of set $\mathcal{E}'$ computed by the recursive call at line 4 have length n and, then, they satisfy the induction

hypothesis. Line 8 updates each embedding of $\mathcal{E}'$ by replacing the first pairing v'_2 with one of its suffix-pairings v''. Since v'_2 is a prefix-pairing of v_2, it follows that $v'' \preceq v_2$. Moreover if v'_2 was also the last pairing of the embedding, since v'' is a suffix-pairing of v'_2, the replacement of v'_2 with v'' preserves the ending positions of the last pairing of the embedding on both P and T. Then, at the same line, a new pairing is added to the front of the embedding. Such pairing is, by construction, a prefix-pairing of v_1, thus it is contained in v_1 and has the same starting positions (on both P and T) of v_1. Since the other pairings of the embeddings are not changed by the procedure, the proof is concluded. □

Our aim is to prove an even stronger result than Lemma 6.10. Indeed, we claim (see Lemma 6.12 and Lemma 6.13) that an (ℓ, g)-embedding is maximal if and only if it is computed by procedure **BuildEmbeddings** called on a extended maximal path (i.e. a path that connects an extended source to an extended sink). The proof of such claim is achieved by a series of intermediate results.

Lemma 6.12. *A maximal (ℓ, g)-embedding $\varepsilon = \langle v'_1, \ldots, v'_m \rangle$ belongs to the set $\mathcal{E}(\mathcal{P})$ where $\mathcal{P} = \langle \varphi(v'_1), \ldots, \varphi(v'_m) \rangle$ is an extended maximal path.*

Lemma 6.13. *An (ℓ, g)-embedding ε is maximal if belongs to a set $\mathcal{E}(\mathcal{P})$, where $\mathcal{P} = \langle v_1, \ldots, v_m \rangle$ is an extended maximal path.*

First, we prove Lemma 6.12, then we claim that embeddings computed by **BuildEmbeddings** on the same extended maximal path or on distinct extended maximal paths are pairwise not comparable, from which Lemma 6.13 follows.

Proof of Lemma 6.12. From Corollary 6.5, embedding $\varepsilon = \langle v'_1, \ldots, v'_m \rangle$ corresponds to a maximal path $\mathcal{P}' = \langle v'_1, \ldots, v'_m \rangle$ of the EMEG. From Definition 6.2, the path $\mathcal{P}'$ induces an extended maximal path $\mathcal{P} = \langle \varphi(v'_1), \ldots, \varphi(v'_m) \rangle$. Since $\mathcal{P}'$ is a maximal path of the EMEG, by Lemmas 6.9 and 6.8, we have that $\mathcal{P}$ is an extended maximal path of the CMEG.

We want to show that $\varepsilon \in \mathcal{E}(\mathcal{P})$. For simplicity we denote with v_i the i-th pairing $\varphi(v'_i)$ of $\mathcal{P}$. Let us proceed by induction on length of the embedding (which is equal to the length of $\mathcal{P}'$ and $\mathcal{P}$). If ε has only one pairing v', then the maximality of ε implies the maximality of v' and, thus, $v' = \varphi(v') = v$. The procedure returns the embedding composed by the single pairing v (Line 2) which is equal to ε. Now let us assume that every maximal embedding of length $n-1$ is computed by procedure **BuildEmbeddings**. Suppose that ε has length n and consider the embedding $\varepsilon' = \langle v'_2, v'_3, \ldots, v'_n \rangle$. The embedding ε' is contained in a maximal embedding $\varepsilon'' = \langle v''_2, v'_3, \ldots, v'_n \rangle$ which belongs to $\mathcal{E}(\langle v_2, \ldots, v_n \rangle)$ by induction hypothesis. Indeed, since ε is maximal, the embeddings ε' and ε'' are equal in every position but the first. As v'_2 is contained in both v_2 and v''_2, by Lemma 6.6, we have $v''_2 \preceq v_2$, and from the maximality of ε'' we have that v''_2 is a prefix-pairing of v_2 and v'_2 is a

suffix-pairing of v_2''. The equivalent set of (v_1, v_2'') contains all the suffix-pairings that can form an (ℓ, g)-embedding, thus ε belongs to $\mathcal{E}(\mathcal{P})$. □

We accomplish the proof of Lemma 6.13 via a series of claims. The first two claims prove that if an extended maximal path $\mathcal{P}'$ is a sub-path of a second extended maximal path $\mathcal{P}$, then an embedding computed from $\mathcal{P}'$ is not contained in an embedding computed from $\mathcal{P}$. Claim 6.16 generalizes the previous results to any pair of embeddings computed from two distinct extended maximal paths. Finally, Claim 6.18 complete the results to any pair of embeddings computed from a single path. Clearly it follows that the embeddings computed by **BuildEmbeddings** are pairwise not-comparable (Corollary 6.19).

Claim 6.14. *Let $\mathcal{P} = \langle v_1, \ldots, v_i, \ldots, v_m \rangle$ be a path of a CMEG of P and T such that an internal vertex v_i is an extended source, and let $\mathcal{P}'$ be the sub-path of $\mathcal{P}$ that starts from v_i and ends to v_m. Then each embedding $\varepsilon' \in \mathcal{E}(\mathcal{P}')$ is not contained in an embedding $\varepsilon \in \mathcal{E}(\mathcal{P})$.*

Proof. Consider the vertex v_{i-1} which precedes v_i in $\mathcal{P}$, and the set of embeddings $\mathcal{E} = \mathcal{E}(\langle v_i, \ldots, v_m \rangle)$ computed during the recursive call of **BuildEmbeddings**. After $\mathcal{E}$ has been computed, the procedure attempts to extend each embedding ε of $\mathcal{E}$ by replacing the first pairing of ε with a pair of pairings of the equivalent set of v_{i-1} and a prefix-pairing of v_i. However, by Lemma 6.9, v_i and a prefix-pairing of v_{i-1} whose length is at least ℓ (as required by the equivalent set definition) are overlapping (on P or T). Thus every embedding ε of $\mathcal{E}(\mathcal{P})$ has a pairing v' that is contained in a strict suffix-pairing of v_i (i.e. a suffix-pairing of v_i that is not equal to v_i). Since the first pairing of an embedding ε' that belongs to $\mathcal{E}(\mathcal{P}')$ is a prefix-pairing of v_i, we obtain that $\varepsilon' \not\subseteq \varepsilon$. □

Claim 6.15. *Let $\mathcal{P} = \langle v_1, \ldots, v_i, \ldots, v_m \rangle$ be a path of a CMEG of P and T such that an internal vertex v_i is an extended sink, and let $\mathcal{P}'$ be the sub-path of $\mathcal{P}$ that starts from v_1 and ends to v_i. Then each embedding $\varepsilon' \in \mathcal{E}(\mathcal{P}')$ is not contained in an embedding $\varepsilon \in \mathcal{E}(\mathcal{P})$.*

Proof. It is a consequence of Claim 6.14 on the CMEG of P' and T', where P' is the reverse of P and T' is the reverse of T. □

Now we can prove that embeddings computed from distinct extended maximal paths are not comparable.

Claim 6.16. *Let $\mathcal{P}_1 = \langle v_1, \ldots, v_m \rangle$ and $\mathcal{P}_2 = \langle u_1, \ldots, u_n \rangle$ be two distinct extended maximal paths of a CMEG. Then $\varepsilon_1 \in \mathcal{E}(\mathcal{P}_1)$ and $\varepsilon_2 \in \mathcal{E}(\mathcal{P}_2)$ are not comparable.*

Proof. Note that for each pairing v'_i of ε_1 we have $v'_i \preceq v_i$, and for each pairing u'_j of ε_2 we have $u'_j \preceq u_j$. Now suppose, to the contrary, that $\varepsilon_1 \subseteq \varepsilon_2$. By the definition of $\subseteq$, for each $v'_i \in \varepsilon_1$ there exists a pairing $u'_j \in \varepsilon_2$ such that $v'_i \preceq u'_j$. Since $u'_j \preceq u_j$, it follows that v'_j is contained in two maximal pairings, v_i and u_j. From Lemma 6.6, we conclude that $v_i = u_j$ and, thus, that $\mathcal{P}_1$ is a proper sub-path of $\mathcal{P}_2$. Therefore v_1 or v_m are internal nodes of $\mathcal{P}_2$ and, since v_1 in an extended source and v_m is an extended sink, at least one among Claim 6.14 or Claim 6.15 applies. It follows that $\varepsilon_1 \not\preceq \varepsilon_2$, contradicting the hypothesis. □

Corollary 6.17. *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two distinct extended maximal paths of a CMEG. Then $\mathcal{E}(\mathcal{P}_1) \cap \mathcal{E}(\mathcal{P}_2) = \varnothing$.*

Two embeddings are incomparable even if they are computed from the same path, as shown by the following claim.

Claim 6.18. *Given a path $\mathcal{P}$ of a CMEG, for every pair of distinct embeddings $\varepsilon_1, \varepsilon_2 \in \mathcal{E}(\mathcal{P})$ we have $\varepsilon_1 \not\subseteq \varepsilon_2$.*

Proof. Assume that $\mathcal{P}$ is at least 2 vertices long (otherwise the procedure returns one embedding and the claim is trivially verified). Thus Line 7 is executed at least once, say on pairings v_i and v'_{i+1}. The embeddings contained in the equivalent set of v_i and v'_{i+1} are pairwise not comparable and, since the ending positions of the prefix-pairings of v_i and the starting positions of the suffix-pairings of v'_{i+1} do not change during the subsequent execution, two embeddings added to the result remain not comparable. □

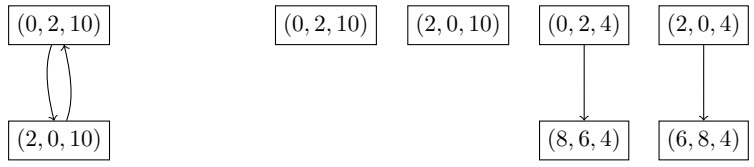
From Claim 6.16 and Claim 6.18, it derives that two embeddings computed by **BuildEmbeddings** are not comparable.

Corollary 6.19. *Let G be a CMEG and $\mathcal{E} = \bigcup_{\mathcal{P}} \mathcal{E}(\mathcal{P})$ for all extended maximal paths $\mathcal{P}$. Then $\varepsilon_1, \varepsilon_2 \in \mathcal{E}$ are not comparable.*

Finally we can prove, from Lemma 6.12 and Corollary 6.19, that procedure **BuildEmbeddings**($\mathcal{P}$) computes only maximal embeddings if $\mathcal{P}$ is an extended maximal path.

Proof of Lemma 6.13. From Lemma 6.12, procedure **BuildEmbeddings** computes all the maximal embeddings. Since, by Corollary 6.19, the embeddings computed are pairwise not comparable, they are all maximal. □

Lemmas 6.12 and 6.13 extend the bijection between maximal embeddings and maximal paths of the MEG to the extended maximal paths of the CMEG. A first strategy that can be used to reconstruct all the maximal embeddings from the CMEG is to call procedure **BuildEmbeddings** on every extended maximal path



(a) The CMEG (b) The MEG

Figure 6.2: An example of a CMEG that contains a cycle and the MEG which it is associated to. The strings P and T are $P = aabbaabbaabb$ and $T = bbaabbaabbaa$.

of the CMEG, as it has been done during the previous proofs. However it is clear that, in general, there could exist several maximal paths that differ for only a small sub-path. In this case, procedure **BuildEmbeddings** is not able to re-use partial results that has been computed on the common part of such paths, and thus it recomputes them independently for each extended maximal paths.

Moreover a second element must be carefully considered. A MEG is, clearly, a directed *acyclic* graph but the associated CMEG could be *cyclic*. Consider the following example.

Example 6.2. Let $P = aabbaabbaabb$ and $T = bbaabbaabbaa$. It is easy to see that the set of maximal pairings of P and T is $V = \{v_1, v_2\}$ where $v_1 = (0, 2, 10)$ and $v_2 = (2, 0, 10)$. If ℓ and g are equal to 4, then the CMEG has two arcs: the first is from v_1 to v_2 and the second one is from v_2 to v_1 (see Figure 6.2(a)). Therefore the CMEG contains one cycle. The MEG, depicted in Figure 6.2(b), does not have any cycle.

Computing the set of extended maximal paths on a directed cyclic graph is not an easy task. Clearly, since the MEG is acyclic, not every extended maximal path of the CMEG has associated a non-empty set of maximal embeddings. It is trivial to see that, for each cycle c, there exists a positive number m_c such that for all extended maximal paths $\mathcal{P}_c^i$ that repeat the cycle c more than m_c times, the set of embeddings $\mathcal{E}(\mathcal{P}_c)$ is empty. Indeed during the construction of the embeddings related to a path, for each vertex of the path one of its sub-pairings of length at least ℓ is added to the result (if such a sub-pairing exists, otherwise the procedure terminates). Thus the length of each maximal pairing of c is an upper-bound of the number m_c.

In the following section we propose an efficient algorithm that, based on the same ideas of **BuildEmbeddings**, visits the CMEG H and reconstruct the MEG G which H is associated to. To maintain efficiency, it explicitly deals with the presence of directed cycles and avoids the recomputation of unnecessary intermediate results.

6.4.3 Efficient Reconstruction of the MEG from the CMEG

Algorithm **VisitCMEG** reconstructs the MEG $G = (V, E)$ starting from the CMEG $H = (V', E')$ by visiting H depth-first order and expanding each edge (and vertex) of H to the set of edges (and vertices) of the original MEG G. Amortized linear time complexity in the size $|V| + |E|$ of the output can be achieved by exploiting a classic algorithmic technique called *memoization* [30].

Algorithm **VisitCMEG** recursively constructs, given a suffix-pairing v' of a pairing v of the CMEG, the subgraph of the MEG that is reachable, in the graph-theoretic sense, from a prefix-pairing of v' (i.e. a sub-pairing of v), and returns the set of sources of such subgraph. Clearly the complete MEG (and thus all the maximal embeddings) is obtained when the procedure is called on all the extended sources of the CMEG.

The correctness of the algorithm derives from that of **BuildEmbeddings** since it basically employs the same ideas, with the only important difference of dealing with the presence of directed cycles.

Essentially the original procedure **BuildEmbeddings** works by trying to add prefix pairings of the first vertex to the front of the embeddings recursively constructed on the tail of the path. Procedure **VisitCMEG**, instead, explicitly enumerates the prefix-pairings of the current suffix-pairing v' that are link-compatible with some suffix-pairing u' of a successor u of the current vertex v, and continues recursively by visiting the suffix-pairing u' of u. In other words, the order of the recursive call and the "extension" of the embedding (Lines 4 and Lines 6–8) of the original procedure **BuildEmbeddings** is reversed in the procedure **VisitCMEG** where the recursive call is performed only for a suffix-pairing of the successor. The recursion always terminates because the starting positions of the suffix-pairings on which the recursive calls are made form a strictly increasing sequence, bounded by the length of the pattern P and the text T.

Since there could be several directed paths connecting a vertex v_i to a vertex v_j in a CMEG, it is possible that the procedure is invocated more than once on the same input data. To avoid to recompute the same subgraph, a lookup table is used to store the results obtained by the procedure on a given suffix-pairing v' of vertex v. Subsequent invocations on v' can be performed in constant time (provided an adequate choice of the lookup table data structure).

The time complexity of the algorithm is linear in the size $|V| + |E|$ of the output MEG $G = (V, E)$ plus the time required by the creation of the lookup table and the subsequent retrieval operations.

In the simplest case, the lookup table can be implemented with an array and its creation requires linear time in the total length $L = \sum_{(p_i,t_i,l_i)} l_i$ of the maximal pairings $(p_i, t_i, l_i) \in V$. However, since in most cases only a part of the array is actually used, a different choice, such as a hash table, could be advisable. In fact

the number of results stored in the array is clearly upper-bounded by the number of output pairings V. Therefore an hash table could improve the performance because it can efficiently store a sparse collection while maintaining amortized constant-time retrieval. In this case the final amortized time-complexity is linear in the size $|\,V\,|+|\,E\,|$ of the output MEG.

6.4.4 Building the CMEG

In the previous sections we have presented the Compact Maximal Embedding Graph as a medium to implicitly represent the set of maximal (ℓ, g)-embeddings of a pattern P in a text T. Algorithm **VisitCMEG** provides an efficient method to explicitly enumerate all the maximal embeddings from the CMEG. However, one of the most important advantage of using a CMEG is that there exists an efficient algorithm to construct it. This section is devoted to describe such algorithm.

The algorithm works in two phases. First it computes the set of maximal pairings (i.e. the vertex set) of the CMEG G using the *Generalized Suffix Tree (GST)* of the pattern P and the text T ([52] provides a good introduction to such index structure). Then it creates the edge set by checking for every pair of vertices if the ℓ- and g-conditions of Property 6.7 are satisfied.

The first phase is the creation of the set of the maximal pairings of P and T (i.e. the vertex set of the CMEG). The key observation, for this phase, is that the generalized suffix tree of strings P and T easily provides the set of common factors between P and T and their occurrences in linear time in the size of the input $|\,P\,|+|\,T\,|$ plus the size of the output (the number of occurrences k). Clearly, given a common factor f, the cartesian product of its occurrences on P and its occurrences on T computes the set of pairings that represent f. However the pairings so computed are not guaranteed to be maximal. Thus we propose an efficient algorithm that visits the generalized suffix tree and produces directly the set of maximal pairings in time $O(|\,P\,|+|\,T\,|+|\,V\,|)$, where V is the final set of maximal pairings.

Observe that the problem of computing all maximal pairings of P and T generalizes the problem discussed in [52] of computing the maximal pairs of a single string S. In the following we discuss the basic steps of the algorithm we propose for computing all maximal pairings.

Let P_i denote the i-th suffix of P, that is the substring $P[i,|\,P\,|]$ of P. Let us first discuss the computation of the right-maximal pairings having i as starting position on P.

This step requires to calculate the longest common prefix P_i^1 of P_i which is a factor of T and it is achieved by finding the deepest node n_i^1 in the generalized suffix tree whose label is equal to P_i^1. (In fact a suffix tree is a tree that represents all suffixes of a given string S as paths from the root to a leaf: each node, instead, can be

considered as labelled by a prefix of a suffix of S.) Clearly, P_i^1 is a common factor between P and T and, by visiting the subtree rooted in n_i^1, the set of occurrences $O_i^1 = \{o_1^{i,1}, \ldots, o_m^{i,1}\}$ of P_i^1 in T can be determined. Let V_i be the set of pairings of the form $(i, o_j^{i,1}, |\, P_i^1\,|)$. It is easy to see that a pairing that belongs to V_i is right-maximal. The remaining right-maximal pairings starting at position i on P are computed by iteratively visiting the ancestors n_i^l of n_i^1 in the generalized suffix tree until the root has been reached. Let P_i^l be the prefix of P_i such that P_i^l is the label of n_i^l. Then add to V_i the pairings $(i, o_j^{i,l}, |\, P_i^l\,|)$ where $o_j^{i,l}$ is an occurrence of P_i^l in T such that $o_j^{i,l} \notin O_i^{l-1}$. Note that the set of occurrences $o_j^{i,l}$ can be obtained by visiting the subtree of the generalized suffix tree rooted in n_i^l that does not include n_i^{l-1}. Since $O_j^{l-1} \cap O_j^l = \varnothing$, the characters that are located immediately to the right of the new pairings on T and P are different, thus also the new pairings are right-maximal. Since we are interested in pairings whose length is at least ℓ, we stop the iterations when a node labelled by a prefix of P_i whose length is less than ℓ has been reached.

Maximal pairings are obtained by ensuring that the pairings added to set V_i are also left-maximal, i.e. pairings that cannot be "extended" to the left. To this aim we consider only the occurrences o_j^i of a prefix of P_i on T that are preceded by a character $T[o_j^i - 1]$ different from $P[i-1]$. Indeed, since $P[i-1] \neq T[o_j^i - 1]$, the pairings (i, o_j^i, l) (where l is the length of the common factor) cannot be extended to the left and, thus, they are also left-maximal.

Iterating the above procedure on every suffix P_i of P we obtain the set of maximal pairings $V = \bigcup_{1 \leq i \leq |P|} V_i$.

The construction of the vertex set achieves linear-time complexity $O(|\, P\,| + |\, T\,| + |\, V\,|)$ by adopting two "tricks": suffix-links and suffix-tree preprocessing. The suffix-links permit to consider each symbol of P once, ensuring the linearity in $|\, P\,|$. The preprocessing, instead, creates an appropriate set of lists of occurrences on T segregated by their preceding character and, thus, permits to avoid multiple visits of the subtree of the generalized suffix tree rooted on a node that is labelled by a common factor of P and T.

The second phase of the algorithm consists in the creation of the edge set. Our approach consider every pair of vertices and check if the two pairings satisfy the ℓ- and g-conditions. The resulting time complexity is $O(|V|^2)$ and, since the number of vertices is usually quite small (indeed we only admit maximal pairings that are at least ℓ characters long), it seems acceptable in practice.

The memory required by the algorithm is $O(|\, P\,| + |\, T\,| + |\, V\,| + |\, E\,|)$, i.e. linear in the input and output size. Although the hidden constant in the big-O notation could be significant due to the use of the suffix tree of T, memory requirements are not a real bottleneck. In fact, if the available memory is not sufficient to construct the suffix tree of T, the vertex set can be computed separately on contiguous (small)

segments of T, and then the various subsets are merged into the final vertex set V. The second phase of the algorithm, edge set construction, can be finally performed on the whole vertex set V since it only requires the memory needed to store the CMEG and not the suffix tree or the strings T and P. Notice that the asymptotic time complexity of the algorithm does not change even if the text T is partitioned in segments. Moreover, computing the vertex set separately on segments of the text T leads also to an efficient strategy for the parallelization of the algorithm.

6.5 From Embeddings to Spliced Alignments of ESTs

In this section the problem of reconstructing a set of spliced alignments, given a maximal (ℓ, g)-embedding, is described. Our goal is to reconstruct the spliced alignments of P on T from the set of the maximal (ℓ, g)-embeddings obtained from the CMEG graph. Maximal embeddings are expected to provide the spliced alignments, among which the biologically meaningful ones can be extracted by using a global agreement approach described in the next chapter. Before describing the problem, we have to introduce some definitions. Given an embedding $\varepsilon = \langle v_1, \ldots, v_m \rangle$ of pairings $v_i = (p_i, t_i, l_i)$, we call a *superfactor* of ε on P (on T, respectively) a substring $P[s_p, d_p]$ ($T[s_p, d_p]$, respectively), where $s_p \leq p_1$ and $d_p \geq p_m + l_m$ ($s_t \leq t_1$ and $d_t \geq t_m + l_m$, respectively). Given a fixed bound b_1, an embedding $\varepsilon = \langle v_1, \ldots, v_m \rangle$ of pairings $v_i = (p_i, t_i, l_i)$ is a *short pairing sequence* if the length of the gap separating v_i and v_{i+1} on T (i.e. the substring $T[t_i + l_i + 1, t_{i+1} - 1]$) is less than b_1, for each $1 \leq i < m$.

A short pairing sequence provides an EST factor corresponding to a genomic factor (or exon). Indeed the b_1 value is an upper bound for the length of gaps representing consecutive mismatches between the EST and the genomic sequence that are inside an exon (i.e. factor f'_j in the spliced alignment $\mathcal{C}$). Therefore pairings in a short pairing sequence must be composed into an exon on the text T.

Moreover, we fix also a lower bound $b_2 > b_1$ for gaps in the pattern P corresponding to gaps on the text T where the splice site corresponding to introns on the genomic sequence must be located. When some gap on T has length between b_1 and b_2, then the embedding is not able to give a meaningful spliced alignment, since that gap cannot reasonably classified as an intron nor as a part of an exon.

Given a maximal (ℓ, g)-embedding $\varepsilon = \langle v_1, \ldots, v_m \rangle$, our goal is to partition ε into k short pairing sequences $\varepsilon_i = \langle v_{q_{j-1}+1}, \ldots, v_{q_j} \rangle$, with $1 \leq q_1 < \ldots < q_k \leq m$ according to the fixed bounds for the gaps. For each ε_i, we want to find a superfactor f_i on P and a superfactor f'_i on T, s.t. $P = pf_1 \cdots f_k s$, $T = p' f'_1 i_1 f'_2 \cdots i_{k-1} f'_k s'$, and the edit distance between f_i and f'_i is minimized as stated in the definition of a spliced alignment.

We implemented a simple procedure which, given a maximal embedding $\varepsilon = \langle v_1, \ldots, v_m \rangle$, scans the sequence of pairings from left to right and merges two adjacent pairings v_i and v_{i+1} to form a single short pairing sequence if the gap between the two common factors is smaller than b_1 or creates a new short pairing sequence starting with v_{i+1} if such a gap is larger than b_2. If the length of the gap between v_i and v_{i+1} on T is included in the interval $[b_1, b_2]$, then the embedding does not represent a reasonably meaningful spliced alignment of the EST on the genomic sequence and, thus, it is discarded. Finally the superfactors are then computed starting from the short pairing sequences by choosing the "best" alignment of the gaps between short pairing sequences. In particular, given two adjacent pairings $v_i = (p_i, t_i, l_i)$ and $v_{i+1} = (p_{i+1}, t_{i+1}, l_{i+1})$ such that v_i and v_{i+1} belong to different short pairing sequences, a triplet $(p'_{i+1}, t'_i, t'_{i+1})$ is computed in such a way that the sum $d(P[p_i + l_i, p'_{i+1} - 1], T[t_i + l_i, t'_i]) + d(P[p'_{i+1}, p_{i+1} - 1], T[t'_{i+1}, t_{i+1} - 1])$, where $d(\cdot, \cdot)$ is the edit distance between two strings, is minimum. The output superfactor that contains v_i ends at $p'_{i+1} - 1$ on P and at t'_i on T, while the output superfactor that contains v_j starts at p'_{i+1} on P and at t'_{i+1} on T. Clearly, since the gap on P, by definition of (ℓ, g)-embedding, is at most g-character long where g is a parameter of the problem, the procedure of minimizing the sum of the edit distances can be performed in $O(1)$ time.

7 Agreement of Spliced Alignments

In the previous chapter we addressed the problem of finding a set of spliced alignments for a single transcript (EST) sequence against a reference genomic sequence. In this chapter we formulate the Minimum Factorization Agreement problem, which determines a single spliced alignment for each transcript in a set of related transcripts based on the global gene structure that they would induce. For this problem, which, given in input a set of transcripts and their putative spliced alignments, determines a single splice alignment for each of them based on a common gene structure induced by the input data, we provide an exact algorithm based on a precomputation step that is specifically designed to greatly reduce the size of real instances. An experimental analysis of the results produced by the algorithms proposed in the second part of thesis, has revealed encouraging results that increased our confidence in this approach.

7.1 Introduction

Producing *spliced alignments* of EST sequences is a fundamental task in the computational problem of reconstructing splice and transcript variants, which are fundamental data for the alternative splicing investigation.

Unfortunately, given an EST sequence, there can be several spliced alignments associated to it, since the original EST sequences may have different alignments against wide genomic regions. In this chapter we address the problem that the previous observation had highlighted: Given a collection C of different spliced alignments that are associated to an initial set S of EST sequences, how can we extract a subset C' of C such that each EST sequence in S has a "putative" spliced alignment in C' and C' agrees on a common alignment region to the genome or gene structure?

We formalize this problem by defining the Minimum Factorization Agreement (MFA) problem over an instance consisting of a set C of factorized sequences, called *compositions*, i.e. sequences that are the concatenation of factors of a finite set F. More precisely, the MFA problem over a set S of sequences asks for a minimum subset $F' \subseteq F$ such that each sequence in S has associated a composition in C using only factors in F'.

In this formalization an EST is a sequence $s \in S$ to which we associate a set of compositions representing the possible spliced alignments of the EST. A composi-

tion is a sequence over a finite alphabet F of genomic factors (that can be exons, prefixes or suffixes of exons of the gene). Then, the MFA problem formalizes the idea of computing a minimum cardinality set of genomic factors (exons) that can explain at least one spliced alignment for each sequence s in S.

The MFA problem also allows to face the more general problem of reconciling a set of EST sequence data with respect to different types of biological information. Indeed, the alphabet F by which a sequence is factorized could represent splice sites or splice patterns instead of exons and thus the MFA problem would ask for the minimum set of factors (common splice sites or significant patterns) on which all input sequences agree.

The rest of the chapter is structured as follows. We first show that the MFA problem is NP-hard even over simple instances. Successively we provide a two-step algorithm for the MFA problem that, first, reduces the instance size and then it solves exactly the problem on the reduced instance. Though the worst-case time complexity of the algorithm is exponential in the number of factors, the algorithm is efficient over practical instances, as the size-reduction step quickly and effectively trims the set of possible factors. The combination of the algorithms presented in the previous chapter for the maximal embedding problem with the algorithm that we introduce for the MFA problem, forms an approach to tackle tasks related to gene structure prediction. Therefore, we finally present an experimental evaluation of this combined approach to solve a subtask of gene structure prediction. The results we obtained from the experimental evaluation support the soundness of both the new formulation and the algorithmic solutions we provide.

7.2 The Minimum Factorization Agreement Problem

In this section we formulate the Minimum Factorization Agreement problem for a set C of factorized sequences is proposed.

Let $F = \{f_1, f_2, \ldots, f_{|F|}\}$ be an ordered finite set of *factors*, and let S be a set of sequences. We can assume to have an explicit enumeration of the sequences in S, i.e. $S = \{s_1, s_2, \ldots, s_{|S|}\}$. A *composition* c is a sequence over an alphabet F where each factor appears at most once in c and c respects the ordering of F. Equivalently c can be thought of as a subsequence of F. The *factor set* of a composition c is the set of factors belonging to c.

Informally, each sequence in S represents an EST sequence while the composition, as defined here, is a simplification of the concept of spliced alignment introduced in the previous chapter. Indeed, previously, we were interested in the sequence of both the genomic factors and the EST factors. Since in this chapter we are primarily focused on the determination of a common structure over the genomic sequence, we can safely ignore the EST factors and considering only the sequence of genomic

factors. Determining the actual nucleotide sequence of each genomic factor is a task performed by the previous step when compositions are reconstructed from the embeddings. Therefore, instead of representing the nucleotide sequence that composes each genomic factor, we consider the genomic factor as whole, and we represent it with a single symbol of F. Since each sequence s has associated a certain number of spliced alignments, each composition is colored by some sequences in S, that is for each composition we specify from which sequences the composition is derived. Notice that the same composition can be derived from more than one original EST sequence. In the following we will assume that each composition c is actually colored by some sequences in S, and we denote by $\lambda(c)$ such set of sequences. Conversely, given a sequence $s \in S$, we denote by $C(s)$ the set of compositions that are colored by s. By a slight abuse of language, given a subset $S_1 \subseteq S$, we denote by $C(S')$ the set $\bigcup_{s \in S'} C(s)$. Therefore $C(S)$ consists of all compositions.

Given the set $C(S)$, a subset $F' \subseteq F$ of factors is a *factorization agreement set for* $C(S)$ iff for each sequence $s \in S$, there exists a composition $c \in C(s)$ such that the factor set of c is entirely contained in F'.

Now we can formally define our main computational problem: Minimum Factorization Agreement problem.

Problem 13. Minimum Factorization Agreement (MFA).
Input: A set F of factors, a set S of sequences, and a set $C(S)$ of (colored) compositions.
Output: A minimum cardinality subset F' of F such that F' is a factorization agreement set for $C(S)$.

First we will prove, via an L-reduction, that MFA shares all known inapproximability results of Minimum Set Cover [6, 96], defined as follows.

Problem 14. Minimum Set Cover (Min Set Cover).
Input: A universe set U and a collection $\mathcal{C}$ of subsets of U.
Output: A minimum cardinality subset $\mathcal{C}_1$ of $\mathcal{C}$ such that the union of all sets in $\mathcal{C}_1$ is exactly U.

We provide an alternative formulation of the MFA problem based on binary matrices. Given an instance $(F, S, C(S))$ of the MFA problem, we construct a binary matrix M where the columns (rows, respectively) are in a one-to-one correspondence with the factors in F (with the compositions in $C(S)$, resp.). In the following we identify each column (row, resp.) with a factor (a composition, resp.). Moreover each row is colored and the set of colors is the set S of input sequences. (Notice that each row can be colored by more than one color.) The entry $M[\alpha, f]$ (i.e. the entry of the row identified by the factor f and the composition α) is equal to 1 if and only if the composition α contains the factor f.

Sequence	Compositions
1	A, B, D
2	A, C
3	B, C

Colors	Factors	A	B	C	D
$1,2$	A	1	0	0	0
$1,3$	B	0	1	0	0
$2,3$	C	0	0	1	0
1	D	0	0	0	1

Figure 7.1: Example of instance of MFA on factors $F = \{A, B, C, D\}$

Given a composition α, we denote by $1(\alpha)$ and $0(\alpha)$ respectively the set of columns of M containing 1 and 0 in row α. The MFA problem then asks for a minimum cardinality subset $F_1 \subseteq F$ such that, for each possible color c, there is a composition α with $1(\alpha) \subseteq F_1$ and the row α of M is colored by c. (Notice that this row can also be colored with different colors.)

The matrix formulation leads immediately to a reduction from Minimum Set Cover. In fact let $(U, \mathcal{C})$ be an instance of Min Set Cover. (We recall that U is the universe set and $\mathcal{C}$ is a collection of subsets of U.) Then let U be the color set, while the factor set and the composition set are both equal to $\mathcal{C}$. The entries of the matrix M are equal to the entries of the identity matrix and the set of colors of a row (identified by a subset $\mathcal{C}_1 \in \mathcal{C}$) is $\mathcal{C}_1$ itself. Notice that any agreement factor set F is also a set cover on $(U, \mathcal{C})$. An illustrative example where the instance of Min Set Cover is over universe $\{1, 2, 3\}$ and $\mathcal{C}$ consists of four subsets $A = \{1, 2\}$, $B = \{1, 3\}$, $C = \{2, 3\}$, $D = \{1\}$, is represented in Figure 7.1.

It is immediate to notice that the covers of $(U, \mathcal{C})$ are also factorizations agreement sets of M and vice versa. Moreover the reduction produces instances of MFA where all compositions consists of only one factor. Also, the sizes of approximate and optimal solutions of the instances of Min Set Cover and MFA are the same, hence all known inapproximability results for Min Set Cover (namely there exists a constant $c > 0$ such that no polynomial-time approximation algorithm can achieve a guaranteed approximation ratio $c \log n$, unless $\mathsf{P} = \mathsf{NP}$ [96]) hold for such a restriction.

7.3 An Algorithm for Solving the MFA Problem

In this section we study the matrix formulation of the MFA problem and we present an algorithm that is of practical interest. The first step of the algorithm consists of reducing the matrix, that is removing some rows and/or columns, while keeping at least an optimal solution.

More precisely, we provide some rules that must be enforced, if possible, on the matrix M. The reduction process is iterated until no such rule applies any more. At the end we will obtain a reduced matrix M_1 together with a set S_F of factors that must be added to any feasible solution of M_1 to obtain a solution of M. Let us list the five rules.

1. If there exist two rows r_1, r_2 such that $M[r_1, c] \geq M[r_2, c]$ for each column c of M and $\lambda(r_1) \cap \lambda(r_2) \neq \varnothing$, then remove all sequences in $\lambda(r_1) \cap \lambda(r_2)$ from the colors of r_1. If $\lambda(r_1) = \varnothing$ then remove r_1 from M. (Rationale: all feasible solutions that include all factors in r_1 also include all factors in r_2.)

2. If there exists a column c containing only 1s, then add c to S_F and remove c from M. (Rationale: c must necessarily belong to all solutions of M.)

3. If there exists a column c and a color i such that all rows of M that are colored i have value 1 in c, then add c to S_F and remove c from M. (Rationale: since at least one of the compositions colored by i have to be factorized by a solution, c must necessarily belong to all solutions of M.) Notice that this rule is more general than Rule 1, which is nonetheless explicitly stated since it is computationally easier to enforce.

4. If there exists a row r containing only 0s and r is colored C, then remove from M all colors in $\lambda(r)$. Remove from M all rows r such that $\lambda(r)$ becomes empty. (Rationale: S_F contains all factors of a composition of r, therefore r can be removed from M without enlarging the size of a feasible solution. Moreover, if $\lambda(r) \neq \varnothing$ and the row r has only zeroes for each color $x \in \lambda(r)$, then S_F contains all factors used in a composition colored by x.)

5. If there exists a column c containing only 0s, then remove c from M. (Rationale: no actual composition contains c, which therefore is not an interesting factor any more.)

Consequently, in the following we can assume that our algorithms deal with reduced matrices.

7.3.1 A Naïve Algorithm

A simple approach for solving exactly the MFA problem consists of computing all possible subsets $F_1 \subseteq F$ in non-decreasing order of cardinality of such subsets, and checking if F_1 is a factorization agreement set. As we are interested in a minimum-cardinality agreement set, the algorithm halts as soon as a factorization agreement set is found.

The analysis of the time complexity is straightforward. In fact, given a subset F_1 of factors, deciding if F_1 is a factorization agreement set for S requires

time $O(|F||C(S)|)$. Moreover, since the factor subsets are enumerated in non-decreasing order of cardinality and denoting by *opt* the cardinality of a minimum factorization agreement set, the algorithm stops after considering $\sum_{k=1}^{opt} \binom{|F|}{k}$ subsets. Thus, the overall time complexity is $O\left(\sum_{k=1}^{opt}\left(\binom{|F|}{k}|F||C(S)|\right)\right)$ which is clearly $O\left(2^{|F|}|F||C(S)|\right)$.

Even if the time is bounded only by an exponential function, two considerations are in order. First of all, on real data the reduction process usually results in a matrix much smaller than the original, therefore lessening the impact of the exponential factor (which is relative to the reduced matrix). Moreover, the implementation of such simple method actually exploits some features of real-world architectures, such as bit-parallelism and data locality (in fact any reduced matrix with at most 100 factors and 10^5 compositions resides in the cache memory of a mainstream PC).

7.3.2 A Refined Algorithm

In this section we describe an algorithm which is more refined and somewhat more complex than the one of the previous section. While we expect the refined algorithm to outperform the naïve one on hard and/or large instances, we have currently been unable to find some real-world instances where those improvements have resulted in appreciable decreases in the running times.

The algorithm is based on the idea of building a binary matrix $FM(\cdot,\cdot)$, where each element $FM(X,s)$ is equal to 1 if and only if the set of factors $X \subseteq F$ contains a composition of the sequence $s \in S$. The following Lemma leads immediately to a recursive definition of $FM(\cdot,\cdot)$.

Lemma 7.1. *Let X be a subset of F and let s be a sequence in S. Then $FM(X,s) = 1$ iff X is a minimal factor set of a composition of s or there exists $x \in X$ with $FM(X-\{x\},s) = 1$.*

Proof. Assume initially that $FM(X,s) = 1$ and that X is not a minimal set that is the factor set of a composition of s. Then, by definition of minimality, there exists $x \in X$ such that $X - \{x\}$ is composition of s. By definition of $FM(\cdot,\cdot)$, $FM(X-\{x\},s) = 1$. The other direction is trivial. □

If we have an algorithm for checking if a subset $X \subseteq F$ is a minimal factor set of a composition of a sequence $s \in S$, then we can describe a simple exponential time, yet effective, algorithm for the MFA problem, whose correctness is an immediate consequence of Lemma 7.1. Our algorithm, whose pseudocode is represented in Algorithm 8, is called MFA2 and is bit-parallel. In fact, we will use the array *Fact*$(\cdot)$ instead of the matrix $FM(\cdot,\cdot)$, where *Fact* is indexed by a subset X of F and each element of *Fact* is actually the vector corresponding to an entire row of

Algorithm 8: MFA2(F, S)

Data: An instance $(F, S, C(S))$ of MFA
Compute *MinimalFA*;
Fact$(\varnothing) \leftarrow \bar{0}$;
foreach $F_1 \subseteq F$ *in non-decreasing order of cardinality,* $F \neq \varnothing$ **do**
    Fact$(F_1) \leftarrow \bigvee_{x \in F_1}$ *Fact*$(F_1 - \{x\}) \bigvee$ *MinimalFA*(F_1);
    if *Fact*$(F_1) \leftarrow \bar{1}$ **then**
        return F_1
    end
end

$FM(\cdot, \cdot)$. Therefore, *Fact*(X) is the characteristic binary vector of the sequences which have a composition whose factor set is contained in X.

We will also use another array *MinimalFA*$(\cdot)$, which is also indexed by a subset X of F and each *MinimalFA*(X) is the characteristic binary vector of the sequences which have a composition whose factor set is exactly X (it is immediate to notice that it is equivalent to the fact that X is a minimal set such that there exists a composition whose factor set is X, as the algorithm works on a reduced matrix.) We recall that a sequence can color more than one composition.

It is paramount for the efficiency of the MFA2 algorithm that we are able to compute and query efficiently *MinimalFA*$(\cdot)$. Since almost all the vectors in *MinimalFA*$(\cdot)$ are equal to $\bar{0}$, it is more efficient to actually use a hash for storing *MinimalFA*$(\cdot)$, assuming that an entry not found in the hash is equal to $\bar{0}$. It is immediate to notice that the time required by Alg. 9 is $O(|S||C(S)||F|)$, assuming that we have chosen an implementation of hash tables where each operation can be performed in time logarithmic in the number of elements, and noticing that $|C(S)| \leq 2^{|F|}$.

The analysis of Algorithm 8 is based on the simple observation that all bit-level operations can be performed in $O(|S|)$ time. Moreover, since the subsets of F are generated by non-decreasing cardinality, at most $O(|F|^{opt})$ such subsets are considered, where *opt* is the size of the optimal solution. The overall time complexity is therefore $O(|S||F|^{opt})$. The time for reducing the original matrix is clearly polynomial in the size of the matrix, as enforcing each rule results in the removal of at least a row or a column. The most trivial implementation of the reduction rules has an $O(|S||F|^3|C(S)|^3)$ overall time complexity.

Algorithm 9: Build *MinimalFA*

Data: An instance $(F, S, C(S))$ of MFA

```
1 MinimalFA ← empty hash;
2 foreach c ∈ C(s) do
3     foreach s ∈ S do
4         if c is a composition of s then
5             Pose to 1 the bit of MinimalFA(f) corresponding to sequence s;
6         end
7     end
8 end
9 return MinimalFA(·)
```

7.4 Experimental Analysis

The combination of the algorithm proposed in the previous chapter to solve the maximal embedding problem (from which spliced alignments can be easily recovered) with the algorithms that we proposed in this chapter to solve the MFA algorithm, represents a new approach for the task of predicting the exon-intron structure of a gene. Indeed, given a set of EST sequence and a genomic region, we can compute a set of possible spliced alignments for each of them by using the algorithm for the maximal embedding problem. Then, the resulting set represents a set of compositions from which the MFA problem extracts a minimum cardinality factorization agreement set. Each factor of the factorization agreement set should correspond to an exon or a part of exon. We designed an experiment to evaluate this hypothesis. Clearly, if such hypothesis is true, then more refined algorithms can be applied on this set of exons in order to predict, for example, full-length transcript isoforms.

We believe that the results we obtained support the soundness of our approach. Especially the running times are encouraging, as the experiments show that our approach is able to complete the analysis of several genes of a chromosome in only few hours, using modest computational resources.

The experiment is structured as follows. We randomly selected 350 genes from the human chromosome 22 and we retrieved the gene structure of each gene from ASPicDB [24]. We eliminated single-exon genes and genes whose genomic sequence retrieved from ASPicDB was significantly different from the sequence of the last revision on the NCBI database. 337 genes remained. We considered, for each gene, a genomic sequence of approximately 10^6 nucleotides centered in the position of gene as described by the NCBI Gene database. For each gene, we considered the associated clusters in the UniGene database as input transcripts. The genomic sequence and the transcript sequences for each gene have been used to compute a

set of spliced alignments as described in the previous chapter, by setting $\ell = 15$ and $g = 31$. Only a few basic biological criteria have been used to filter some incorrect sliced alignment, such as introns must be of length less than a fixed value (that is $100,000$ bp in the experimentation described in the following), the coverage of the alignment, with respect to the total transcript length, must be greater than a fixed threshold and the transcript prefix/suffix discarded in a composition must be of limited length (in the experimentation up to a 17% of the transcript length). Then, the spliced alignments have been "translated" in a MFA instance. During the "translation", factors that overlapped significantly have been merged in a single "macro"-factor. Finally the reduction algorithm and the naïve algorithm have been used to solve the translated instance.

The total number of processed ESTs for the 337 input genes is 126,995, with an average of 363 ESTs for each gene. We tested our software on a standard PC running Linux. The total running time has been approximately 8 hours, with average time of 83 seconds for each gene. In 7 cases the algorithm reported an excessive amount of spliced alignments for a single EST sequence and it has been terminated. A more stringent parameter setting would probably solve this problem, but we preferred to maintain the same parameters for all genes and reporting the issue. These genes have not been included in computation of the running time.

We compared our results on the 337 input genes of the chromosome 22 with the gene structure data provided by ASPicDB in order to evaluate if the resulting factors were compatible with the gene structure predicted by ASPicDB. For each gene, let the *benchmark set* B and the *test set* F be the set of exons from ASPicDB and the set of genomic factors predicted by our method respectively. We estimate the similarity, between an exon e in B and a genomic factor f in F, by means of three measures: (*i*) the number of genomic positions covered by e and f (or true positive $TP(e, f)$), (*ii*) the number of genomic positions covered by f but not by e (or false positive $FP(e, f)$), and (*iii*) the number of genomic positions covered by e but not by f (or false negative $FN(e, f)$). We say that an exon e from B is present in the test set, if there exists a genomic factor f in F with the maximum value of $TP(e, f)$, s.t. $FP(e, f) + FN(e, f)$ is less than a fixed bound of 10bp. We estimate the sensitivity of our method as the ratio between the number of benchmark exons that are also present in the test set, and the total number of benchmark exons. Sensitivity results are summarized in the first five columns of Table 7.1 at page 116 for the 337 genes that we have studied. For each gene the table reports the number of exons in the benchmark set, the number of benchmark exons that are also present in the test set, the value of sensitivity and the average value for $(FP + FN)$ on the benchmark exons present in the test set. On the whole set of the 337 input genes our method reaches a sensitivity of 76% (5,917 out of the 7,825 benchmark exons are present in the test set). 324 input genes have a sensitivity greater than 50% and, among these genes, 43 genes have a sensitivity of 100%.

At its current state, our method overpredicts the exons of a gene. The main reason of the overprediction is that it does not merge different parts of the same exons into the same class. Indeed the "macro"-factors described above have been used only during the creation of the MFA instance, and not for this comparison, where we considered all the original components of the "macro"-factor. At this stage, we do not believe that the overprediction is a real issue, since the main aim of the experimentation was to evaluate if the factors were compatible with the gene structure provided by ASPicDB. For this reason, we say that a factor f in the test set can predict an exon e in the benchmark set, if it is a meaningful part of e and then, if the coverage of f w.r.t. e (provided by the value $TP(e, f)$) is at least 90% of its length. On this basis, we estimate the specificity of our method as the ratio of the number of predicted factors having a coverage of at least 90% w.r.t. the benchmark set, and the total number of predicted factors in the test set. Specificity results are summarized in the last three columns of Table 7.1. For each gene the table reports the total number of factors in the test set, the number of factors having a coverage of at least 90% w.r.t. the benchmark set and the value of specificity. On the whole set of the 337 input genes our method reaches a specificity of 87% (56,416 out of the 64,802 predicted factors have a coverage of at least 90% w.r.t. the benchmark set). 318 input genes have a specificity greater than 50% and, among these genes, 17 genes have a specificity of 100%.

We would like to stress the fact that our implementation is only at a preliminary stage, especially since a number of reasonable biological criteria have not been incorporated yet. Still, our experimental analysis has shown the computational efficiency of the implementation. At the same time, this preliminary analysis shows that our method roughly confirms the gene structure data of ASPicDB [24] which gives us high expectations on the quality of the results that can be obtained once a more refined implementation can be produced and executed on an even large genomic region.

Table 7.1: Sensitivity and Specificity results

	Sensitivity				Specificity		
Gene	#exons ASPicDB	#exons ASPicDB in Test	Sensitivity (%)	Average $FP + FN$	#exons in Test	#exons covered 90%	Specificity (%)
A4GALT	13	13	100	1	123	117	95
ACO2	42	31	74	3	750	481	64
ACR	7	6	86	2	15	12	80
ADM2	7	4	57	2	50	50	100
ADORA2A	25	17	68	3	163	158	97
ADRBK2	37	27	73	1	255	250	98
ADSL	29	24	83	3	247	233	94
AIFM3	43	36	84	3	126	124	98

(continue)

Table 7.1: Sensitivity and Specificity results (continued)

	Sensitivity				Specificity		
Gene	#exons ASPicDB	#exons ASPicDB in Test	Sensitivity (%)	Average $FP + FN$	#exons in Test	#exons covered 90%	Specificity (%)
ALG12	17	12	71	2	151	147	97
ANKRD54	20	14	70	3	157	155	99
AP1B1	36	29	81	3	163	156	96
APOBEC3A	10	9	90	2	49	36	73
APOBEC3B	12	11	92	1	59	45	76
APOBEC3C	16	10	63	4	221	152	69
APOBEC3D	13	7	54	4	42	33	79
APOBEC3F	15	12	80	3	104	79	76
APOBEC3G	26	19	73	3	291	248	85
APOBEC3H	10	10	100	2	35	35	100
APOL2	23	15	65	2	216	177	82
APOL3	29	19	66	2	227	184	81
APOL6	11	8	73	2	110	106	96
ARFGAP3	23	22	96	2	675	250	37
ARVCF	34	28	82	3	271	158	58
ASCC2	55	48	87	3	330	310	94
ASPHD2	4	4	100	3	95	93	98
ATF4	24	17	71	2	751	746	99
ATP6V1E1	18	16	89	1	296	292	99
ATXN10	32	26	81	2	338	327	97
BAIAP2L2	19	16	84	2	52	47	90
BCL2L13	34	29	85	2	379	365	96
BID	23	16	70	2	182	179	98
BIK	5	5	100	4	42	40	95
BPIL2	22	21	95	2	30	27	90
BRD1	29	23	79	3	186	165	89
C1QTNF6	21	17	81	2	139	132	95
C22orf42	13	12	92	2	37	37	100
C22orf43	15	12	80	3	32	26	81
CABIN1	56	49	88	2	288	275	95
CABP7	7	5	71	1	28	27	96
CACNA1I	38	38	100	3	66	63	95
CACNG2	4	2	50	4	12	7	58
CARD10	36	33	92	2	159	156	98
CBX6	13	8	62	3	210	201	96
CBX7	13	9	69	2	184	173	94
CBY1	19	18	95	2	182	159	87
CCDC116	8	7	88	2	32	28	88
CCDC117	8	7	88	1	196	191	97
CCDC134	9	9	100	2	52	41	79
CDC42EP1	9	9	100	3	188	182	97
CDC45L	32	26	81	2	138	130	94
CECR1	26	23	88	2	235	229	97
CECR5	27	22	81	2	284	269	95
CECR6	2	1	50	3	51	49	96
CELSR1	48	44	92	2	154	148	96
CERK	21	19	90	2	275	254	92
CHCHD10	13	10	77	4	104	99	95
CHEK2	37	31	84	2	130	122	94

(continue)

Table 7.1: Sensitivity and Specificity results (continued)

	Sensitivity				Specificity		
Gene	#exons ASPicDB	#exons ASPicDB in Test	Sensitivity (%)	Average $FP + FN$	#exons in Test	#exons covered 90%	Specificity (%)
CLTCL1	51	40	78	3	141	126	89
CRELD2	24	18	75	2	159	154	97
CRKL	3	3	100	3	409	369	90
CRYBA4	7	6	86	1	83	82	99
CRYBB1	8	8	100	2	95	91	96
CRYBB2	9	8	89	2	105	105	100
CRYBB3	9	9	100	3	25	24	96
CSDC2	7	6	86	1	94	92	98
CSF2RB	19	18	95	3	114	112	98
CSNK1E	39	0	0	ND			
CYP2D6	23	16	70	3	215	127	59
CYTH4	31	25	81	3	162	151	93
CYTSA	31	0	0	ND			
DDT	10	9	90	2	92	73	79
DDTL	10	7	70	1	91	34	37
DEPDC5	70	60	86	2	188	169	90
DGCR14	24	22	92	2	192	186	97
DGCR2	30	28	93	2	708	657	93
DGCR6	16	9	56	3	120	108	90
DGCR6L	11	10	91	3	129	122	95
DGCR8	25	22	88	2	522	236	45
DMC1	21	19	90	1	40	38	95
DNAL4	13	12	92	3	161	155	96
DRG1	18	13	72	2	216	198	92
DUSP18	12	9	75	3	115	101	88
EFCAB6	51	48	94	2	118	112	95
EIF3D	42	39	93	3	385	375	97
EIF3L	48	0	0	ND			
EIF4ENIF1	38	32	84	2	269	246	91
ELFN2	11	9	82	1	48	47	98
EMID1	24	18	75	3	94	88	94
ENTHD1	14	13	93	1	43	43	100
EP300	43	35	81	2	325	303	93
EWSR1	50	42	84	3	559	542	97
FAM109B	5	4	80	1	85	84	99
FAM118A	33	26	79	2	218	191	88
FAM83F	9	9	100	3	58	57	98
FBLN1	44	0	0	ND			
FBXO7	25	20	80	2	360	331	92
FOXRED2	21	15	71	2	258	226	88
GAB4	14	12	86	2	20	20	100
GAL3ST1	22	16	73	3	103	99	96
GALR3	2	2	100	2	10	10	100
GAS2L1	23	17	74	1	203	200	99
GCAT	20	18	90	3	123	121	98
GGA1	60	48	80	2	411	378	92
GGT1	31	26	84	2	142	109	77
GGT5	22	20	91	3	197	192	97
GNAZ	11	8	73	2	151	146	97

(continue)

Table 7.1: Sensitivity and Specificity results (continued)

	Sensitivity				Specificity		
Gene	#exons ASPicDB	#exons ASPicDB in Test	Sensitivity (%)	Average $FP + FN$	#exons in Test	#exons covered 90%	Specificity (%)
GP1BB	30	22	73	3	378	313	83
GRAMD4	30	22	73	3	123	121	98
GRAP2	32	29	91	2	126	117	93
GSC2	3	3	100	2	5	4	80
GSTT1	18	13	72	3	179	143	80
GSTT2	18	10	56	3	71	27	38
GSTT2B	5	4	80	4	62	48	77
GTPBP1	23	21	91	2	286	273	95
GTSE1	31	25	81	2	254	242	95
H1F0	6	1	17	0	468	37	8
HDAC10	50	33	66	2	162	156	96
HIC2	8	8	100	4	149	147	99
HIRA	43	38	88	2	228	216	95
HMGXB4	22	18	82	2	203	200	99
HMOX1	9	8	89	2	194	191	98
HORMAD2	16	14	88	2	47	39	83
HPS4	36	28	78	2	383	342	89
HSCB	15	11	73	3	51	44	86
IGLL1	5	5	100	3	34	33	97
IGLL3	5	3	60	3	9	8	89
IL17RA	19	13	68	2	144	82	57
IL17REL	15	4	27	3	7	6	86
IL2RB	24	19	79	2	149	146	98
ISX	4	4	100	2	21	19	90
JOSD1	13	9	69	3	276	260	94
KCNJ4	3	3	100	2	26	26	100
KCTD17	19	14	74	1	103	92	89
KDELR3	10	7	70	2	526	145	28
KIAA1644	5	5	100	1	71	70	99
KLHL22	28	20	71	2	282	263	93
L3MBTL2	32	25	78	2	201	189	94
LARGE	37	25	68	2	174	164	94
LDOC1L	2	2	100	2	235	228	97
LGALS1	17	13	76	3	155	126	81
LGALS2	4	4	100	1	24	15	63
LIF	7	5	71	3	67	67	100
LIMK2	47	33	70	3	446	408	91
LMF2	31	16	52	3	156	150	96
LZTR1	44	37	84	3	299	282	94
MAFF	10	0	0	ND			
MAP3K7IP1	30	24	80	2	279	236	85
MAPK1	24	15	63	2	812	798	98
MAPK11	21	18	86	2	82	82	100
MAPK12	26	21	81	2	172	164	95
MAPK8IP2	20	18	90	3	117	114	97
MB	18	13	72	1	418	407	97
MCAT	11	9	82	2	108	108	100
MCHR1	8	7	88	1	92	87	95
MED15	54	40	74	2	451	416	92

(continue)

Table 7.1: Sensitivity and Specificity results (continued)

	Sensitivity				Specificity		
Gene	#exons ASPicDB	#exons ASPicDB in Test	Sensitivity (%)	Average $FP + FN$	#exons in Test	#exons covered 90%	Specificity (%)
MEI1	56	51	91	2	126	112	89
MFNG	26	22	85	1	189	184	97
MICALL1	25	22	88	2	204	185	91
MIF	5	5	100	5	144	126	88
MIOX	22	18	82	2	75	70	93
MKL1	32	27	84	2	224	211	94
MLC1	42	36	86	2	388	386	99
MMP11	24	16	67	3	292	271	93
MN1	3	3	100	2	113	111	98
MORC2	33	31	94	2	151	137	91
MOV10L1	39	36	92	2	130	113	87
MRPL40	10	8	80	1	137	130	95
MTMR3	34	28	82	2	377	349	93
MTP18	10	6	60	2	426	147	35
MYH9	76	64	84	2	1065	1026	96
MYO18B	57	54	95	3	122	113	93
NAGA	22	18	82	2	263	261	99
NCAPH2	44	34	77	3	178	177	99
NCF4	14	12	86	2	98	85	87
NDUFA6	12	9	75	3	195	179	92
NEFH	11	5	45	2	215	214	100
NF2	31	28	90	2	255	253	99
NFAM1	11	9	82	2	60	60	100
NHP2L1	23	16	70	2	407	379	93
NIPSNAP1	25	19	76	3	347	340	98
NOL12	21	16	76	3	158	151	96
NUP50	31	24	77	2	335	326	97
ODF3B	17	12	71	3	80	62	78
OSBP2	37	23	62	3	180	169	94
OSM	4	4	100	3	32	30	94
P2RX6	22	17	77	3	53	46	87
PACSIN2	29	22	76	3	675	357	53
PANX2	9	7	78	1	51	49	96
PARVB	35	30	86	2	177	159	90
PARVG	37	30	81	2	154	144	94
PATZ1	14	11	79	2	236	230	97
PDGFB	11	10	91	2	82	77	94
PDXP	7	3	43	3	191	190	99
PES1	29	27	93	1	293	286	98
PEX26	14	12	86	2	141	76	54
PHF21B	23	23	100	2	84	80	95
PHF5A	8	5	63	1	121	116	96
PICK1	36	32	89	2	178	171	96
PIK3IP1	21	18	86	3	331	311	94
PIM3	14	7	50	3	162	151	93
PISD	44	33	75	2	313	277	88
PIWIL3	23	22	96	2	34	31	91
PLA2G3	7	7	100	2	15	14	93
PLA2G6	50	46	92	2	265	242	91

(continue)

Table 7.1: Sensitivity and Specificity results (continued)

	Sensitivity				Specificity		
Gene	#exons ASPicDB	#exons ASPicDB in Test	Sensitivity (%)	Average $FP + FN$	#exons in Test	#exons covered 90%	Specificity (%)
PLXNB2	57	51	89	2	410	403	98
PMM1	30	23	77	3	166	163	98
PNPLA3	17	11	65	3	75	64	85
PNPLA5	14	12	86	2	30	26	87
POLDIP3	31	28	90	2	492	462	94
POLR2F	20	16	80	2	148	104	70
POLR3H	23	16	70	3	380	365	96
PPARA	23	22	96	2	207	196	95
PPIL2	53	0	0	ND			
PPM1F	22	18	82	2	307	300	98
PPPDE2	18	8	44	1	129	102	79
PRAME	24	19	79	2	269	263	98
PRODH	31	20	65	2	117	108	92
PVALB	12	9	75	2	38	37	97
RAB36	14	11	79	2	70	68	97
RABL2B	29	17	59	3	119	109	92
RABL4	25	14	56	2	147	135	92
RAC2	15	11	73	2	195	188	96
RANBP1	21	17	81	2	171	161	94
RANGAP1	43	36	84	2	413	400	97
RASD2	4	3	75	2	69	66	96
RASL10A	5	5	100	2	39	38	97
RBM9	51	34	67	2	610	590	97
RBX1	20	15	75	2	120	99	83
RFPL1	2	2	100	0	15	6	40
RFPL3	3	3	100	1	13	11	85
RGL4	22	20	91	2	62	50	81
RHBDD3	19	16	84	1	157	130	83
RIBC2	11	11	100	2	52	52	100
RNF215	13	12	92	3	37	34	92
RP3-402G11.5	26	21	81	3	155	139	90
RP3-474I12.5	7	7	100	2	13	12	92
RPS19BP1	12	11	92	2	144	136	94
RRP7A	28	22	79	3	716	431	60
RTDR1	16	14	88	1	63	59	94
RTN4R	11	9	82	2	115	113	98
SAPS2	58	0	0	ND	379	10	3
SCARF2	14	13	93	1	91	79	87
SCO2	4	4	100	1	80	75	94
SCUBE1	29	27	93	2	73	72	99
SDF2L1	5	5	100	2	108	99	92
SEC14L2	34	29	85	3	281	242	86
SEC14L3	21	17	81	3	51	44	86
SEC14L4	17	8	47	2	18	18	100
SEPT3	24	20	83	2	297	286	96
SERHL	14	13	93	2	143	73	51
SERHL2	22	17	77	3	285	131	46
SERPIND1	11	8	73	3	134	127	95
SEZ6L	25	22	88	2	142	136	96

(continue)

Table 7.1: Sensitivity and Specificity results (continued)

	Sensitivity				Specificity		
Gene	#exons ASPicDB	#exons ASPicDB in Test	Sensitivity (%)	Average $FP + FN$	#exons in Test	#exons covered 90%	Specificity (%)
SF3A1	33	25	76	2	555	549	99
SFI1	63	53	84	2	248	213	86
SGSM1	33	28	85	2	95	90	95
SGSM3	54	38	70	3	331	308	93
SH3BP1	34	33	97	3	109	103	94
SHANK3	34	20	59	2	113	100	88
SLC16A8	7	7	100	1	21	21	100
SLC25A1	21	19	90	3	219	211	96
SLC25A18	25	16	64	2	113	105	93
SLC2A11	35	26	74	2	162	147	91
SLC35E4	8	7	88	2	92	90	98
SLC5A1	20	19	95	2	84	83	99
SLC5A4	15	14	93	2	19	17	89
SLC7A4	12	8	67	3	43	41	95
SMARCB1	19	16	84	3	255	236	93
SMC1B	25	24	96	2	58	51	88
SMCR7L	27	22	81	2	372	363	98
SMTN	77	64	83	1	452	435	96
SNAP29	13	10	77	2	197	195	99
SNRPD3	19	14	74	2	180	157	87
SOX10	13	10	77	3	151	137	91
SREBF2	56	40	71	2	587	559	95
SRRD	11	9	82	2	84	74	88
SSTR3	4	3	75	1	20	18	90
ST13	24	19	79	3	424	415	98
SULT4A1	19	14	74	2	122	116	95
SUSD2	18	17	94	2	165	162	98
SYN3	23	17	74	2	76	59	78
SYNGR1	19	16	84	1	243	231	95
TBC1D10A	23	23	100	2	203	187	92
TBX1	15	15	100	2	33	32	97
TCF20	8	7	88	4	110	100	91
TCN2	19	15	79	2	163	152	93
TEF	13	12	92	2	237	232	98
TFIP11	35	31	89	2	319	297	93
THAP7	23	12	52	2	272	261	96
THOC5	45	40	89	2	247	233	94
TIMP3	16	9	56	2	985	971	99
TMEM184B	24	23	96	2	381	344	90
TMPRSS6	26	24	92	2	48	44	92
TNFRSF13C	3	3	100	0	9	9	100
TNRC6B	36	30	83	2	356	325	91
TOB2	4	4	100	0	232	229	99
TOMM22	7	6	86	2	137	134	98
TOP3B	52	46	88	3	253	236	93
TPST2	18	15	83	2	242	222	92
TRABD	34	25	74	2	220	210	95
TRIOBP	46	27	59	2	322	300	93
TRMT2A	35	27	77	3	320	267	83

(continue)

Table 7.1: Sensitivity and Specificity results (continued)

	Sensitivity				Specificity		
Gene	#exons ASPicDB	#exons ASPicDB in Test	Sensitivity (%)	Average $FP + FN$	#exons in Test	#exons covered 90%	Specificity (%)
TRMU	24	21	88	2	164	129	79
TSPO	12	9	75	2	159	154	97
TST	7	6	86	3	139	136	98
TTC38	32	19	59	3	70	69	99
TTLL1	22	18	82	2	109	98	90
TTLL12	23	20	87	3	275	274	100
TTLL8	20	20	100	1	33	33	100
TUBA8	17	11	65	2	82	72	88
TUBGCP6	41	26	63	2	160	154	96
TXN2	14	11	79	2	240	232	97
TXNRD2	46	36	78	2	208	203	98
TYMP	36	23	64	3	255	238	93
UBE2L3	20	14	70	3	594	591	99
UFD1L	31	25	81	2	169	135	80
UNC84B	44	38	86	2	470	450	96
UPB1	22	18	82	2	99	88	89
UPK3A	8	6	75	2	21	20	95
USP18	20	16	80	2	118	87	74
VPREB1	4	4	100	1	9	9	100
VPREB3	4	3	75	1	22	21	95
WBP2NL	21	20	95	2	65	48	74
XPNPEP3	32	24	75	3	227	122	54
XRCC6	43	35	81	2	616	595	97
YDJC	18	14	78	3	187	183	98
YPEL1	13	11	85	2	244	192	79
YWHAH	14	11	79	3	521	499	96
ZBED4	2	2	100	2	137	130	95
ZC3H7B	37	33	89	2	208	189	91
ZDHHC8	20	17	85	3	147	133	90
ZMAT5	17	17	100	3	89	88	99
ZNF280A	2	2	100	3	9	8	89
ZNF280B	10	6	60	2	102	100	98
ZNF70	2	2	100	0	30	23	77
ZNF74	23	17	74	2	157	153	97
ZNRF3	18	11	61	1	139	120	86
Total	7825	5917	76	2	64802	56416	88

Bibliography

[1] G. R. Abecasis, S. S. Cherny, W. O. Cookson, and L. R. Cardon, "Merlin–rapid analysis of dense genetic maps using sparse gene flow trees," *Nature Genetics*, vol. 30, no. 1, pp. 97–101, Jan. 2002. [Online]. Available: http://dx.doi.org/10.1038/ng786

[2] G. R. Abecasis and J. E. Wigginton, "Handling marker-marker linkage disequilibrium: pedigree analysis with clustered markers," *American J. Human Genetics*, vol. 77, no. 5, pp. 754–767, Nov. 2005. [Online]. Available: http://dx.doi.org/10.1086/497345

[3] P. Alimonti and V. Kann, "Some APX-completeness results for cubic graphs," *Theoretical Computer Science*, vol. 237, no. 1–2, pp. 123–134, 2000. [Online]. Available: http://dx.doi.org/10.1016/S0304-3975(98)00158-3

[4] N. Alon, R. Panigrahy, and S. Yekhanin, "Deterministic approximation algorithms for the nearest codeword problem," in *Proc. APPROX and RANDOM 2009*, ser. LNCS, I. Dinur, K. Jansen, J. Naor, and J. D. P. Rolim, Eds., vol. 5687. Springer, 2009, pp. 339–351. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-03685-9_26

[5] S. Arora, L. Babai, J. Stern, and Z. Sweedyk, "The hardness of approximate optima in lattices, codes, and systems of linear equations," *J. Computer and System Sciences*, vol. 54, no. 2, pp. 317–331, 1997.

[6] G. Ausiello, P. Crescenzi, V. Gambosi, G. Kann, A. Marchetti-Spaccamela, and M. Protasi, *Complexity and Approximation: Combinatorial optimization problems and their approximability properties.* Springer-Verlag, 1999.

[7] T. Barzuza. GREAL - software for the graph realization problem. [Online]. Available: http://acgt.cs.tau.ac.il/greal/

[8] T. Barzuza, J. S. Beckmann, R. Shamir, and I. Pe'er, "Computational problems in perfect phylogeny haplotyping: Xor-genotypes and tag SNPs," in *Proc. 15th Symp. on Combinatorial Pattern Matching (CPM)*, ser. LNCS, vol. 3109. Springer, Jul. 5–7, 2004, pp. 14–31. [Online]. Available: http://www.springerlink.com/content/lfv09pmjxbkcak3p

[9] T. Barzuza, J. S. Beckmann, R. Shamir, and I. Pe'er, "Computational problems in perfect phylogeny haplotyping: Typing without calling the allele," *IEEE Transactions on Computational Biology and Bioinformatics*, vol. 5, no. 1, pp. 101–109, 2008. [Online]. Available: http://doi.acm.org/10.1145/1343571.1343580

[10] J. R. Bitner, G. Ehrlich, and E. M. Reingold, "Efficient generation of the binary reflected Gray code and its applications," *Communications of the ACM*, vol. 19, no. 9, pp. 517–521, 1976. [Online]. Available: http://doi.acm.org/10.1145/360336.360343

[11] R. E. Bixby and D. K. Wagner, "An almost linear-time algorithm for graph realization," *Mathematics of Operations Research*, vol. 13, pp. 99–123, 1988.

[12] P. Bonizzoni, R. Rizzi, and G. Pesole, "Computational methods for alternative splicing prediction." *Briefings in Functional Genomics and Proteomics Advance*, vol. 5:1, pp. 46–51, 2006.

[13] P. Bonizzoni, "A linear-time algorithm for the perfect phylogeny haplotype problem," *Algorithmica*, vol. 48, no. 3, pp. 267–285, 2007. [Online]. Available: http://dx.doi.org/10.1007/s00453-007-0094-3

[14] P. Bonizzoni, G. Della Vedova, R. Dondi, and J. Li, "The haplotyping problem: An overview of computational models and solutions," *J. Computer Science and Technology*, vol. 18, no. 6, pp. 675–688, 2003. [Online]. Available: http://dx.doi.org/10.1007/BF02945456

[15] P. Bonizzoni, G. Della Vedova, R. Dondi, Y. Pirola, and R. Rizzi, "Minimum factorization agreement of spliced ESTs," in *Proc. 9th Int. Workshop on Algorithms in Bioinformatics (WABI)*, ser. LNCS, S. L. Salzberg and T. Warnow, Eds., vol. 5724. Springer, 2009, pp. 1–12. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-04241-6_1

[16] P. Bonizzoni, G. Della Vedova, R. Dondi, Y. Pirola, and R. Rizzi, "Pure parsimony xor haplotyping," in *Proc. 5th Int. Symp. on Bioinformatics Research and Applications (ISBRA)*, ser. LNCS, I. I. Mandoiu, G. Narasimhan, and Y. Zhang, Eds., vol. 5542. Springer, 2009, pp. 186–197. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-01551-9_19

[17] P. Bonizzoni, G. Della Vedova, Y. Pirola, and R. Rizzi, "Fast spliced alignment via maximal pairings of a pattern and a text," 2009, submitted.

[18] P. Bonizzoni, G. D. Vedova, R. Dondi, and J. Li, "The haplotyping problem: An overview of computational models and solutions," in *Genome Sequencing*

Technology and Algorithms, 1st ed., S. Kim, H. Tang, and E. R. Mardis, Eds. Artech House, Inc., 2007, pp. 151–181.

[19] D. Brett, J. Hanke, G. Lehmann, S. Haase, S. Delbruck, S. Krueger, J. Reich, and P. Bork, "EST comparison indicates 38% of human mRNAs contain possible alternative splice forms." *FEBS Letters*, vol. 474(1), pp. 83–86, 2000.

[20] D. G. Brown and I. M. Harrower, "Integer programming approaches to haplotype inference by pure parsimony," *IEEE Transactions on Computational Biology and Bioinformatics*, vol. 3, no. 2, pp. 141–154, 2006.

[21] J. Caceres and A. Kornblihtt, "Alternative splicing: multiple control mechanisms and involvement in human disease," *Trends Genet.*, vol. 18(4), pp. 186–193, 2002.

[22] Z. Cai, H. Sabaa, Y. Wang, R. Goebel, Z. Wang, J. Xu, P. Stothard, and G. Lin, "Most parsimonious haplotype allele sharing determination," *BMC Bioinformatics*, vol. 10, no. 1, pp. 115+, Apr. 2009. [Online]. Available: http://dx.doi.org/10.1186/1471-2105-10-115

[23] C. Cannings, E. A. Thompson, and M. H. Skolnick, "Probability functions on complex pedigrees," *Ann. Appl. Prob.*, vol. 10, pp. 26–61, 1978.

[24] T. Castrignanò, M. D'Antonio, A. Anselmo, D. Carrabino, A. D. D. Meo, A. M. D'Erchia, F. Licciulli, M. Mangiulli, F. Mignone, G. Pavesi, E. Picardi, A. Riva, R. Rizzi, P. Bonizzoni, and G. Pesole, "ASPicDB: A database resource for alternative splicing analysis," *Bioinformatics*, vol. 24, no. 10, pp. 1300–1304, 2008. [Online]. Available: http://dx.doi.org/10.1093/bioinformatics/btn113

[25] T. Castrignanò, R. Rizzi, I. G. Talamo, P. D. D. Meo, A. Anselmo, P. Bonizzoni, and G. Pesole, "ASPIC: a web resource for alternative splicing prediction and transcript isoforms characterization," *Nucleic Acid Research*, vol. 34, no. Web-Server-Issue, pp. 440–443, 2006. [Online]. Available: http://dx.doi.org/10.1093/nar/gkl324

[26] B. M.-Y. Chan, J. W.-T. Chan, F. Y. L. Chin, S. P. Y. Fung, and M.-Y. Kao, "Linear-time haplotype inference on pedigrees without recombinations," in *Proc. 6th Int. Workshop on Algorithms in Bioinformatics (WABI)*, ser. LNCS, P. Bucher and B. M. E. Moret, Eds., vol. 4175. Springer, 2006, pp. 56–67. [Online]. Available: http://dx.doi.org/10.1007/11851561_6

[27] M. Y. Chan, W.-T. Chan, F. Y. L. Chin, S. P. Y. Fung, and M.-Y. Kao, "Linear-time haplotype inference on pedigrees without recombinations and

mating loops," *SIAM J. on Computing*, vol. 38, no. 6, pp. 2179–2197, 2009. [Online]. Available: http://dx.doi.org/10.1137/080680990

[28] R. Cilibrasi, L. van Iersel, S. Kelk, and J. Tromp, "The complexity of the single individual SNP haplotyping problem," *Algorithmica*, vol. 49, no. 1, pp. 13–36, 2007. [Online]. Available: http://dx.doi.org/10.1007/s00453-007-0029-z

[29] A. Clark, "Inference of haplotypes from PCR-amplified samples of diploid populations," *Molecular Biology and Evolution*, vol. 7, no. 2, pp. 111–122, 1990.

[30] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. MIT Press, 2001.

[31] Q. D. and L. Beckmann, "Minimum-recombinant haplotyping in pedigrees," *American J. Human Genetics*, vol. 70, no. 6, pp. 1434–1445, 2002.

[32] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 39, no. 1, pp. 1–38, 1977. [Online]. Available: http://dx.doi.org/10.2307/2984875

[33] R. Diestel, *Graph Theory*, 3rd ed., ser. Graduate Texts in Mathematics. Springer-Verlag, Heidelberg, 2005, vol. 173.

[34] Z. Ding, V. Filkov, and D. Gusfield, "A linear-time algorithm for the perfect phylogeny haplotyping (pph) problem," *J. Computational Biology*, vol. 13, no. 2, pp. 522–553, 2006. [Online]. Available: http://dx.doi.org/10.1089/cmb.2006.13.522

[35] K. Doi, J. Li, and T. Jiang, "Minimum recombinant haplotype configuration on tree pedigrees," in *Proc. 3rd Int. Workshop on Algorithms in Bioinformatics (WABI)*, ser. LNCS, G. Benson and R. D. M. Page, Eds., vol. 2812. Springer, 2003, pp. 339–353. [Online]. Available: http://www.springerlink.com/content/ulnhtntqy14hjxg6

[36] R. Downey and M. Fellows, *Parameterized Complexity.* Springer Verlag, 1999.

[37] M. Elkin, Y. Emek, D. A. Spielman, and S.-H. Teng, "Lower-stretch spanning trees," *SIAM J. on Computing*, vol. 38, no. 2, pp. 608–628, 2008. [Online]. Available: http://link.aip.org/link/?SMJ/38/608/1

[38] R. C. Elson and J. Stewart, "A general model for the analysis of pedigree data," *Human Heredity*, vol. 21, pp. 523–542, 1971.

[39] L. Excoffier and M. Slatkin, "Maximum-likelihood estimation of molecular haplotype frequencies in a diploid population," *Molecular Biology and Evolution*, vol. 12, no. 5, pp. 921–927, 1995. [Online]. Available: http://mbe.oxfordjournals.org/content/12/5/921.abstract

[40] E. Eyras, M. Caccamo, V. Curwen, and M. Clamp, "ESTGenes: alternative splicing from ESTs in Ensembl," *Genome Research*, vol. 14, pp. 976–987, 2004.

[41] E. Fredkin, "Trie memory," *Communications of the ACM*, vol. 3, no. 9, pp. 490–499, 1960. [Online]. Available: http://doi.acm.org/10.1145/367390.367400

[42] S. Fujishige, "An efficient PQ-graph algorithm for solving the graph realization problem," *Journal of Computer and System Science*, vol. 21, pp. 63–68, 1980.

[43] P. Galante, N. Sakabe, N. Kirschbaum-Slager, and S. de Souza, "Detection and evaluation of intron retention events in the human transcriptome," *RNA*, vol. 10(5), pp. 757–65, 2004.

[44] R. G. Gallager, *Low-Density Parity-Check Codes*. M.I.T. Press, 1963.

[45] G. Gao, I. Hoeschele, P. Sorensen, and F. X. Du, "Conditional probability methods for haplotyping in pedigrees," *Genetics*, vol. 167, pp. 2055–2065, 2004.

[46] G. Gao, D. B. Allison, and I. Hoeschele, "Haplotyping methods for pedigrees," *Human heredity*, vol. 67, no. 4, pp. 248–266, 2009. [Online]. Available: http://dx.doi.org/10.1159/000194978

[47] M. Garey and D. Johnson, *Computer and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman, 1979.

[48] N. Garg, V. V. Vazirani, and M. Yannakakis, "Approximate max-flow min-(multi)cut theorems and their applications," *SIAM J. on Computing*, vol. 25, no. 2, pp. 235–251, 1996. [Online]. Available: http://link.aip.org/link/?SMJ/25/235/1

[49] F. Gavril and R. Tamari, "An algorithm for constructing edge-trees from hypergraphs," *Networks*, vol. 13, no. 3, pp. 377–388, 1983. [Online]. Available: http://dx.doi.org/10.1002/net.3230130306

[50] D. F. Gudbjartsson, T. Thorvaldsson, A. Kong, G. Gunnarsson, and A. Ingolfsdottir, "Allegro version 2," *Nature Genetics*, vol. 37, no. 10, pp. 1015–1016, Oct. 2005. [Online]. Available: http://dx.doi.org/10.1038/ng1005-1015

[51] S. Gupta, D. Zink, B. Korn, M. Vingron, and S. Haas, "Genome wide identification and classification of alternative splicing based on EST data," *Bioinformatics*, vol. 20(16), pp. 2579–2585, 2004.

[52] D. Gusfield, *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology.* Cambridge: Cambridge University Press, 1997.

[53] D. Gusfield, "Inference of haplotypes from samples of diploid populations: Complexity and algorithms," *J. Computational Biology*, vol. 8, no. 3, pp. 305–323, 2001.

[54] D. Gusfield, "Haplotyping as perfect phylogeny: Conceptual framework and efficient solutions," in *Proc. 6th Ann. Conf. on Research in Computational Molecular Biology (RECOMB)*, 2002, pp. 166–175.

[55] D. Gusfield, "Haplotype inference by pure parsimony," in *Proc. 14th Symp. on Combinatorial Pattern Matching (CPM)*, 2003, pp. 144–155.

[56] D. Gusfield, "An overview of combinatorial methods for haplotype inference," in *Computational Methods for SNPs and Haplotype Inference, DIMACS/RECOMB 2002 Satellite Workshop, Revised Papers*, ser. LNCS, S. Istrail, M. S. Waterman, and A. G. Clark, Eds., vol. 2983. Springer, 2004, pp. 9–25. [Online]. Available: http://www.springerlink.com/content/0yry96gwq40y67q1

[57] D. Gusfield and S. H. Orzack, "Haplotype inference," in *CRC Handbook on Bioinformatics*, S. Aluru, Ed., 2006, pp. 18/1–18/25.

[58] B. V. Halldórsson, V. Bafna, N. Edwards, R. Lippert, S. Yooseph, and S. Istrail, "A survey of computational methods for determining haplotypes," in *Computational Methods for SNPs and Haplotype Inference, DIMACS/RECOMB 2002 Satellite Workshop, Revised Papers*, ser. LNCS, S. Istrail, M. S. Waterman, and A. G. Clark, Eds., vol. 2983. Springer, 2004, pp. 26–47.

[59] E. Halperin and E. Eskin, "Haplotype reconstruction from genotype data using imperfect phylogeny," *Bioinformatics*, vol. 20, no. 12, pp. 1842–1849, Aug. 2004. [Online]. Available: http://dx.doi.org/10.1093/bioinformatics/bth149

[60] S. Heber, M. A. Alekseyev, S.-H. Sze, H. Tang, and P. A. Pevzner, "Splicing graphs and EST assembly problem," in *Proc. 10th Int. Conf. on Intelligent Systems for Molecular Biology (ISMB) (Suppl. of Bioinformatics)*, vol. 18, 2002, pp. 181–188.

[61] Y.-T. Huang, K.-M. Chao, and T. Chen, "An approximation algorithm for haplotype inference by maximum parsimony," *J. Computational Biology*, vol. 12, no. 10, pp. 1261–1274, 2005. [Online]. Available: http://dx.doi.org/10.1089/cmb.2005.12.1261

[62] R. R. Hudson, "Generating samples under a Wright-Fisher neutral model of genetic variation," *Bioinformatics*, vol. 18, no. 2, pp. 337–338, Feb. 2002. [Online]. Available: http://dx.doi.org/10.1093/bioinformatics/18.2.337

[63] T. Jiang and Y. Pirola, "Haplotype inference in pedigrees with recombinations and mutations," 2009, in preparation.

[64] Z. Kan, E. C. Rouchka, W. R. Gish, and D. J. States, "Gene structure prediction and alternative splicing analysis using genomically aligned ESTs," *Genome Research*, vol. 11(5), pp. 889–900, 2001.

[65] S. Khot, "On the power of unique 2-prover 1-round games," in *Proc. 34th Symp. Theory of Computing (STOC)*, 2002, pp. 767–775. [Online]. Available: http://doi.acm.org/10.1145/509907.510017

[66] N. Kim, S. Shin, and LeeSanghyuk, "ECgene: genome-based EST clustering and gene modeling for alternative splicing," *Genome Research*, vol. 15, no. 4, pp. 566–576, 2005.

[67] B. Kirkpatrick, J. Rosa, E. Halperin, and R. M. Karp, "Haplotype inference in complex pedigrees," in *Proc. 13th Ann. Conf. on Research in Computational Molecular Biology (RECOMB)*, ser. LNCS, S. Batzoglou, Ed., vol. 5541. Springer, 2009, pp. 108–120. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02008-7_8

[68] A. Kong, G. Masson, M. L. Frigge, A. Gylfason, P. Zusmanovich, G. Thorleifsson, P. I. Olason, A. Ingason, S. Steinberg, T. Rafnar, P. Sulem, M. Mouy, F. Jonsson, U. Thorsteinsdottir, D. F. Gudbjartsson, H. Stefansson, and K. Stefansson, "Detection of sharing by descent, long-range phasing and haplotype imputation," *Nature Genetics*, vol. 40, no. 9, pp. 1068–1075, Sep. 2008. [Online]. Available: http://dx.doi.org/10.1038/ng.216

[69] L. Kruglyak, M. J. Daly, M. P. Reeve-Daly, and L. E., "Parametric and nonparametric linkage analysis: a unified multipoint approach," *American J. Human Genetics*, vol. 58, no. 6, pp. 1346–1363, Jun. 1996.

[70] G. Lancia, "The phasing of heterozygous traits: Algorithms and complexity," *Computers & Mathematics with Applications*, vol. 55, no. 5, pp. 960–969, Mar. 2008. [Online]. Available: http://dx.doi.org/10.1016/j.camwa.2006.12.089

[71] G. Lancia, M. C. Pinotti, and R. Rizzi, "Haplotyping populations by pure parsimony: Complexity of exact and approximation algorithms," *INFORMS Journal on Computing*, vol. 16, no. 4, pp. 348–359, 2004.

[72] G. Lancia and R. Rizzi, "A polynomial case of the parsimony haplotyping problem," *Operations Research Letters*, vol. 34, no. 3, pp. 289–295, 2006.

[73] G. Lancia and P. Serafini, "A set-covering approach with column generation for parsimony haplotyping," *INFORMS J. on Computing*, vol. 21, no. 1, pp. 151–166, Jan. 2009. [Online]. Available: http://dx.doi.org/10.1287/ijoc.1080.0285

[74] E. Lander and P. Green, "Construction of multilocus genetic linkage maps in human," *Proceedings of the National Academy of Sciences USA*, vol. 84, pp. 2363–2367, 1987.

[75] J. Leipzig, P. Pevzner, and S. Heber, "The Alternative Splicing Gallery (ASG): bridging the gap between genome and transcriptome," *Nucleic Acid Research*, vol. 32, no. 13, pp. 3977–3983, 2004. [Online]. Available: http://nar.oxfordjournals.org/cgi/reprint/32/13/3977.pdf

[76] J. Li and T. Jiang, "Efficient inference of haplotypes from genotypes on a pedigree," *J. Bioinformatics and Computational Biology*, vol. 1, no. 1, pp. 41–69, Apr. 2003.

[77] J. Li and T. Jiang, "Efficient rule-based haplotyping algorithms for pedigree data," in *Proc. 7th Ann. Conf. on Research in Computational Molecular Biology (RECOMB)*, 2003, pp. 197–206. [Online]. Available: http://doi.acm.org/10.1145/640075.640101

[78] J. Li and T. Jiang, "Computing the minimum recombinant haplotype configuration from incomplete genotype data on a pedigree by integer linear programming," *J. Computational Biology*, vol. 12, no. 6, pp. 719–739, 2005. [Online]. Available: http://dx.doi.org/10.1089/cmb.2005.12.719

[79] J. Li and T. Jiang, "A survey on haplotyping algorithms for tightly linked markers," *Journal of bioinformatics and computational biology*, vol. 6, no. 1, pp. 241–259, Feb. 2008. [Online]. Available: http://view.ncbi.nlm.nih.gov/pubmed/18324755

[80] X. Li and J. Li, "An almost linear time algorithm for a general haplotype solution on tree pedigrees with no recombination and its extensions." *J. Bioinformatics and Computational Biology*, vol. 7, no. 3, pp. 521–545, Jun. 2009. [Online]. Available: http://view.ncbi.nlm.nih.gov/pubmed/19507288

[81] X. Li and J. Li, "Efficient haplotype inference from pedigrees with missing data using linear systems with disjoint-set data structures." *Computational Systems Bioinformatics Conference*, vol. 7, pp. 297–308, 2008. [Online]. Available: http://view.ncbi.nlm.nih.gov/pubmed/19642289

[82] Z.-P. Li, W. Zhou, X.-S. Zhang, and L. Chen, "A parsimonious tree-grow method for haplotype inference," *Bioinformatics*, vol. 21, no. 17, pp. 3475–3481, 2005. [Online]. Available: http://dx.doi.org/10.1093/bioinformatics/bti572

[83] L. Liu, X. Chen, J. Xiao, and T. Jiang, "Complexity and approximation of the minimum recombination haplotype configuration problem," in *Proc. 16th Int. Symp. on Algorithms and Computation (ISAAC)*, 2005, pp. 370–379. [Online]. Available: http://dx.doi.org/10.1007/11602613_38

[84] L. Liu, X. Chen, J. Xiao, and T. Jiang, "Complexity and approximation of the minimum recombinant haplotype configuration problem," *Theoretical Computer Science*, vol. 378, no. 3, pp. 316–330, Jun. 2007. [Online]. Available: http://dx.doi.org/10.1016/j.tcs.2007.02.036

[85] L. Liu and T. Jiang, "Linear-time reconstruction of zero-recombinant mendelian inheritance on pedigrees without mating loops." in *Proc. 17th Int. Conf. on Genome Informatics*, vol. 19, 2007, pp. 95–106. [Online]. Available: http://view.ncbi.nlm.nih.gov/pubmed/18546508

[86] A. Makhorin. GLPK - the GNU Linear Programming Kit. GNU Project. [Online]. Available: http://www.gnu.org/software/glpk/

[87] C. Meyer, *Matrix Analysis and Applied Linear Algebra.* Philadelphia: SIAM, 2000.

[88] S. Myers, L. Bottolo, C. Freeman, G. McVean, and P. Donnelly, "A fine-scale map of recombination rates and hotspots across the human genome." *Science*, vol. 310, no. 5746, pp. 321–324, Oct. 2005. [Online]. Available: http://dx.doi.org/10.1126/science.1117196

[89] T. Niu, Z. Qin, X. Xu, and J. Liu, "Bayesian haplotype inference for multiple linked single-nucleotide polymorphisms," *American J. Human Genetics*, vol. 710, pp. 157–169, 2002.

[90] T. Niu, "Algorithms for inferring haplotypes," *Genetic epidemiology*, vol. 27, no. 4, pp. 334–347, Dec. 2004. [Online]. Available: http://dx.doi.org/10.1002/gepi.20024

[91] J. R. O'Connell, "Zero-recombinant haplotyping: applications to fine mapping using SNPs," *Genetic Epidemiolgy*, vol. 19, no. Suppl. 1, pp. S64–S70, 2000.

[92] S. H. Orzack, D. Gusfield, J. Olson, S. Nesbitt, L. Subrahmanyan, and V. P. Stanton, "Analysis and exploration of the use of rule-based algorithms and consensus methods for the inferral of haplotypes," *Genetics*, vol. 165, no. 2, pp. 915–928, Oct. 2003. [Online]. Available: http://www.genetics.org/content/165/2/915.abstract

[93] C. H. Papadimitriou, *Computational Complexity*. Addison Wesley, 1993.

[94] C. Papadimitriou and M. Yannakakis, "Optimization, approximation and complexity classes," *J. Computer and System Sciences*, vol. 43, pp. 425–440, 1991.

[95] J. Pearl, "Reverend bayes on inference engines: A distributed hierarchical approach," in *Proc. of the American Ass. of Artificial Intelligence National Conference on AI*, Pittsburgh, PA, 1982, pp. 133–136.

[96] R. Raz and S. Safra, "A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP," in *Proc. 29th Symp. Theory of Computing (STOC)*, 1997, pp. 475–484. [Online]. Available: http://doi.acm.org/10.1145/258533.258641

[97] R. V. Satya and A. Mukherjee, "An optimal algorithm for perfect phylogeny haplotyping," *J. Computational Biology*, vol. 13, no. 4, pp. 897–928, 2006. [Online]. Available: http://dx.doi.org/10.1089/cmb.2006.13.897

[98] C. Savage, "A survey of combinatorial Gray codes," *SIAM Review*, vol. 39, no. 4, pp. 605–629, 1997. [Online]. Available: http://dx.doi.org/10.1137/S0036144595295272

[99] D. J. Schaid, S. K. McDonnell, L. Wang, J. M. Cunningham, and T. S. N., "Caution on pedigree haplotype inference with software that assumes linkage equilibrium," *American J. Human Genetics*, vol. 71, no. 4, pp. 992–995, Oct. 2002.

[100] T. Smith and M. Waterman, "Identification of common molecular subsequences," *J. Molecular Biology*, vol. 147, pp. 195–197, 1981.

[101] E. Sobel and K. Lange, "Descent graphs in pedigree analysis: applications to haplotyping, location scores, and marker-sharing statistics." *American J. Human Genetics*, vol. 58, no. 6, pp. 1323–1337, Jun. 1996. [Online]. Available: http://view.ncbi.nlm.nih.gov/pubmed/8651310

[102] E. Sobel, K. Lange, J. R. O'Connell, and D. E. Weeks, "Haplotyping algorithms," in *Genetic mapping and DNA sequencing*, ser. IMA Volumes in Mathematics and its Applications, vol. 81, 1996, pp. 89–110.

[103] Y. S. Song, Y. Wu, and D. Gusfield, "Algorithms for imperfect phylogeny haplotyping (IPPH) with a single homoplasy or recombination event," in *Proc. 5th Int. Workshop on Algorithms in Bioinformatics (WABI)*, ser. LNCS, R. Casadio and G. Myers, Eds., vol. 3692. Springer, 2005, pp. 152–164. [Online]. Available: http://dx.doi.org/10.1007/11557067_13

[104] M. Stephens and P. Donnelly, "A comparison of bayesian methods for haplotype reconstruction from population genotype data," *American J. Human Genetics*, vol. 73, no. 5, pp. 1162–1169, Nov. 2003. [Online]. Available: http://dx.doi.org/10.1086/379378

[105] M. Stephens, N. Smith, and P. Donnelly, "A new statistical method for haplotype reconstruction from population data," *American J. Human Genetics*, vol. 68, pp. 978–989, 2001.

[106] The International HapMap Consortium, "A haplotype map of the human genome," *Nature*, vol. 437, no. 7063, pp. 1299–1320, 2005. [Online]. Available: http://dx.doi.org/10.1038/nature04226

[107] The International HapMap Consortium, "A second generation human haplotype map of over 3.1 million SNPs," *Nature*, vol. 449, no. 7164, pp. 851–861, Oct. 2007. [Online]. Available: http://dx.doi.org/10.1038/nature06258

[108] W. T. Tutte, "An algorithm for determining whether a given binary matroid is graphic," *Proceedings of the American Mathematical Society*, vol. 11, no. 6, pp. 905–917, 1960.

[109] L. van Iersel, J. Keijsper, S. Kelk, and L. Stougie, "Shorelines of islands of tractability: Algorithms for parsimony and minimum perfect phylogeny haplotyping problems," *IEEE Transactions on Computational Biology and Bioinformatics*, vol. 5, no. 2, pp. 301–312, 2008. [Online]. Available: http://doi.acm.org/10.1145/1371585.1371599

[110] W.-B. Wang and T. Jiang, "Efficient inference of haplotypes from genotypes on a pedigree with mutations and missing alleles (extented abstract)," in

Proc. 20th Symp. on Combinatorial Pattern Matching (CPM), ser. LNCS, G. Kucherov and E. Ukkonen, Eds., vol. 5577. Springer, 2009, pp. 353–367. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02441-2_31

[111] S. Wheelan, D. Church, and J. Ostell, "Spidey: a tool for mRNA-to-genomic alignments," *Genome Research*, vol. 11(11), pp. 1952–1957, 2001.

[112] E. M. Wijsman, "A deductive method of haplotype analysis in pedigrees." *American J. Human Genetics*, vol. 41, no. 3, pp. 356–373, Sep. 1987. [Online]. Available: http://view.ncbi.nlm.nih.gov/pubmed/3115093

[113] T. D. Wu and C. K. Watanabe, "GMAP: a genomic mapping and alignment program for mRNA and EST sequence," *Bioinformatics*, vol. 21, no. 9, pp. 1859–1875, 2005. [Online]. Available: http://dx.doi.org/10.1093/bioinformatics/bti310

[114] J. Xiao, L. Liu, L. Xia, and T. Jiang, "Fast elimination of redundant linear equations and reconstruction of recombination-free mendelian inheritance on a pedigree," in *Proc. 18th ACM/SIAM Symp. on Discrete Algorithms (SODA)*, N. Bansal, K. Pruhs, and C. Stein, Eds. SIAM, 2007, pp. 655–664. [Online]. Available: http://doi.acm.org/10.1145/1283383.1283454

[115] J. Xiao, L. Liu, L. Xia, and T. Jiang, "Efficient algorithms for reconstructing zero-recombinant haplotypes on a pedigree based on fast elimination of redundant linear equations," *SIAM J. on Computing*, vol. 38, no. 6, pp. 2198–2219, 2009. [Online]. Available: http://dx.doi.org/10.1137/070687591

[116] J. Xiao, T. Lou, and T. Jiang, "An efficient algorithm for haplotype inference on pedigrees with a small number of recombinants (extended abstract)," in *Proc. 17th European Symp. on Algorithms (ESA)*, ser. LNCS, A. Fiat and P. Sanders, Eds., vol. 5757. Springer, 2009, pp. 325–336. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-04128-0_30

[117] W. Xiao and P. J. Oefner, "Denaturing high-performance liquid chromatography: A review," *Human Mutation*, vol. 17, no. 6, pp. 439–474, 2001. [Online]. Available: http://dx.doi.org/10.1002/humu.1130

[118] H. Xie, W. Zhu, A. Wasserman, V. Grebinskiy, A. Olson, and L. Mintz, "Computational analysis of alternative splicing using EST tissue information," *Genomics*, vol. 80(3), pp. 326–330, 2002.

[119] Y. Xing, A. Resch, and C. Lee, "The multiassembly problem: reconstructing multiple transcript isoforms from EST fragment mixtures," *Genome Research*, vol. 14, no. 3, pp. 426–441, Mar. 2004. [Online]. Available: http://dx.doi.org/10.1101/gr.1304504

[120] Q. Xu, B. Modrek, and C. Lee, "Genome-wide detection of tissue-specific alternative splicing in the human transcriptome," *Nucleic Acid Research*, vol. 30(17), pp. 3754–3766, 2002.

[121] Y. Xue, Q. Wang, Q. Long, B. L. Ng, H. Swerdlow, J. Burton, C. Skuce, R. Taylor, Z. Abdellah, Y. Zhao, D. G. MacArthur, M. A. Quail, N. P. Carter, H. Yang, and C. Tyler-Smith, "Human Y chromosome base-substitution mutation rate measured by direct sequencing in a deep-rooting pedigree," *Current Biology*, vol. 19, no. 17, pp. 1453–1457, Sep. 2009. [Online]. Available: http://dx.doi.org/10.1016/j.cub.2009.07.032

[122] M. Yannakakis, "Node-and edge-deletion NP-complete problems," in *Proc. 10th Symp. Theory of Computing (STOC)*. ACM, 1978, pp. 253–264. [Online]. Available: http://dx.doi.org/10.1145/800133.804355

[123] K. Zhang, F. Sun, and H. Zhao, "HAPLORE: a program for haplotype reconstruction in general pedigrees without recombination," *Bioinformatics*, vol. 21, no. 1, pp. 90–103, Jan. 2005. [Online]. Available: http://dx.doi.org/10.1093/bioinformatics/bth388