UNIVERSIDADE TÉCNICA DE LISBOA

INSTITUTO SUPERIOR TÉCNICO

Satisfiability-based Algorithms for Haplotype Inference

Ana Sofia Soeiro da Graça
(Licenciada)

Dissertação para obtenção do Grau de Doutor em
Engenharia Informática e de Computadores

Orientadora:	Doutora Maria Inês Camarate de Campos Lynce de Faria
Co-orientadores:	Doutor João Paulo Marques da Silva Doutor Arlindo Manuel Limede de Oliveira

Júri

Presidente:	Presidente do Conselho Científico do IST
Vogais:	Doutor Arlindo Manuel Limede de Oliveira Doutor João Paulo Marques da Silva Doutor Eran Halperin Doutor Ludwig Krippahl Doutor Luís Jorge Brás Monteiro Guerra e Silva Doutora Maria Inês Camarate de Campos Lynce de Faria

Janeiro de 2011

Abstract

Identifying genetic variations has an important role in human genetics. Single Nucleotide Polymorphisms (SNPs) are the most common genetic variations and haplotypes encode SNPs in a single chromosome. Unfortunately, current technology cannot obtain directly the haplotypes. Instead, genotypes, which represent the combined data of two haplotypes on homologous chromosomes, are acquired. The haplotype inference problem consists in obtaining the haplotypes from the genotype data. This problem is a key issue in genetics and a computational challenge.

This dissertation proposes two Boolean optimization methods for haplotype inference, one population-based and another pedigree-based, which are competitive both in terms of accuracy and efficiency. The proposed method for solving the haplotype inference by pure parsimony (HIPP) approach is shown to be very robust, representing the state of the art HIPP method. The pedigree-based method, which takes into consideration both family and population information, shows to be more accurate than the existing methods for haplotype inference in pedigrees. Furthermore, this dissertation contributes to better understanding the HIPP problem by providing a vast comparison between HIPP methods.

Keywords: haplotype inference, Boolean satisfiability (SAT), pseudo-Boolean optimization (PBO), maximum satisfiability (MaxSAT), pure parsimony (HIPP), minimum recombinant (MRHC).

Sumário

A identificação de variações genéticas desempenha um papel fundamental no estudo da genética humana. Os polimorfismos de nucleótido único (SNPs) correspondem às variações genéticas mais comuns entre os seres humanos e os haplótipos codificam SNPs presentes num único cromossoma. Actualmente, por limitações de índole tecnológica, a obtenção directa de haplótipos não é exequível. Por outro lado, são obtidos os genótipos, que correspondem à informação conjunta do par de cromossomas homólogos. O problema da inferência de haplótipos consiste na obtenção de haplótipos a partir de genótipos. Este paradigma representa um aspecto fundamental em genética e também um interessante desafio computacional.

Esta dissertação propõe dois métodos de optimização Booleana para inferência de haplótipos, um populacional e outro em linhagens genealógicas, cujo desempenho é assinalável em eficiência e precisão. O método para inferência de haplótipos por parcimónia pura (HIPP) em populações mostra-se muito robusto e representa o estado da arte entre os modelos HIPP. O modelo proposto para inferência de haplótipos em linhagens genealógicas, que combina informação familiar e populacional, representa um método mais preciso que os congeneres. Ademais, esta dissertação representa um contributo relevante na compreensão do problema HIPP, fornecendo uma extensa comparação entre métodos.

Palavras chave: inferência de haplótipos, satisfação Booleana (SAT), optimização pseudo-Booleana (PBO), satisfação máxima (MaxSAT), parcimónia pura (HIPP), recombinação mínima (MRHC).

Acknowledgments

"Bear in mind that the wonderful things that you learn in your schools are the work of many generations, produced by enthusiastic effort and infinite labor in every country of the world. All this is put into your hands as your inheritance in order that you may receive it, honor it, and add to it, and one day faithfully hand it on to your children."

Albert Einstein

I have always been fascinated by science, its mysteries and curiosities. Understanding the world where we live and, in particular, the human beings, requires the effort and hard work of many generations. This thesis represents a modest contribution to the development of science, my little drop in the ocean. Studying such a multidisciplinary topic was a big challenge and achieving this goal represents a very important step for me. However, I could not have reached this point without the help of several people.

First of all, I owe a special thanks to Professor Inês Lynce for being such a dedicated supervisor and for her contributions to this project. She taught me how to do research. She is an example of an excellent researcher and also a very good friend. She motivated me from the very beginning and always gave me the right words.

I also thank Professor João Marques da Silva for his enthusiasm and motivation at work, for his interesting contributions and for sharing with me the vast experience he has in the SAT area.

I am also grateful to Professor Arlindo Oliveira for suggesting me this PhD topic, for his interesting suggestions and his help with the biological issues - the hardest for me as a mathematician.

I thank all my colleges and professors at Instituto Superior Técnico, INESC-ID, University of Southampton and University College of Dublin.

I am eternally grateful to my parents, Otília and António, who always help me following my dreams and support my decisions, sometimes even without understanding. I dedicate my thesis to them.

I owe a word of gratitude to my brother and to my sister-in-law. Paulo and Cristina gave me strength to follow this project and they gave me the best advices when I needed them. I admire their perseverance in life. Most of all, I thank them for giving me three lovely nephews that bring so much happiness into my life.

A special thanks to my nephews for renewing in me the imagination and curiosity, typical from children, and so important for who works in research.

I am very grateful to Renato. He is able to solve my problems even before I could realize them. He supported me in the hardest moments and always believed I could do a good work out of this project.

Finally, I also thank all my friends for being present.

"We ourselves feel that what we are doing is just a drop in the ocean. But the ocean would be less because of that missing drop."

Mother Teresa

Contents

List of Figures

List of Tables

List of Algorithms

1

Introduction

"Reason is the pace; increase of science, the way; and the benefit of mankind, the end."

Thomas Hobbes

Recent advances in sequencing technologies have enabled decoding the genome of thousands of individuals efficiently and inexpensively. Such information has offered investigators new opportunities of understanding the genetic differences between human beings, and afterward mapping such differences to common human diseases. The International HapMap Project [1] [153, 154, 155] and the 1000 Genomes Project [2] represent significant efforts to catalog the genetic variations among human beings.

The DNA, *deoxyribonucleic acid*, is a long polymer whose structural units are the nucleotides: adenine (A), thymine (T), guanine (G) and cytosine (C). Moreover, the DNA contains the genetic information and is passed on from parents to offspring. Differences in the sequence of nucleotides that constitutes the DNA explain the genetic variations that influence how people differ in their risk of disease or their response to drugs.

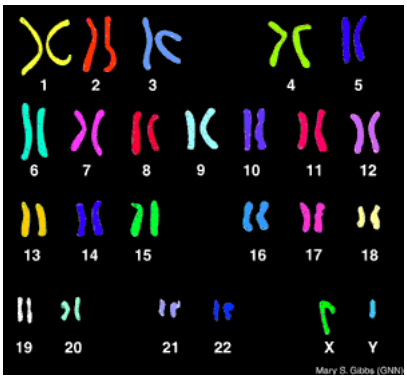
At the forehead of human variation at genetic level are Single Nucleotide Polymorphisms, or SNPs (pronounced *snips*). A SNP is a single DNA position where a mutation has occurred and one nucleotide was replaced by a different one. For example, a SNP occurs when a sequence AG**T**TCG is modified to AG**A**TCG. Moreover, the least frequent nucleotide must be present in a significant percentage of the population (e.g. 1%). SNPs are the most common genetic variations. The human genome has millions of SNPs [155], which are cataloged in dbSNP [3], the public repository for DNA

[1] *http://www.hapmap.org*

[2] *http://www.1000genomes.org*

[3] *http://www.ncbi.nlm.nih.gov/projects/SNP*

Figure 1.1: Human genome [a]



[a]Figure taken from *http://www.genomenewsnetwork.org/resources/whats_a_genome/Chp1_2_1.shtml*

variations [146].

What is the haplotype inference problem?

Humans are diploid organisms, which means that our genome is organized in pairs of homologous chromosomes, representing the maternal and paternal chromosome. Figure 1.1 illustrates the 23 pairs of human chromosomes. Homologous chromosomes have the same gene sequences, each one being inherited from one parent.

Haplotypes correspond to the sequence of correlated SNPs in a single chromosome. Therefore, each individual has two haplotypes for a given stretch of the genome. Technological limitations prevent geneticists from acquiring experimentally the data from a single chromosome, the haplotypes. Instead, genotypes are obtained. *Genotypes* correspond to the mixed data of homologous haplotypes. This means that, at each DNA position, it is not possible to know whether the individual has inherited the same nucleotide from both parents (homozygous position) or whether the individual has inherited distinct nucleotides from each parent (heterozygous position). An example of a genotype, with seven positions of which three are heterozygous positions, is $T\ A/G\ C\ C/T\ G\ A\ C/T$. The problem of obtaining the haplotypes from the genotypes is known as *haplotype inference*, *haplotyping* or *phasing*.

Figure 1.2 presents an example of the haplotype inference problem. Given the genotype g, the haplotyping problem aims at choosing the pair of haplotypes, among the four possible solutions, A-D, that originated genotype g.

Figure 1.2: Example of the haplotype inference problem

	genotype g	T	A/G	C	C/T	G	A	C/T
Solution A	haplotype h^a	T	G	C	T	G	A	C
	haplotype h^b	T	A	C	C	G	A	T
Solution B	haplotype h^a	T	A	C	T	G	A	T
	haplotype h^b	T	G	C	C	G	A	C
Solution C	haplotype h^a	T	A	C	T	G	A	C
	haplotype h^b	T	G	C	C	G	A	T
Solution D	haplotype h^a	T	A	C	C	G	A	C
	haplotype h^b	T	G	C	T	G	A	T

Why is haplotype inference important?

Although a number of disease association studies can be performed using only single-locus alleles or genotype frequencies, haplotype information is essential to the detailed analysis of the mechanisms of a disease. The identification of haplotypes enables to perform haplotype-based association tests with diseases. This is particularly important in genome-wide association studies. Indeed, haplotypic association studies have found DNA positions associated with diseases that are not genome-wide significant using single-marker tests [18]. Moreover, most imputation methods require haplotypic data rather than genotypes [81].

Information about human haplotypes is significantly important in clinic medicine [30]. Haplotypes are more informative than genotypes and, in some cases, can better predict the severity of a disease or even be responsible for producing a specific phenotype. In some cases of medical transplants, patients who closely match the donor haplotypes are predicted to have more success on the transplant outcome [128]. Moreover, medical treatment can be customized, based on the genetic information of the patient, because individual responses to drugs can be associated with a specific haplotype [48]. Furthermore, haplotypes help inferring population demographic histories [112].

Why a computer science dissertation in haplotype inference is of interest?

Computational biology is a research field in extensive expansion, where computer science and mathematics knowledge is applied to biological problems.

Despite being an important biological problem, haplotype inference turned also to be a challenging mathematical problem, and, therefore, has deserved significant attention by the mathematical and computer science communities. The mathematical approaches to haplotype inference can be statistical [151, 19] or combinatorial [26, 61, 62]. Within the combinatorial approaches, it is noteworthy the haplotype inference by pure parsimony (HIPP) [62] and the minimum recombinant haplotype configuration (MRHC) problems [64, 138]. Given a set of genotypes, the HIPP approach aims at finding a haplotype inference solution that uses the smallest number of distinct haplotypes. Given the set of genotypes from individuals of the same family, the MRHC approach searches the haplotype inference solution that minimizes the number of recombination events. Both HIPP and MRHC are challenging computational problems belonging to the complexity class of APX-hard problems [92, 110].

Since these problems are computationally hard, there has been significant effort towards producing efficient solvers, using distinct types of methodologies, including branch-and-bound [164], integer linear programming (ILP) [62, 16, 17] and answer set programming (ASP) [39]. Also Boolean satisfiability (SAT) has been successfully applied to haplotype inference [114, 130], producing very competitive results when compared to alternative methods [116].

Why is this dissertation relevant?

This dissertation contributes with alternative approaches for solving the haplotype inference problem, which are competitive both in terms of efficiency and accuracy. We explored the well-known HIPP approach, suggesting an efficient method which addresses the state of the art for solving the problem. In addition, an accurate family-based approach for haplotype inference is proposed.

Alternative approaches to haplotype inference are important for complementing the existing methods, because there exists no approach which can infer haplotypes without inaccuracies. Moreover, this dissertation contributes to better understanding the haplotype inference problem, in particular, the HIPP approach.

The next section outlines the main contributions of this thesis.

1.1 Contributions

Several contributions were developed in the context of this thesis. All the contributions refer to the haplotype inference problem.

- First, we propose a novel method, named *RPoly*, for solving the haplotype inference by pure parsimony problem, with the goal of outperforming the efficiency of the existing methods. Indeed, this goal has been achieved. The evaluation of exact HIPP solvers, in a considerable set of well-known instances, has shown that our method is, in general, faster than other methods and, furthermore, that RPoly is able to solve a significantly larger set of instances, in a reasonable amount of time. In addition, RPoly is able to deal with missing genotype sites. A first version of this work has resulted in a publication in the *Algebraic Biology 2007* conference [55], whereas an improved version of the model was published in the *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming 2008* [56]. This contribution also resulted in a publication in the *Annals of Operations Research* [57].

- A second main contribution is a novel approach to haplotype inference in pedigrees, with the main goal of obtaining an accurate method for solving this problem. The resulting haplotype inference model, named *PedRPoly*, is shown to be quite competitive and more accurate than the existing methods for haplotype inference in pedigrees. A preliminary version was presented at the *Workshop on Constraint Based Methods for Bioinformatics 2009* [51], whereas a further improved version was published in the proceedings of the *Algebraic and Numeric Biology 2010* conference [52].

- Third, we contributed to better understanding the haplotype inference by pure parsimony problem, providing a complete study of the approach. This work resulted in three publications: a chapter in the book *Mathematical Approaches to Polymer Sequence Analysis* [54], a survey in the *Journal of Computational Biology* [53] and an overview in the proceedings of the *International Conference on Tools with Artificial Intelligence* [113].

In addition, an approximation and upper bound algorithm for haplotype inference by pure parsimony was proposed in the context of this thesis [125].

Finally, the different haplotype inference models developed during this research work represent challenging combinatorial optimization problems and provide an interesting new set of challenging

Boolean optimization benchmarks, some of which cannot yet be solved by state of the art pseudo-Boolean optimization and maximum satisfiability solvers. Also, these challenging instances have inspired recent research work in the development of new solvers [124].

The work developed in the scope of this dissertation has already tens of citations by other authors.

1.2 Thesis Outline

This dissertation has a total of nine chapters. The first chapter is the introduction and the last chapter presents the conclusions and future work. Chapters 2, 3 and 4 describe the problems related to the subject of this thesis. Chapters 5, 6, 7 and 8 present the main contributions of this thesis. More precisely, this dissertation is organized as follows:

Chapter 2 reviews techniques used to solve discrete constraint problems, including integer linear programming, Boolean satisfiability, pseudo-Boolean optimization and maximum satisfiability. These techniques will be used in the following chapters in order to solve haplotype inference models.

Chapter 3 provides a description of the haplotype inference problem and overviews the population and pedigree-based approaches to tackle the haplotype inference problem.

Chapter 4 reviews related work in haplotype inference by pure parsimony, including preprocessing techniques, exact and heuristic methods and complexity studies.

Chapter 5 describes our haplotype inference by pure parsimony model, named *RPoly*. The new approach is a pseudo-Boolean optimization model based on previous models but with further improvements. These improvements include eliminating key symmetries, reducing the size of the model, integrating lower bounds and cardinality constraints. Moreover, RPoly is able to deal with missing data.

Chapter 6 presents experimental results regarding the pure parsimony haplotyping. First, the experimental data and setup is presented. Second, the performance of all existent HIPP solvers is compared in terms of efficiency. We conclude that RPoly is the most efficient HIPP method. Third, the tightness of lower and upper bounds is tested. Finally, the accuracy of the pure parsimony

approach is compared with other haplotype inference methods.

Chapter 7 describes a new approach to haplotype inference in pedigrees. The new approach is named minimum recombinant maximum parsimony (MRMP). A new Boolean constraint optimization model for solving the MRMP problem, named *PedRPoly*, is proposed, including the use of advanced modeling techniques and an adequate constraint solver, which results in an efficient haplotype inference tool.

Chapter 8 presents the experimental results regarding haplotype inference in pedigrees. We conclude that PedRPoly has better accuracy than existing haplotype inference methods for pedigrees.

Chapter 9 concludes the dissertation and suggests future research directions.

2

Discrete Constraint Solving

This chapter introduces a number of existent mathematical formalisms for solving discrete constraint problems, with special focus on Boolean constraint problems. These formalisms will be relevant later in this dissertation for modeling the haplotype inference problem.

The following sections aim at being an introduction to integer linear programming (ILP), Boolean satisfiability (SAT), pseudo-Boolean optimization (PBO) and maximum satisfiability (MaxSAT). All these problems are solved by satisfying a number of constraints. Moreover, the problems can include a cost function which must be optimized. There are other discrete constraint solving approaches, such as constraint programming (CP) and approximation methods (e.g. local search), but they are out of the scope of this thesis.

2.1 Integer Linear Programming (ILP)

Linear programming (LP) is a well-known methodology for solving constraint optimization problems, first proposed by Kantorovich, in 1939 [83]. LP is a field of research rather studied and with several important results. In this dissertation, we only provide a brief introduction to the subject.

2.1.1 Preliminaries

Many practical optimization problems can be expressed using linear programming formulations. The goal of linear programming is to find an assignment of values to a set of variables, $x_1, \ldots, x_n$, which optimizes (minimizes or maximizes) a linear function f,

$$f(x_1, \ldots, x_n) = \sum_{j=1}^{n} c_j x_j, \tag{2.1}$$

subject to a set of m linear constraints (equalities or inequalities),

$$\sum_{j=1}^{n} a_{i\,j} x_j \geq b_i, \tag{2.2}$$

where $c_j, a_{i\,j}, b_i \in \mathbb{R}$, $1 \leq i \leq m$, $1 \leq j \leq n$ and $n, m \in \mathbb{N}$.

Function f is known as the *objective function* or *cost function*, derived from the importance of LP in microeconomics and business management. A standard example of an LP application consists in maximizing the business profit of a company subject to constraints evolving the cost of production schemes.

In 1947, Dantzig developed the *simplex* method to solve LP problems. Although the complexity of simplex is exponential in the worst case, this algorithm is efficient in most cases and is very useful in practice. In 1980, the LP problem was proved to be solvable in polynomial time [86] and some polynomial methods have been presented. Nonetheless, *simplex* is still widely used. CPLEX [76] [1] is the most well-known commercial tool used to solve the LP problem applying the simplex procedure. Indeed, CPLEX is able to solve various problems, including linear programming, quadratic programming, quadratically constrained programming and mixed integer programming problems.

Although many problems can be naturally formulated in linear programming, there are other problems which require further restrictions. *Integer linear programming* (ILP) is a special case of linear programming (LP) with the additional constraint that variables must take on integral values. This restriction to a discrete domain is sufficient to increase the complexity of the problem to NP-complete [135].

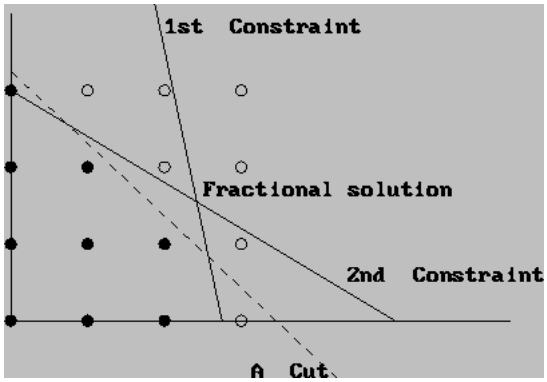
2.1.2 ILP Algorithms

Despite the complexity of ILP, several techniques have been studied in order to solve ILP problems using a reasonable amount of time, in several practical situations [49].

The usual method for solving ILP problems is *branch-and-bound* search. As the name indicates, the process consists in branching the problem recursively in smaller subproblems. Several branching heuristics have been studied, and its choice can have a relevant impact in the performance of the search algorithm. Different choices on branching conduct to different parts of the search space that must be explored.

[1] *www.cplex.com*

Figure 2.1: Example of a valid cut, during a branch-and-cut search [a]



[a]Figure taken from http://mat.gsia.cmu.edu/orclass/integer/node14.html#figcut

Usually, to solve an ILP problem, the respective *LP relaxation*, which is obtained removing the integrality constraints on the variables, is first considered, and next, an optimal integer solution, close to the fractional solution, is searched. The solution of the LP relaxation must satisfy the following properties. First, in a minimization problem, the LP relaxation provides a lower bound on the optimal value of the ILP problem whereas in a maximization problem, the LP relaxation provides an upper bound on the solution of the ILP problem. In addition, all integer solutions are also solutions of the LP relaxation problem. Moreover, if the LP relaxation provides a solution with integer variables, then the solution is also valid for the ILP problem.

Without loss of generality, consider a maximization problem. The LP relaxation provides an upper bound on the value of the feasible solution. On each branch of the search tree, if the LP solution is greater than the upper bound, then we can discard this branch and stop exploring it. In this case, we say that the subproblem is discarded by a bounding argument.

The branch-and-bound search algorithm works as follows. First, the linear relaxation of the problem is solved. If the solution is integer, then the algorithm terminates returning the solution. Otherwise, new subproblems are created by branching on a fractional variable. If x is a variable which could take $n < +\infty$ distinct integer values, then one possible choice is to create n ILPs obtained by setting x to each possible value. Let a subproblem be *not active* when any of the following occurs: you used the subproblem to branch on, all variables in the solution are integer, the subproblem is infeasible, or you can discard the subproblem by a bounding argument. Otherwise, the subproblem is *active*. Hence, an active subproblem and a fractional variable are chosen to branch on. The process should be repeated until there are no more active subproblems.

An alternative to the general branch-and-bound algorithm is called branch-and-cut. A *branch-and-cut* algorithm is a branch-and-bound procedure where, at each step, valid cuts are added to improve the bounds. A *valid cut* is a constraint which every feasible integer solution must satisfy and the current fractional solution should not satisfy. Figure 2.1 provides an example of a valid cut. In general, branch-and-cut is an efficient method to solve ILP problems. Nonetheless, given the NP-hardness of ILP, the branch-and-cut procedure has exponential complexity in the worst case.

2.2 Boolean Satisfiability (SAT)

The Boolean satisfiability (SAT) problem has many applications in several fields, among which are electronic design automation [123], model checking [14], bioinformatics [115], software modeling and verification [77]. Consequently, a great effort has been done to study the SAT theory and to develop efficient SAT solvers.

2.2.1 Preliminaries

In *Boolean logic* (or *propositional logic*), each variable x may take one of two values, 1 (for true) or 0 (for false). A propositional formula φ is said to be in *conjunctive normal form* (CNF) if it is the conjunction ($\wedge$) of disjunctions ($\vee$) of literals, i.e. $\bigwedge_i(\bigvee_j l_{ij})$, where a *literal* l is either a variable, x, or its complement, $\neg x$. A *clause* is a disjunction of literals, $(\bigvee_j l_j)$. Every propositional formula φ can be converted to CNF, as explained in the next subsection.

A complete Boolean assignment maps each variable x to one of two values, either 0 or 1. For a non-complemented literal, the value of the literal is the value of the variable. For a complemented literal, the value of the literal is the complement of the value of the variable. The value of a clause is the disjunction of the values of its literals. The value of a CNF formula is the conjunction of the values of its clauses. A clause is satisfied if any of its literals is assigned value 1. A CNF formula φ is satisfied if all of its clauses are satisfied.

Definition 2.1. ***Boolean Satisfiability***

Given a propositional formula, φ, the Boolean SATisfiability (SAT) problem consists in deciding whether there exists a Boolean assignment to the variables of φ, such that φ becomes satisfied and, in that case, the satisfying assignment can be given.

For example, consider the CNF formula $\varphi = ((x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee \neg x_3))$, with three variables and three clauses. A possible SAT solution to the formula φ assigns $x_1 = 1$, $x_2 = 0$ and

$x_3 = 1$.

The SAT problem was the first problem to be proved NP-complete, in 1971 [28].

2.2.2 CNF Encodings

Every propositional formula is equivalent to a CNF formula. Propositional logic formulae can be transformed to CNF in several ways, but some encodings may be better than others. In particular, some encodings keep structural information while others do not. Encodings which keep the structural information tend to be preferred because it can be used to improve the performance of the search procedure.

The most simple way to transform a propositional formula to CNF is by using the rules of Boolean algebra: definitions of the connectives, commutativity, associativity, distributivity and idempotence of $\wedge$ and $\vee$, *De Morgan*'s laws, laws of absorption, complements, etc. For example, consider the formula

$$\varphi = ((a \wedge b) \Rightarrow (c \wedge d)).$$

By definition of the implication connective, the formula can be rewritten as

$$(\neg(a \wedge b) \vee (c \wedge d)),$$

and, using a De Morgan's law, φ is equivalent to

$$(\neg a \vee \neg b \vee (c \wedge d)).$$

Applying distributivity, the following CNF formula is achieved,

$$((\neg a \vee \neg b \vee c) \wedge (\neg a \vee \neg b \vee d)).$$

The main drawback of this method is that, in general, it yields to exponentially large formulae.

The most usual conversion to CNF is obtained by the Tseitin transformation [157]. Two formulas, φ and φ', are *equisatisfiable* if φ is satisfiable exactly when φ' is satisfiable. The Tseitin encoding transforms a propositional formula φ into an equisatisfiable CNF formula φ' with a linear increase in the number of clauses. This is obtained by adding a linear number of new variables. For example, consider the formula $\varphi = ((a \wedge b) \Rightarrow (c \wedge d))$. Introduce a new variable f_1 such that $f_1 \Leftrightarrow (a \wedge b)$,

which is equivalent to the CNF formula

$$((\neg f_1 \vee a) \wedge (\neg f_1 \vee b)).$$

Moreover, introduce a new variable f_2 such that $f_2 \Leftrightarrow (c \wedge d)$ which is equivalent to the CNF formula

$$((\neg f_2 \vee c) \wedge (\neg f_2 \vee d)).$$

Therefore, φ is equisatisfiable to $f_1 \Rightarrow f_2$, i.e.,

$$(\neg f_1 \vee f_2),$$

which means that the CNF formula obtained by the Tseitin transformation is

$$((\neg f_1 \vee a) \wedge (\neg f_1 \vee b) \wedge (\neg f_2 \vee c) \wedge (\neg f_2 \vee d) \wedge (\neg f_1 \vee f_2)).$$

2.2.3 SAT Algorithms

The first well-known SAT algorithm, which was proposed in 1960, is the *Davis-Putman* (DP) algorithm [33] and was based on the resolution inference rule. This algorithm has the drawback of its exponential space complexity, thus requiring exponentially large memory resources.

The algorithm that inspired almost all modern SAT solvers is the *Davis-Putnam-Logemann-Loveland* (DPLL) algorithm [32], from 1962, which first introduces the notion of a search tree for solving SAT.

SAT solving algorithms have significantly improved in the last years. SAT competitions [2] have been held each year from 2002. These competitions establish every year the state of the art SAT solvers in different categories and have contributed to rapid advances in the quality of SAT solvers. Examples of well-known SAT solvers are MiniSat [37] [3], PicoSAT [13] [4] and SAT4J [11] [5].

[2] *http://www.satcompetition.org*
[3] *http://minisat.se*
[4] *http://fmv.jku.at/picosat*
[5] *http://www.sat4j.org*

The DPLL Algorithm

The *Davis-Putnam-Logemann-Loveland* (DPLL) algorithm, also known as DPLL backtrack search, introduced the searching procedure exploring the nodes of a decision tree, using depth-first search. In a *decision tree*, each node corresponds to a variable assignment, specified as a *decision assignment*. A *decision level* is associated with each node according to its depth in the search tree, starting from the first variable selection which corresponds to the decision level 1. Assignments during preprocessing are associated with decision level 0. The search algorithm alternates between three main processes: decision process, deduction process and diagnosis process, which are briefly described in the following paragraphs. The DPLL algorithm eliminates the exponential memory problem of the previous DP algorithm based on resolution. However, exponential time is still a problem and developing efficient techniques are crucial for building an efficient SAT solver.

Decision Process

In the *decision process* a variable is selected and assigned, i.e. it is chosen a node in the search tree and a branch to explore. The selected variable is referred to as *decision* variable. Choosing different branches of the search tree can produce significantly different sized search trees and consequently different running times may be necessary to solve the problem. Given the importance of choosing a good branching variable and value, several heuristics have been studied to choose the most appropriate decision variable and corresponding assignment. Early heuristics give preference to assignments that simplify the largest number of clauses [171, 79, 22], or directly satisfy the maximum number of clauses [127]. The *literal count heuristics* [122] give more relevance to the variables that appear more times in unsatisfied clauses. The *VSIDS* heuristic [129] keeps a score for each variable assignment, which is increased when a variable appears in a learned clause (diagnosis process).

Deduction Process

In the deduction process, the current assignment is extended by following the logical consequences of the assignments made so far. Let a *unit clause* be a clause in which all literals except one are assigned false. Then, in a unit clause, the unassigned literal must be assigned true in order to satisfy the clause (*unit clause rule*) [33]. *Boolean constraint propagation* (BCP), also known as unit propagation, is the procedure that iteratively applies the unit clause rule until there are no unit clauses available or a conflict is found.

In the BCP process, it is necessary to detect unit clauses and conflicting clauses after a variable assignment. This mechanism can be very time consuming if some optimized techniques are not considered. If each time a new variable is assigned, every clause needs to be visited to check whether the clause has became unit, then the algorithm can be very slow. Therefore, to improve the BCP engine some techniques have been suggested, using a *counter-based* BCP [127, 10]. During the search, a clause is important only in two steps: when the clause goes from two non-falsified literals to one non-falsified literal; or when the clause goes from one non-falsified literal to zero non-falsified literals. In the former case, the clause becomes unit and the unit clause rule must be applied. In the latter case, a conflict is reached and backtrack must be done (diagnosis process). So, in practice, it is sufficient to keep a counter for each clause and, each time a literal is assigned false, the counter of each clause which contains that literal is increased by one. For a clause with N literals, only when the counter goes from N-2 to N-1 it is necessary to visit the clause, because it is the only case where the clause may become unit.

Alternatively, a two *watch-literal strategy* [129] can be applied. For each clause, pick two literals not yet assigned and watch them. Note that for a clause to be unit, it is required that at least one of those literals be assigned value 0. Then, it is only necessary to visit a clause when one of those literals takes value 0. In this case, one of two situations can happen. Either the clause becomes unit and then, the other watched literal, which is the only one not yet assigned, must be true; or at least two literals in the clause are not assigned to 0. In the latter case, one of the non-watched literals, l, is not assigned to 0, and then, literal l should be chosen to replace the watch-literal just assigned to value 0. The watch-literal BCP has the advantage that fewer clauses are visited when a literal is assigned. Moreover, unassignment is done in constant time and frequent re-assignments of literals are faster.

Diagnosis Process

A *conflict* occurs when all literals in a clause evaluate to false, resulting on an unsatisfiable clause, also called *conflicting clause*. In this case, the *conflicting assignment* must be undone and the subtree below the current node can be pruned. The mechanism of undoing an assignment and going back to a lower level on the decision tree is called the *backtracking process*.

The original DPLL algorithm performs *chronological backtracking* which corresponds to backtracking to the highest decision level where the corresponding variable has not been tried with both values. This procedure, proposed in the original DPLL algorithm, works well for randomly generated instances but, in general, performs poorly for instances coming from practical applications.

Actually, chronological backtracking can waste too much time exploring a region of the search tree with no satisfying assignments.

After a conflict, the BCP engine can do a *learning* process [127, 10], to ensure that the conflicting assignment is not tried again in the future. Normally, only a subset of the assigned variables are responsible for the conflict. During the diagnosis process, such subsets of variables must be identified and clauses with information about the conflict, *learned clauses*, are added to the model. Furthermore, learned clauses are used in non-chronological backtracking. *Non-chronological backtracking* [127] allows to jump some levels in the search tree when doing backtrack. This process plays an important role in pruning the search space, in particular for structured problems.

The algorithm terminates successfully when all clauses became satisfied by a certain assignment or unsuccessfully if all assignments have been tested, thus being an unsatisfiable formula.

2.3 Pseudo-Boolean Optimization (PBO)

A *pseudo-Boolean optimization* (PBO) problem consists in finding an assignment to a set of Boolean variables which satisfy a given set of constraints, called PB-constraints, and optimize a given PB objective function.

Pseudo-Boolean optimization problems have been studied since 1968 [67], in several contexts including operations research, artificial intelligence and electronic design automation. In the last years, extensive work has been published in the context of PBO.

2.3.1 Preliminaries

In this dissertation, we only consider linear constraints. Linear PBO problems can be seen as a particular case of ILP problems, where all variables are Boolean and all coefficients are integers. For this reason, PBO problems are also called 0-1 ILP problems.

A pseudo-Boolean constraint (PB-constraint) is a linear inequality with integer coefficients,

$$\sum_{j=1}^{n} a_j l_j \geq b, \tag{2.3}$$

with $a_j, b \in \mathbb{Z}$ and $l_j \in \{x_j, \bar{x}_j\}$, and b is the *right-hand-side* (*rhs*) of the constraint. A *term* is an expression $a_j l_j$, where $a_j \in \mathbb{Z}$ is a coefficient and $l_j \in \{x_j, \bar{x}_j\}$ is a literal, with $\bar{x}_j = \neg x_j$. Hence, constraint (2.3) has n terms.

Definition 2.2. ***Pseudo-Boolean Optimization***

A Pseudo-Boolean Optimization (PBO) problem consists in finding the values for a set of n binary variables x_j, $1 \leq j \leq n$ *(i.e. a Boolean assignment), that minimizes a linear objective function* $\sum_{j=1}^{n} c_j x_j$, *subject to a set of m pseudo-Boolean constraints,* $\sum_{j=1}^{n} a_{ij} l_j \geq b_i$, *with* $1 \leq i \leq m$ *and* $c_j, a_{ij}, b_i \in \mathbb{Z}$.

PBO problems can be seen as a generalization of SAT problems. Clauses are particular cases of PB-constraints. Moreover, PBO problems are enriched with an optimization function. Pseudo-Boolean constraints have more expressive power than clauses, but PBO is close enough to SAT to benefit from the recent advances in SAT. On the other hand, PBO is a particular case of ILP and, therefore, can benefit from the large body of work in ILP.

PBO is a NP-hard problem, since it contains the SAT problem as a particular case [45].

Any PB-constraint can be written in the normal form. A PB-constraint is in the *normal form* when it is expressed as

$$\sum_{j=1}^{n} a_j l_j \geq b, \text{ with } a_j, b \in \mathbb{Z}^+, \tag{2.4}$$

where l_j denotes either x_j or $\bar{x}_j$. Clearly, every $\leq$-constraint can be transformed into a $\geq$-constraint by negating all its constants. Furthermore, negative coefficients can be eliminated changing x_j into $\bar{x}_j$ and updating *rhs* (note that $\bar{x}_j$ is equivalent to $1 - x_j$).

A particular but interesting case of pseudo-Boolean constraints are cardinality constraints, which impose bounds on the number of literals with value 1.

Definition 2.3. ***Cardinality Constraint***

A Boolean cardinality constraint, also known as counting constraint, is a PB-constraint $\sum_{i=1}^{n} a_i l_i \geq b$, *with* $l_i \in \{x_i, \bar{x}_i\}$, *where for every i,* $a_i = 1$.

Every CNF clause is equivalent to a PB-constraint. Indeed, a CNF clause $l_1 \vee l_2 \vee \ldots \vee l_n$, with $l_i \in \{x_i, \bar{x}_i\}$ and $1 \leq i \leq n$, is equivalent to the PB-constraint $l_1 + l_2 + \ldots + l_n \geq 1$. However, in general, a PB-constraint is not equivalent to a single clause. In fact, in some cases, a single PB-constraint corresponds to an exponential number of CNF clauses. One simple example of an exponential case is the cardinality constraint $\sum_{i=1}^{n} l_i \leq k$, which is true when at most k literals among $l_1, \ldots, l_n$ are true. This constraint translates to $\binom{n}{k+1}$ clauses which correspond to all possible clauses obtained by choosing $k+1$ literals among $\{l_1, \ldots, l_n\}$.

A binate covering problem [162] is a PBO problem where each constraint can be translated into SAT using a single clause.

Definition 2.4. ***Binate Covering Problem***

A Binate Covering Problem is a PBO problem such that:

- $a_{ij} \in \{-1, 0, 1\}$ *for* $1 \leq j \leq n$ *and* $1 \leq i \leq m$*; and*
- $b_i = 1 - |\{a_{ij} = -1, 1 \leq j \leq n\}|$ *for* $1 \leq i \leq n$.

There are three types of PB-constraints. Clauses are particular cases of cardinality constraints, which are particular cases of general pseudo-Boolean constraints. Examples of each type of constraints are

$$\begin{aligned} \text{propositional clause} \quad & x_1 + \bar{x}_2 + \bar{x}_3 + x_4 + \bar{x}_5 \geq 1, \\ \text{cardinality constraint} \quad & x_1 + \bar{x}_2 + \bar{x}_3 + x_4 + \bar{x}_5 \geq 4, \\ \text{pseudo-Boolean constraint} \quad & 3x_1 + \bar{x}_2 + 5\bar{x}_3 + x_4 + 2\bar{x}_5 \geq 4. \end{aligned}$$

2.3.2 PBO Algorithms

PBO algorithms can be based on techniques from the ILP domain and from the SAT domain. The integration of both formalisms translates to a promising method for solving constraint problems.

Several approaches based on SAT techniques have emerged for solving PBO problems. Mainly, one needs to think how to handle the objective function and how to solve the PB-constraints.

Dealing with the optimization function

There exist two main ideas to deal with the objective function. A branch-and-bound procedure can be developed, pruning the search space with better estimates for the objective function value. Alternatively, the objective function can be handled as a PB-constraint whose *rhs* is iteratively modified optimizing its value. After determining the value of the current solution, a new constraint is added to the formulation such that a new solution must improve the best solution found so far. The process is iterated until an unsatisfiable formula is obtained, which means that the optimal value of the objective function is the last one to be achieved.

Consider that the problem consists in minimizing a function $f(\mathbf{x})$. In the linear search, an upper bound, UB, on the optimal value of the function $f(\mathbf{x})$ must be calculated, for example the maximum of function f. Afterward, the PB-constraint $f(\mathbf{x}) < UB$ must be added to the problem formulation. Each time a solution S is found, UB is set to $f(S)$, so that the corresponding constraint imposes finding a solution strictly better than S. The process is iterated until the set of PB-constraints

becomes unsatisfiable. In that case, the last solution (if it exists) is the optimal solution.

Solving PBO using SAT techniques

There are two main approaches for using SAT techniques with PB-constraints. First, PB-constraints can be directly translated to SAT. Alternatively, the SAT engine can be modified to deal with PB-constraints directly.

There exist a few methods for translating PB-constraints to SAT [38, 8].

The most simple approach is to translate PB-constraints into a semantically equivalent set of clauses, using the same variables. However, the number of clauses generated may be exponential. Nonetheless, the size of the CNF can be reduced to linear using auxiliary variables. Practical approaches are translation through *binary decision diagrams* (BDDs), *sorter networks* or *adder networks* [38].

The translation of PB-constraints to clauses using BDDs is interesting in many cases. In order to translate a BDD into a CNF formula, extra variables may be introduced. Otherwise, even polynomial BDDs can have an exponential translation to clauses. Therefore, the goal of introducing new variables is to produce a compact representation of PB-constraints. Nonetheless, in general, translating PB-constraints into CNF using BDDs is a problem with exponential complexity [8]. Moreover, new variables also allow to preserve more implications between the literals of a PB-constraint. When translating a PB-constraint to a CNF formula, if an assignment can be propagated on the PB-constraint, then the same assignment should also be propagated, by unit propagation, on the corresponding CNF formula. This concept, widely used on constraint programming, is called *arc-consistency*. On the other hand, the introduction of variables can slow down the solver. Therefore, one should choose a CNF encoding more likely to imply unit propagation, but without adding too many variables. Assuming P $\neq$ NP, which is vastly believed to be true, in general no encoding can make all implications derivable without adding an exponential number of extra constraints, otherwise there would be a polynomial algorithm for SAT. However, cardinality constraints can be translated efficiently maintaining arc-consistency. Translating PB-constraints through BDDs maintains arc-consistency. In general, conversion through adders or sorters does not maintain arc-consistency, but in particular, sorters preserve arc-consistency for cardinality constraints.

Recent work proves that there exists a polynomial size CNF encoding of PB-constraints such that generalized arc-consistency is maintained through unit propagation [9].

PBO Solvers

Five PBO competitions [6] have been held since 2005, with the goal of assessing the state of the art in the field of pseudo-Boolean solvers.

Three examples of PBO solvers which follow quite different approaches, still being competitive, are MiniSat+, Pueblo and BSOLO.

MiniSat+ [38] [7] is a pseudo-Boolean solver which handles PB-constraints through translation to SAT, without modifying the SAT procedure itself. To encode each individual PB-constraint, the most appropriate representation (BDDs, adders or sorters) is chosen. In addition, the objective function is satisfied by iteratively calling the SAT solver where for each new iteration the objective function is updated until the problem is unsatisfiable.

Pueblo [145] is a PB-solver which handles directly PB-constraints, modifying the SAT engine to apply propagation and learning to PB-constraints.

BSOLO [119] [8], that was first specially developed for binate covering problems, but afterward generalized for all PBO problems, adapts techniques from the SAT domain into a branch-and-bound algorithm.

2.4 Maximum Satisfiability (MaxSAT)

SAT solvers are clearly important in proving satisfiability and providing satisfiability assignments. However, SAT solvers provide little information about unsatisfiable instances. In several unsatisfiable real world problems, it would be desirable to find an assignment which maximizes the number of satisfied constraints, or that satisfies all constraints from a given set and the maximum number of constraints from another set. Therefore, some extensions of SAT have been considered. MaxSAT and its variants have been used in a wide range of applications, namely scheduling [159], electrical engineering [170], bioinformatics [152], among others.

2.4.1 Preliminaries

The maximum satisfiability problem is an extension of SAT, that aims at finding an assignment which maximizes the number of satisfiable clauses.

[6] *http://www.cril.univ-artois.fr/PB10*
[7] *http://minisat.se/MiniSat+.html*
[8] *http://sat.inesc-id.pt/~vmm/research*

Definition 2.5. ***Maximum Satisfiability***

Given a CNF formula, φ, the Maximum Satisfiability (MaxSAT) problem consists in finding a Boolean assignment to the variables of φ, such that the maximum number of clauses in φ become satisfied.

For example, consider the formula $\varphi = (x_1 \wedge (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2))$. φ is unsatisfiable but a MaxSAT solver points out that the maximum number of satisfiable clauses is two, and a possible solution is to satisfy the first two clauses by assigning $x_1 = 1$ and $x_2 = 1$.

MaxSAT is a NP-hard problem because SAT can be reduced to MaxSAT in polynomial time.

There are three important extensions of MaxSAT named weighted MaxSAT, partial MaxSAT and weighted partial MaxSAT. For these extensions, consider that a weighted clause is a pair (C_i, w_i), where C_i is a clause and w_i is a positive natural number ($w_i \in \mathbb{N}^+$). In this case, w_i is named the weight of clause C_i.

Definition 2.6. ***Weighted MaxSAT***

Given a set of weighted clauses, the Weighted Maximum Satisfiability (Weighted MaxSAT) problem aims at finding a Boolean assignment that maximizes the sum of weights of satisfied clauses.

For example, consider the weighted formula $\{(x_1, 1), ((\neg x_1 \vee x_2), 2), ((\neg x_1 \vee \neg x_2), 4)\}$. To maximize the sum of the weights of the satisfied clauses, the second and the third clauses must be satisfied. This fact requires the assignment $x_1 = 0$.

Another MaxSAT variant is the Partial MaxSAT problem, which can be described as follows.

Definition 2.7. ***Partial MaxSAT***

Given a set of clauses, in which some clauses are declared hard clauses, C_H, and some clauses are declared soft clauses, C_S, the Partial Maximum Satisfiability (Partial MaxSAT) problem aims at finding a Boolean assignment that satisfies all the hard clauses and maximizes the number of satisfied soft clauses.

For example, consider the formula containing the soft clauses $C_S = \{(x_1), (\neg x_1 \vee x_2)\}$ and the hard clauses $C_H = \{(\neg x_1 \vee \neg x_2)\}$. Clauses in C_H must be satisfied, and the number of satisfied clauses in C_S must be maximized. Hence, one possible solution is to assign $x_1 = 1$ and $x_2 = 0$, thus satisfying clauses (x_1) and $(\neg x_1 \vee \neg x_2)$.

Definition 2.8. ***Partial Weighted MaxSAT***

Given a set of hard clauses and a set of weighted soft clauses, the Partial Weighted Maximum

Satisfiability (Partial Weighted MaxSAT) problem aims at finding a Boolean assignment that satisfies all the hard clauses and maximizes the sum of the weights of the satisfied soft clauses.

For example, consider the formula containing the soft clauses $C_S = \{((x_1), 1), ((\neg x_1 \vee x_2), 3)\}$ and the hard clauses $C_H = \{(\neg x_1 \vee \neg x_2)\}$. In order to satisfy all hard clauses and maximize the sum of the weight of the satisfied soft clauses, the solution must contain the assignment $x_1 = 0$.

Note that the MaxSAT problem is a particular case of the partial weighted MaxSAT problem, where all clauses are soft and have the same weight.

2.4.2 MaxSAT Algorithms

MaxSAT solvers have improved significantly in the last few years. In particular, the existence of a MaxSAT evaluation [9], held annually since 2006, has contributed to the development of new efficient MaxSAT solvers.

The first approach proposed to solve MaxSAT problems uses the branch-and-bound scheme with lower bounding. In this approach, sophisticated lower bounds on the number of unsatisfiable clauses are computed, using dedicated inference rules, unit propagation and inconsistent subsets. The approach can be improved using good variable selection heuristics and suitable data structures. Solvers following the branch-and-bound approach include MaxSatz [98], IncMaxSatz [97], WMaxSatz [6] and MiniMaxSAT [71].

A different approach is to convert MaxSAT into a different formalism. For example, MaxSAT can be solved using pseudo-Boolean optimization solvers. SAT4J-MaxSAT [10] [11] applies this translation procedure. Indeed, PBO and MaxSAT are equivalent formalisms and there are algorithms to convert one problem into the other problem [3, 70].

An alternative method for solving the MaxSAT problem and its variants uses an unsatisfiability-based approach. Unsatisfiability-based MaxSAT was first proposed by Fu&Malik [43], in 2006, and recently, MSUnCore [126] has shown that the approach can actually be very competitive and practical in real application domains. Other very recent unsatisfiability-based MaxSAT solvers are WBO [120], WPM1 [5] and PM2 [5]. The unsatisfiability-based approach works as follows. Iteratively, a conflict-driven clause learning SAT solver is used to identify unsatisfiable subformulae (unsatisfiability cores) in the input formula. Each clause in each unsatisfiable subformula should be relaxed by adding a fresh variable. Moreover, a new cardinality constraint is added to the formula

[9] *http://www.maxsat.udl.cat*
[10] *http://www.sat4j.org*

requiring that exactly one relaxation variable should be assign true. The process is iterated until there are no more unsatisfiable cores. The solution to the MaxSAT problem corresponds to the number of clauses which do not need to be satisfied with the help of a relaxation variable. For example, consider the formula $\varphi = (x_1 \wedge (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2))$. A conflict-driven SAT solver would say that the unsatisfiability core includes all the three clauses. Indeed, note that you can satisfy each pair of clauses, but you cannot satisfy the three of them. Therefore, each clause of the formula should be relaxed with a new variable and a new cardinality constraint should be included in the formulation $\varphi' = \{((x_1 \vee b_1) \wedge (\neg x_1 \vee x_2 \vee b_2) \wedge (\neg x_1 \vee \neg x_2 \vee b_3))\} \bigcup \{(\sum_{i=1}^{3} b_i = 1)\}$. This new formula is satisfiable and, therefore, the MaxSAT solution is 3-1=2 satisfied clauses.

Decomposition-based solvers use another alternative approach which has not been shown to be competitive. Examples of decomposition-based MaxSAT algorithms include Clone [137] and SR(w) [139].

2.5 Conclusions

This section describes four formalisms for solving discrete constraint problems, namely integer linear programming (ILP), Boolean satisfiability (SAT), pseudo-Boolean optimization (PBO) and maximum Satisfiability (MaxSAT). ILP, PBO and MaxSAT are optimization problems, whereas SAT is a decision problem. All of these formalisms correspond to NP-complete problems.

ILP is a mature approach for solving linear programming problems with integer variables. In particular, in 0-1 ILP problems, all variables must take Boolean values. ILP problems are normally solved using branch-and-bound. CPLEX [76] is a well-known commercial tool for solving ILP problems.

SAT aims at finding a Boolean assignment to the variables of a formula φ, such that φ becomes satisfied. Every Boolean formula is equivalent to a formula in conjunctive normal form (CNF). The Tseitin transformation [157] allows a conversion to CNF by adding a linear increase in the number of clauses. The DPLL algorithm [32] aims at solving SAT using a search procedure which explores the nodes of a decision tree using depth-first search. The search procedure alternates between three main processes, namely decision, deduction and diagnosis. In the decision process, a variable is selected and assigned to a Boolean value, using a heuristic decision. In the deduction process, the current assignment is extended by following the logical consequences of the assignments made so far, using Boolean constraint propagation. The third step, the diagnosis process, must occur after a conflict has been reached. After a conflict, it is necessary to backtrack (chronologically or non-

chronologically) and the learned clauses are added to the model [127, 10]. These three steps are iterated until a solution is achieved or all branches of the search tree have been explored without finding a solution, in which case the formula is unsatisfiable.

PBO aims at finding a Boolean assignment to a set of variables such that a set of PB-constraints is satisfied and, in addition, a PB-function is optimized. PBO can be seen as a generalization of SAT and a particularization of ILP. Therefore, PBO solvers can integrate techniques from both domains of SAT and ILP, taking both advantage of the recent advances in SAT and the large experience in the ILP field. Each PB-constraint can be translated to a CNF formula with a polynomial number of clauses, only if new variables are added [8]. Moreover, it is important to maintain arc-consistency through unit propagation during a translation of PB-constraints to a CNF formula. The simplest case is the binate covering problem [162], in which each constraint can be translated into a single clause. The cost function of the PBO approach can be handled by iteratively adding a new constraint which requires a better solution to the model, or, alternatively, using a branch-and-bound search procedure. The most well-known PBO solver is MiniSat+ [38].

MaxSAT aims at finding a Boolean assignment to the variables of a formula φ, such that the maximum number of clauses of φ becomes satisfied. There are three extensions of the MaxSAT problem. In the weighted MaxSAT problem, each clause has a weight, and the goal is to find an assignment which maximizes the weight of the satisfied clauses. In the partial MaxSAT problem, some clauses are declared hard and some clauses are declared soft, and the goal is to satisfy all hard clauses and the maximum number of soft clauses. The partial weighted MaxSAT problem is a variant of the partial MaxSAT where a weight is associated with each soft clause and the goal is to satisfy all hard clauses and to maximize the sum of the weights of the soft clauses. MaxSAT and its extensions can be solved using a branch-and-bound scheme with lower bounding [98, 97, 6, 71], translating the problem directly to PBO [11], using unsatisfiability-based algorithms [43, 126, 120, 5] or using decomposition-based solvers [137, 139]. The state of the art solvers, which have shown to have a better performance in solving industrial instances, are the solvers based on the identification of unsatisfiable cores [126].

3

Haplotype Inference

Humans are diploid organisms, which mean that our genome is organized in pairs of homologous chromosomes, each one inherited from one parent. Due to technological limitations, it is infeasible to obtain the genetic data of each chromosome separately. Instead, the combined data of the two homologous chromosomes is obtained. The genetic data of a single chromosome is called a haplotype, whereas the mixed genetic data of both homologous chromosomes is called a genotype. The haplotype inference problem aims at obtaining the haplotypic data based on the genotypic information.

Haplotype inference is an important field of research, which has relevant impact in clinic medicine [30]. Haplotypes are more informative than genotypes and, in some cases, can predict better the severity of a disease or even be responsible for producing a specific phenotype.

This dissertation considers two types of haplotype inference approaches. Population-based methods apply to sets of unrelated individuals from the same population, whereas pedigree-based methods apply to sets of individuals from the same family.

The first computational method for solving the haplotype inference problem was proposed in 1990 [26]. However, it was in the last decade that the haplotyping problem was devoted large effort and attention from the scientific community. Several computational approaches have been developed to solve the haplotype inference problem, either combinatorial or statistical. The population-based combinatorial approaches are the Clark's method [26], the perfect phylogeny haplotyping [61] and the pure parsimony approach [62]. The pedigree-based combinatorial approaches are the zero recombinant [132, 173] and the minimum recombinant [99, 64, 138] haplotype configurations. The statistical approaches can be subdivided in methods using expectation-maximization algorithms [41, 58, 1, 148] and methods using Bayesian algorithms [149, 131, 19, 34, 42, 88].

There are two main criteria for evaluating the haplotype inference methods: accuracy and efficiency. The accuracy of the haplotyping methods is measured comparing the obtained solution with the real one, which is typically obtained by simulation studies. Because haplotyping methods are computationally very demanding, the efficiency of the methods is also an important additional criterion, with is measured by the computer running time.

This chapter explains the haplotype inference problem, from the biological and mathematical point of views. First, the concepts related with SNPs, haplotypes and genotypes are detailed and the population-based and pedigree-based haplotype inference problems are explained. Follows a resumed description of the major computational approaches, both combinatorial and statistical. A final section resumes the chapter.

3.1 Preliminaries

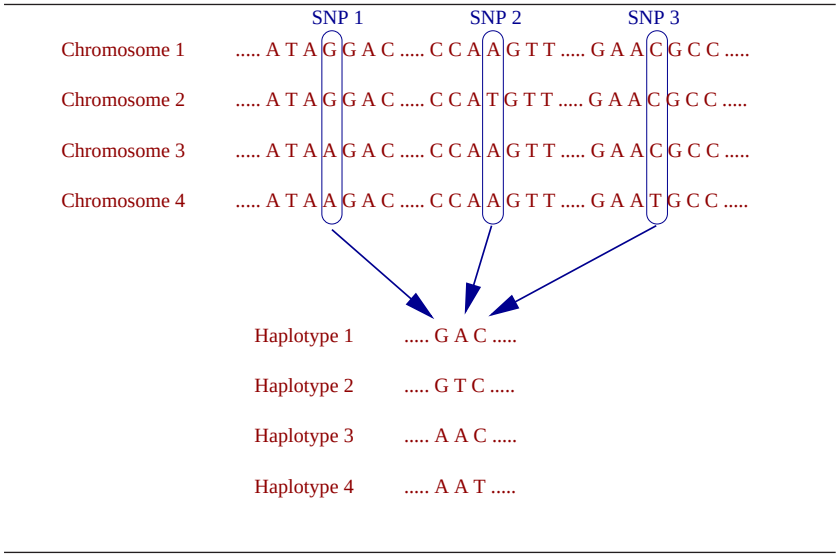
3.1.1 SNPs, Haplotypes and Genotypes

The genome constitutes the hereditary data of an organism and is encoded in the DNA (*deoxyribonucleic acid*), where it is specified by the sequence of bases of nucleotides that it contains: A (*adenine*), C (*cytosine*), T (*thymine*) and G (*guanine*).

The coding part of the genome is organized in DNA segments called *genes*. Each gene encodes a specific protein and the variants of a single gene are named *alleles*. Despite the considerable similarity between our genes, no two individuals have the same genome. The human genome has roughly three billion nucleotides but about 99.9% of them are the same for all human beings. On average, the sequence of bases of two individuals differ one in every 1200 bases, but the variations are not uniformly distributed along all the DNA. Variations in the DNA define the differences between human beings and, in particular, influence their susceptibility to diseases. Consequently, a critical step in genetics is the understanding of the differences between the genetic code of human beings. *Single Nucleotide Polymorphisms*, or SNPs (pronounced *snips*) correspond to differences in a single position of the DNA where mutations have occurred, and that present a minor allele frequency which is equal to or greater than a given value (e.g. 1%). For example, a SNP occurs when a sequence AG**T**TCG is modified to AG**A**TCG.

SNPs which are close in the genome tend to be inherited together in blocks. Hence, SNPs within a block are statistically associated. These blocks of SNPs are known as haplotypes. Therefore, *haplotypes* are sequences of correlated SNPs, in a single chromosome, as illustrated in Figure 3.1. Haplotype blocks exist because the crossing-over phenomenons do not occur randomly along the

Figure 3.1: Identifying SNPs and haplotypes



DNA, but instead are rather concentrated in small regions called recombination hotspots. Recombination does not occur in every hotspot at every generation and, consequently, individuals within the same population tend to have large haplotype blocks in common. Furthermore, due to the association of SNPs, it is often possible to identify a small subset of SNPs which represent all the remaining SNPs within the haplotype. For this reason, the SNPs of this subset are called *tagSNPs* [80].

The human genome is organized into 23 pairs of homologous chromosomes, each chromosome being inherited from one parent. There are a few methods for sequencing separately homologous chromosomes, named allele specific polymerase chain reaction (AS-PCR) and somatic cell hybrids. Nonetheless, these methods are very expensive and time consuming [21], and consequently, it is infeasible to obtain experimentally haplotypes, which correspond to genetic data at a single chromosome, for large regions.

Instead, *genotypes*, which correspond to the conflated data of two haplotypes on homologous chromosomes, are obtained. In general, the genotype data does not make it possible to distinguish between the alleles inherited from each one of the parents. The *haplotype inference* problem corresponds to finding the set of haplotypes which originate a given set of genotypes.

Almost all human SNPs are *biallelic*, which means that only two different alleles are allowed for each position. SNPs with more than two different alleles are called *polyallelic*. In what follows, we will only consider biallelic SNPs, with two possible alleles. The *wild type nucleotide* corresponds to the most common allele, and the *mutant type nucleotide* corresponds to the least frequent allele.

3.1.2 Haplotyping

Formally, a haplotype can be described as a binary string, where the value 0 represents the wild type nucleotide and the value 1 represents the mutant type nucleotide. Genotypes can be described as strings over the alphabet $\{0, 1, 2\}$. Each SNP (also called *site* or *position*) in a genotype g_i of size m is represented by $g_{i\,j}$, with $1 \leq j \leq m$. A site $g_{i\,j}$ is *homozygous* if $g_{i\,j} = 0$ or $g_{i\,j} = 1$. Otherwise, when $g_{i\,j} = 2$, the site is *heterozygous*. A *homozygous* genotype is a genotype g_i such that every site in g_i is homozygous, i.e.

$$\forall_{j:\ 1 \leq j \leq m}\ (g_{i\,j} = 0 \vee g_{i\,j} = 1). \tag{3.1}$$

A genotype g_i is *explained* (also called *resolved*) by a non-ordered pair of haplotypes $(h_i^a,\ h_i^b)$ of the same size, which is represented by $g_i = h_i^a \oplus h_i^b$, if

$$g_{i\,j} = \begin{cases} h_{i\,j}^a & \text{if } h_{i\,j}^a = h_{i\,j}^b \\ 2 & \text{if } h_{i\,j}^a \neq h_{i\,j}^b \end{cases}. \tag{3.2}$$

If $g_i = h_i^a \oplus h_i^b$, then haplotypes h_i^a and h_i^b are declared *conjugates* with respect to g_i.

The concept of (in)compatible genotypes is important in the following chapters.

Definition 3.1. ***Compatible Genotypes***

Two genotypes $g_i,\ g_k \in \mathcal{G}$ are compatible if $\forall_{j:\ 1 \leq j \leq m}\ g_{i\,j} + g_{k\,j} \neq 1$. Otherwise, the genotypes are incompatible.

Clearly, incompatible genotypes cannot have haplotypes in common in their explanation.

In general, if a genotype g_i has r_i heterozygous sites, then the potential number of non-ordered haplotype pairs that explain g_i is c_i, where

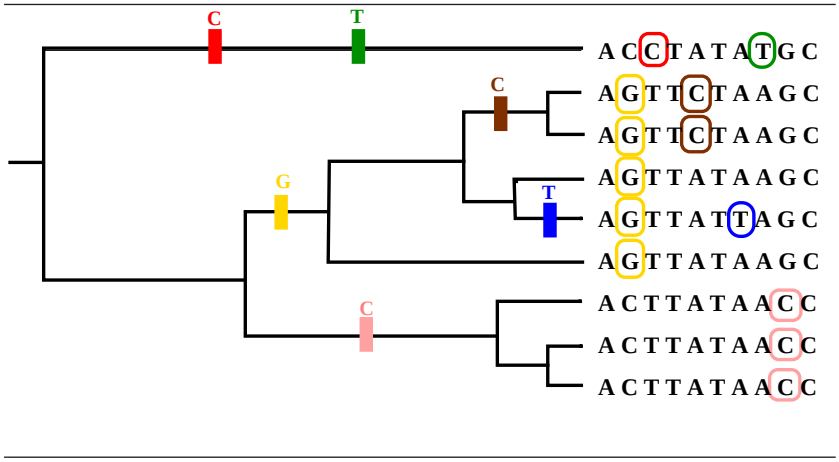
$$c_i = \begin{cases} 2^{r_i - 1} & \text{if } r_i > 0 \\ 1 & \text{if } r_i = 0 \end{cases}. \tag{3.3}$$

Therefore, homozygous genotypes or genotypes with only one heterozygous site are explained by only one pair of haplotypes and, therefore, are called *unambiguous*. Genotypes with more than one heterozygous position are called *ambiguous* genotypes.

Definition 3.2. ***Haplotype Inference***

Given a set with n genotypes, $\mathcal{G}$, each with size m, the haplotype inference problem consists in obtaining the set of haplotypes, $\mathcal{H}$, which explain the genotypes in $\mathcal{G}$ and associating a pair of

Figure 3.2: Mutations within a population



haplotypes $(h_i^a,\ h_i^b)$*, with* $h_i^a,\ h_i^b \in \mathcal{H}$*, to each genotype* $g_i \in \mathcal{G}$*, such that* $g_i = h_i^a \oplus h_i^b$*.*

Example 3.3. *(Haplotype Inference) Consider genotype 02212 having 5 sites, of which one site is homozygous with value 0, one site is homozygous with value 1 and the remaining three sites correspond to heterozygous sites. There are four different possible explanations for this genotype:* (00010, 01111)*,* (00110, 01011)*,* (00111, 01010) *and* (00011, 01110)*.*

There are several approaches to solve the haplotype inference problem, usually directly or indirectly related with the coalescent model [74]. The coalescent model assumes that, within a population, all haplotypes come from a single ancestor and, therefore, haplotypes tend to be arranged in groups according to the mutations that have occurred. Figure 3.2 illustrates a coalescent tree of a population, where mutations originate groups of similar haplotypes. Each leaf node corresponds to a haplotype and each branch represents one or more mutations.

There are two major computational approaches to solve the haplotype inference problem, which can be classified as combinatorial or statistical. Section 3.2 describes these haplotype inference approaches.

3.1.3 Pedigrees

Incorporating genotypes of related individuals in the haplotype inference problem input brings a number of relevant advantages. Pedigree information provides new valuable knowledge to the haplotype inference problem. A pedigree can be defined formally as follows.

Definition 3.4. ***Pedigree***

A pedigree is a directed acyclic graph $G = (V, E)$, where $V = M \cup F$ and M stands for male nodes and F stands for female nodes. The in-degree of each node is 0 or 2. Nodes with in-degree 0 are called founders*, whereas nodes with in-degree 2 are called* non-founders*. In non-founder nodes, one incoming edge must start at a male node (called* father*) and the other incoming edge must start at a female node (called* mother*) and the node itself is called a* child *of its* parents *(mother and father).*

A subgraph constituted by a mother, a father and a respective child is a *trio*. A pedigree has a *mating loop* if there are two different paths from a node x to a node y.

According to the *Mendelian laws of inheritance*, every site in a single haplotype is inherited from the same parent, assuming no mutations within a pedigree. Hence, one chromosome comes from the mother, whereas the other chromosome comes from the father.

Definition 3.5. ***Haplotype Inference in Pedigrees***
Given a set of n genotypes, $\mathcal{G}$, each with size m, organized in p pedigrees, the haplotype inference problem consists in obtaining the set of haplotypes, $\mathcal{H}$, which explain the genotypes in $\mathcal{G}$ and associating a pair of haplotypes $(h_i^a,\ h_i^b)$, with $h_i^a,\ h_i^b \in \mathcal{H}$, to each genotype $g_i \in \mathcal{G}$, such that $g_i = h_i^a \oplus h_i^b$, satisfying the Mendelian laws of inheritance and assuming no mutations within pedigrees.

We assume that haplotype h^a is inherited from the father and h^b is inherited from the mother. However, a recombination may occur, where the two haplotypes of a parent get shuffled and the shuffled haplotype is passed on to the child. For example, suppose a father has the haplotype pair $(011, 100)$ and the haplotype that he passed on to his child is 111. Hence one recombination event must have occurred: haplotypes 011 and 100 have mixed together and originated a new haplotype $h = 111$. Although every site of the child's haplotype h was inherited from the father, the first site came from the paternal grandmother, while the second and third sites came from the paternal grandfather.

Approaches to solve pedigree-based haplotype inference can be grouped as combinatorial and statistical. These approaches are introduced in Section 3.3.

3.2 Population-based Haplotype Inference

There is a rich and growing literature methods for solving the haplotype inference problem. This section overviews the methods for haplotype inference in unrelated individuals. The methods can be grouped in combinatorial and statistical approaches.

3.2.1 Combinatorial Approaches

The combinatorial approaches to solve the haplotype inference problem can be subgrouped in Clark's method, the perfect phylogeny haplotyping and the pure parsimony approach.

Clark's Method

Clark's method [26] is the first computational algorithm for haplotyping. The method is non-deterministic and its performance depends on the algorithmic choices.

The Clark's algorithm is as follows. Unambiguous genotypes, i.e. homozygous genotypes and genotypes with only one heterozygous site, can be explained in only one way. Consequently, the haplotypes that explain these genotypes must be in the final solution $\mathcal{H}$. Then, iteratively, the other genotypes are explained using one haplotype already used, i.e. from $\mathcal{H}$, and another one which must be included in $\mathcal{H}$. This is called the *Clark's inference rule*.

Definition 3.6. ***Clark's Inference Rule***
Let $\mathcal{NR}$ represent the set of unresolved genotypes and $\mathcal{H}$ be the set of haplotypes already in the solution. If a genotype $g \in \mathcal{NR}$ can be resolved using haplotypes h^a and h^b ($g = h^a \oplus h^b$), such that $h^a \in \mathcal{H}$ and h^b is the conjugate of h^a with respect to g, then consider g to be resolved by pair (h^a, h^b), add h^b to $\mathcal{H}$ (i.e. $\mathcal{H} = \mathcal{H} \cup \{h^b\}$) and remove g from the set of unresolved haplotypes (i.e. $\mathcal{NR} = \mathcal{NR} - \{g\}$).

The algorithm terminates when all genotypes have been explained or no further genotypes can be explained using one haplotype from $\mathcal{H}$. The genotypes that remain unresolved are called *orphans*.

Algorithm 1 resumes the Clark's method. The $InitialSetResolvedGenotypes$ represents the set of unambiguous genotypes and the $InitialSetHaplotypes$ is the set of haplotypes that explain genotypes in the $InitialSetResolvedGenotypes$.

Example 3.7. *(Clark's Method) Consider the set of genotypes $\mathcal{G} = \{1010, 0002, 2211, 2222\}$. G has two unambiguous genotypes, 1010 and 0002. Therefore, the initial set of resolved genotypes is $InitialSetResolvedGenotypes = \{1010, 0002\}$, the initial set of selected haplotypes is $InitialSetHaplotypes = \{1010, 0000, 0001\}$ and, initially, $\mathcal{NR} = \{2211, 2222\}$. Genotype 2222 $\in \mathcal{NR}$ can be explained using haplotype 1010 $\in InitialSetHaplotypes$ and another haplotype 0101. Therefore, $\mathcal{H} = \{1010, 0000, 0001, 0101\}$ and $\mathcal{NR}' = \{2211\}$. Since 2211 cannot be resolved using any haplotype in $\mathcal{H}$, it remains as an orphan.*

There are several different ways of applying the inference rule, since the order in which the genotypes are chosen is relevant and also because, in general, for a single genotype g, there exist

Algorithm 1 Clark's method

ClarkMethod($\mathcal{G}$)

1 $\mathcal{NR} \leftarrow (\mathcal{G} - InitialSetResolvedGenotypes)$

2 $\mathcal{H} \leftarrow InitialSetHaplotypes$

3 **for each** $g \in \mathcal{NR}$

4 **do if** $(\exists_{h^a \in \mathcal{H}}\ (g = h^a \oplus h^b))$

5 $\mathcal{H} \leftarrow \mathcal{H} \cup h^b$

6 $\mathcal{NR} \leftarrow \mathcal{NR} - \{g\}$

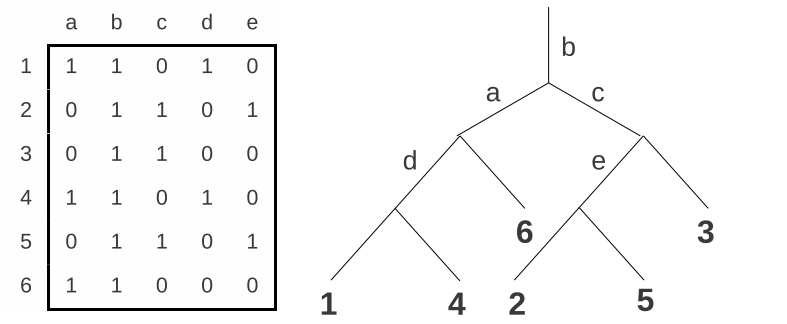
7 **return** $\mathcal{H}$

many choices for haplotypes in $\mathcal{H}$ that explain g. Clark suggested to run the algorithm several times with different genotype orders and to choose the solution which resolves a larger number of genotypes. However, usually only a small number of possibilities can be tried. Moreover, different solutions resolving the same number of genotypes can be generated and it is not clear how to use the obtained results. The *Maximum Resolution* (MR) problem aims at finding a solution to the Clark's method which leaves the minimum number of orphan genotypes. This problem is NP-hard as shown by Gusfield [60], who also proposed an integer linear programming approach for solving the MR problem. Additional heuristics [133] have been studied regarding the way in which the haplotype list should be defined and the genotype list should be analyzed to produce a better solution.

Perfect Phylogeny

The *perfect phylogeny haplotyping* (PPH) is directly based on the coalescent model. Models for PPH represent the evolutionary history of haplotypes as a direct, acyclic graph, where the lengths of the edges represent the passage in time (in the number of mutations). Furthermore, the approach is based on two biological assumptions: no recombination within long blocks and the infinite sites model. Assuming *no recombination in long blocks*, each haplotype of an individual is a copy of one of the haplotypes of his parents, and thus, the backwards history of a single haplotype is just a path. The *infinite sites assumption* [87] states that the number of sites in a genome is so large that the probability of occurring more than one mutation at a single site is infinitely small and, therefore, one can assume that it has never occurred. Based on these assumptions, the set of haplotypes should satisfy a perfect phylogeny. The definition of a perfect phylogeny is as follows.

Figure 3.3: Binary matrix, $\mathcal{H}$, and respective perfect phylogeny



Definition 3.8. ***Perfect Phylogeny***

A perfect phylogeny for a binary matrix, $\mathcal{H}_{2n\times m}$, with $2n$ rows and m columns, is a rooted tree, $\mathcal{T}$, with the following properties:

- *$\mathcal{T}$ has $2n$ leaves, each one labeled by exactly one of the $2n$ rows of $\mathcal{H}$,*
- *each of the m columns of $\mathcal{H}$ labels exactly one edge of $\mathcal{T}$,*
- *every interior edge of $\mathcal{T}$ is labeled by at least one column of $\mathcal{H}$,*
- *the columns of $\mathcal{H}$ that label the edges of the path from the root to the leaf i, specify the columns of $\mathcal{H}$ that have value 1 in the row i, giving a compact representation of the row i.*

Figure 3.3 presents an example of a perfect phylogeny for a binary matrix.

The perfect phylogeny haplotyping was first proposed by Gusfield, in 2002 [61]. Under the previous assumptions, the evolutionary history of $2n$ haplotypes can be described as a perfect phylogeny with $2n$ leaves, where the most ancestral haplotype labels the root of the tree, each SNP labels an edge, and each of the $2n$ haplotypes labels a leaf. A label j in an edge represents the only point in the history where a mutation has occurred at site j.

Definition 3.9. ***Perfect Phylogeny Haplotyping***

The perfect phylogeny haplotyping (PPH) problem consists in finding a set of haplotypes $\mathcal{H}$, which explains a given set of genotypes $\mathcal{G}$, and such that $\mathcal{H}$ satisfies a perfect phylogeny.

Example 3.10. *(Perfect Phylogeny Haplotyping) Consider the set of genotypes $\mathcal{G} = \{012, 201, 222\}$. There exist four solutions which satisfy the haplotype inference problem but only one solution satisfies a perfect phylogeny: $\mathcal{H} = \{010, 011, 101, 001, 101, 010\}$.*

Given a binary matrix $\mathcal{H}$, to test the existence of a perfect phylogeny and, in the affirmative case, to construct the respective tree can be done in polynomial time [59]. The *four-gamete test* states that a binary matrix $\mathcal{H}$ has a perfect phylogeny if and only if for each pair of columns, do not exist four rows with 00, 01, 10 and 11 in those two columns. Moreover, in this case, the perfect phylogeny is unique if and only if all columns of $\mathcal{H}$ are distinct.

It is interesting to note that the genotype matrix contains several informations about the paths in the perfect phylogeny tree, allowing an efficient deduction of the tree structure. For example,

- If $g_{i\,j} = 1$, then the edge labeled with j must be in the path from the root to both leaves h_i^a and h_i^b. In other words, the set of positions with value 1 in the genotype g_i defines the unordered labels of the edges from the root to the most recent ancestor of haplotypes h_i^a and h_i^b.

- If $g_{i\,j} = 2$, then the edge labeled with j has to be in the path from the root to exactly one of the leaves labeled with h_i^a or h_i^b.

- If $g_{i\,j} = 0$, then the edge labeled with j *must not* be in the path from the root to leaves labeled with h_i^a and h_i^b.

- Columns which label the same edge must be equal.

Several algorithms have been proposed for haplotype inference using perfect phylogeny. The first algorithm for PPH is based on an explicit reduction from the PPH problem to the graph realization problem [1] [61]. The algorithm is theoretically almost linear in the size of the matrix $\mathcal{G}$.

A different contribution is the algorithm described in [40], which can be seen as a specialization of the general graph realization method to the PPH case. This algorithm constructs a tree *top-down* from the root. The complexity of this algorithm is $\mathcal{O}(nm^2)$.

The algorithm proposed in [158] is based on the combinatorial structure of the PPH problem, exploring the relations between columns, and relying in the standard four-gamete test. Since the algorithm has to explore the relations between every pair of columns, the complexity of the method is $\mathcal{O}(nm^2)$. Some other algorithms use specific data structures to avoid the comparison between every two columns, but being able to maintain the same information on the data structure. The construction of such a structure allows the algorithms to be linear ($\mathcal{O}(nm)$) [111, 35, 160, 161].

[1] The graph realization problem is defined as follows: given a family of paths, determine a tree where each of the path sets is realized or determine that no such tree exists.

Although recombination events and recurrent mutations are rare, the perfect phylogeny assumptions are not realistic in general. A PPH relaxed method considers an imperfect phylogeny algorithm [66]. This method is implemented in a haplotype inference tool called HAP [2]. This approach performs a partition of the given genotypes into blocks. For each block, the imperfect phylogeny model finds a haplotype inference solution which fits the perfect phylogeny model for the common haplotypes.

Pure Parsimony

A well-known combinatorial approach, which is indirectly related with the coalescent model, is pure parsimony. This approach was first proposed by Gusfield in 2003 [62]. The haplotype inference by pure parsimony (HIPP) approach aims at finding a minimum-cardinality set of haplotypes $\mathcal{H}$ that can explain a given set of genotypes $\mathcal{G}$.

The motivation for searching a haplotype inference solution with the smallest number of haplotypes is biologically motivated by the fact that individuals from the same population have the same ancestors and mutations do not occur often. In addition, it is a well-known fact that the number of haplotypes in a population is much smaller than the number of genotypes. Moreover, experimental results support that iterations of the Clark's method return more accurate solutions when the number of haplotypes is smaller [63].

Definition 3.11. ***Haplotype Inference by Pure Parsimony***

The haplotype inference by pure parsimony (HIPP) problem consists in finding a minimum-size set $\mathcal{H}$ of haplotypes that explain all genotypes in $\mathcal{G}$.

Example 3.12. *(Haplotype Inference by Pure Parsimony)Consider the set of genotypes $\mathcal{G} = \{g_1, g_2, g_3\} = \{022, 221, 222\}$. There are solutions using 6 different haplotypes: $\mathcal{H}_1 = \{000, 001, 010, 011, 101, 111\}$, such that $g_1 = 001 \oplus 010$, $g_2 = 011 \oplus 101$ and $g_3 = 000 \oplus 111$. However, the HIPP solution only requires 4 distinct haplotypes: $\mathcal{H}_2 = \{000, 011, 101, 111\}$ such that $g_1 = 011 \oplus 000$, $g_2 = 011 \oplus 101$ and $g_3 = 011 \oplus 100$.*

In general, there may exist several solutions which satisfy the HIPP criterion.

Finding a solution to the HIPP problem is an APX-hard (and consequently, NP-hard) problem [92]. Extensive work has been developed in order to produce an efficient solver to the HIPP problem. Some HIPP algorithms are described in Chapter 4 and the state of the art HIPP solver is detailed in Chapter 5.

[2] http://research.calit2.net/hap

3.2.2 Statistical Approaches

Generally, statistical methods for haplotype inference solve the problem by assuming that the observed genotypes have been obtained by a random combination of haplotypes from an (unknown) distribution. A crucial point of the statistic haplotype inference methods is to estimate that distribution, i.e. the frequencies of the haplotypes. After that, the pair of haplotypes which resolve genotype $g \in \mathcal{G}$ and has higher probability is associated with g.

There are two main groups of statistical approaches for haplotype inference: expectation-maximization algorithms and Bayesian algorithms. Expectation-maximization methods proceed iteratively estimating the haplotype frequencies and maximizing the probability of observing the given genotype data. Bayesian methods incorporate assumptions or prior information as a guide to haplotype inference.

Expectation-Maximization

The maximum likelihood estimation is a well-known method used to estimate unknown parameters that best fit the model in study. Roughly, for a given data and a given probability model, the maximum likelihood estimation finds the parameters of the model which make the data "more likely" to be observed.

Let $\mathcal{G} = \{g_1, g_2, \ldots, g_n\}$ be the set of (known) genotypes and $\mathcal{H} = \{H_1, H_2, \ldots, H_n\}$ denote the set of corresponding (unknown) haplotype pairs, with $H_i = (h_i^a, h_i^b)$. Let $F = \{F_1, F_2, \ldots, F_M\}$ denote the set of (unknown) population haplotype frequencies and $f = \{f_1, f_2, \ldots, f_M\}$ be the set of (unknown) sample haplotype frequencies to estimate, where M is the number of possible haplotypes. We first estimate population haplotype frequencies, F, and then use these frequencies to estimate the sample frequencies, f.

Assuming random mating, the probability of observing a certain genotype, g_i, is given by the sum of the probabilities of all possible haplotype pairs that resolve g_i. Therefore, let $P(i)$ denote the probability of observing a genotype g_i, then

$$P(i) = \sum_{j=1}^{c_i} P(H_j), \tag{3.4}$$

where c_i is the number of haplotype pairs that explain g_i and H_j, with $1 \leq j \leq c_i$, represent pairs of haplotypes (h_k, h_l) which explain g_i.

Algorithm 2 Expectation-maximization algorithm

EXPECTATION-MAXIMIZATION($\mathcal{G}$)

1 Choose some initial values $F_1^{(0)}, F_2^{(0)}, \ldots, F_M^{(0)}$ for haplotype frequencies
$\triangleright$ M is the number of haplotypes

2 $t \leftarrow 0$

3 **if** $(t > 0)$

4 **do**

5 **while** $(P(\mathcal{G}|\hat{F}^{(t+1)}) - P(\mathcal{G}|\hat{F}^{(t)}) \geq \varepsilon)$

6 $t \leftarrow (t + 1)$

7 **do**

8 a) Probability of resolving each genotype by each haplotype pair
$P_j(h_k, h_l)^{(t)} = \frac{n_j}{n} \frac{P(h_k,h_l)^{(t)}}{P(j)^{(t)}}$

9 b) Estimated frequency of each haplotype
$\hat{F}_b^{(t+1)} = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{c_i} \delta_{ib} P_j(H_j)^{(t)}$

10 c) Vector of estimated frequencies
$\hat{F}^{(t+1)} = (\hat{F}_1^{(t+1)}, \hat{F}_2^{(t+1)}, \ldots, \hat{F}_M^{(t+1)})$

11 $\hat{f} \leftarrow \hat{F}^{(t+1)}$

12 **return** $\hat{f}$ (estimate of the haplotype frequencies)

The probability of a pair of haplotypes (h_k, h_l) is given by

$$P(h_k, h_l) = \begin{cases} F_k^2 & \text{if } k = l \\ 2F_kF_l & \text{if } k \neq l \end{cases}, \tag{3.5}$$

where F_k and F_l are the population haplotype frequencies (unknown) of the k^{th} and l^{th} haplotypes.

The *Expectation-Maximization* (EM) procedure for haplotype inference is described by Algorithm 2. The EM algorithm is an iterative method used to calculate maximum-likelihood estimates of the haplotype frequencies which maximize the sample probabilities [41].

The EM algorithm is initialized with some arbitrary values for the haplotype frequencies: $F_1^{(0)}$, $F_2^{(0)}$, ..., $F_M^{(0)}$. In the *expectation step* the frequencies of the haplotype pairs, $P(h_k, h_l)$, are estimated.

At each step, t, the estimated probability of resolving genotype g_i with pair (h_k, h_l) is given by

$$P_i(h_k, h_l)^{(t)} = \frac{n_i}{n} \frac{P(h_k, h_l)^{(t)}}{P(i)^{(t)}} \tag{3.6}$$

where $\frac{n_i}{n}$ is the frequency of genotypes g_i in the sample, $P(h_k, h_l)$ is given by equation (3.5) and $P(i)$ is given by equation (3.4).

In the *maximization step*, the haplotype frequencies at the following iteration are estimated based on the expected frequencies,

$$\hat{F}_b^{(t+1)} = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{c_i} \delta_{jb} P_j(H_j)^{(t)}, \tag{3.7}$$

where δ_{jb} is the number of times haplotype h_b is in the haplotype pair H_j (0, 1 or 2).

The method progresses iteratively until convergence is reached, i.e. $P(\mathcal{G}|\hat{F}^{(t+1)}) - P(\mathcal{G}|\hat{F}^{(t)}) < \varepsilon$, for a small ε predefined.

The EM algorithm is limited by the number of heterozygous sites, since the number of haplotype pairs, which probability must be calculated, is exponential in the number of heterozygous sites. Moreover, one of the main drawbacks of the EM algorithm is that it can converge to a local optimum, instead of converging to a global optimum.

Methods for haplotype inference following the EM algorithms are presented in [41, 69]. Another statistical model which follows the EM approach is the FastPHASE model [143], which is based on clusters of similar haplotypes. Actually, haplotypes tend to cluster into local groups of similar haplotypes. In practice, haplotypes tend to be organized in local clusters, i.e. instead of associating a cluster of origin to a haplotype, one should associate a cluster to each site (SNP) of the haplotype. The cluster of origin of each SNP varies continuously along the haplotype, originating some local patterns. A hidden Markov model is used to model this local clustering of haplotypes.

Bayesian Algorithms

Bayesian algorithms aim at estimating the posterior distribution of parameters given the observed data, assuming some prior knowledge about the distribution of the parameters.

The posterior distribution can be estimated using, for example, Gibbs sampling techniques. Gibbs sampling is a type of Markov chain Monte Carlo procedure. The goal is to draw samples from a posterior distribution. Gibbs sampling constructs a Markov chain which stationary distribution is the true joint distribution.

Algorithm 3 Pseudo-Gibbs sampling algorithm

PHASE($\mathcal{G}$)

1 Take some initial values to the haplotype reconstruction $\mathcal{H}^{(0)}$

2 **for** $t = 0$ **to** y $\triangleright$ y is the maximum number of iterations

3 **do**

4 a) Choose, uniformly and randomly, an ambiguous genotype g_i from $\mathcal{G}$

5 b) Sample $\mathcal{H}_i^{(t+1)}$ from $P(\mathcal{H}_i|\mathcal{G}, \mathcal{H}_{-i}^{(t)})$

6 c) $\mathcal{H}_j^{(t+1)} \leftarrow \mathcal{H}_j^{(t)}$ for $j = 1, \ldots, n, j \neq i$

7 **return** $\mathcal{H}$

The main difference between the Gibbs sampling and the EM algorithm is that the Gibbs algorithm samples from the conditional distribution while the goal of the EM approach is to maximize conditional distributions.

The Pseudo-Gibbs Sampling (PGS) algorithm to haplotype inference [151] is implemented in the well-known statistical haplotype inference solver named PHASE. The PGS algorithm obtains an approximate haplotype sample from the posterior distribution $P(\mathcal{H}|\mathcal{G})$ using Gibbs sampling. The PGS algorithm works as follows. Some initial haplotype reconstruction $\mathcal{H}^{(0)}$ should be guessed. Then, iteratively, for each iteration $t \geq 0$ choose a genotype g_i from $\mathcal{G}$ and calculate $P(\mathcal{H}_i|\mathcal{G}, \mathcal{H}_{-i}^{(t)})$, where $\mathcal{H}_{-i}$ is the set of haplotype pairs excluding the pair for genotype g_i. Then, let $\mathcal{H}_i^{(t+1)}$ be the haplotype pair which maximizes the conditional distribution, i.e.

$$\mathcal{H}_i^{(t+1)} = \max_{\mathcal{H}_i} P(\mathcal{H}_i|G, \mathcal{H}_{-i}^{(t)}),$$

and $H_j^{(t+1)} = \mathcal{H}_j^{(t)}$ for each $j \neq i$. Algorithm 3 systematizes this idea.

The crucial point of the Gibbs sampling method is the computation of the conditional distribution $P(\mathcal{H}_i|\mathcal{G}, \mathcal{H}_{-i})$ because this probability depends on a prior of population haplotype frequencies, as well as knowledge of the genetic and demographic models, which are usually unknown.

Stephens and Donnely [149] proposed an approximation to that distribution based on the coalescent, which corresponds to the probability of choosing one haplotype h_α from $\mathcal{H}$ uniformly at random and then applying a geometric number s of mutations, with parameter $\frac{\theta}{r+\theta}$, according to a

mutation matrix P, i.e.

$$\pi(h|\mathcal{H}) = \sum_{h_\alpha \in E} \sum_{s=0}^{\infty} \frac{r_{h_\alpha}}{r} (\frac{\theta}{r+\theta})^s \frac{r}{r+\theta} (P^s)_{h_\alpha h} \tag{3.8}$$

where r_{h_α} is the number of haplotypes h_α in $\mathcal{H}$, r is the total number of haplotypes in $\mathcal{H}$, and θ is the overall scaled mutation rate across sites.

With this distribution, each haplotype tends to be similar to the previous sampled haplotypes. The similarity between haplotypes grows as the size of $\mathcal{H}$, r, increases and the mutation rate, θ, decreases.

The algorithm implemented in the software tool HAPLOTYPER [131] also follows the Gibbs sampling techniques. The main difference between the PHASE and the HAPLOTYPER algorithms regards the prior distribution. PHASE uses a prior approximating the coalescent model, whereas HAPLOTYPER uses the Dirichlet prior.

The Gibbs sampling method for haplotype inference is impractical for a large number of haplotypes. In order to handle this problem, a partition-ligation technique, which is an application of the divide-conquer technique, is used to infer haplotypes in long SNP sequences [131]. This idea has also been incorporated in PHASE version 2 and the EM algorithm.

Other statistical method, which is commonly used for genetic association studies, is implemented in the BEAGLE tool [19]. The implemented method uses localized haplotype clustering and a hidden Markov model.

A very recent contribution to haplotype inference is the algorithm Shape-IT [34]. This statistical algorithm is based on PHASE version 2, i.e. Shape-IT follows the genetic model of coalescence with recombination. The major algorithmic improvement of Shape-IT is the use of binary trees to represent the set of candidate haplotypes for each individual. With this new representation, the speed of the computations of PHASE can be improved.

3.3 Pedigree-based Haplotype Inference

In many studies, some individuals are closely related with each other and pedigree information is available. When the considered individuals are organized in pedigrees, additional information may be associated with the haplotype inference problem, and can be used to improve the results of the inference methods.

A large number of methods has been proposed for solving haplotype inference problems within

pedigrees. Among those, some are statistical approaches [1, 42, 166, 91, 90, 148, 88] and others are combinatorial approaches [132, 173, 99, 168, 109, 118, 104, 103]. For a more complete survey of haplotyping methods for pedigrees, we refer to [44].

3.3.1 Combinatorial Approaches

Combinatorial approaches for haplotype inference infer haplotypes using the Mendelian law of inheritance and further optimize the solution based on some reasonable assumptions such as minimizing the number of recombinant events or assuming no recombination over haplotypes within a pedigree.

Zero Recombinant Haplotype Configuration (ZRHC)

Recombination events are rare in DNA regions with high linkage disequilibrium. Therefore, most rule-based haplotype inference methods for pedigrees assume no recombination among SNPs within each pedigree [167, 173, 102].

Definition 3.13. ***Zero Recombinant Haplotype Configuration***
The zero recombinant haplotype configuration (ZRHC) problem searches for a solution allowing no recombinations.

The ZRHC problem was first proposed by Wijsman [167] in 1987, who defined a set of twenty logic rules necessary and sufficient to derive haplotypes in pedigrees, under the assumption of no recombination. These rules are still the basis of many other combinatorial algorithms [173]. It can be proved that the ZRHC problem with missing sites is a NP-hard problem [108].

Minimum Recombinant Haplotype Configuration (MRHC)

Although the assumption of no recombination is valid in many cases, this assumption can be violated even for some dense markers [99]. Therefore, the problem of minimizing the number of recombinations was suggested [64, 138, 100].

Definition 3.14. ***Minimum Recombinant Haplotype Configuration***
The Minimum Recombinant Haplotype Configuration (MRHC) problem aims at finding a haplotype inference solution for a pedigree which minimizes the number of required recombination events [64, 138].

The MRHC problem is a well-known approach to solve the haplotype inference problem in pedigrees and it has been shown to be a NP-hard [99, 110] problem. The PedPhase tool [100] implements an integer linear programming model for MRHC with missing sites.

3.3.2 Statistical Approaches

Similar to the population-based haplotype inference, there are two main groups of statistical approaches for haplotype inference: expectation-maximization and Bayesian algorithms.

Expectation-Maximization

Maximum likelihood [42] algorithms aim at finding a solution to the haplotype inference problem which has maximum probability given the observed data on the pedigree.

Definition 3.15. ***Maximum Likelihood Haplotype Configuration***
The maximum likelihood haplotype configuration (MLHC) problem aims at finding the haplotyping solution for all members of the pedigree which maximizes the probability of observing the given genotypes.

There are a number of methods which exactly solve the MLHC approach, among those Genehunter [90], Allegro [58] and Merlin [1], and there are others which perform approximate algorithms, for instance, SimWalk2 [148].

Bayesian Algorithms

Although there are more Bayesian algorithms [169], this section focuses in the Superlink [42] and PhyloPed [88] methods.

Superlink [42] is also a statistical method for haplotype inference in pedigrees, that finds an exact MLHC solution. Superlink uses Bayesian networks as the internal representation of pedigrees, which enables to handle more complex pedigrees.

PhyloPed [88] is a recent statistical approach, and should be used in genome regions which shows little evidence of recombination events. PhyloPed is based in a blocked Gibbs sampler algorithm. Two cases may happen. If there is little evidence of ancestral recombination or recurrent mutations in the founding haplotypes, then the perfect phylogeny model (see Section 3.2.1) is used. Contrariwise, if there is evidence of ancestral recombinations, then founder haplotypes are not restricted to a perfect phylogeny. The algorithm is polynomial in the former case, whereas it is exponential in the

latter case. The PhyloPed method allows more accurate haplotype inference for small number of SNPs.

3.4 Conclusions

Haplotype inference is an important biological problem and a computational challenge. Diploid organisms, such as humans, have pairs of homologous chromosomes, each one inherited from one parent. Although homologous chromosomes are very similar, they differ for some positions, in particular they can differ at sites named single nucleotide polymorphisms (SNPs). Whereas haplotypes correspond to the set of SNPs in a single chromosome, genotypes represent the conflated data of haplotypes in homologous chromosomes. Due to technological limitations, it is not practical to obtain the haplotypes directly. Instead, genotypes are usually obtained. The haplotype inference problem consists in inferring the set of haplotypes that originate a given set of genotypes.

The input data for the haplotype inference problem can be divided into two groups, thus originating two categories of haplotype inference methods: population-based methods and pedigree-based methods. The population-based methods tackle genotypes from unrelated individuals of the same population, whereas the pedigree-based methods tackle pedigree genotype data. Each category of methods can be partitioned in combinatorial and statistical approaches.

Population-based combinatorial approaches include Clark's method [26], the perfect phylogeny haplotyping [61] and the pure parsimony approach [62].

Clark's method [26] was the first computational approach proposed to solve the haplotype inference problem. The algorithm starts with the haplotypes which resolve unambiguous genotypes and proceeds iteratively using an inference rule which aims at resolving each genotype using at least one already used haplotype. The main drawbacks of this approach is that it can leave orphan genotypes and, in addition, there are many ways in which the inference rule can be applied and it is not clear which way produces more accurate results.

The perfect phylogeny haplotyping [61] is directly based on the coalescent model. Assuming the infinite sites model and that there are no recombinations within haplotypes, the evolutionary history of haplotypes can be described in a perfect phylogeny tree. Thus, the perfect phylogeny haplotyping aims at explaining the given genotypes using a set of haplotypes which satisfies a perfect phylogeny. The main drawback of this approach is that perfect phylogeny assumptions may not be realistic, in general. The HAP software implements a relaxed approach which follows an imperfect phylogeny.

The pure parsimony approach [62] aims at finding the minimum number of haplotypes which

can explain a given set of genotypes. This approach is also indirectly based in the coalescent model. Indeed, due to the fact that mutation and recombination events do not occur often, the number of haplotypes in a population tend to be small. This problem is NP-hard but efficient methods, which will be described in the following chapters of this dissertation, have been developed to tackle the problem.

Population-based statistical approaches include expectation-maximization and Bayesian methods.

The expectation-maximization algorithm [41] is used to calculate the maximum likelihood estimates for the input data. Basically, the goal is to obtain the haplotype frequencies which maximize the probability of observing the given genotype data. A drawback of this method is that it can converge to a local optimum instead of converging to the global optimum. Moreover, the algorithm can be impractical because the number of haplotypes is exponential with respect to the number of heterozygous sites.

Bayesian approaches [149, 131, 19, 34] are popular methods for haplotype inference. The main goal is to estimate the posterior distribution of haplotype frequencies given the observed genotypes, assuming some prior knowledge about the haplotype frequencies. The Gibbs sampling method is commonly used to obtain estimates of the posterior distribution of haplotype frequencies. PHASE [149] is a well-known approach which uses a prior approximating the coalescent model, while HAPLOTYPER uses the Dirichlet prior. Moreover, HAPLOTYPER [131] introduces a partition-ligation technique which aims at improving the efficiency of the methods for long regions. More recent Bayesian methods are implemented in the BEAGLE and Shape-IT software. BEAGLE [19] implements localized haplotype clustering and hidden Markov models. Shape-IT [34] is very similar to PHASE but an improvement in the data structures allows the algorithm to be more efficient.

Pedigree-based combinatorial approaches include the zero recombinant haplotype configuration (ZRHC) and the minimum recombinant haplotype configuration (MRHC) problems. The ZRHC problem aims at finding a haplotype inference solution allowing no recombination within the pedigree. Examples of methods that solve the ZRHC are Zaplo [132] and HAPLORE [173]. The MRHC problem aims at finding the haplotype inference solution which minimizes the number of recombination events within pedigrees. Examples of methods that solve the MRHC problem are MRH [138] and PedPhase [100].

The pedigree-based statistical approaches can be grouped in methods that follow an expectation-maximization algorithm and methods that follow Bayesian approaches. Examples of expectation-maximization methods are Allegro [58], Merlin [1] and SimWalk2 [148]. Examples of Bayesian methods are Superlink [42] and PhyloPed [88].

4

Haplotype Inference by Pure Parsimony (HIPP)

The Haplotype Inference by Pure Parsimony (HIPP) problem aims at finding a minimum-cardinality set of haplotypes $\mathcal{H}$ that can explain a given family of genotypes $\mathcal{G}$.

The motivation for chosing a HIPP solution to the haplotype inference problem is based on the coalescent model [74] and the assumption that the mutation rate at each site is small and recombination rates are low. Indeed, few haplotypes seem to be found in genome blocks with small recombination rates. Practical experience confirm that the number of haplotypes in large populations is typically very small, although genotypes exhibit a great diversity [63].

The HIPP problem is APX-hard [92]. Consequently, a significant effort has been made to produce an efficient and exact method for solving the HIPP problem and a considerable number of approaches have been proposed to solve the problem [62, 16, 17, 164, 114, 130, 39, 55, 56, 134]. In addition, several approaches propose heuristic algorithms for solving the HIPP problem [92, 93, 165, 72, 82, 105, 163, 46, 125, 12, 134]. Despite its APX-hardness, the HIPP problem is tractable in some particular cases. Some works study the islands of tractability of the problem [93, 144].

This chapter is an overview of the work that has been developed to study and solve the pure parsimony problem. The first section describes preprocessing techniques. These techniques include structural simplifications which can be used to simplify the problem instance and can be applied before using any HIPP method. Moreover, preprocessing techniques include the computation of lower bounds and upper bounds on the number of haplotypes required, which are used by some HIPP methods. Secondly, a considerable number of exact HIPP methods are detailed. These methods include modulation of the problem with a wide variety of different constraint solving paradigms: integer linear programming, branch-and-bound, Boolean satisfiability and answer set programming.

Algorithm 4 Procedure for eliminating duplicated genotypes

```
RemoveDuplicatedGenotypes(G_{n×m})
1  ▷ G_{n×m} is the set of n genotypes with m sites
2  G' ← G_{n×m}
3  for each g_i ∈ G'
4       do for each g_k ∈ G', k > i
5            if (∀_{1≤j≤m} g_{i j} = g_{k j})
6               then G' ← G_k                    ▷ G_k: set G' without row k
7  return G'
```

Some published work also overviews the HIPP methods [113, 24, 53]. Furthermore, we also describe some techniques which are used to solve the complete HIPP problem. The complete HIPP problem aims at finding *all* parsimonious solutions for a given haplotype inference problem. Follows a section which summarizes the results regarding the complexity of the HIPP problem and its special cases, and another section which overviews heuristic methods for HIPP. A final section concludes the chapter.

4.1 Preprocessing Techniques

When solving the HIPP problem, there are a number of techniques that may be applied during preprocessing. These techniques are inexpensive and empirical evidence shows that they can significantly speed-up the performance of HIPP solvers.

4.1.1 Structural Simplifications

Structural simplifications on the set of genotypes can be performed on all HIPP solvers as a preprocessing technique. These simplifications consist of using the structural properties of genotypes with the purpose of reducing the search space [17, 116].

The set of genotypes given to HIPP solvers contains genotypes from individuals that belong to the same population. Not surprisingly, these sets often contain repeated genotypes, even though each of them refers to a different individual. Clearly, two equal genotypes can be explained by the same pair of haplotypes, and consequently one of them can be discarded. Therefore, for each subset of repeated genotypes only one of them needs to be kept. The procedure to remove duplicated genotypes is described in Algorithm 4.

Moreover, other techniques for reducing the size of a problem instance entail removing sites of

Algorithm 5 Procedure for eliminating duplicated sites

```
RemoveDuplicatedColumns(G_{n×m})
1  ▷ G_{n×m} is the set of n genotypes with m sites
2  G' ← G_{n×m}
3  for j = 1 to m − 1
4        do for j' = j + 1 to m
5           if (∀_{g_i ∈ G'} g_{i j} = g_{i j'})
6              then G' ← G^{j'}                    ▷ G^{j'}: set G' without site j'
7  return G'
```

Algorithm 6 Procedure for eliminating complemented sites

```
RemoveComplementedColumns(G_{n×m})
1  ▷ G_{n×m} is the set of n genotypes with m sites
2  G' ← G_{n×m}
3  for j = 1 to m − 1
4        do for j' = j + 1 to m
5           if (∀_{g_i ∈ G'} (g_{i j} = 2 ∧ g_{i j'} = 2) ∨ (g_{i j} = 1 − g_{i j'}))
6              then G' ← G^{j'}                    ▷ G^{j'}: set G' without site j'
7  return G
```

the genotypes. Indeed, duplicate sites can be discarded. If there are two sites with exactly the same value for each genotype, then one of them can be removed. Algorithm 5 describes the procedure to eliminate duplicated sites.

Furthermore, complemented sites can also be discarded. Two sites, $g_{i\,j}$ and $g_{i\,j'}$, are said to be complemented if for each genotype the two sites are either homozygous with different values or heterozygous, i.e.

$$\forall_{g_i \in \mathcal{G}}(g_{i\,j} = 2 \ \wedge \ g_{i\,j'} = 2) \ \vee \ (g_{i\,j} = 1 - g_{i\,j'}). \tag{4.1}$$

In this case, one of the sites, j or j, may be discarded. Algorithm 6 presents the procedure for removing complemented sites.

Example 4.1. *(Structural Simplifications) Consider the set of genotypes* $\mathcal{G} = \{10111, 10121, 21022, 10111, 12211\}$. *By removing duplicated genotypes, the fourth genotype is removed and the set becomes* $\mathcal{G}' = \{10111, 10121, 21022, 12211\}$. *This set is further reduced by removing duplicated sites, which implies removing the fifth site for being equal to the first site, thus becoming* $\mathcal{G}'' = \{1011, 1012,$

Algorithm 7 Lower bound procedure

LowerBound(G)

1 $\triangleright$ G is the set of genotypes

2 $(G_{LB},\ lb) \leftarrow$ CliqueLowerBound(G)

3 $(G_{LB},\ lb) \leftarrow$ ImprovedLowerBound($G,\ G_{LB},\ lb$)

4 $(G_{LB},\ lb) \leftarrow$ FurtherImprovedLowerBound($G,\ G_{LB},\ lb$)

5 **return** lb

Algorithm 8 Clique lower bound

CliqueLowerBound(G)

1 $\triangleright$ G is the set of genotypes

2 Create an incompatible graph $I = (G, E)$

- each vertex is a genotype
- set of edges is $E = \{(g_i, g_k) :\ g_i, g_k \in G\ \wedge\ \neg\text{Compatible}(g_i, g_k)\}$

3 Compute maximal clique of mutually incompatible genotypes: G_C

- $\forall_{g_i,g_k \in G_C}\ g_i \neq g_k \Rightarrow \neg\text{Compatible}(g_i, g_k)$

4 $lb \leftarrow (2 \times \#G_C - \sigma)$ $\triangleright$ lb: lower bound obtained from G_C

$\triangleright$ $\sigma = \#\{g \in G_C :\ \neg\text{Heterozygous}(g)\}$

5 **return** $(G_C,\ lb)$

2102, 1221}. *Finally, we may remove the third site for being complemented with the second site, thus getting the simplified set* $\mathcal{G}''' = \{101, 102, 212, 121\}$.

Assuming that information about the genotypes and sites discarded is kept, it is straightforward to construct a solution for the original set of genotypes once a solution to the simplified set of genotypes has been discovered.

4.1.2 Lower Bounds

A key issue in some HIPP approaches [114, 39] is to find tight lower bounds on the minimum number of haplotypes required.

This section describes a method for computing lower bounds which integrates three different techniques. The procedure is described in Algorithm 7, and consists of three routines: CliqueLowerBound, ImprovedLowerBound and FurtherImprovedLowerBound, which are described in the following paragraphs. The first two lower bounds have been proposed in the context of the SHIPs approach [114]. The SHIPs algorithm will be explained in Section 4.4.1.

Algorithm 9 Improved lower bound

```
ImprovedLowerBound(G, G_C, lb)
 1  ▷ G: set of genotypes; G_C: set of genotypes on the clique lower bound
 2  ▷ lb: lower bound obtained from G_C
 3  Sort G_NC by increasing number of heterozygous sites                ▷ Optional step
 4  G_NC ← G − G_C                                  ▷ G_NC: set of non-clique genotypes
 5  G_LB ← G_C                          ▷ G_LB: set of genotypes contributing to lower bound
 6  G_S ← G_C                                    ▷ Working set of genotypes starts with G_C
 7  for each g_i ∈ G_NC                              ▷ Analyze genotypes in (sorted) order
 8        do S ← {cg ∈ G_S : Compatible(g, cg)}
 9           if (∃_{1≤j≤m} (g_i j = 2) ∧ ∃_{v∈{0,1}} ∀_{s_k∈S} s_k j = v)
10              then
11                   lb ← lb + 1
12                   ▷ Set sites with differing values to 2 and update G_S
13                   ng ← MergeGenotypes(S, g)
14                   G_S ← (G_S − S) ∪ {ng}
15                   G_LB ← G_LB ∪ {g}
16  return (G_LB, lb)

MergeGenotypes(S, g)
1  ▷ S: set of genotypes; g: a genotype
2  ng_i ← g
3  for each s_k ∈ S
4        do for j = 1 to m
5              do if (s_k j ≠ ng_i j)
6                    then ng_i j ← 2
7  return ng
```

The techniques for computing lower bounds rely on information regarding (in)compatible genotypes. Two genotypes of the same length are *compatible* if there is no site such that one genotype has value 0 and the other genotype has value 1 (see definition 3.1).

A lower bound can be computed from a maximal clique [114]. Clearly, for two incompatible genotypes, g_i and g_l, the haplotypes that explain g_i must be distinct from the haplotypes that explain g_l. Given the incompatibility relation we can create an incompatibility graph I, where each vertex is a genotype, and two vertexes are connected with an edge if they are incompatible. Suppose I has a clique of size k. Then the number of required haplotypes is at least $2k - \sigma$, where σ is the number of genotypes in the clique which do not have heterozygous sites. In theory, a maximal clique

Algorithm 10 Further improved lower bound

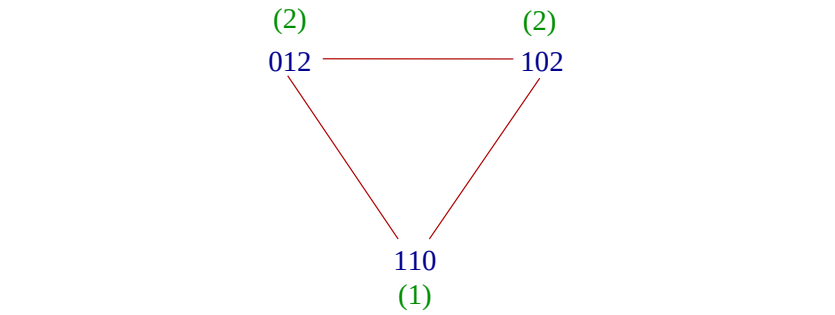
```
FurtherImprovedLowerBound(G, G_LB, lb)
 1  ▷ G is the set of genotypes
 2  ▷ G_LB is the set of genotypes contributing to the lower bound
 3  ▷ lb is the lower bound obtained from G_LB
 4  G_NLB ← G − G_LB                          ▷ G_NLB: set of genotypes not in G_LB
 5  for each g_i ∈ G_NLB
 6       do
 7          for each j, j', j'' : 1 ≤ j < j' < j'' ≤ m
 8               do
 9                  if (g_{i j} = g_{i j''} = g_{i j'''} = 2)
10                     then
11                          S ← {cg ∈ G_LB : Compatible(g, cg)}
12                          if (∃_{c∈{0,1}} ∀_{gl_k∈S} ∃_{k1,k2∈{j,j',j''}} (gl_{k k1} = gl_{k k2} = c))
13                             then
14                                  lb ← lb + 1
15                                  G_LB ← G_LB ∪ {g}
16                                  G_NLB ← G_NLB − {g}
17  return (G_LB, lb)
```

should be found, in order to provide a tighter lower bound. However, since the problem of computing a maximal clique is NP-hard [45], we use the size of a clique computed using a simple greedy heuristic. The genotype with the highest number of incompatible genotypes is first selected. At each step, the selected genotype is one that is still incompatible with all the already selected genotypes, and preference is given to the haplotype with the highest number of incompatible genotypes. Algorithm 8 illustrates the procedure CliqueLowerBound, which computes the clique-based lower bound.

Example 4.2. *(Clique-based Lower Bounds) Consider the following set of genotypes:* $\mathcal{G} = \{110, 012, 102\}$. *The three genotypes are incompatible, which is represented in the incompatibility graph in Figure 4.1.2, along with each genotype contribution to the lower bound. Genotype* 110 *in homozygous and thus contributes with 1 to the lower bound. Each one of the genotypes* 012 *and* 102 *contributes with 2 for the lower bound. Hence, the number of required haplotypes is at least 5 (twice the clique size less the number of genotypes with no heterozygous sites).*

In addition, the analysis of the structure of the genotypes allows the lower bound to be further increased, by identifying heterozygous sites which require at least one additional haplotype given a set of previously chosen genotypes [116]. The procedure starts from the clique-based lower bound

Figure 4.1: Clique lower bound



and grows the lower bound by searching for heterozygous sites among genotypes not yet considered for lower bounding purposes. For each genotype g_i that is not in the clique, if the genotype has a heterozygous site and all compatible genotypes have the same value at that site (either 0 or 1), then g_i is guaranteed to require one additional haplotype to be explained. Hence the lower bound can be increased by 1. Algorithm 9 presents the pseudo-code of procedure ImprovedLowerBound for calculating the improved lower bound.

Furthermore, another improvement to the lower bound consists in identifying genotypes with triples of heterozygous sites, among the genotypes not used in the clique lower bound [113]. Algorithm 10 presents the pseudo-code of the algorithm FurtherImprovedCliqueLowerBound for calculating the lower bound based on the triples of heterozygous sites.

Example 4.3. *(Improved Lower Bounds) Consider the following set of genotypes:* $\mathcal{G} = \{200, 020, 002, 222\}$. *Given that there are no two incompatible genotypes, the clique-based lower bound would give a lower bound of 2 corresponding to a unique vertex (e.g. with the first genotype). The analysis of the structure of the remaining genotypes requires one additional haplotype for the second and the third genotype, thus increasing the lower bound to 4 haplotypes. This lower bound can be further improved by analyzing the fourth haplotype 222. Any of the haplotypes already included in the lower bound requires* at least *two positions with value 0. But the pair of haplotypes explaining 222 will require one haplotype with* at most *one position with value 0. Hence, the lower bound can be increased to 5.*

4.1.3 Upper Bounds

The computation of tight upper bounds on the HIPP solution is also an important issue on some exact approaches [12, 39].

In general, every heuristic method for the HIPP approach can be used to provide an upper bound. This section focus on two heuristic methods used for computing upper bounds: the delayed haplotype selection algorithm and the CollHaps approach. Both of these methods are based on the Clark's method, described in Section 3.2.1.

Delayed Selection

Clark's method may be used to compute an upper bound to the HIPP problem. However, this method is often too greedy, at each step seeking to explain each non-explained genotype with the most recently chosen haplotype. The *Delayed Selection* (DS) [125] is an alternative algorithm whose main motivation is to avoid the excessive greediness of Clark's method in selecting new haplotypes.

The DS algorithm maintains two sets of haplotypes: the *selected* haplotypes, which represent haplotypes that have been chosen to be included in the target solution, and the *candidate* haplotypes, which represent haplotypes that can explain one or more genotypes not yet explained by a pair of selected haplotypes.

Algorithm 11 presents the pseudo-code of the procedure to calculate the upper bound using delayed selection. The initial set of selected haplotypes, H_S, corresponds to all haplotypes which are required to explain unambiguous genotypes, i.e. genotypes with no more than one heterozygous site. The set of unexplained genotypes, G, is updated during the algorithm and genotypes which can be totally explained by haplotypes in H_S are removed.

The initial set of candidate haplotypes, H_C, is then calculated. If a genotype g can be explained by one selected haplotype h_s, then the haplotype h_c such that $g = h_s \oplus h_c$ is added to the set of candidate haplotypes, H_C.

At each step, the DS algorithm chooses the candidate haplotype h_c which can explain the largest number of genotypes. The chosen haplotype h_c is then used to identify additional candidate haplotypes. Moreover, h_c is added to the set of selected haplotypes, and all genotypes which can be explained by a pair of selected haplotypes are removed from the set of unexplained genotypes. The algorithm terminates when all genotypes have been explained.

Each time the set of candidate haplotypes becomes empty, and there are still genotypes to be explained, a new candidate haplotype is generated. The new haplotype is selected greedily as the haplotype which can explain the largest number of genotypes not yet explained.

Observe that the proposed organization allows selecting haplotypes which will not be used in the final solution. The last step of the algorithm is to remove from the set of selected haplotypes all haplotypes which are not used for explaining any genotypes.

Algorithm 11 Delayed haplotype selection

```
DelayedHaplotypeSelection(G)
 1  ▷ H_S is the set of selected haplotypes; H_C is the set of candidate haplotypes
 2  H_S ← CalcInitialHaplotypes(G)
 3  G ← RemoveExplainedGenotypes(G, H_S)
 4  for each h_s ∈ H_S
 5        do for each g ∈ G
 6              do if CanExplain(h_s, g)
 7                    then h_c ← CalcExplainPair(h, g)
 8                         H_C ← H_C ∪ {h_c}
 9                         Associate h_c with g
10  while (G ≠ ∅)
11        do if (H_C = ∅)
12              then
13                   h_c ← PickCandHaplotype(G)
14                   H_C ← {h_c}
15           h ← h_c ∈ H_C associated with largest number of genotypes
16           H_C ← H_C − {h}
17           H_S ← H_S ∪ {h}
18           G ← RemoveExplainedGenotypes(G, H_S)
19           for each g ∈ G
20                do if CanExplain(h, g)
21                      then h_c ← CalcExplainPair(h, g)
22                           H_C ← H_C ∪ {h_c}
23                           Associate h_c with g
24  H_S ← RemoveNonUsedHaplotypes(H_S)
25  return H_S
```

Example 4.4. *(Upper Bounds - Delayed Selection) Consider the set of genotypes* $G = \{1010, 0002, 2211, 2222\}$. *$G$ has two genotypes with no more than one heterozygous site,* 1010 *and* 0002. *Therefore, the initial set of selected haplotypes is* $H_S = \{1010, 0000, 0001\}$. *Hence, the set of unexplained genotypes is reduced to* $G' = \{2211, 2222\}$. *Using H_S to partially explain the genotypes, a set of candidate haplotypes is defined* $H_C = \{0101, 1111, 1110\}$. *The candidate haplotype* $h_c \in H_C$ *which explains the largest number of genotypes is selected,* $h_c = 1111$. *The set of selected haplotypes becomes* $H'_S = \{1010, 0000, 0001, 1111\}$, *the new explained genotype is removed and the set of unexplained genotypes becomes* $G'' = \{2211\}$. *Finally, using the selected haplotype* 1111, *a new candidate haplotype is selected,* 0011, *and* $H''_S = \{1010, 0000, 0001, 1111, 0011\}$. *At this point all*

genotypes have been explained and, therefore, the algorithm terminates. Hence, the upper bound computed by the DS algorithm is 5.

The DS algorithm runs in polynomial time in the number of genotypes (n) and sites (m). A straightforward analysis yields a run time complexity in $\mathcal{O}(n^2\, m)$.

CollHaps

CollHaps is an alternative approach [156] which can be used as a heuristic method for solving the haplotype inference problem based on pure parsimony, but also as an upper bound algorithm for HIPP solutions. Indeed, the CollHaps algorithm is used as a preprocessor to obtain upper bounds by some exact HIPP solvers [12].

The main concept in which CollHaps relies is the *collapse rule* which is a generalization of the Clark's rule [26]. The CollHaps algorithm consists in the iterative application of collapse rules.

In practice, CollHaps associates two symbolic haplotypes, h_{2i-1} and h_{2i}, with each genotype $g_i \in \mathcal{G}$, defined by

$$h_{2i-1\,j} = \begin{cases} 0 & \text{if} \quad g_{i\,j} = 0 \\ 1 & \text{if} \quad g_{i\,j} = 1 \\ x_{\mu(i,j)} & \text{if} \quad g_{i\,j} = 2 \end{cases} \tag{4.2}$$

and

$$h_{2i\,j} = \begin{cases} 0 & \text{if} \quad g_{i\,j} = 0 \\ 1 & \text{if} \quad g_{i\,j} = 1 \\ \neg x_{\mu(i,j)} & \text{if} \quad g_{i\,j} = 2 \end{cases}, \tag{4.3}$$

where $\mu(i,j)$ is the bijective mapping that given a pair (i,j), such that $g_{i\,j} = 2$, returns an integer p, with $1 \leq p \leq r$, where r is the number of heterozygous sites in $\mathcal{G}$, with $1 \leq i \leq n$ and $1 \leq j \leq m$. The matrix $H_{2n\times m}$ whose rows are the symbolic haplotypes h_k, with $1 \leq k \leq 2n$, is referred to as the *symbolic haplotype matrix*. This matrix is continuously updated during the algorithm's execution until there are no more variables on the symbolic matrix. The pure parsimony criterion aims at determining an assignment to the variables x which minimizes the number of distinct rows. The CollHaps approach suggests an heuristic for the problem.

CollHaps is based on successive applications of the collapse rule to the rows of matrix H. Given a pair of symbolic haplotypes, the *collapse rule* corresponds to the set of variable assignments, which lead the two haplotypes to become identical, although requiring the minimum number of assignments to constant values. Thus, it is important to note that some variables may be assigned

to other variables. For example, consider two symbolic haplotypes $h_i = (1\ x_1\ 0\ x_2)$ and $h_k = (1\ x_3\ x_4\ x_5)$. A collapse rule for haplotypes h_i and h_k will be the assignment $\rho_{i\,k}$ such that $\rho_{i\,k}(x_1) = x_3$, $\rho_{i\,k}(x_2) = x_5$ and $\rho_{i\,k}(x_4) = 0$. Two symbolic haplotypes, h_i and h_k, are compatible iff

$$\forall_{j:\ 1\leq j\leq m}\ (h_{i\,j} \neq 0 \vee h_{k\,j} \neq 1) \wedge \forall_{p:\ 1\leq p\leq r}\ (h_{i\,j} \neq x_p \vee h_{k\,j} \neq \bar{x}_p))$$

Clearly, only pairs of compatible symbolic haplotypes can be considered for collapsing. Moreover, note that Clark's rule is a special case of the collapse rule where one of the two haplotypes to be collapsed is unambiguous. This rule implies, in general, a larger number of variables assigned to constants.

In general, the same variable has multiple occurrences within H, and the assignment expressed by a collapse rule has to be propagated to all occurrences of that variable. This process is named the *collapse step*.

It can be proved that every pure parsimony solution can be obtained by a successive applications of collapse steps [156]. However, it is clear that exhaustive exploration of all collapse sequences would be infeasible. Hence, a heuristic strategy is required. The basic idea of the CollHaps heuristic is to delay as much as possible the assignments of the variables to constants, in order to avoid increasing the number of incompatibilities.

CollHaps opts by a randomized quasi-greedy choice of the haplotypes to be collapsed. Let the distance between two haplotypes, h_i and h_k, be the number of variables that must be assigned to constants to collapse h_i and h_k. Pairs of haplotypes at a lower distance have a higher probability of being randomly chosen than those pairs at higher distance.

After applying the sequence of collapsing steps, a set of haplotypes $\mathcal{H}$ is obtained. Afterward, a simple greedy technique is used to further reduce the number of haplotypes used. Note that some genotypes $g \in \mathcal{G}$ may be explained by more than one pair of haplotypes in $\mathcal{H}$. Hence, for each genotype, the "most useful" pair of haplotypes, i.e. the pair whose haplotypes can explain more genotypes, is chosen.

Example 4.5. *(Upper Bounds - CollHaps) Consider the same set of genotypes of example 4.4, i.e.* $\mathcal{G} = \{1010, 0002, 2211, 2222\}$. *One possible sequence of collapse rules is* $\rho_{5,7}$ *and* $\rho_{3,8}$, *such that* $\rho_{5,7}(x_2) = x_4$, $\rho_{5,7}(x_3) = x_5$, $\rho_{5,7}(x_6) = 1$, $\rho_{5,7}(x_7) = 1$ *and* $\rho_{3,8}(x_1) = 0$, $\rho_{3,8}(x_4) = 1$, $\rho_{3,8}(x_5) = 1$.

The sequence of corresponding symbolic haplotype matrices is

$$\begin{pmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \end{pmatrix} = \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ 1 & 0 & 1 & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{x_1} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{\bar{x}_1} \\ \mathbf{x_2} & \mathbf{x_3} & \mathbf{1} & \mathbf{1} \\ \mathbf{\bar{x}_2} & \mathbf{\bar{x}_3} & \mathbf{1} & \mathbf{1} \\ \mathbf{x_4} & \mathbf{x_5} & \mathbf{x_6} & \mathbf{x_7} \\ \mathbf{\bar{x}_4} & \mathbf{\bar{x}_5} & \mathbf{\bar{x}_6} & \mathbf{\bar{x}_7} \end{pmatrix} \xrightarrow{\rho_{5,7}} \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ 1 & 0 & 1 & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{x_1} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{\bar{x}_1} \\ \mathbf{x_4} & \mathbf{x_5} & \mathbf{1} & \mathbf{1} \\ \mathbf{\bar{x}_4} & \mathbf{\bar{x}_5} & \mathbf{1} & \mathbf{1} \\ \mathrm{x}_4 & \mathrm{x}_5 & 1 & 1 \\ \mathbf{\bar{x}_4} & \mathbf{\bar{x}_5} & \mathbf{0} & \mathbf{0} \end{pmatrix} \xrightarrow{\rho_{3,8}} \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ 1 & 0 & 1 & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{1} \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Therefore, the upper bound calculated by the CollHaps algorithm is 5.

Finally, note also that some haplotypes in $\mathcal{H}$ may still be symbolic, i.e, containing one or more variables. A final step makes the assignment for these variables. All assignments originate solutions with the same number of distinct haplotypes. Hence, the strategy aims at improving the accuracy of the method and is based on the coalescent model. For each symbolic haplotype $h_s \in \mathcal{H}$, the variable-free haplotype $h_v \in \mathcal{H}$ requiring the fewest SNP switches to be transformed into h_s is searched. The variables in h_s are then assigned according to the constant values in h_v.

4.2 Integer Linear Programming Models

4.2.1 RTIP

The first proposed approach to solve the HIPP problem is an exponential integer linear programming (ILP) model called RTIP [62]. For each genotype $g_i \in \mathcal{G}$, the conceptual formulation enumerates all pairs of haplotypes which can explain g_i. Note that if the genotype g_i has r_i heterozygous sites, then the number of explaining pairs is 2^{r_i-1}. Consequently, the space complexity of the model is $\mathcal{O}(2^m)$, where m is the number of sites of each genotype, which represents the maximum number of heterozygous sites per genotype.

In practice, a Boolean variable $y_{i\,j}$ is associated with each haplotype pair capable of explaining genotype g_i, with $1 \leq i \leq n$ and $1 \leq j \leq 2^{r_i-1}$. Moreover a Boolean variable x_k is associated with each distinct haplotype $h_k \in H$, where H is the set of different haplotypes in this expansion. A variable has value 1 when the respective pair/haplotype is chosen to be in the solution.

The objective function ensures that the number of x_k variables assigned value 1 is the minimum

Table 4.1: The RTIP formulation

minimize:	$\sum_{k=1}^{\lvert H\rvert} x_k$	(4.4)
subject to:		
$\sum_{j=1}^{2^{r_i-1}} y_{i\,j} = 1,$	$\forall_{i:\ 1\leq i\leq n}$	(4.5)
$y_{i\,j} - x_{k_{i\,j_1}} \leq 0,$	$\forall_{i:\ 1\leq i\leq n},\ \forall_{j\ :1\leq j\leq m}$	(4.6)
$y_{i\,j} - x_{k_{i\,j_2}} \leq 0,$	$\forall_{i:\ 1\leq i\leq n},\ \forall_{j:\ 1\leq j\leq m}$	(4.7)

possible, implying that the minimum number of distinct haplotypes is selected,

$$\min \sum_{k=1}^{|H|} x_k, \tag{4.4}$$

where $|H|$ is the cardinality of set H. The constraints are as follows. Each genotype g_i must be explained by exactly one pair of haplotypes, i.e.

$$\sum_{j=1}^{2^{r_i-1}} y_{i\,j} = 1. \tag{4.5}$$

In addition, the utilization of a specific pair of haplotypes implies the selection of the associated haplotypes, i.e.

$$y_{i\,j} - x_{k_{i\,j_1}} \leq 0 \tag{4.6}$$

$$y_{i\,j} - x_{k_{i\,j_2}} \leq 0, \tag{4.7}$$

where $x_{k_{i\,j_1}}$ and $x_{k_{i\,j_2}}$ represent the two haplotypes (not necessarily distinct) which belong to pair $y_{i\,j}$. The RTIP formulation is summarized in Table 4.1.

As stated previously, this formulation, and thus the running time required to create it, increases exponentially with the problem size, and therefore it is impractical to use without additional improvements. For this reason, RTIP considers only haplotype pairs where at least one haplotype could partly explain more than one genotype. This simplification reduces significantly the size of the formulation, specially when the value of recombinations are higher. As the level of recombination rises, haplotypes become more differentiated and there are more pairs that can be discarded in the RTIP approach, and a smaller model is produced. Anyway, RTIP remains an exponential-sized model in the worst case and, consequently, requires significant computational memory resources.

Table 4.2: The PolyIP formulation

minimize:	$\sum_{i=1}^{2n} u_i$	(4.8)
subject to:		
$t_{2i-1\,j} + t_{2i\,j} = \begin{cases} 0 & \text{if} \quad g_{i\,j} = 0 \\ 2 & \text{if} \quad g_{i\,j} = 1 \\ 1 & \text{if} \quad g_{i\,j} = 2 \end{cases}$	$\forall_{i:\ 1 \le i \le n},\ \forall_{j:\ 1 \le j \le m}$	(4.9)
$x_{i\,k} \ge t_{i\,j} - t_{k\,j}$	$\forall_{i,k:\ 1 \le i < k \le 2n},\ \forall_{j:\ 1 \le j \le m}$	(4.10)
$x_{i\,k} \ge t_{k\,j} - t_{i\,j}$	$\forall_{i,k:\ 1 \le k < i \le 2n},\ \forall_{j:\ 1 \le j \le m}$	(4.11)
$u_i \ge 2 - i + \sum_{k=1}^{i-1} x_{i\,k}$	$\forall_{i:\ 1 \le i \le 2n}$	(4.12)

4.2.2 PolyIP

PolyIP [16] is an ILP approach to solve the HIPP problem which uses a formulation that is polynomial on the number of genotypes and sites.

Different authors independently and almost simultaneously suggest similar models for HIPP [65, 92]. Despite the similarities between the models, here we focus on the PolyIP formulation from [16] which, to the best of our knowledge, was the only one which was implemented and tested in practice by the authors.

The PolyIP model is as follows. PolyIP associates two haplotypes, h_{2i-1} and h_{2i}, with each genotype $g_i \in \mathcal{G}$. A variable is associated with each site of h_{2i-1} and h_{2i}. A Boolean variable $t_{k\,j}$ represents site j of haplotype h_k ($1 \le k \le 2n$, $1 \le j \le m$), where n is the number of genotypes on set $\mathcal{G}$. Moreover, each haplotype is associated with a Boolean variable u_i ($1 \le i \le 2n$) which has value 1 when haplotype h_i is different from all haplotypes used to explain genotypes with lower index.

The objective function consists in minimizing the number of distinct haplotypes used, i.e. minimize the number of variables u_i assigned value 1,

$$\min \sum_{i=1}^{2n} u_i. \tag{4.8}$$

Constraints should ensure that h_{2i-1} and h_{2i} really explain g_i $(1 \leq i \leq n)$,

$$t_{2i-1\,j} + t_{2i\,j} = \begin{cases} 0 & \text{if} \quad g_{i\,j} = 0 \\ 2 & \text{if} \quad g_{i\,j} = 1 \\ 1 & \text{if} \quad g_{i\,j} = 2 \end{cases} . \tag{4.9}$$

In order to count the number of different haplotypes used, Boolean variables $x_{i\,k}$ $(1 \leq k < i \leq 2n)$ are introduced, such that $x_{i\,k}$ has value 1 when h_i is different from haplotype h_k. If h_i is different from h_k, there must exist a site j $(1 \leq j \leq m)$ such that $t_{i\,j} \neq t_{k\,j}$. Hence the constraints on variables $x_{i\,k}$ are

$$x_{i\,k} \geq t_{i\,j} - t_{k\,j}, \tag{4.10}$$

$$x_{i\,k} \geq t_{k\,j} - t_{i\,j}. \tag{4.11}$$

In addition, variable u_i $(1 \leq i \leq 2n)$ must have value 1 when haplotype h_i is different from all haplotypes used to explain genotypes with lower index,

$$u_i \geq 2 - i + \sum_{k=1}^{i-1} x_{i\,k}. \tag{4.12}$$

Observe that $\sum_{k=1}^{i-1} x_{i\,k} = i - 1$ if h_i is distinct from all previous haplotypes, by the definition of $x_{i\,k}$.

The PolyIP formulation is summarized in Table 4.2. The number of constraints and variables of the PolyIP model are, respectively, in $\Theta(n^2 m)$ and $\Theta(n^2 + nm)$.

4.2.3 HybridIP

The same authors of PolyIP [16] also suggest an alternative polynomial-size ILP model, named HybridIP [17], which represents a hybrid between the RTIP and the PolyIP models. Although being an exponential model, RTIP is, in general, faster than the polynomial model for solving small instances. HybridIP was formulated to join the strength of RTIP and PolyIP, in an approach with both practical size and able to run within reasonable run times. Nonetheless, experimental results, to be given in Section 6, show that no significant improvements were achieved by this new model, when compared with PolyIP.

Table 4.3: The HaploPPH formulation

minimize:	$\sum_{i=1}^{2n} u_i$				(4.13)
subject to:					
$u_{2i+1} \leq u_{2i}$				$\forall_{i:\ 1\leq i\leq n}$	(4.14)
$\sum_{i=1}^{2n}\sum_{i'=i}^{2n} y^k_{i\,i'} \geq 1$				$\forall_{k:\ 1\leq k\leq n}$	(4.15)
$\sum_{i'=i}^{n} y^k_{i\,i'} + \sum_{i'=1}^{i-1} y^k_{i'\,i} \leq u_i$				$\forall_{k:\ 1\leq k\leq n},\ \forall_{i:\ 1\leq i\leq 2n}$	(4.16)
$y^i_{2i\,2i+1} \leq u_{2i+1}$				$\forall_{i:\ 1\leq i\leq n}$	(4.17)
$t_{2k-1\,j} + t_{2k\,j} = \begin{cases} 0 & \text{if} \quad g_{k\,j} = 0 \\ 2 & \text{if} \quad g_{k\,j} = 1 \\ 1 & \text{if} \quad g_{k\,j} = 2 \end{cases}$				$\forall_{k:\ 1\leq k\leq n},\ \forall_{j:\ 1\leq j\leq m}$	(4.18)
$1 - \sum_{i'=i}^{n} y^k_{i\,i'} - \sum_{i'=1}^{i-1} y^k_{i'\,i} \geq t_{i\,j}$	if	$g_{k\,j} = 0$		$\forall_{i:\ 1\leq i\leq n},\ \forall_{j:\ 1\leq j\leq m}$	(4.19)
$\sum_{i'=i}^{n} y^k_{i\,i'} + \sum_{i'=1}^{i-1} y^k_{i'\,i} \leq t_{i\,j}$	if	$g_{k\,j} = 1$		$\forall_{k:\ 1\leq k\leq n},\ \forall_{i,i':\ 1\leq i\leq i'\leq 2n},\ \forall_{j:\ 1\leq j\leq m}$	(4.20)
$t_{i\,j} + t_{i'\,j} \geq y^k_{i\,i'}$	if	$g_{k\,j} = 2$		$\forall_{k:\ 1\leq k\leq n},\ \forall_{i,i':\ 1\leq i\leq i'\leq 2n},\ \forall_{j:\ 1\leq j\leq m}$	(4.21)
$t_{i\,j} + t_{i'\,j} \leq 2 - y^k_{i\,i'}$	if	$g_{k\,j} = 2$		$\forall_{k:\ 1\leq k\leq n},\ \forall_{i,i':\ 1\leq i\leq i'\leq 2n},\ \forall_{j:\ 1\leq j\leq m}$	(4.22)

4.2.4 HaploPPH

A more recent ILP polynomial model to the HIPP problem is the HaploPPH model [23].

This model is based on the following idea. Each HIPP solution can be represented by a bipartite graph between genotypes and haplotypes [15]. There is an edge between haplotype h and genotype g if h is chosen to explain g. Hence, each heterozygous genotype g has exactly degree two, corresponding to the two haplotypes which explain g. Each haplotype h in the final solution has degree at least one because h must explain at least one genotype. Moreover, genotypes can be grouped into classes. Each class S is induced by a haplotype h. Therefore, each class S contains the genotypes that are explained by haplotype h.

An index is associated with each subset S of genotypes induced by a haplotype h. Specifically, let i be the smallest index associated with a genotype in S, i.e. $i = \min\{k :\ g_k \in S,\ 1 \leq k \leq n\}$. However, note that each genotype g can belong to two classes and it may happen that g is the genotype with the smallest index in both subsets. Hence, $2i$ is associated with S if S_{2i} is still not defined; otherwise, $2i + 1$ is associated with S. The maximum number of classes is $2n$ because at most $2n$ haplotypes are used to explain a set with n genotypes.

The HaploPPH model is defined as follows. Firstly, $2n$ symbolic haplotypes are created. A variable u_i $(1 \leq i \leq 2n)$ is associated with each set S_i such that u_i is assigned value 1 when there exists a haplotype inducing class S_i. The HIPP approach searches for the minimum number of haplotypes which induce classes. Therefore, the objective function of the model is

$$\min \sum_{i=1}^{2n} u_i, \tag{4.13}$$

such that

$$u_{2i+1} \leq u_{2i}, \tag{4.14}$$

because class S_{2i+1} can only be defined after class S_{2i} is defined. Moreover, a variable $y^k_{i\,i'}$ is associated with each genotype g_k and two classes S_i and $S_{i'}$, with $i < i'$. Variable $y^k_{i\,i'}$ is assigned value 1 when genotype g_k belongs both to class S_i and class $S_{i'}$. Every genotype g_k must belong exactly to two classes, and therefore,

$$\sum_{i=1}^{2n} \sum_{i'=i}^{2n} y^k_{i\,i'} \geq 1, \tag{4.15}$$

for each k, with $1 \leq k \leq n$.

If a genotype g_k belongs to a class S_i, the value of u_i (representing whether class S_i is defined) must be taken into account,

$$\sum_{i'=i}^{n} y^k_{i\,i'} + \sum_{i'=1}^{i-1} y^k_{i'\,i} \leq u_i \tag{4.16}$$

When a genotype g_i belongs to both S_{2i} and S_{2i+1}, then u_{2i+1} must be assigned 1,

$$y^i_{2i\,2i+1} \leq u_{2i+1}. \tag{4.17}$$

Moreover, the model includes variables $t_{i\,j}$ associated with the j^{th} site of the haplotype which induces set S_i, with $1 \leq i \leq 2n$ and $1 \leq j \leq m$. Therefore, it must be imposed that the haplotypes explain the genotypes,

$$t_{2k-1\,j} + t_{2k\,j} = \begin{cases} 0 & \text{if} \quad g_{k\,j} = 0 \\ 2 & \text{if} \quad g_{k\,j} = 1 \\ 1 & \text{if} \quad g_{k\,j} = 2 \end{cases}, \tag{4.18}$$

for $1 \leq k \leq n$ and $1 \leq j \leq m$. Furthermore, if the haplotype h inducing set S_i explains a genotype

g_k belonging to S_i, then h must be such that, for $1 \leq j \leq m$, if $g_{k\,j} = 0$ then $h_{i\,j} = 0$ and if $g_{k\,j} = 1$ then $h_{i\,j} = 1$:

$$1 - \sum_{i'=i}^{n} y^k_{i\,i'} - \sum_{i'=1}^{i-1} y^k_{i'\,i} \geq t_{i\,j} \quad \text{if} \quad g_{k\,j} = 0, \tag{4.19}$$

$$\sum_{i'=i}^{n} y^k_{i\,i'} + \sum_{i'=1}^{i-1} y^k_{i'\,i} \leq t_{i\,j} \quad \text{if} \quad g_{k\,j} = 1. \tag{4.20}$$

Finally, if g_k belongs to class $S_i \cap S_{i'}$ and $g_{k\,j} = 2$ then exactly one of the haplotypes inducing sets S_i and $S_{i'}$ must be 1 at site j,

$$t_{i\,j} + t_{i'\,j} \geq y^k_{i\,i'}, \tag{4.21}$$

$$t_{i\,j} + t_{i'\,j} \leq 2 - y^k_{i\,i'}. \tag{4.22}$$

The HaploPPH formulation is summarized in Table 4.3. All variables of this formulation are Boolean. The number of variables is $\Theta(n^3 + nm)$ and the number of constraints is $\Theta(n^2 m)$.

The presented formulation corresponds to the basic model. The original model was reduced noting that a significant number of variables $y^k_{i\,i'}$ are actually redundant. Moreover, also the value of variables $t_{i\,j}$ associated with homozygous sites can be fixed. Finally, some inequalities are added to the formulation in order to strengthen the model.

4.2.5 P^{UB}_{max}

A new polynomial ILP formulation is referred to as P^{UB}_{max} [12]. The main characteristic of this approach is that the HIPP problem is turned into a maximization problem. Starting from an upper bound UB on the number of haplotypes, the formulation aims at finding the maximum number of genotypes which can be explained using $r = UB - 1$ haplotypes. If the solution for the formulation is $n = |\mathcal{G}|$, i.e. the total number of genotypes of the problem, then the value of the upper bound is decremented, $UB = UB - 1$, and the procedure is repeated. Otherwise, if the solution for the formulation is lower than n then the number of haplotypes required by the parsimony problem is $s = UB$.

The upper bound UB on the optimal solution is provided by the heuristic approach Coll-Haps [156], described in Section 4.1.3.

At each iteration, considering $r = UB - 1$ haplotypes, the problem is modeled by an ILP formulation. The P^{UB}_{max} model only uses two types of binary variables. Similarly to RTIP, the model

Table 4.4: The P^{UB}_{max} formulation (considering r haplotypes)

maximize:	$\sum_{k=1}^{n}\sum_{i=1}^{r-1}\sum_{i'=i+1}^{r} y^k_{i\,i'}$			(4.23)
subject to:				
$\sum_{i=1}^{r-1}\sum_{i'=i+1}^{r} y^k_{i\,i'} \leq 1$			$\forall_{k:\ 1\leq k\leq n}$	(4.24)
$\sum_{k=1}^{n} y^k_{i\,i'} \leq 1$			$\forall_{i,i':\ 1\leq i<i'\leq r}$	(4.25)
$t_{i\,j} + \sum_{i'=1}^{i-1} y^k_{i'\,i} + \sum_{i'=i+1}^{r} y^k_{i\,i'} \leq 1$	if	$g_{k\,j}=0$	$\forall_{k:\ 1\leq k\leq n}, \forall_{i:\ 1\leq i\leq r}, \forall_{j:\ 1\leq j\leq m}$	(4.26)
$t_{i\,j} \geq \sum_{i'=1}^{i} y^k_{i'\,i} + \sum_{i'=i+1}^{r} y^k_{i\,i'}$	if	$g_{k\,j}=1$	$\forall_{k:\ 1\leq k\leq n}, \forall_{i:\ 1\leq i\leq r}, \forall_{j:\ 1\leq j\leq m}$	(4.27)
$t_{i\,j} + t_{i'\,j} \geq y^k_{i\,i'}$	if	$g_{k\,j}=2$	$\forall_{k:\ 1\leq k\leq n}, \forall_{i,i':\ 1\leq i<i'\leq r}, \forall_{j:\ 1\leq j\leq m}$	(4.28)
$t_{i\,j} + t_{i'\,j} \leq 2 - y^k_{i\,i'}$	if	$g_{k\,j}=2$	$\forall_{k:\ 1\leq k\leq n}, \forall_{i,i':\ 1\leq i<i'\leq r}, \forall_{j:\ 1\leq j\leq m}$	(4.29)

associates a variable $y^k_{i\,i'}$ with each triple $(g_k,\ h_i,\ h_{i'})$, for each $1 \leq k \leq n$ and $1 \leq i < i' \leq r$, such that $y^k_{i\,i'} = 1$ exactly when h_i and $h_{i'}$ are used to explained g_k. Moreover, similar to PolyIP, a Boolean variable $t_{i\,j}$ is associated with each haplotype site $h_{i\,j}$, for each $1 \leq i \leq r$ and $1 \leq j \leq m$, representing the value of the haplotype at that position.

For each number of haplotypes $r = UB - 1$, the formulation is as follows. The objective function maximizes the number of genotypes which can be explained using r haplotypes,

$$\max \sum_{k=1}^{n}\sum_{i=1}^{r-1}\sum_{i'=i+1}^{r} y^k_{i\,i'}. \tag{4.23}$$

Each genotype $g_k \in \mathcal{G}$ is explained by at most one pair of haplotypes,

$$\sum_{i=1}^{r-1}\sum_{i'=i+1}^{r} y^k_{i\,i'} \leq 1, \tag{4.24}$$

for $1 \leq k \leq n$. Moreover, considering that all genotypes are different, each pair of haplotypes can explain at most one genotype,

$$\sum_{k=1}^{n} y^k_{i\,i'} \leq 1, \tag{4.25}$$

for each $1 \leq i < i' \leq r$. Observe that equations (4.24) and (4.25) guarantee that the value of the objective function is the number of genotypes explained by r haplotypes. Furthermore, we need to assure that if a haplotype pair $(h_i,\ h_{i'})$ explains genotype g_k, then $g_k = h_i \oplus h_{i'}$. Therefore, if

$g_{k\,j} = 0$, then each haplotype h_i which explains g_k must have value 0 at position $h_{i\,j}$,

$$t_{i\,j} + \sum_{i'=1}^{i-1} y^k_{i'\,i} + \sum_{i'=i+1}^{r} y^k_{i\,i'} \leq 1, \tag{4.26}$$

for each $1 \leq k \leq n$, $1 \leq i \leq r$ and $1 \leq j \leq m$. Similarly, if $g_{k\,j} = 1$, then haplotype h_i which explains g_k must have value 1 at position $h_{i\,j}$.

$$t_{i\,j} \geq \sum_{i'=1}^{i} y^k_{i'\,i} + \sum_{i'=i+1}^{r} y^k_{i\,i'}, \tag{4.27}$$

for each $1 \leq k \leq n$, $1 \leq i \leq r$ and $1 \leq j \leq m$. Finally, if $g_{k\,j} = 2$, then we must ensure that the haplotypes, h_i and $h_{i'}$, which explain g_k have distinct values at position $h_{i\,j}$ and $h_{i'\,j}$. This fact is guaranteed by the following two equations:

$$t_{i\,j} + t_{i'\,j} \geq y^k_{i\,i'} \tag{4.28}$$

and

$$t_{i\,j} + t_{i'\,j} \leq 2 - y^k_{i\,i'}, \tag{4.29}$$

for each $1 \leq k \leq n$, $1 \leq i < i' \leq r$ and $1 \leq j \leq m$. The P^{UB}_{max} formulation is summarized in Table 4.4.

Concerning the size of formulation P^{UB}_{max}, the number of variables is $\mathcal{O}(n \times r^2 + rm)$ Thus, the number of variables is $\mathcal{O}(n^3 + nm)$ since $r = \mathcal{O}(n)$. The number of constraints is $\mathcal{O}(n \times r^2 \times m)$, which is $\mathcal{O}(n^3 \times m)$.

In addition, the formulation is strengthen using additional constraints, which take into consideration the graph of incompatibilities between genotypes. Moreover, the lexicographic order is applied to haplotypes in order to break symmetries.

The performance of the P^{UB}_{max} approach is significantly related with the quality of the upper bound. Clearly, a tight upper bound reduces the number of iterations of the algorithm and the number of ILP problems to solve.

4.3 Branch-and-Bound Models

4.3.1 HAPAR

HAPAR [164] is a branch-and-bound algorithm designed to solve the HIPP problem. Inspired

Algorithm 12 HAPAR algorithm

```
HAPAR(𝒢)
 1  for each g_i ∈ 𝒢
 2       do List all possible explaining pairs of haplotypes in Array(i)
 3          s_i ← length of Array(i)
 4  S ← φ                                              ▷ S is the set of resolutions
 5  Use the greedy algorithm to get a solution S* and set f*(S) to be the size of the solution
    ▷ f(S) is the number of distinct haplotypes in S
 6  Search for an optimal solution as follows:
 7  for j_1 = 1 to s_1
 8       do S ← {Array(1)[j_1]}
 9          if (f(S) > f*(S))
10             then go to 7 and try next j_1
11          for j_2 = s_1 to s_2
12               do S ← {Array(1)[j_1], Array(2)[j_2]}.
13                  if (f(S) > f*(S))
14                     then go to 11 and try next j_2;
15                  ......
16                  for j_n = 1 to s_n
17                       do S ← {Array(1)[j_1], Array(2)[j_2], ..., Array(n)[j_n]}
18                          if (f(S) < f*(S))
19                             then f*(S) = f(S);
20  return S
```

by the RTIP model [62], HAPAR also enumerates all possible pairs of haplotypes explaining each genotype $g \in \mathcal{G}$. Listing all haplotype pairs is a critical task, because the number of pairs is exponential. Hence, the method includes several improvements in order to reduce the size of the lists of haplotype pairs. One significant optimization consists in eliminating pairs of haplotypes that are guaranteed not to yield solutions better than the solutions produced by other pairs of haplotypes.

Algorithm 12 presents the pseudo-code for HAPAR, which is the branch-and-bound general procedure. The initial upper bound solution to the branch-and-bound algorithm is given by a greedy algorithm which associates each genotype with the haplotype pair with maximum coverage. The *coverage* of a haplotype h is the number of genotypes $g \in \mathcal{G}$ that h can explain and the coverage of a haplotype pair is the sum of the coverage of both haplotypes in the pair. The solution of this greedy algorithm is often close to the optimal solution. Afterward, standard branch-and-bound search is performed. Starting with the initial greedy solution, the algorithm searches for solutions

Table 4.5: The Set Covering formulation

minimize:	$\sum_{h_k \in H_G} x_k$		(4.30)
subject to:			
$\sum_{h_k \in H_j^a(g_i)} x_k \geq 1$	if $g_{i\,j} = 2$	$\forall_{g_i \in \mathcal{G}}, \forall_{j:\ 1 \leq j \leq m}, a \in \{0,\ 1\}$	(4.31)
$x(\mathcal{C}(g_i, H')) \geq 1$		$\forall_{(g_i, H') \in \mathcal{N}'}$	(4.32)

with a lower number of distinct haplotypes, cutting off and pruning the search space where a solution smaller than the current upper bound is guaranteed not to be found in that branch of the search tree.

The complexity of the algorithm is $\mathcal{O}(2^{nm})$, where n is the number of genotypes in the sample and m is the number of sites of each genotype.

4.3.2 Set Covering

A recent HIPP method is the Set Covering approach [94], which is based on an implicit representation of the solution space. The Set Covering approach proposes an integer programming model which has an exponential formulation. Nonetheless, variables and constraints are added dynamically to the model in order to obtain, at each step, a tractable approach. In addition, haplotypes are also generated at run-time.

In order to describe the Set Covering model, some definitions should be presented. Let $H(g)$ be the set of haplotypes which are compatible with genotype $g \in \mathcal{G}$ and $H_{\mathcal{G}} = \bigcup_{g \in \mathcal{G}} H(g)$. Moreover, $H_j^v(g)$ ($v \in \{0,\ 1\}$) represents the set of haplotypes compatible with g and which have value v at position j, i.e. $H_j^v(g) = \{h_k \in H(g) :\ h_{k\,j} = v\}$.

The model is based on the *covering condition* which states that a HIPP solution $\mathcal{H}$ must be such that, for each $g_i \in \mathcal{G}$, position j with $g_{i\,j} = 2$ and value $v \in \{0, 1\}$, there must exist a haplotype $h_k \in \mathcal{H} \cap H(g_i)$ with $h_{k\,j} = v$:

$$\forall_{g_i \in \mathcal{G}} \forall_{j \in \{j:\ g_{i\,j}=2,\ 1 \leq j \leq m\}} \forall_{v \in \{0,1\}} \exists_{h_k \in \mathcal{H} \cap H(g_i)}\ h_{k\,j} = v.$$

Clearly, in the final solution, there must exist haplotypes compatible with g_i with complemented values in the positions where g_i is heterozygous. The covering condition is not sufficient, but it is necessary in every haplotype inference solution. This means that there may exist sets of haplotypes

H which satisfy the covering condition but still do not explain all genotypes $g_i \in \mathcal{G}$.

The Set Covering model is as follows. For each haplotype that can explain at least one genotype in $\mathcal{G}$, i.e. for each $h_k \in H_{\mathcal{G}}$, a variable x_k is defined. Moreover, $x_k = 1$ if haplotype h_k belongs to the HIPP solution $\mathcal{H}$. These variables are similar to the variables x_k in RTIP.

Also similarly to RTIP, the goal consists in minimizing the function

$$\min \sum_{h_k \in H_{\mathcal{G}}} x_k. \tag{4.30}$$

In addition, for each genotype g_i, the covering condition is represented by the constraint

$$\sum_{h_k \in \mathcal{H}_j^a(g_i)} x_k \geq 1, \tag{4.31}$$

for every j such that $g_{i\,j} = 2$, and $a \in \{0,\ 1\}$. As the covering condition is not sufficient to obtain a feasible solution for haplotype inference, more restrictions are necessary in order to eliminate unfeasible solutions. If H' is a set of haplotypes which does not solve $\mathcal{G}$, then there are unresolved genotypes g_i in $\mathcal{G}$. Let $\mathcal{C}(g_i, H') = H(g) - H'$. There must be at least one haplotype on set $\mathcal{C}(g, H')$ on the final solution. Let N' be the set of pairs of unresolved genotypes and insufficient sets of haplotypes, i.e $N' = \{(g_i, H') :\ H'$ does not solve $\mathcal{G},\ g_i$ unresolved genotype$\}$. Therefore, for each $(g_i, H') \in N'$ it must be valid that,

$$x(\mathcal{C}(g_i, H')) \geq 1. \tag{4.32}$$

The Set Covering formulation is summarized in Table 4.5.

The Set Covering formulation has an exponential number of variables and constraints. In order to optimize its LP-relaxation, variables and constraints are generated at run-time. In particular, at run-time only a subset of N' is considered in equations (4.32).

For solving the covering condition formulation, a dedicated branch-and-bound algorithm is used, applying a pricing procedure where the haplotypes are implicitly represented. At each node of the branch-and-bound tree either new variables or new constraints are added to the problem.

Algorithm 13 Top-level SHIPs algorithm

```
SHIPs(G, lb)
1  r ← lb
2  while (true)
3        do Generate φ^r given G and r
4           if SAT(φ^r) = true
5              then return r
6              else  r ← r + 1
```

4.4 Boolean Satisfiability Models

4.4.1 SHIPs

SHIPs [114] is the first SAT-based HIPP solver and represents a remarkable improvement on the efficiency of the HIPP solvers. The SHIPs formulation iteratively models whether there exists a set of distinct haplotypes $\mathcal{H}$ with size r which explains the set of genotypes $\mathcal{G}$. The algorithm considers increasing values for $r = |\mathcal{H}|$, starting from a lower bound and stopping when achieves a feasible solution.

Algorithm 13 summarizes the top-level operations of SHIPs. Starting with a lower bound on the number of required haplotypes, r, the SHIPs algorithm models the problem for r haplotypes, into a CNF formula φ^r, and uses a SAT solver to find a solution (identified by the function call SAT(φ^r)). If the model is unsatisfiable, then the value of r is increased and a new model is generated; otherwise, the model is satisfiable and an optimal solution is guaranteed to be achieved.

Considering r candidate haplotypes, the model is as follows. Let $h_{k\,j}$ represent the j^{th} site of haplotype h_k, for $1 \leq k \leq r$ and $1 \leq j \leq m$. The model also uses the so called *selector* variables, $s^a_{k\,i}$ and $s^b_{k\,i}$ ($1 \leq k \leq r$, $1 \leq i \leq n$), which, for each genotype g_i, select two (possibly equal) haplotypes to explain g_i. Hence, g_i is explained by h_{k_1} and h_{k_2} exactly when $s^a_{k_1\,i} = 1$ and $s^b_{k_2\,i} = 1$.

If $g_{i\,j} = 0$, then the haplotypes selected to explain g_i must have value 0 at site j,

$$(\neg h_{k\,j} \vee \neg s^a_{k\,i}) \wedge (\neg h_{k\,j} \vee \neg s^b_{k\,i}). \tag{4.33}$$

Similarly, if $g_{i\,j} = 1$ and h_k is chosen by selector variables $s^a_{k\,i}$ or $s^b_{k\,i}$ to explain g_i then $h_{k\,j}$ must be

Table 4.6: The SHIPs formulation (considering r haplotypes)

$(\neg h_{k\,j} \vee \neg s^a_{k\,i}) \wedge (\neg h_{k\,j} \vee \neg s^b_{k\,i})$	if	$g_{i\,j} = 0$	$\forall_{i:\ 1\leq i\leq n},\ \forall_{j:\ 1\leq j\leq m},\ \forall_{k:\ 1\leq k\leq r}$	(4.33)
$(h_{k\,j} \vee \neg s^a_{k\,i}) \wedge (h_{k\,j} \vee \neg s^b_{k\,i})$	if	$g_{i\,j} = 1$	$\forall_{i:\ 1\leq i\leq n},\ \forall_{j:\ 1\leq j\leq m},\ \forall_{k:\ 1\leq k\leq r}$	(4.34)
$(g^a_{i\,j} \vee g^b_{i\,j}) \wedge (\neg g^a_{i\,j} \vee \neg g^b_{i\,j})$	if	$g_{i\,j} = 2$	$\forall_{i:\ 1\leq i\leq n},\ \forall_{j:\ 1\leq j\leq m},\ \forall_{k:\ 1\leq k\leq r}$	(4.35)
$(h_{k\,j} \vee \neg g^a_{i\,j} \vee \neg s^a_{k\,i}) \wedge (\neg h_{k\,j} \vee g^a_{i\,j} \vee \neg s^a_{k\,i})$ $\wedge (h_{k\,j} \vee \neg g^b_{i\,j} \vee \neg s^b_{k\,i}) \wedge (\neg h_{k\,j} \vee g^b_{i\,j} \vee \neg s^b_{k\,i})$	if	$g_{i\,j} = 2$	$\forall_{i:\ 1\leq i\leq n},\ \forall_{j:\ 1\leq j\leq m},\ \forall_{k:\ 1\leq k\leq r}$	(4.36)
$(\sum_{k=1}^{r} s^a_{k\,i} = 1) \wedge (\sum_{k=1}^{r} s^b_{k\,i} = 1)$			$\forall_{i:\ 1\leq i\leq n},\ \forall_{j:\ 1\leq j\leq m},\ \forall_{k:\ 1\leq k\leq r}$	(4.37)

assigned value 1,

$$(h_{k\,j} \vee \neg s^a_{k\,i}) \wedge (h_{k\,j} \vee \neg s^b_{k\,i}). \tag{4.34}$$

Moreover, when $g_{i\,j} = 2$, we require the chosen haplotypes to have opposing values at site j. To achieve this goal, two Boolean variables, $g^a_{i\,j}$, $g^b_{i\,j}$, are created such that $g^a_{i\,j} \neq g^b_{i\,j}$,

$$(g^a_{i\,j} \vee g^b_{i\,j}) \wedge (\neg g^a_{i\,j} \vee \neg g^b_{i\,j}). \tag{4.35}$$

As a result, for the case $g_{i\,j} = 2$, the model requires that

$$\begin{aligned}(h_{k\,j} \vee \neg g^a_{i\,j} \vee \neg s^a_{k\,i}) \wedge (\neg h_{k\,j} \vee g^a_{i\,j} \vee \neg s^a_{k\,i}) \\ \wedge (h_{k\,j} \vee \neg g^b_{i\,j} \vee \neg s^b_{k\,i}) \wedge (\neg h_{k\,j} \vee g^b_{i\,j} \vee \neg s^b_{k\,i}).\end{aligned} \tag{4.36}$$

Finally, each genotype g_i must be explained by exactly one pair of haplotypes, and therefore, there exist exactly one haplotype h_{k_1} and exactly one haplotype h_{k_2} such that $s^a_{k_1\,i} = 1$ and $s^b_{k_2\,i} = 1$, i.e. the following constraints must be valid:

$$(\sum_{k=1}^{r} s^a_{k\,i} = 1) \wedge (\sum_{k=1}^{r} s^b_{k\,i} = 1). \tag{4.37}$$

The SHIPs formulation, considering r haplotypes, is summarized in Table 4.6.

SHIPs includes several important modeling techniques for improving the effectiveness of using SAT for solving the HIPP problem, namely identification and breaking of key problem symmetries [115] and also procedures for computing lower bounds on the number of haplotypes [117], which are described in Section 4.1.2.

The complexity of the SHIPs formulation is $\mathcal{O}(rnm)$. Since $r = \mathcal{O}(n)$, each SAT model is in $\mathcal{O}(n^2 m)$.

4.4.2 SATlotyper

The majority of the haplotype inference methods can only handle biallelic SNP data of diploid species. The SATlotyper method [130] is an interesting contribution which corresponds to a generalization of the SAT-based approach SHIPs to handle polyallelic SNPs (which have more than two different alleles) and polyploid species (which have more than two homologous chromosomes), which is the case of some species of plants. Therefore, the constraints generated by SATlotyper are extensions of the constraints generated by SHIPs.

This section presents the SAT model for biallelic polyploids. Although SATlotyper is also able to handle SNPs with more than two possible values, for that case we refer to the original paper [130].

Let p be the ploidy of the considered specie, i.e. the specie has p-tuples of homologous chromosomes. Then each polyploid genotype g_i, with $1 \leq i \leq n$ is represented by a sequence of m vectors with size p, where each vector encodes one SNP site of the given individual, $g_{i\,j} = (g^1_{i\,j}, g^2_{i\,j}, \ldots, g^p_{i\,j})$, with $1 \leq j \leq m$. A site $g_{i\,j}$ is heterozygous if there are two components of vector $g_{i\,j}$, $g^{l_1}_{i\,j}$ and $g^{l_2}_{i\,j}$, such that $g^{l_1}_{i\,j} \neq g^{l_2}_{i\,j}$, with $1 \leq l_1, l_2 \leq p$. The haplotype inference problem must be reformulated.

Definition 4.6. ***Haplotype Inference - Polyploid Species***
Given a set $\mathcal{G}$ with n genotypes, each of length m, the haplotype inference problem aims at finding a set of haplotypes $\mathcal{H}$, such that, for each genotype $g_i \in \mathcal{G}$ there exists a non-ordered tuple of p haplotypes $(h^1_i, \ldots, h^p_i)$, with $h^1_i, \ldots, h^p_i$ explaining genotype g_i.

A set with p haplotypes explains a genotype g_i if the p haplotypes and the genotype g_i have the same allele composition at each SNP site.

Example 4.7. *(Haplotype Inference - Polyploid Species) An example of a tetraploid genotype with three biallelic SNP sites is $g_i = (1,0,0,1)(0,0,0,1)(1,1,1,1)$. This genotype can have 24 possible explanations, corresponding to all possible permutations of alleles at each position $g_{i\,j}$.*

The core algorithm of SATlotyper is the same as in the basic SHIPs model. The model is defined iteratively from a lower bound to an upper bound. Trivial lower and upper bounds are, respectively, 1 and $p \cdot n$. As in SHIPs, the model uses selector variables for selecting which haplotypes are used for explaining each genotype. For each genotype, the model uses p sets of selector variables, $s^l_{k\,i}$, for $1 \leq l \leq p$. Haplotypes $h_{k_1}, \ldots, h_{k_p}$ are selected to explain g_i if $s^l_{k_1\,i} = 1$, …, $s^l_{k_p\,i} = 1$.

For each genotype site $g_{i\,j}$, the selector constraints are described as follows. If $g^l_{i\,j} = 0$, for all $1 \leq l \leq p$, then every haplotype which is chosen to explain g_i must have value 0 at site j,

$$(\neg h_{k\,j} \vee \neg s^l_{k\,i}), \tag{4.38}$$

with $1 \leq k \leq r$. If $g^l_{i\,j} = 1$, for all $1 \leq l \leq p$, then every haplotype which is chosen to explain g_i must have value 1 at site j,

$$(h_{k\,j} \vee \neg s^l_{k\,i}), \tag{4.39}$$

with $1 \leq k \leq r$. Otherwise, if the genotype site $g_{i\,j}$ is heterozygous, it is necessary to create p Boolean variables $g^1_{i\,j}, \ldots, g^p_{i\,j}$, which represent the possible arrangements of $1s$ and $0s$ at site j. Hence, the following formula must be satisfied:

$$(h_{k\,j} \vee \neg g^l_{i\,j} \vee \neg s^l_{k\,i}) \wedge (\neg h_{k\,j} \vee g^l_{i\,j} \vee \neg s^l_{k\,i}), \tag{4.40}$$

where $1 \leq k \leq r$ and $1 \leq l \leq p$. Finally, for each value of i and l it is necessary that exactly one haplotype is selected. This is represented by the cardinality constraint

$$\sum_{k=1}^{r} s^l_{k\,i} = 1. \tag{4.41}$$

Furthermore, the model applies symmetry breaking to haplotypes and genotypes, similarly to SHIPs. Nonetheless, the existing version of SATlotyper does not include the computation of lower bounds, which has a crucial contribution to the efficiency of SHIPs.

The number of variables of the SATlotyper model is $\mathcal{O}(nmp^2 \log_2 p)$ and the number of constraints is $\mathcal{O}(r_f nmp)$, where r_f is the final (most parsimonious) number of haplotypes.

4.5 Answer Set Programming Models

4.5.1 HAPLO-ASP

A recent contribution to the HIPP problem is the HAPLO-ASP approach [39], based on Answer Set Programming (ASP). ASP is a declarative programming paradigm that provides a high-level language for representing combinatorial search problems, and efficient solvers to compute solutions.

The HAPLO-ASP solver, similarly to SHIPs, is an iterative algorithm. A binary search is performed in order to find the optimal value between the lower bound (lb) and the upper bound

(ub). At each iteration, an ASP formulation is solved, which decides whether there exists a solution to the haplotype inference using k distinct haplotypes, $lb \leq k \leq ub$. Clearly, if there is a haplotype inference solution using k distinct haplotypes and there exists no solution using $k-1$ haplotypes, then k corresponds to the number of haplotypes in the HIPP solution.

The ASP formulation is explained in the input language of the answer set solver CMODELS [47] and the grounder LPARSE [147]. The core search engine for CMODELS is a SAT solver. In the original paper, the authors have chosen to use the MINISAT solver [37].

4.6 Complete HIPP

In general, the HIPP problem has more than one possible solution for a single instance. In fact, the number of HIPP solutions can be significantly large [27].

The Complete HIPP [78] problem aims at finding *all* haplotype inference solutions which minimize the number of used haplotypes, i.e., all HIPP solutions.

Definition 4.8. ***Complete Haplotype Inference by Pure Parsimony***
The Complete Haplotype Inference by Pure Parsimony (CHIPP) aims at finding all *minimum-cardinality sets of haplotypes $\mathcal{H}_k$ which can explain a given set of genotypes $\mathcal{G}$.*

In theory, every algorithm for solving HIPP can be easily modified for solving the CHIPP problem. For instance, consider the RTIP formulation. RTIP can be used to find the minimum number of requested haplotypes, r, and achieve one HIPP solution, $\mathcal{H}_0 = x_{0_1}, \ \ldots, \ x_{0_r}$. Afterward, a new HIPP solution can be obtained eliminating the actual solution from the problem. Adding a new constraint to the model,

$$\sum_{k=1}^{r} x_{0_k} < r, \tag{4.42}$$

guarantees that the actual solution is not going to be found again, because the set of selected haplotypes must be different. The procedure is repeated while there are haplotype inference solutions using r haplotypes, i.e. while the optimization function has value r. This procedure allows determining all HIPP solutions.

However, note that each iteration of the procedure needs to solve a NP-hard problem. Moreover the number of solutions can be significantly large. Hence, this process can be very inefficient. In order to improve the efficiency of the procedure, some optimization techniques are suggested [78].

Backbone Haplotypes

A *backbone haplotype* is a haplotype which appears in every HIPP solution. Moreover, a genotype which can be explained using two backbone haplotypes is referred to as *backbone genotype*. Note that backbone genotypes can be omitted from the formulation. Indeed, backbone genotypes can be explained by every solution that the procedure will return. Some backbone haplotypes are easily identified. For example, homozygous genotypes or genotypes with only one heterozygous site have only one possible explanation, and therefore, induce haplotypes that must appear in all HIPP solutions, the *trivial backbone haplotypes*. Moreover, the procedure to identify all backbone haplotypes, without having to compute all HIPP solutions, is as follows. First, a HIPP solution is found, using r haplotypes. Next, iteratively remove, one at a time, haplotype h_i $(1 \leq i \leq r)$ from the solution, by including a constraint in the model which does not allow haplotype h_i to be assigned value 1. If a new parsimonious solution exists without using haplotype h_i, then h_i is not a backbone haplotype. Otherwise, h_i is a backbone haplotype. The complexity of this procedure is r times the complexity of finding a HIPP solution.

Duplicated sites

The preprocessing technique for eliminating duplicated sites, described in Section 4.1.1, cannot be directly applied for solving the CHIPP problem. Indeed, some optimal solutions may be lost by simply considering the explanation to the removed sites equal to the explanation of the original site. Therefore, when completing the removed column, every possibility which maintains the number of haplotypes and still explains the set of genotypes must be considered as optimal solution.

Decomposability

A HIPP problem instance is *decomposable* when the set of genotypes can be partitioned in disjoint sets of genotypes such that each genotype is incompatible with every genotype in the other sets. Note that two incompatible genotypes cannot share explaining haplotypes with each other. Clearly, for a decomposable problem instance, HIPP can be solved by independently finding a solution for each partition. The CHIPP problem can also be solved for each sub-problem independently. The number of HIPP solutions is the product of the number of solutions in each sub-problem, corresponding to all combinations of sub-problem solutions.

In [78], algorithms for solving CHIPP based on RTIP and on HAPAR, and implementing the

techniques described previously, are explained. The elimination of duplicated sites and the elimination of backbone genotypes are said to be the most important techniques to improve the efficiency of the resolution of the CHIPP's problem. Nonetheless, run times are not presented in the paper.

4.7 Complexity

The HIPP problem is NP-hard [73] and, furthermore, proved to be APX-hard [92]. Therefore, there is a constant $\lambda > 1$ for which does not exist a λ-approximation for the HIPP problem, unless P=NP. The proof of APX-hardness is based on a reduction from the NODE-COVER problem which is known to be APX-hard [136].

The HIPP problem is APX-hard even when each genotype is restricted to possess at most three heterozygous sites [92]. However, the case in which each genotype has at most two ambiguous positions can be solved in polynomial time[93].

In [144] the complexity and the approximability of the HIPP problem are studied. The problem is APX-hard even in very restricted cases. The HIPP problem is proved to be APX-hard for instances with at most four heterozygous sites per genotype and at most three heterozygous sites per column (SNP). On the other hand, the HIPP problem is tractable if the number of haplotypes in the solution is fixed to an integer k.

A clique instance is an instance where every two genotypes are compatible. Note that in a clique instance each column must not have both 0 and 1 values. The pure parsimony haplotyping is proved to be NP-hard even on clique instances [144]. Nonetheless there are some islands of tractability. A clique instance for which columns have at most two ambiguous sites is tractable. Moreover, in a clique instance where each column has at most k heterozygous sites per column yields an approximation ratio of (k+1)/2. Some other islands of tractability have also been identified in the case of a very particular type of instances. A polynomial algorithm is given for the case of enumerable instances [1] where the compatibility graph has bounded tree-width. Finally, HIPP is proved to be APX-hard even when the compatibility graph is bipartite [144].

4.8 Heuristic Methods

Finding a solution to HIPP is an APX-hard problem [92], and therefore a significant number of heuristic and meta-heuristic algorithms have been developed to approximate pure parsimony

[1] In an enumerable instance, a polynomial number of haplotypes is compatible with each genotype.

solutions [92, 93, 165, 72, 82, 105, 163, 46, 125, 12]. Interesting contributions use, for instance, genetic algorithms [165], iterative semi-definite programming-based algorithms [72, 82] and the tree-grow method [105]. Heuristic algorithms can be used as haplotyping methods that approximate the pure parsimony criterion but also as upper bounds to the HIPP solution.

In [92], two approximation algorithms to HIPP are presented. These algorithms are polynomial assuming that the number of heterozygous sites per genotype is limited by an integer k. The first approximation algorithm is based on the RTIP formulation. This algorithm performs a LP relaxation of the problem and then applies a heuristic to obtain an integer solution. The solution obtained is a 2^{k-1}-approximation. In the same article, the authors describe a randomized approximation algorithm which is a $2^{k+1}(1+\lfloor \log n \rfloor (1+\lceil \ln n \rceil))$-approximation, where n is the number of genotypes. For the case that the number of heterozygous sites per genotype is at most k, different approximation algorithms were proposed [93]. Another approximation method, based on RTIP, uses a simple heuristic that chooses among the candidate haplotypes the one that can resolve more genotypes [163].

Two interesting heuristics to HIPP are based on semidefinite programming [72, 82]. An iterative semi-definite programming-based algorithm [72] is shown to find an $\mathcal{O}(\log n)$ approximation to the optimal solution, where n is the number of genotypes. In [82], the heuristic method to HIPP is obtained with a vector program (semi-definite programming) relaxation of a quadratic integer programming formulation.

In addition, meta-heuristic methods have been developed to approximate the HIPP solution. A stochastic local search method is described in [46]. Exploiting the graphs representing the compatibility between genotypes, a reduction procedure is developed which, starting from a set of haplotypes, attempts to reduce its cardinality. The search space of the local search procedure is described by a complete representation of the collection of sets of the pairs of haplotypes that explain the genotypes of the problem instance. The choice of the state to move to can be done according to different local search strategies, namely best improvement, stochastic first improvement, simulated annealing and tabu search.

Another meta-heuristic approach to the HIPP problem makes use of a genetic algorithm [165] whose population space corresponds to the set of all different possible genotype explanations. Starting with a random initial population, the genetic operators - namely selection, tournament, crossover and mutation - are performed in different algorithmic iterations. At each step, the best individual of the current population is selected.

The parsimonious tree-grow (PTG) method [105] is another heuristic algorithm to HIPP. The PTG method resolves the genotype matrix columns one by one. Successive layers of the constructed

growing tree correspond to successive columns of the genotype matrix. This constructive heuristic approach keeps all genotypes (or genotype fragments) resolved during the process.

Finally, the delayed selection algorithm [125] and the CollHaps algorithm [156], described in Section 4.1.3 to compute upper bounds to the HIPP problem, can also be used as greedy algorithms to approximate the HIPP solution. These algorithms are based on the well-known Clark's method [26] but include more sophisticated methods for the selection of explaining haplotypes, in order to explicitly introduce a bias towards parsimonious solutions.

4.9 Conclusions

The goal of the haplotype inference by pure parsimony approach is to minimize the number of haplotypes used for explaining a given set of genotypes. This problem is APX-hard [92].

Several techniques can be used in a preprocessing step aiming at simplifying the problem instances and thus contributing to speed-up the performance of HIPP solvers [17, 116]. Structural simplifications include eliminating duplicated genotypes, duplicated sites and complemented sites. Applying these simplifications to the set of genotypes does not compromise the goal of obtaining a pure parsimony solution for the initial set of genotypes because it is trivial to obtain a HIPP solution for the initial set of genotypes after finding a HIPP solution for the simplified set of genotypes.

In addition, lower and upper bounds to the HIPP number of haplotypes can be computed before applying a HIPP solver. The general lower bound procedure includes three lower bound techniques. The CliqueLowerBound [114] technique produces a lower bound based on the largest clique of incompatible genotypes. The ImprovedLowerBound [114] method improves the former lower bound by identifying genotypes not included in the clique of incompatible genotypes but which require at least one more distinct haplotype. Moreover, the FurtherImprovedLowerBound technique further improves the lower bound by identifying genotypes with triples of heterozygous sites which require haplotypes not previously considered.

In general, every heuristic method for solving the HIPP problem can be used as a polynomial algorithm for computing an upper bound. The delayed haplotype selection [125] method and the CollHaps [156] method are both based on the well-known Clark's rule and represent two interesting algorithms for computing upper bounds for the HIPP problem.

A significant number of exact models have been suggested for solving the HIPP problem. The large majority of the models are integer linear programming formulations. The first model proposed, published in 2003, is named RTIP [62] and is exponential on the number the number of genotypes

and sites. PolyIP [16], published in 2004, represents the first polynomial ILP formulation modeling the HIPP problem. HybridIP [17] represents a hybrid between RTIP and PolyIP. Two recent ILP polynomial models which aim at solving the HIPP problem are HaploPPH [23], which was published in 2009, and P_{max}^{UB} [12] which is from 2008. HaploPPH is a class representative model. In addition, P_{max}^{UB} starts with an upper bound on the number of required haplotypes and solves a formulation which aims at finding a maximum number of genotypes, s, which can be explained using $UB - 1$ haplotypes. If the solution is $s < n$, where n is the total number of genotypes, then UB is the HIPP solution; otherwise, if $s = n$, then the process is iterated decreasing the value of the UB.

Dedicated branch-and-bound algorithms have also been proposed for solving the HIPP problem. HAPAR [164] performs a branch-and-bound search by enumerating the exponential number of possible pairs of haplotypes. The Set Covering [94] method proposes an ILP formulation which is also exponential but where variables and constraints are added dynamically to the model while performing a dedicated branch-and-bound search.

The idea of considering Boolean satisfiability methods for tackling the HIPP problem arose with the SHIPs [114] method, in 2006. Starting with a lower bound LB on the number of required haplotypes, the SHIPs formulation models whether there exist LB haplotypes which can explain the given set of genotypes. The value of the lower bound is iteratively increased until a feasible solution is found. In addition, the SATlotyper [130] model extends the SHIPs formulation to handle polyploid and polyallelic data. Furthermore, HAPLO-ASP [39] is an answer set programming model that solves the HIPP problem.

The number of HIPP solutions can be significantly large [27]. The complete HIPP [78] problem aims at finding *all* HIPP solutions for a given set of genotypes. In general, the process is very time consuming because a large number of NP-hard problems are needed to be solved. Nonetheless, some techniques, namely determining backbone haplotypes, duplicated sites and decomposability, can help improving the efficiency.

Although the HIPP problem is an APX-hard problem, some small islands of tractability can be found [93, 144]. In addition, a significant number of heuristic approaches have been developed to provide an approximation to the general HIPP problem [92, 93, 165, 72, 82, 105, 163, 46, 125, 12, 134]. However, given the complexity of the problem, none of these approaches can guarantee a good approximation.

5

A New Approach for HIPP

This chapter describes one of the main contributions of this thesis. The idea is to formulate a new exact HIPP model and to produce a new efficient approach that can be used as a practical alternative to the approaches described in Chapter 4. The new proposed model is referred to as the *RPoly* model.

RPoly was first published in 2007 [55]. Further improvements to the model were published in 2008 [56]. In addition, the model is explained in detail in a journal publication [57]. We would like to point out that some methods described in Chapter 4 are posterior to RPoly. Note that HAPLO-ASP [39], SATlotyper [130] and P_{max}^{UB} [12] were published in 2008 and that HaploPPH [23] and the Set Covering [94] approaches were published in 2009.

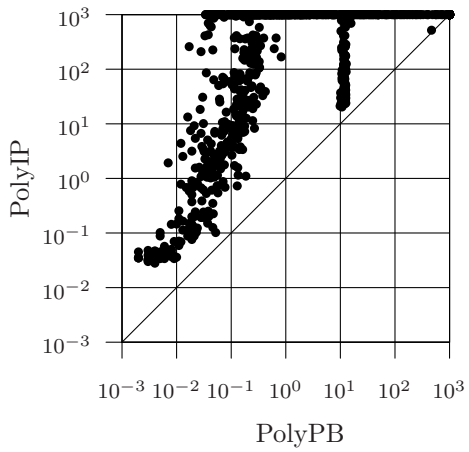
The proposed model, RPoly, is based on some prior formulations, namely PolyIP and SHIPs. The RPoly model is based on the PolyIP model but with notable improvements. A significant number of proposed modifications reduce the size of the model and are shown to be very effective. Moreover, solving the HIPP problem using SAT techniques (SHIPs) has motivated the consideration of other SAT-based procedures, such as pseudo-Boolean optimization (PBO) methods.

The RPoly model has been implemented in C code. The algorithm produces a PBO model which is handled by a PBO solver, e.g. MiniSat+ [1] [38]. The RPoly solver can be obtained from *http://sat.inesc-id.pt/∼assg/rpoly*.

The next sections describe the new PBO approach for solving the HIPP problem, RPoly, and its features: use of a PBO solver, symmetry breaking, reduction of the size of the model, integration of lower bounds and cardinality constraints. Moreover, each section evaluates each of the features included in the final version of RPoly. Each feature contributes for RPoly to be the state of the

[1] *http://minisat.se/MiniSat+.html*

Figure 5.1: Comparison of PolyIP (CPLEX) and PolyPB (MiniSat+) on the CPU time



art HIPP solver. Section 5.5 extends the RPoly model to include genotypes with missing sites. Section 5.6 studies the impact of the constraint solver in RPoly. A final section resumes the chapter.

5.1 Solving ILP HIPP Models with PBO

In the context of the integer linear programming (ILP) based HIPP models - RTIP (Section 4.2.1), PolyIP (Section 4.2.2) and HybridIP (Section 4.2.3) - all variables are Boolean and all coefficients are integers. These facts imply that the ILP-based HIPP models are also pseudo-Boolean optimization (PBO) models, and therefore PBO solvers can be considered.

The RTIP model is known to be inadequate for large problem instances due to its exponential size and the performances of PolyIP and HybridIP models are similar, as shown in Section 6.2. Hence, we only evaluate the performance of the PolyIP model using a PBO solver (MiniSat+ [38]), which we refer to as *PolyPB*.

Figure 5.1 summarizes a comparison between PolyPB and PolyIP. The set of used instances corresponds to the 1183 instances described in Table 6.1. Each point in the scatter plot represents one problem instance, where the x-axis corresponds to the CPU time required by PolyPB to solve HIPP and the y-axis corresponds to the CPU time required by PolyIP to solve the HIPP problem. A log scale is used for both axis. Instances represented by points located above the traced diagonal represent instances where PolyPB performs better than PolyIP. Points in the 10^3 lines represent instances where the solver has reached the time limit of 1000 seconds without giving the solution. Clearly, PolyPB is faster than PolyIP for all instances and, in addition, PolyPB is able to solve a

significant larger number of instances. Indeed, PolyPB solves 623 instances which PolyIP aborts. Overall, PolyIP solves only 40% of the instances whereas PolyPB solves 93% of the problem instances. We conclude that the use of MiniSat+ with the Poly model represents by far a more efficient solution than using the Poly model with CPLEX. The results obtained suggest that the PBO techniques are, in fact, very good at solving the HIPP problem.

5.2 RPoly Model

The *reduced Poly model (RPoly)* [55] is a new PBO model based on the PolyPB model but extended with further optimizations.

RPoly introduces two key modifications to the PolyPB model: the elimination of key symmetries and the reduction of the size of the model.

5.2.1 Eliminating Key Symmetries

It is a well-known fact that the SHIPs model would not be competitive if it was not for some specific optimizations, which include breaking key symmetries [114]. Eliminating symmetries prunes the search space, which in general accelerates the performance of the solvers.

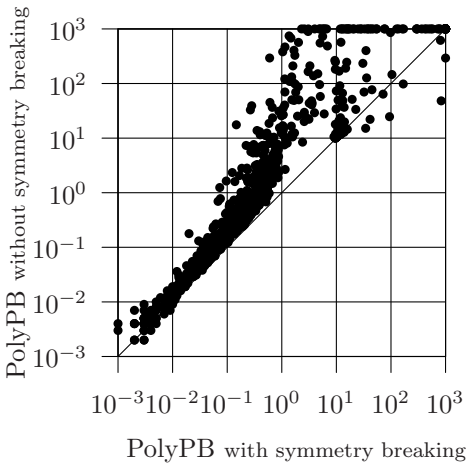
Clearly, if a genotype $g_i \in \mathcal{G}$ is explained by the haplotype pair (h_i^a, h_i^b), then g_i is also explained by the haplotype pair (h_i^b, h_i^a). Eliminating this symmetry reduces the number of solutions and consequently reduces the search space.

In practice, this symmetry is broken by adding additional constraints to the model. Considering that each haplotype is trivially mapped in a binary number, a lexicographic order can be imposed on the elements of each haplotype pair. Hence, for each site g_{ij} in a genotype $g_i \in \mathcal{G}$, we must ensure that $h_i^a < h_i^b$, i.e. if g_{ij} is the first heterozygous site of g_i then $h_{ij}^a = 0$ and $h_{ij}^b = 1$.

For example, consider the genotype $g_1 = 0212$. Eliminating symmetry in the pairs of haplotypes, the pair of haplotypes (h_1^a, h_1^b) explaining genotype g_1 must be such that $h_{12}^a = 0$ and $h_{12}^b = 1$.

Figure 5.2 compares the performance of the PolyPB model with and without the symmetry breaking constraints. Clearly, with a few exceptions (72 out of 1183 instances), eliminating symmetries accelerates the performance of the PBO solver. The new model is faster than the PolyPB model for 90% of the instances and up to 2 orders of magnitude. This result comes as no surprise, given the success of the same technique when implemented in the SHIPs model. This result is indeed significant, as the new model only aborts 32 instances, whereas the PolyPB model aborts 82 instances. Thus, the elimination of symmetries reduces in more than 60% the number of aborted

Figure 5.2: Run times for PolyPB with and without symmetry breaking



instances.

5.2.2 Reducing the Model

The PBO instances generated by the PolyPB model are significantly larger than the SAT instances generated with the SHIPs model [55]. The number of constraints in the PBO model can be up to an order of magnitude larger than the number of constraints in the SAT model, whereas the number of variables in the PBO model can be a factor of three larger than the number of variables in the SAT model.

The differences on the size of the models suggest the integration of some specific reductions used in the SHIPs model into the PolyPB model. For this reason, the resulting model is referred to as reduced Poly model (RPoly).

The organization of RPoly follows the organization of PolyIP. The RPoly model also associates two haplotypes (h_i^a, h_i^b) with each genotype $g_i \in \mathcal{G}$, and conditions are defined which capture when a different haplotype is used for explaining a given genotype. However, RPoly presents effective reductions. First, the set of variables is different. Instead of associating variables with each site of each genotype, RPoly associates variables only with heterozygous sites. Observe that homozygous sites do not require variables because the value of the haplotypes explaining homozygous sites is known beforehand and so can be implicitly assumed. Therefore, a Boolean variable $t_{i\,j}$ is associated

with each heterozygous site $g_{i\,j}$, such that,

$$t_{i\,j} = \begin{cases} 1 & \text{if } h^a_{i\,j} = 1 \wedge h^b_{i\,j} = 0 \\ 0 & \text{if } h^a_{i\,j} = 0 \wedge h^b_{i\,j} = 1 \end{cases} . \tag{5.1}$$

This alternative definition of the variables associated with the sites of genotypes reduces the number of variables by a factor of 2. In addition, the model only creates variables for heterozygous sites, and therefore the number of variables associated with sites equals the total number of heterozygous sites. As a result, the conditions described in equation (4.8) are not necessary. It should be mentioned that this definition of the variables associated with sites follows the SHIPs model [114, 115].

The RPoly model also includes variables relating pairs of candidate haplotypes. In practice, for two candidate haplotypes h^p_i and h^q_k ($p, q \in \{a, b\}$ and $1 \leq k < i \leq n$), a Boolean variable $x^{p\,q}_{i\,k}$ is defined, such that $x^{p\,q}_{i\,k}$ is 1 if haplotype h^p_i explaining genotype g_i and haplotype h^q_k explaining genotype g_k are different. Comparing with PolyIP, the key difference is that the candidate haplotypes for each genotype are related with candidate haplotypes for other genotypes only if the two genotypes are compatible. Two incompatible genotypes are guaranteed not to be explained by the same haplotype, and therefore, for the four possible combinations of p and q, $x^{p\,q}_{i\,k} = 1$.

In addition, in order to count the number of distinct haplotypes used, Boolean variables u^p_i are defined such that u^p_i is 1 if haplotype h^p_i explaining genotype g_i is different from every haplotype which explains genotype g_k with $k < i$, for each $p \in a, b$ and $2 \leq i \leq n$.

The goal of haplotype inference by pure parsimony is to minimize the number of distinct haplotypes. Hence, the cost function of the RPoly model is given by

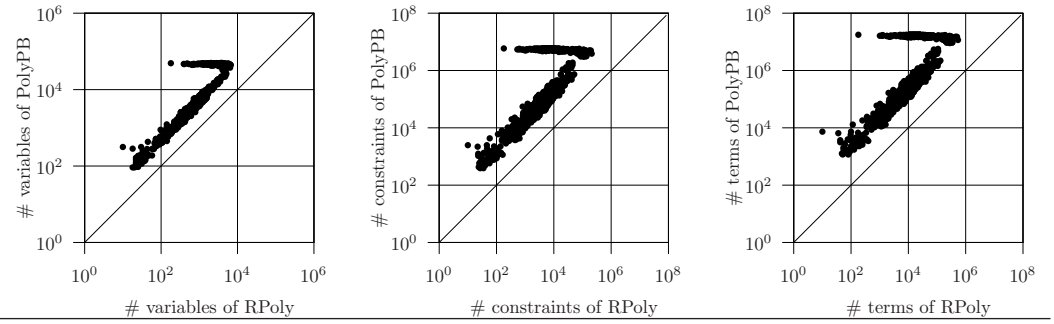
$$\min \sum_{i=1}^{n} (u^a_i + u^b_i). \tag{5.2}$$

The symmetry in haplotype pairs described in Section 5.2.1 is broken by considering that $t_{i\,j} = 0$ for each first heterozygous site $g_{i\,j}$ of each genotype g_i,

$$(g_{i\,j} = 2 \wedge \forall_{1 \leq j' < j}\ g_{i\,j'} \neq 2) \Longrightarrow \neg t_{i\,j}. \tag{5.3}$$

Two genotypes g_i and g_k are related only with respect to sites for which either g_i or g_k is

Figure 5.3: Number of variables, constraints and terms for PolyPB and RPoly models



heterozygous at that site. The conditions on the $x_{i\,k}^{p\,q}$ variables are all of the following form,

$$\neg(R \Leftrightarrow S) \Rightarrow x_{i\,k}^{p\,q}, \tag{5.4}$$

where the predicates R and S depend on the values of the sites $g_{i\,j}$ and $g_{k\,j}$, and also on the haplotype being considered, i.e. either h^a or h^b. Observe that $1 \leq k < i \leq n$, $1 \leq j \leq m$, and $p, q \in \{a, b\}$. Accordingly, the R and S predicates are defined as follows:

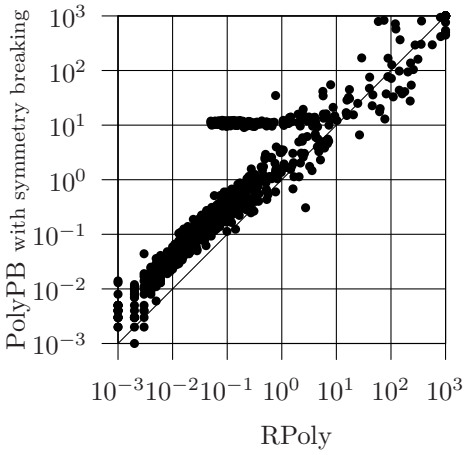
- If $g_{i\,j} \neq 2 \wedge g_{k\,j} = 2$, then $R = (g_{i\,j} \Leftrightarrow (q \Leftrightarrow a))$ and $S = t_{k\,j}$.
- If $g_{k\,j} \neq 2 \wedge g_{i\,j} = 2$, then $R = (g_{k\,j} \Leftrightarrow (p \Leftrightarrow a))$ and $S = t_{i\,j}$.
- If $g_{i\,j} = 2 \wedge g_{k\,j} = 2$, then $R = (p \Leftrightarrow q)$ and $S = (t_{i\,j} \Leftrightarrow t_{k\,j})$.

Moreover, the conditions on variables u_i^p are based on the conditions for the x_i variables. Note that if genotypes g_i and g_k are incompatible then their explaining haplotypes need not to be compared. Let predicate $\kappa(i, k)$ be defined true if and only if genotypes g_k and g_i are compatible. In addition, let $\mathcal{K}_{<i}$ be the cardinality of the set $\{g_k \in \mathcal{G} : k < i \wedge \kappa(i, k)\}$. Therefore, the RPoly model integrates the following constraints:

$$\bigwedge_{\substack{1 \leq k < i \\ \kappa(i,k)}} (x_{i\,k}^{p\,a} \wedge x_{i\,k}^{p\,b}) \Rightarrow u_i^p. \tag{5.5}$$

In practice, the proposed modifications result in significantly smaller PBO problem instances, thus originating the Reduced Poly (RPoly) model. Figure 5.3 compares the number of variables, constraints and terms for the PolyPB (extended with symmetry breaking) and the RPoly models. The results are consistent and show that the number of variables in RPoly can be up to a factor of three smaller than the number of variables in PolyPB; the number of constraints in RPoly is up to

Figure 5.4: Run times for PolyPB with symmetry breaking and RPoly



a factor of five smaller than in PolyPB; and the number of terms in RPoly is up to a factor of five smaller than in PolyPB. We should note that the *phasing* class of instances (see Table 6.1) exhibits a different behavior: most of these instances have around 10^5 variables, 10^7 constraints and 10^7 terms in the PBO model with symmetry breaking predicates. The number of variables, constraints and terms in RPoly is not reduced by a constant factor, as it is for the other classes of instances. These instances have a higher number of incompatible genotypes when compared with the other classes of instances. Hence, the impact of the reduced model is much more significant. For the same reason, the impact on the run times is also more significant (see Figure 5.4 where the run time for the *phasing* instances using the PBO model with symmetry breaking is around 10^1 seconds). As a result, for these instances RPoly can outperform PolyPB by two orders of magnitude.

In addition, we evaluate the effect of the reductions with respect to the run times. Figure 5.4 compares the PolyPB model extended with symmetry breaking constraints and the RPoly model, both using the PBO solver MiniSat+, on the set of 1183 problem instances and with a timeout of 1000 seconds. RPoly significantly outperforms PolyPB. RPoly is faster than PolyPB for 92.6% of the instances and the speedup can reach two orders of magnitude.

5.3 Optimized RPoly Model

Other techniques have been developed in order to further optimize the RPoly model [56]. The new modifications allow significant additional improvements in performance.

5.3.1 Integrating Lower Bounds

The two lower bound procedures used in SHIPs (see Section 4.1.2) can also be integrated in the RPoly model.

The FurtherImprovedLowerBound procedure provides a list of genotypes with an indication of the contribution of each genotype to the lower bound. Each genotype either contributes with +2, indicating that 2 new haplotypes will be required for explaining this genotype, or with +1, indicating that 1 new haplotype will be required for explaining this genotype. For each genotype with an associated *fixed* haplotype, the corresponding u variable is assigned value 1, and the clauses used for constraining the value of u, constraints (5.5), *need not* be generated. Remember that the u variables denote whether a haplotype used for explaining a genotype is different from the haplotypes considered so far.

It should be observed that the order of the genotypes is crucial for correctness. The RPoly model needs to be generated in such a way that the first genotypes correspond to genotypes used in the lower bound: first the genotypes used in the clique of mutually incompatible genotypes, G_C, and then the genotypes contributing to the improved lower bound. Moreover, for each genotype not contributing to the lower bound, *all fixed* compatible haplotypes must be considered as candidate haplotypes for explaining the genotype.

Similarly to the advantages of using lower bounds in SHIPs [117], the integration of lower bounds in RPoly offers a few relevant advantages. First, several u variables become fixed with value 1. This allows the PBO solver to focus on the remaining u variables. Second, the size of the generated PBO problem instances becomes significantly smaller. For the more complex problem instances, the integration of lower bound information reduces the size of the generated PBO instances by a factor between 2 and 3, on average.

In addition, we have tried the integration of an upper bound on RPoly. A constraint which explicitly imposes an upper bound on the cost function was included in the formulation. However, the obtained results suggest that this approach does not improve the performance of the HIPP solver. This fact may happen because the upper bound constraint prunes too much the state space which, in this case, seems not to represent an advantage to the PBO solver. Consequently, the integration of upper bounds is not included in the RPoly model.

5.3.2 Integrating Cardinality Constraints

For the problem of HIPP, as well as other combinatorial problems, adding new constraints to a model prunes the search and is therefore likely to contribute to the efficiency of the solver. This observation motivates an additional improvement which imposes cardinality constraints on the x variables.

Clearly, unless genotypes g_i and g_k are equal, they cannot be explained by the same pair of haplotypes. Therefore, two different genotypes must be explained by at most one common haplotype. In practice, this constraint is integrated in the model by adding cardinality constraints on the x variables. (Recall that the x variables capture whether two haplotypes are distinct.) Moreover, for incompatible pairs of genotypes, the constraints on the x variables are automatically guaranteed. Hence, for each pair of distinct non-homozygous compatible genotypes, g_i and g_k, at least three of their four pairwise haplotypes must be different,

$$\text{if } \kappa(i,k) \wedge g_i \neq g_k \wedge \exists_{1 \leq j,j' \leq m}(g_{i\,j} = 2 \wedge g_{i\,j'} = 2), \text{ then } \sum_{p,q \in \{a,b\}} x_{i\,k}^{p\,q} \geq 3.$$

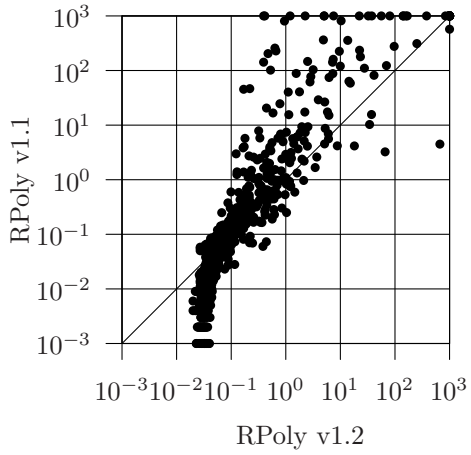
Since this cardinality constraint involves four x variables, the most effective solution is to encode the cardinality constraint using the pairwise encoding. As a result, for each pair of compatible genotypes g_i and g_k, the CNF representation of the previous cardinality constraint becomes

$$(x_{i\,k}^{a\,a} \vee x_{i\,k}^{a\,b}) \wedge (x_{i\,k}^{a\,a} \vee x_{i\,k}^{b\,a}) \wedge (x_{i\,k}^{a\,a} \vee x_{i\,k}^{b\,b}) \wedge (x_{i\,k}^{a\,b} \vee x_{i\,k}^{b\,a}) \wedge (x_{i\,k}^{a\,b} \vee x_{i\,k}^{b\,b}) \wedge (x_{i\,k}^{b\,a} \vee x_{i\,k}^{b\,b}). \tag{5.6}$$

Figure 5.5 evaluates the gains in the performance of RPoly achieved by integrating the lower bound and cardinality constraints. The optimized RPoly model is referred to as RPoly version 1.2 (v1.2), whereas the original RPoly is RPoly version 1.1 (v1.1). For very easy instances, RPoly v1.1 is clearly faster, but for difficult instances, RPoly v1.2 is consistently faster. RPoly v1.2 is able to solve 19 instances that RPoly v1.1 is not able to solve in 1000 seconds. There is only one problem instance that RPoly v1.1 is able to solve and RPoly v1.2 aborts. RPoly v1.1 is able to solve 97% of the problem instances within 1000 seconds and the improved solver, RPoly v1.2, is able to solve 98.5% of the instances, for the same timeout. Overall, the improved RPoly model reduces in 50% the number of instances aborted.

The major gain is due to the integration of lower bounds. This optimization allows solving 18 instances that were previously aborted. The addition of cardinality constraints allows solving one more instance than without these constraints.

Figure 5.5: Comparison of RPoly v1.1 and RPoly v1.2 on the CPU time



We may conclude that RPoly v1.2 is more robust and more effective on solving the hardest instances than the previous version of RPoly. On the following, RPoly refers to RPoly v1.2.

5.4 Additional Remarks

The complete RPoly model is summarized in Table 5.1. It is interesting to observe that all constraints are special cases of a constraint with the form

$$\sum_{i=0}^{n} a_i x_i \geq b, \tag{5.7}$$

where $a_i \in \{-1, 1\}$ and $b = 1 - |\{i : a_i = -1,\ 1 \leq i \leq n\}|$. This means that each RPoly constraint is equivalent to a single SAT clause, i.e. solving the RPoly model is a binate covering problem. This fact also suggests that PBO solvers based on SAT may be adequate for solving the RPoly model.

Regarding the size of the model, the following space complexity theorem may be formulated.

Theorem 5.1. *(Space Complexity) Let n be the number of genotypes in $\mathcal{G}$, m the number of positions of each genotype and θ the number of heterozygous positions in the instance. Then, the number of variables of the PBO model is $\mathcal{O}(n^2+\theta)$, which corresponds to $\mathcal{O}(n^2+nm)$. In addition, the number of constraints of the PBO model is $\mathcal{O}(n^2m)$.*

Proof. The number of t variables is $\mathcal{O}(\theta)$ because there exists one variable for each heterozygous site. The number of x variables is $\mathcal{O}(n^2)$ and the number of u variables is $\mathcal{O}(n)$. Hence, the number

Table 5.1: The RPoly formulation [a]

minimize:	$\sum_{i=1}^{n} u_i^a + u_i^b$			(5.2)
subject to:				
$t_{i\,j} = 0$	if	$g_{i\,j} = 2$ $\forall_{j'<j}\ g_{i\,j'} \neq 2$	$\forall_{i:\ 1\leq i\leq n},\ \forall_{j',j:\ 1\leq j'<j\leq m}$	(5.3)
$(-1)^{g_{i\,j}+1} t_{k\,j} + x_{i\,k}^{p\,a} \geq g_{i\,j}$ $(-1)^{g_{i\,j}} t_{k\,j} + x_{i\,k}^{p\,b} \geq 1 - g_{i\,j}$	if	$\kappa(i,k)$ $g_{i\,j} \neq 2$ $g_{k\,j} = 2$	$\forall_{k,i:\ 1\leq k<i\leq n},\ \forall_{j:\ 1\leq j\leq m},\ p \in \{a,b\}$	(5.4)
$(-1)^{g_{k\,j}+1} t_{i\,j} + x_{i\,k}^{a\,q} \geq g_{k\,j}$ $(-1)^{g_{k\,j}} t_{i\,j} + x_{i\,k}^{b\,q} \geq 1 - g_{k\,j}$	if	$\kappa(i,k)$ $g_{i\,j} = 2$ $g_{k\,j} \neq 2$	$\forall_{k,i:\ 1\leq k<i\leq n},\ \forall_{j:\ 1\leq j\leq m},\ q \in \{a,b\}$	(5.4)
$t_{i\,j} - t_{k\,j} + x_{i\,k}^{p\,q} \geq 0$ $t_{k\,j} - t_{i\,j} + x_{i\,k}^{p\,q} \geq 0$	if	$g_{i\,j} = 2$ $g_{k\,j} = 2$ $p = q,\ \kappa(i,k)$	$\forall_{k,i:\ 1\leq k<i\leq n},\ \forall_{j:\ 1\leq j\leq m},\ p,q \in \{a,b\}$	(5.4)
$t_{i\,j} + t_{k\,j} + x_{i\,k}^{p\,q} \geq 1$ $-t_{i\,j} - t_{k\,j} + x_{i\,k}^{p\,q} \geq -1$	if	$g_{i\,j} = 2$ $g_{k\,j} = 2$ $p \neq q,\ \kappa(i,k)$	$\forall_{k,i:\ 1\leq k<i\leq n},\ \forall_{j:\ 1\leq j\leq m},\ p,q \in \{a,b\}$	(5.4)
$\sum_{\substack{k<i \\ \kappa(i,k)}} \sum_{q\in\{a,b\}} x_{i\,k}^{p\,q} - u_i^p \leq 2\mathcal{K}_{<i} - 3$			$\forall_{i:\ 1<i\leq n},\ p \in \{a,b\}$	(5.5)
$\sum_{p,q\in\{a,b\}} x_{i\,k}^{p\,q} \geq 3$	if	$g_k \neq g_i,\ \kappa(i,k)$ $\neg\omega(g_k),\ \neg\omega(g_i)$	$\forall_{k,i:\ 1\leq k<i\leq n}$	(5.6)

[a] Genotypes must be sorted, starting with the genotypes used in the lower bound. The predicate $\kappa(i,k)$ is defined true iff genotypes g_k and g_i are compatible. $\mathcal{K}_{<i} = |\{g_k \in \mathcal{G} : k < i \wedge \kappa(i,k)\}|$. The predicate $\omega(g_i)$ is defined true iff genotype g_i is homozygous.

of variables is $\mathcal{O}(n^2 + \theta)$, which is equal to $\mathcal{O}(n^2 + nm)$ since $\theta = \mathcal{O}(nm)$.

Denote by $\theta_1, \ldots, \theta_n$ the number of heterozygous positions of genotypes $g_1, \ldots, g_n$, respectively. The number of constraints generated by equation (5.4) is

$$\sum_{i=2}^{n} \sum_{k=1}^{i-1} (\kappa(i,k)\mathcal{O}(\theta_i + \theta_k)), \tag{5.8}$$

which is $\mathcal{O}(n^2 m)$. The number of constraints generated by equation (5.5) is $2n$ and the number of constraints generated by equation (5.6) is $\mathcal{O}(n^2)$, taking into consideration the ranges for the i and k indexes. Hence, the number of constraints is $\mathcal{O}(n^2 m)$. □

Observe that the number of constraints and variables of the PolyIP model are, respectively, in $\Theta(n^2m)$ and $\Theta(n^2 + nm)$, hence exhibiting the same worst-case complexity as the RPoly model. Nevertheless, θ tends to be in practice much smaller than nm and $\mathcal{K}_{<i} = |\{g_k \in \mathcal{G} : k < i \wedge \kappa(i,k)\}|$ tends to be much smaller than i, and so the RPoly model can yield a significantly more compact representation than the PolyIP model.

Approximate Solutions

When RPoly exceeds the given resources without finding the optimal answer, then RPoly provides the best solution which it was able to find within its allocated resources.

5.5 Missing Data

Most often, real genotype data contains a significant percentage of unknown data. Even with modern automated DNA analysis techniques, generating data with missing alleles is not an uncommon situation [84].

One useful feature of the RPoly tool is to be able to deal with unspecified genotype sites. Genotyping procedures often leave a percentage of missing genotype positions, and so haplotype inference tools need to be able to deal with missing sites. RPoly can handle SNPs with unspecified values, inferring the values for the missing sites and still guaranteeing a parsimonious solution.

The formulation is as follows. Two Boolean variables are associated with each missing site to represent the four possible values for the haplotypes: two homozygous values (one for each allele) and two heterozygous values (one for each haplotype phase). The constraints for unspecified genotype sites are similar to the constraints for heterozygous genotype sites.

Missing SNPs are represented by '?'. Hence, the alphabet of the genotypes is extended to $\{0, 1, 2, ?\}$. In practice, two variables, $t^a_{i\,j}$ and $t^b_{i\,j}$, are associated with each missing site $g_{i\,j} =?$. Then, $t^p_{i\,j} = 0$ indicates that $h^p_{i\,j} = 0$, whereas $t^p_{i\,j} = 1$ indicates that $h^p_{i\,j} = 1$, with $p \in \{a, b\}$.

The conditions on the $x^{p\,q}_{i\,k}$ variables, which correspond to equation (5.4), are updated to

$$\neg(R \Leftrightarrow S) \Rightarrow x^{p\,q}_{i\,k}, \tag{5.9}$$

where the predicates R and S depend on the values of the sites $g_{i\,j}$ and $g_{k\,j}$, and also on which haplotype is being considered, i.e. either a or b. Observe that $1 \leq k < i \leq n$, $1 \leq j \leq m$ and $p, q \in \{a, b\}$. Accordingly, the R and S predicates are defined as follows:

- If $g_{i\,j} \neq 2 \wedge g_{k\,j} = 2$, then $R = (g_{i\,j} \Leftrightarrow (q \Leftrightarrow a))$ and $S = t_{k\,j}$.
- If $g_{k\,j} \neq 2 \wedge g_{i\,j} = 2$, then $R = (g_{k\,j} \Leftrightarrow (p \Leftrightarrow a))$ and $S = t_{i\,j}$.
- If $g_{i\,j} = 2 \wedge g_{k\,j} = 2$, then $R = (p \Leftrightarrow q)$ and $S = (t_{i\,j} \Leftrightarrow t_{k\,j})$.
- If $g_{i\,j} =? \wedge g_{k\,j} \notin \{2, ?\}$, then $R = t^p_{i\,j}$ and $S = g_{k\,j}$.
- If $g_{k\,j} =? \wedge g_{i\,j} \notin \{2, ?\}$, then $R = t^q_{k\,j}$ and $S = g_{i\,j}$.
- If $g_{i\,j} =? \wedge g_{k\,j} = 2$, then $R = (q \Leftrightarrow a)$ and $S = (t^p_{i\,j} \Leftrightarrow t_{k\,j})$.
- If $g_{k\,j} =? \wedge g_{i\,j} = 2$, then $R = (p \Leftrightarrow a)$ and $S = (t^q_{k\,j} \Leftrightarrow t_{i\,j})$.
- If $g_{i\,j} =? \wedge g_{k\,j} =?$, then $R = t^p_{i\,j}$ and $S = t^q_{k\,j}$.

5.6 Impact of the Constraint Solver

RPoly is a PBO model which uses as its engine the PBO solver MiniSat+ [38]. Nonetheless, other PBO solvers can be considered to solve the model. In addition, given that PBO is a particular case of ILP, PBO models may also be solved by generic ILP solvers. Moreover, MaxSAT solvers can also be used. Indeed, there are well-known mappings from PBO to weighted MaxSAT and vice-versa [68, 71]. The fact that all constraints in the RPoly model represent instances of the binate covering problem [29] makes the conversion very simple since every constraint is equivalent to a single clause.

Therefore, this section evaluates the performance of RPoly using a number of different solvers. This evaluation shows the sensitivity of the RPoly model to the actual constraint solver. Part of this evaluation is published in [50] but in this dissertation a few more solvers are considered.

For the experimentation, a well-known commercial ILP solver as well as the best performing PBO and MaxSAT solvers of the 2009 evaluations [2] were considered. The evaluated solvers are MiniSat+ [3] [37], Pueblo [145], BSOLO version 3.1 [119], PBS4 [4] [3], Sat4j-pb [10] [11], WBO [120], SCIP [5] [2], glpPB release 0.2 [6] and CPLEX version 11.2 [7]. Moreover, we also

[2] *http://www.maxsat.udl.cat* and *http://www.cril.univ-artois.fr/PB09*
[3] *http://minisat.se/MiniSat+.html*
[4] *http://www.aloul.net/Tools/pbs/pbs4.html*
[5] *http://scip.zib.de*
[6] *http://www.eecs.umich.edu/~hsheini/tools/glpPB.tar.gz*
[7] *http://www.ilog.com/products/cplex*

Table 5.2: Percentage of instances solved by RPoly using different solvers

PBO Solver	% solved	**MaxSAT Solver**	% solved	**ILP Solver**	% solved
MiniSat+	98.5%	WPM1	95.9%	CPLEX	78.3%
Pueblo	95.3%	MSUnCore	94.4%		
WBO	94.6%	Sat4j-maxsat	94.3%		
Sat4j-pb	94.3%	MiniMaxSat	83.9%		
BSOLO	92.3%	IncWMaxSatz	68.1%		
PBS4	91.1%				
SCIP	79.6%				
glpPB	48.3%				

considered weighted MaxSAT solvers in the experiments: WPM1 [5], Sat4j-maxsat [10] [11], MSUnCore version 3 [8] [126], MiniMaxSat [71] and IncWMaxSatz [106].

The RPoly model was adapted to be usable by the solvers described above. Table 5.2 summarizes the results. MiniSat+, which is the solver used by default with the RPoly model, aborts 18 out of 1183 instances. Follows the weighted MaxSAT solver WPM1, which aborts 49 instances, Pueblo which aborts 56 instances, WBO which aborts 64 instances and MSUnCore which aborts 66 instances. Both Sat4j-maxsat and Sat4j-pb abort 68 instances. With a poorer performance, BSOLO aborts 91 instances, PBS4 aborts 105 instances, MiniMaxSat aborts 191 instances and CPLEX aborts 257 instances. The solvers with worst performance are IncWMaxSatz which aborts 377 instances and glpPB which aborts 612 instances.

Clearly, MiniSat+ is the best performing solver. The techniques used by MiniSat+ which consist in translating the PBO model into SAT, result in an approach that is particularly suited for problems that can be naturally encoded into SAT. Indeed, this is the case for the HIPP problem. Hence, one may expect to get a more competitive solver with MiniSat+ rather than by applying other PBO solver, not optimized towards Boolean satisfiability.

The glpPB solver is the worst performing solver. This ILP-based solver implements some of the techniques also used by CPLEX, but glpPB is not as optimized as CPLEX. This can justify the poor performance of the glpPB solver.

Overall, these experiments show that the SAT-based PB and MaxSAT solvers are, in general,

[8] *http://www.csi.ucd.ie/staff/jpms/soft*

more suitable than the state of the art generic ILP solver CPLEX and the ILP-based PB solver GLPPB. The results for CPLEX and GLPPB suggest that ILP techniques are not as adequate as PB techniques for solving the HIPP problem.

Nonetheless, excluding GLPPB and INCWMAXSATZ, all the remaining solvers when applied to the RPoly model are able to solve more instances than the majority of HIPP solvers (see Section 6.2).

5.7 Conclusions

RPoly represents an innovative method for solving the HIPP problem [55, 56]. The RPoly model is based on the PolyIP [16] model but improved with a number of modifications. First, the RPoly model is a pseudo-Boolean model and uses a pseudo-Boolean optimization solver for finding a solution to the problem. Second, RPoly includes the elimination of key symmetries, reducing significantly the search space. Moreover, inspired by the SHIPs [114] model, RPoly uses some ideas which contribute to a significantly more compact model, and that is the reason for the model to be named the *Reduced Poly* model. These reductions include reducing the number of variables and constraints of the model. Moreover, RPoly integrates the computation of a lower bound which allows further reducing the model by fixing the value of a number of variables. Finally, cardinality constraints are joined to the model with the purpose of improving the run times by pruning the search space.

These improvements contribute significantly for the efficiency of RPoly. The use of a PBO Solver, MINISAT+, contributes for solving the large majority of the instances, more precisely 93%. The elimination of key symmetries, reduction of the size of the model, integration of lower bounds and cardinality constraints, contribute for an efficient HIPP solver, which is able to solve 98.5% of the problem instances.

The RPoly model has the interesting particularity that each constraint is equivalent to a single SAT clause. This fact suggests that SAT-based solvers are the most adequate for solving this model.

The number of variables of the RPoly model is $\mathcal{O}(n^2 + nm)$ and the number of constraints is $\mathcal{O}(n^2m)$. Although RPoly exhibits the same space complexity of PolyIP, the RPoly model is significantly more compact in practice.

In addition, RPoly is enriched with the possibility of having missing data. The achievement of a HIPP solution is not compromised by including new constraints which deal with missing sites. Unknown value sites are very common in real genotype data, and thus this feature is an added value to the RPoly model.

A significant number of constraint solvers, including ILP (CPLEX), PBO (MiniSat+, Pueblo, WBO, Sat4j-pb, BSOLO, PBS4, SCIP, glpPB) and MaxSAT (WPM1, MSUnCore, Sat-4j-maxsat, MiniMaxSat, IncWMaxSatz) solvers, were applied to the RPoly model. The main conclusion is that the SAT-based PB and MaxSAT solvers are the most suitable for solving the RPoly model. The best performing solver is MiniSat+, whereas the worst performing solver is the glpPB solver. Nonetheless, excluding glpPB and IncWMaxSatz, all the remaining solvers, when applied to the RPoly model can solve more instances than the large majority of HIPP solvers. This result confirms the robustness of the RPoly model.

6

HIPP: Experimental Results

This chapter illustrates the experimental results obtained using the current exact HIPP solvers. First, in Section 6.1, the experimental setup, including the origin of the considerably large number of instances is described. Section 6.2 compares the efficiency of all solvers which exactly respond to the pure parsimony criterion. Furthermore, in Section 6.3, the effectiveness of the lower and upper bounds described previously in this thesis, is tested. Section 6.4 evaluates the accuracy of the pure parsimony criterion, by comparing the accuracy of RPoly with the accuracy of the mostly used statistical haplotype inference tools. A final section resumes the conclusions acquired from the results presented in this chapter.

6.1 Experimental Setup

6.1.1 Datasets

A significant number of instances is used for testing the performance of the HIPP algorithms. A set with 1183 problem instances [55] is considered. The problem instances can be divided in two sets: the synthetic data and the real data. Table 6.1 contains the considered classes of instances and summarizes their sizes. For each class, we give the number of instances, and the minimum and maximum number of SNPs and genotypes. All instances were simplified using the techniques described in Section 4.1.1 and, therefore, the presented sizes correspond to the simplified instances.

Part of the synthetic data was generated using the Hudson's program *ms* [1] [75]. This software is able to simulate haplotype samples following the coalescent approach and under a variety of assumptions about migration, recombination rate and population size. Randomly, the haplotypes have been paired uniformly (equal haplotypes are removed, so that every haplotype is sampled

[1] *http://home.uchicago.edu/∼rhudson1/source/mksamples.html*

Table 6.1: Classes of instances (I): number of SNPs and genotypes

Class	# Instances	*min*SNPs	*max*SNPs	*min*GENs	*max*GENs
ms	380	4	57	9	94
phasing	329	14	188	34	90
hapmap	24	4	29	5	68
biological	450	4	77	4	49
Total	1183	4	188	4	94

with the same frequency) and non-uniformly (repeated haplotypes have higher probability of being sampled than others) [17].

Additional synthetic data was generated by the simulation software *cosi* [2] [142]. These instances (class *phasing*) were used recently to evaluate phasing algorithms [121]. The phasing instances correspond to the SU-100kb, SU1, SU2 and SU3 classes available online [3].

Part of the real data (class *hapmap*) was obtained from the HapMap project [4] [153, 154], which provides a comprehensive source of genotype data over four populations. These instances were provided by D. Brown and I. Harrower [17]. The data was generated with DNA sequence from regions with a small amount of recombination, of chromosomes 10 and 21. For all four populations, regions with 30, 50 and 75 SNPs were chosen, therefore providing a total of 24 hapmap instances.

The class of *biological* benchmarks was obtained with the haplotypes from five well-studied genes, available from scientific publications [85, 140, 89, 31, 36]. For each of the five sets of real haplotypes, sets of genotypes with different sizes were generated. Genotypes have been generated by randomly pairing two haplotypes.

6.1.2 Setting-up

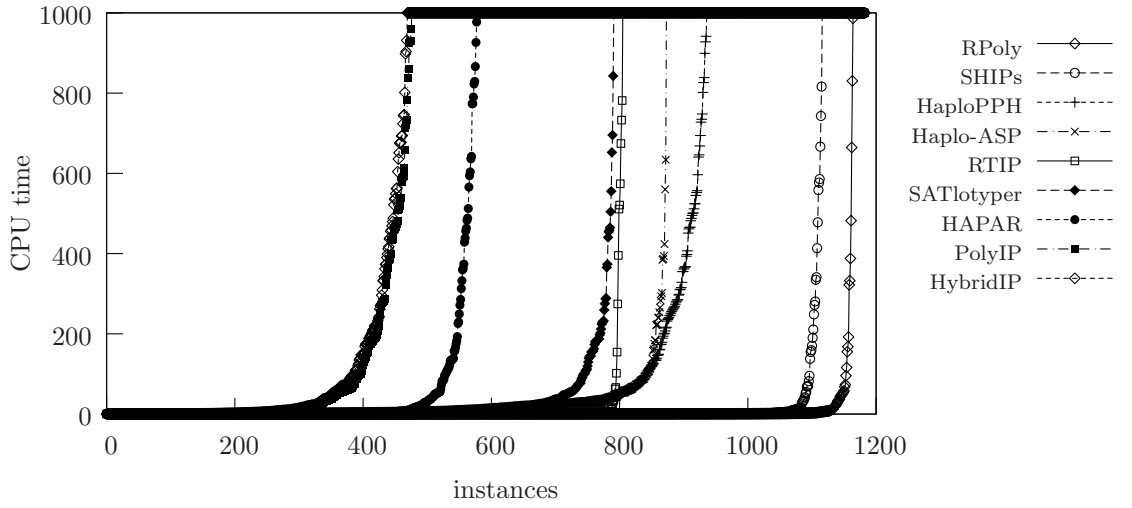
Except when contrarily stated, all experiments were run on an Intel Xeon 5160 server (3.0GHz, 1333Mhz, 4GB) running Red Hat Enterprise Linux WS 4. The timeout for each instance was set to 1000 seconds and the memory limit was set to 2 GB.

[2] *http://www.broad.mit.edu/∼sfs/cosi*

[3] *http://www.stats.ox.ac.uk/∼marchini/phaseoff.html*

[4] *http://www.hapmap.org*

Figure 6.1: Relative performance of HIPP solvers



6.2 Efficiency of HIPP Solvers

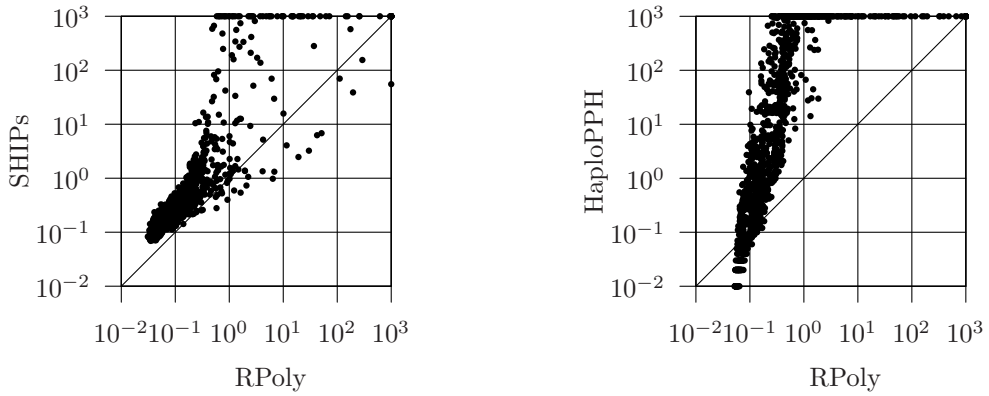
In this section, an extensive comparison of the efficiency of RPoly with the other exact HIPP solvers is performed. A significant number of HIPP solvers were tested, including RTIP [62], PolyIP [16], HybridIP [17], HAPAR [164], SHIPs [116], SATlotyper [130], HAPLO-ASP [39] and HaploPPH [23]. The results were obtained with the tools provided by the authors, except for the RTIP tool. This tool was provided by the authors of PolyIP and HybridIP. To the best of our knowledge, the author of RTIP has not made the software available. For the ILP approaches, CPLEX version 11.2 was used. The most recent version of RPoly (version 1.2) and SHIPs (version 2) were used. SATlotyper version 0.1.1b was used. Both SAT-based HIPP solvers use MiniSat 2 [5]. HAPLO-ASP uses Cmodels (version 3.75) [6] and LParse version (1.0.17) [7], HaploPPH uses Xpress-Mosel (version 3.0.2), from FICO Xpress [8].

6.2.1 Comparing Performances

Figure 6.1 presents the results for the HIPP solvers running in all instances. For each solver, instances are sorted and plotted according to their running times, thus giving a figure about the

[5] *http://minisat.se/MiniSat.html*
[6] *http://www.cs.utexas.edu/users/tag/cmodels.html*
[7] *http://www.tcs.hut.fi/Software/smodels*
[8] *http://www.fico.com*

Figure 6.2: CPU time comparison between RPoly and SHIPs/HaploPPH



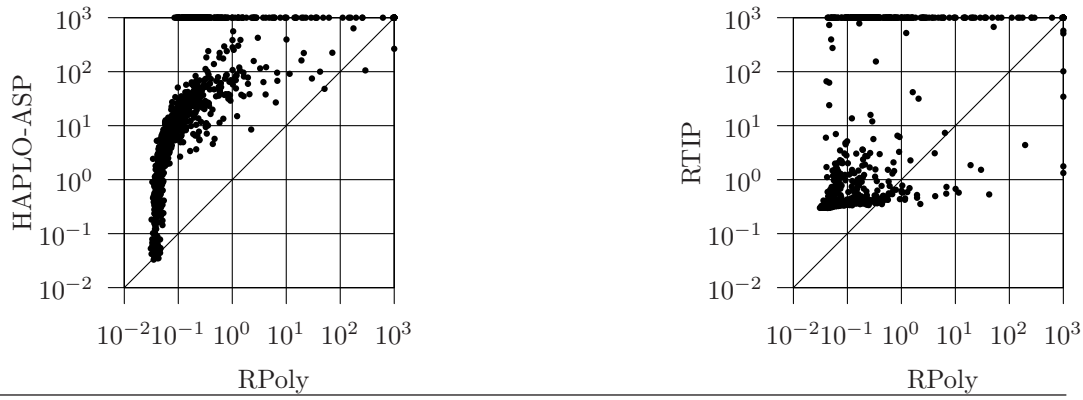
number of instances each solver is able to solve in less than 1000 seconds.

As the figure shows, RPoly is the HIPP tool capable of solving the largest number of instances. RPoly solves 1165 out of 1183 instances within 1000 seconds, which represents more than 98% of the instances. SHIPs is the second more robust HIPP solver, being able to solve more than 94% of the problem instances. Follows HaploPPH which solves 79%, HAPLO-ASP which is able to solve 74%, RTIP which solves about 68% and SATlotyper which solves about 67% of the problem instances. HAPAR solves about 49% of the instances. PolyIP and HybridIP have very similar performances, solving both about 40% of the instances.

Taking into account the amount of time that each solver requires, on average, to solve the non-aborted instances, we can conclude that RPoly is also the fastest solver. RPoly takes on average 3.5 seconds for solving the non-aborted instances followed by RTIP that takes on average 6.8 seconds and by SHIPs that takes 7.9 seconds. SATlotyper takes 24.0 seconds, HAPLO-ASP takes 30.2 seconds, HAPAR takes 39.9 seconds, HaploPPH takes 42.3 seconds, HybridIP takes 72.9 seconds and PolyIP takes 73.4 seconds, on average.

Although RTIP is able to solve a large set of instances in a surprising small amount of time, RTIP is not a robust solver. Due to the exponential size of the formulation, RTIP either solves the problem instance in a few seconds or causes memory exhaustion. Indeed, 96% of the problem instances aborted by RTIP are due to memory exhaustion and on average an instance is aborted due to memory exhaustion in 37.5 seconds. Also HAPLO-ASP has a problem of memory exhaustion with large instances, and consequently aborts most of the instances of the phasing class. Indeed, 96.8% of the instances aborted by HAPLO-ASP abort due to memory exhaustion.

Figure 6.3: CPU time comparison between RPoly and Haplo-ASP/RTIP



Figures 6.2 and 6.3 compare RPoly with the other best performing solvers: SHIPs, HaploPPH, HAPLO-ASP and RTIP, with respect to the CPU time. Points on the 10^3 lines represent instances where either the solvers have reached the time limit of 1000 seconds without giving the solution, or the solvers have aborted due to memory exhaustion. The case of memory exhaustion only happens for RTIP and HAPLO-ASP.

The scatter plot on the left of Figure 6.2 compares RPoly and SHIPs on the CPU time. As can be observed, RPoly has in general better performance than each of the other solvers. RPoly is faster than SHIPs for 96% of instances. Furthermore, RPoly is able to solve 50 instances that SHIPs aborts. Nonetheless, SHIPs is able to solve one instance within 1000 seconds which RPoly does not.

Figure 6.2 (right) compares RPoly with HaploPPH. RPoly is able to solve 229 instances that HaploPPH aborts and RPoly can solve all instances that HaploPPH is able to solve. Moreover, RPoly is faster than HaploPPH for 80% of the problem instances.

Figure 6.3 (left) compares RPoly with HAPLO-ASP. RPoly is faster than HAPLO-ASP for 98% of instances. Moreover, RPoly is able to solve 293 instances that HAPLO-ASP aborts. There is one instance which HAPLO-ASP is able to solve within 1000 seconds and RPoly does not.

Finally, the scatter plot on the right of Figure 6.3 compares RPoly and RTIP on the CPU time. Overall, RPoly is faster than RTIP for 96% of the problem instances. Moreover, RPoly is able to solve 366 instances which RTIP aborts. However, RTIP is able to solve 6 out of the 18 instances aborted by RPoly. This fact is interesting, albeit not significant.

Overall, RPoly is the most robust solver being able to solve the largest number of instances in a

reduced amount of time.

6.2.2 Additional Remarks

The previous sections show that RPoly is clearly the most robust solver. For the large majority of instances, RPoly is faster than any other exact HIPP solver. Furthermore, RPoly is the state of the art HIPP solver which can solve the largest set of instances, aborting only 18 instances out of 1183 of different sources, reducing in 73% the number of instances aborted by SHIPs.

The fact that the best performing solvers are RPoly, SHIPs and, for some classes of instances, HAPLO-ASP and SATlotyper, suggests that the SAT-based techniques are the most adequate for solving the HIPP problem. It is interesting to note that each of these four solvers uses the SAT solver MiniSat [37] as its core engine.

SATlotyper is probably less optimized than SHIPs due to the fact that SATlotyper was not created to compete with exact HIPP solvers. SATlotyper is more general, being able to solve haplotype inference problems on polyploid species and polyallelic SNPs. In particular, SATlotyper does not include the computation of the lower bound, which is a crucial point to the good performance of SHIPs.

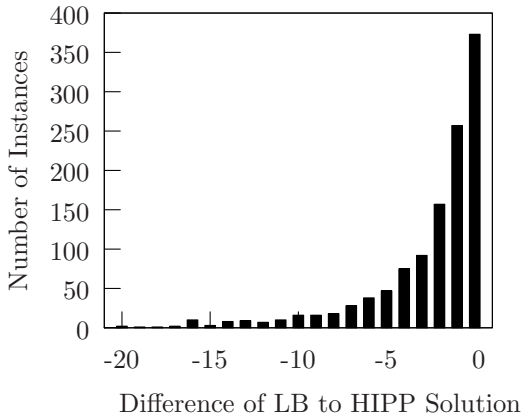
The authors of the Set Covering [94] approach, described in Section 4.3.2, did not turn available their haplotype inference software and therefore, we could not present the experimental results. Nonetheless, according to [94], the Set Covering approach is not able to solve some *ms* instances with 30 genotypes and 50 SNPs. Note that *ms* instances of these dimensions are trivially solved by RPoly. These results suggest that the Set Covering approach would not perform so well in the more challenging instances, as those from the *phasing* class.

Moreover, we also do not have experimental results for the P_{max}^{UB} [12] approach, described in Section 4.2.5. Nonetheless, the results presented in [12] indicate that P_{max}^{UB} needs, on average, 875 seconds to solve instances generated with the Hudson's program, with 50 genotypes, 10 SNPs and recombination factor $r = 16$. Our *ms* dataset contains instances with these parameters and RPoly is able to solve each in less than 2 seconds. This fact suggests that the P_{max}^{UB} approach is also not competitive with the RPoly method.

6.3 Testing Bounds

This section illustrates the effectiveness of the bounding techniques described in Section 4.1.2 and Section 4.1.3. For this study, only the instances that have been solved by at least one of the

Figure 6.4: Difference between the lower bound and the optimal solution



exact HIPP approaches have been taken into account. (This procedure has eliminated 13 out of the 1183 instances.) Otherwise it would not be possible to compare the values of the computed bounds with the optimal solution.

6.3.1 Lower Bounds

Figure 6.4 provides a comparison between the lower bound given by Algorithm 7, described in Section 4.1.2, and the exact HIPP solution, for the 1170 problem instances whose HIPP solution is known. For around 32% of the instances, the lower bound computes the exact HIPP solution. Moreover, for the large majority of the instances (more precisely 86%) the difference between the HIPP solution and the lower bound is less than or equal to 5.

Analyzing the contribution of each algorithm of Section 4.1.2 to the lower bound, the following results can be obtained. Algorithm 8, CliqueLowerBound, computes the exact HIPP solution for only 1% of the instances. Moreover, the difference between the lower bound and the HIPP solution is less than or equal to 5 for 30% of the instances. Algorithm 9, ImprovedLowerBound, notably improves the previous lower bound. The computed lower bound is equal to the exact HIPP solution for 30% of the considered instances. In addition, for 83% of the instances, the difference between the exact solution and the ImprovedLowerBound solution is less than or equal to 5. Algorithm 10, FurtherImprovedLowerBound, further increases the lower bound and calculates the exact HIPP solution for 32% of the instances and 86% of the problem instances differ from the HIPP solution in less than 6 haplotypes.

Figure 6.5: Difference between the Delayed Selection upper bound and the optimal solution

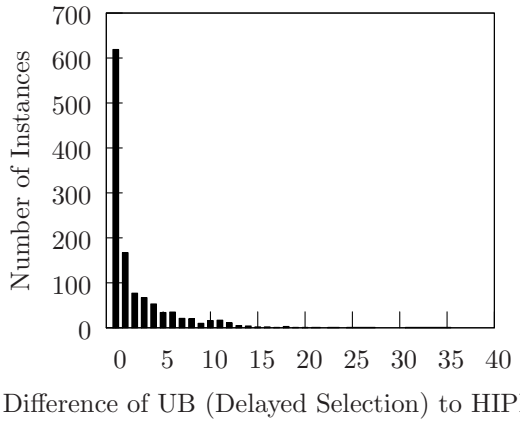
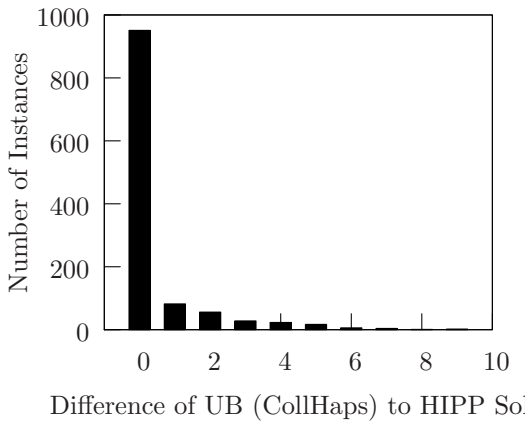


Figure 6.6: Difference between the CollHaps upper bound and the optimal solution

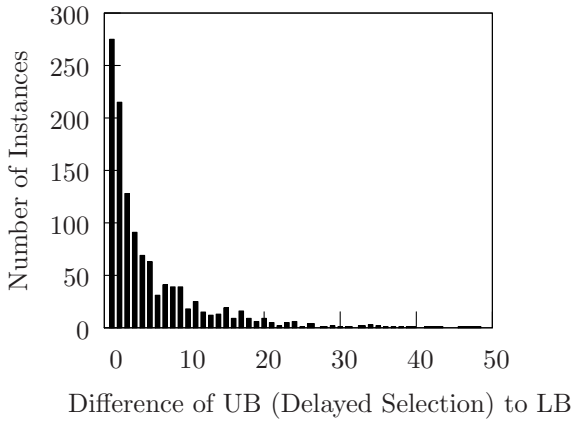


6.3.2 Upper Bounds

The evaluation of the delayed selection upper bound computation is summarized in Figure 6.5. For 53% of the instances, the upper bound algorithm computes the exact HIPP solution. In addition, for 87% of the instances the difference between the computed upper bound and the HIPP solution is less or equal to 5. The distance between the upper bound and the HIPP solution goes up to 33 haplotypes.

Figure 6.6 presents the quality of the CollHaps upper bound. The upper bound value provided by the CollHaps algorithm is equal to the optimum value for 951 instances, i.e. for more than 81% of the problem instances. Moreover, for 99% of the instances the difference between the value provided

Figure 6.7: Difference between the Delayed Selection upper bound and the lower bound



by CollHaps and the HIPP solution is less or equal to 5. The distance between the upper bound and the HIPP solution goes up to 9 haplotypes.

6.3.3 Upper Bounds *vs* Lower Bounds

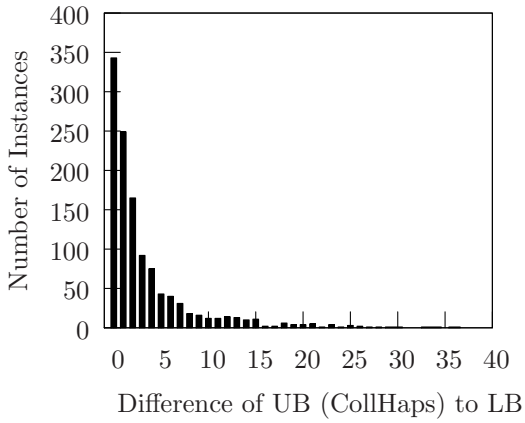
Figure 6.7 compares the lower bound and the delayed selection upper bound values obtained for each instance. For this plot the whole set of 1183 instances was evaluated. We may observe that for more than 23% of the instances both values are exactly the same. This means that computing lower and upper bounds suffices to solve these problem instances, i.e. no search is required. In addition, the difference between the upper bound and the lower bound is less or equal to 5 for 71% of the instances and less or equal to 10 for 85% of the instances, thus predictably not requiring much time to be solved.

Finally, Figure 6.8 compares the lower bound and the CollHaps upper bound values. For 343 instances, i.e. 29% of the problem instances, the value of the upper bound is equal to the value of the lower bound. For 82% of the problem instances, the difference between the upper and lower bounds is less or equal to 5. The difference between the CollHaps upper bound and the lower bound is at most 36 haplotypes.

6.4 Accuracy of Haplotype Inference Methods

In order to have practical impact in genetic studies, a haplotype inference method must be accurate with respect to the reality. Accuracy is measured by the correct association between

Figure 6.8: Difference between the CollHaps upper bound and the lower bound



genotypes and explaining haplotypes. Even though it is not possible, in general, to know the precise solution for the haplotype inference problem, there are a few very well-studied sets of genotypes for which the solution is known. This solution is often obtained using different generations from the same population.

This section aims at comparing the accuracy of the pure parsimony criterion with the accuracy of different haplotype inference approaches. Hence, the accuracy of RPoly is compared with the accuracy of the most well-known statistical approaches and the approach based on perfect phylogeny. Although there are some studies which compare the accuracy of the phasing methods [121, 4, 150], most studies exclude combinatorial methods, as pure parsimony, and recent statistical approaches, as Shape-IT [34] and Beagle [19]. Hence, this section provides also a new accuracy comparison between phasing methods.

Datasets

Two distinct sets of real data are used in this experiment. The first set is constituted of seven instances whose haplotypes have been experimentally determined [133, 4] and correspond to the $A - G$ data sets already used in other haplotyping studies [27]. The set A contains 39 genotypes, whereas all other sets $B - G$ have 80 genotypes. The number of SNPs range from 5 to 47.

The second set of instances corresponds to data obtained from the HapMap site [9]. We have selected the SNPs from a set of nine important genes related with breast cancer, namely BRCA1,

[9] *http://www.hapmap.org*

Table 6.2: Classes of instances (II): number of SNPs and genotypes

Class	# Instances	*min*SNPs	*max*SNPs	*min*GENs	*max*GENs
known datasets: A-G	7	5	47	39	80
HapMap CEU	9	16	320	30	30
HapMap YRI	9	17	308	30	30
Total	25	5	320	30	80

BRCA2, P53, ATM, CYP1B1, HRAS1, RAD51, SOD2 and TSG101. The populations CEU and YRI were chosen because they are subdivided in family trios (mother, father, child). Using the genotypes of the parents, it is possible to infer the haplotypes of the children for the majority of the locus. The number of genotypes is 30 in all instances from this set and the number of SNPs range from 17 to 308.

Table 6.2 summarizes the sizes of the considered classes of instances. Note that haplotyping regions with tens of SNPs are still relevant in several association studies. Moreover, larger regions can always be partitioned into small blocks [172].

Performance Metric

The accuracy of the methods is measured by the well-known *switch error rate*, which measures the percentage of possible switches in haplotype orientation, used to recover the correct phase in an individual [107]. Moreover, we introduce here the *switch accuracy* which is defined by the percentage of successive pairs of heterozygous positions in an individual that are phased correctly with respect to each other, i.e.

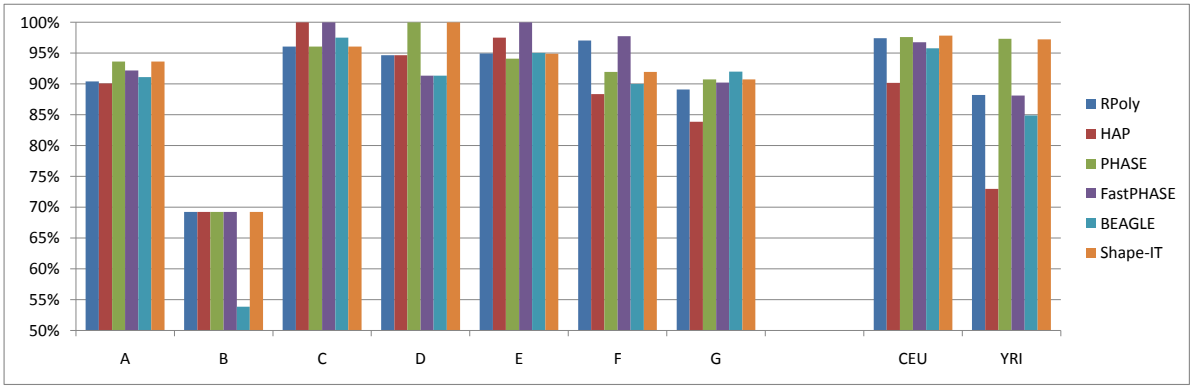
$$\text{(switch accuracy)} = 100\% - \text{(switch error rate)}.$$

Phasing Algorithms

The accuracy of distinct haplotype inference methods are compared here. The following statistical tools are used: PHASE [10] [150] (version 2.0), FastPHASE [11] [143] (version 1.2.3), Beagle [12] [19] (version 3.1) and Shape-IT [13] [34] (version 1.0). PHASE and Shape-IT are Bayesian methods,

[10] *http://depts.washington.edu/uwc4c/express-licenses/assets/phase*
[11] *http://depts.washington.edu/uwc4c/express-licenses/assets/fastphase*
[12] *http://www.stat.auckland.ac.nz/∼bbrowning/beagle/beagle.html*
[13] *http://www.griv.org/shapeit*

Figure 6.9: Switch accuracy of phasing methods in different datasets



and Beagle and FastPHASE use hidden Markov models. Furthermore, the combinatorial method which follows the imperfect phylogeny approach, HAP [14] [66] (version 3.0) is also tested.

6.4.1 Comparing Accuracies

Experimental results are summarized in Figure 6.9 and Table 6.3. Figure 6.9 shows the switch accuracy for the *known datasets* and for *HapMap* instances. The accuracy in the *HapMap* instances is the average of the 9 gene instances for populations CEU and YRI, respectively.

It is not clear which method exhibits better accuracy. Actually, the performance of the solvers may vary significantly with different instances. PHASE and Shape-IT have better accuracy in instances A and D. On the other hand, HAP and FastPHASE have better accuracy in instances C and E and RPoly and FastPHASE have better accuracy in instance F. All methods except HAP have similar accuracy in instance G and in the set of instances from CEU population. Moreover, PHASE, FastPHASE and Shape-IT have better accuracy for instances of YRI population.

Instance B exhibits the smallest switch accuracy for all methods. Instance B is the one with the least number of heterozygous sites. Hence, a small number of errors produce a large percentage in the error rate. All methods have an accuracy rate of 69%, except HAP which have an accuracy of 54%.

Table 6.3 presents the general average switch error and standard deviation in all problem instances. PHASE and Shape-IT present the smallest error rates, more precisely, PHASE presents an average switch error rate of 4.40% and Shape-IT presents a switch error rate of 4.31%. Follows FastPHASE and RPoly which present average switch error rate inferior to 8%. Beagle presents an

[14] *http://research.calit2.net/hap*

Table 6.3: Switch error rate: mean and standard deviation (25 instances)

	RPoly	HAP	PHASE	FastPHASE	Beagle	Shape-IT
Mean switch error	7.91%	16.31%	4.40%	7.81%	10.52%	4.31%
Std deviation	0.085	0.173	0.066	0.162	0.167	0.065

average error rate of 10.52%. Finally, HAP presents the highest average error rate of 16.31%.

RPoly is comparable with the statistical methods in the CEU population, but has less accuracy in the YRI population. This can be explained by the fact that the CEU population is more recent and has less diversity in haplotypes, which follows better the pure parsimony criterion.

HAP presents higher error rates due to the partition in blocks. Actually, the algorithm divides the instances in small blocks of SNPs and solves the perfect phylogeny approach within each block. However, HAP does not use a ligation method to join the solutions obtained within blocks. Therefore, HAP tend to have better accuracy in instances with small number of SNPs.

In addition, we can conclude that PHASE and Shape-IT have very similar accuracy in all datasets. RPoly has a general accuracy which can be compared with the accuracy of Beagle and FastPHASE.

Although these statistical approaches can be considered more accurate than combinatorial approaches, the later can be considered as an alternative, specially in small regions or in less diversified populations.

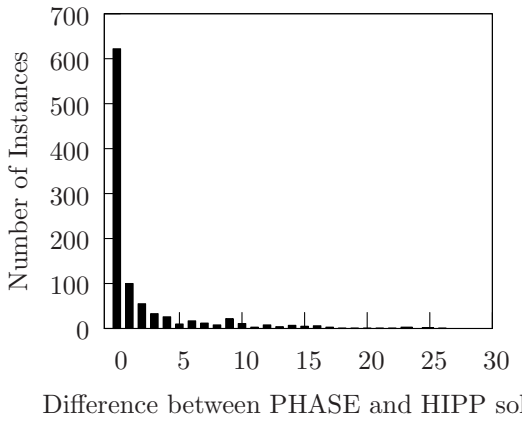
6.4.2 Discussion

This section aims at analyzing the true solutions with respect to the number of haplotypes, in order to measure the distance between the pure parsimony solution and the true solution.

PHASE is a phasing tool which is known to be an accurate method for haplotype inference, although often inefficient [121]. Therefore, firstly in this section, we would like to compare the haplotypes in the solution provided by PHASE with the solution provided by the pure parsimony approach. We used a timeout of 10,000 seconds to run the PHASE algorithm in the 1183 problem instances described in Section 6.1.1. PHASE was able to solve 976 out of 1183 instances within 10,000 seconds.

Figure 6.10 provides a comparison between the PHASE solution and the HIPP solution, regarding the number of haplotypes used in the solution. We used the set of 963 problem instances for which

Figure 6.10: Difference between the number of haplotypes in PHASE and in HIPP solutions



the HIPP solution is known and for which PHASE is able to provide a solution within 10,000 seconds. For approximately 65% of the instances, the PHASE solution and the HIPP solution require exactly the same number of haplotypes. Moreover, for the large majority of the instances (more precisely 88%) the difference between the number of haplotypes in the PHASE solution and in the HIPP solution is less than or equal to 5. In addition, for 34% of the problem instances, the set of haplotypes in the solution provided by the HIPP solver RPoly is exactly the same set of haplotypes provided by PHASE. (This result should be similar using a different HIPP solver because HIPP solvers have no other criterion than parsimony.) Furthermore, on average, 70% of the haplotypes are the same in both the RPoly and PHASE solutions.

In addition, Table 6.4 presents the number of haplotypes in the solution obtained by each phasing method, and the number of haplotypes in the true solution, for the *A*-*G* datasets. For 4 out of the 7 instances, the true solution has the most parsimonious number of haplotypes. Nonetheless, this fact does not guarantee RPoly to be accurate solver in these instances. However, it is very interesting to note that, for each instance except G, the most accurate solver provides a solution which is parsimonious in the number of haplotypes.

These results emphasizes that solutions which tend to be accurate are typically parsimonious or close to parsimonious. However, in general, and for a single instance, the number of solutions satisfying the pure parsimony criterion can be large. The reason for this is that although the HIPP criterion imposes a constraint on the number of haplotypes in the solution, the same set of haplotypes can be used in different ways to explain the genotypes. In addition, there can be solutions with

Table 6.4: Number of haplotypes in the solution of each approach and the real value

	RPoly	HAP	PHASE	FastPHASE	Beagle	Shape-IT	**Real Value**
A	15	20	15	19	22	15	17
B	7	7	7	7	8	7	7
C	12	12	12	12	14	12	12
D	7	7	7	8	8	7	7
E	16	20	17	16	18	17	16
F	17	25	20	17	21	19	18
G	28	40	28	29	31	28	32

different sets of haplotypes that still have minimum size. Future research directions should consider using a criterion to choose the most accurate solution between all possible HIPP solutions.

6.5 Conclusions

The results obtained in a set of 1183 instances from different sources suggest the following conclusions. An extensive comparison of nine HIPP solvers (RPoly [55], RTIP [62], PolyIP [16], HybridIP [17], HAPAR [164], SHIPs [116], SATlotyper [130], HAPLO-ASP [39] and HaploPPH [23]) has confirmed that RPoly is the most efficient solver. Indeed, RPoly is the HIPP tool capable of solving the largest number of instances, more precisely, 98.5% of the total number. Moreover, in general, RPoly is also the fastest solver. The fact that the best performing solvers are RPoly, SHIPs and for some classes of instances, HAPLO-ASP and SATlotyper suggests that the SAT-based techniques are the most adequate for solving the HIPP problem.

The study of the effectiveness of the bounding techniques yields the following conclusions. The lower bound [114] algorithm calculates the exact solution for 32% of the instances. The delayed haplotype selection [125] upper bound algorithm computes the exact HIPP solution for 53% of the instances. The CollHaps [156] algorithm is more robust and provides a better upper bound, being able to coincide with the exact HIPP value for 81% of the instances. Therefore, it is important to evaluate the distribution of the difference between upper and lower bounds. The tighter lower and upper bounds overlap for 29% of the instances.

Finally, the accuracy of RPoly is compared with the accuracy of well-known methods as HAP [66], PHASE [150], FastPHASE [143], Beagle [19] and Shape-IT [34], in a set of real instances. Exper-

imental results suggest that the accuracy of RPoly can be considered as an alternative to other methods. RPoly presents an accuracy which is comparable with the best statistical methods in some classes of instances, in particular, in more recent populations which have less diversity in haplotypes. In particular, the accuracy of RPoly is comparable with the ones of Beagle and FastPHASE. Nonetheless, future work directions include studying new criteria that can be included in the pure parsimony solvers, with the goal of improving the accuracy of the method.

7

A New Approach for Haplotype Inference in Pedigrees

Many haplotype inference methods apply either to unrelated individuals [149, 19, 55] from the same population or to pedigrees [42, 100, 173, 88]. Nonetheless, in real studies and most of the times, the input data contains both closely related and unrelated individuals. Recently, a study comparing the haplotype inference methods using pedigrees and unrelated individuals [101] concluded that taking into consideration both pedigree and population information leads to improvements in the precision of haplotype inference methods. The combination of population and pedigree genetic models has the potential to increase the precision of the inference methods, leading to an increased power of the statistical tests [20, 25] and a reduction in the number of wrong results caused by the existence of a sub-population structure that is ignored by the statistical tests.

This chapter describes a key contribution of this thesis that consists in a new approach for haplotype inference, which applies to sets of pedigrees from the same population [51, 52]. The new approach is a combination of two well-known haplotype inference approaches: pure parsimony (Section 3.2.1), which is used to phase unrelated individuals; and minimum recombinant (Section 3.3.1), which is used to phase individuals organized in pedigrees. The new approach is called minimum recombinant maximum parsimony (MRMP), and the suggested model for solving the MRMP problem is named PedRPoly model. A first version of the PedRPoly model was first outlined in [51].

The new model, PedRPoly, is enhanced with several constraint modeling techniques. These techniques aim at improving the efficiency of the method and include the identification of lower bounds, symmetry breaking and a heuristic sorting technique. Furthermore, a significant number of constraint optimization solvers is tested. The use of an appropriate optimization solver with the model contributes for an efficient haplotype inference solver. This significantly improved version of

PedRPoly is described in [52].

7.1 Minimum Recombinant Maximum Parsimony

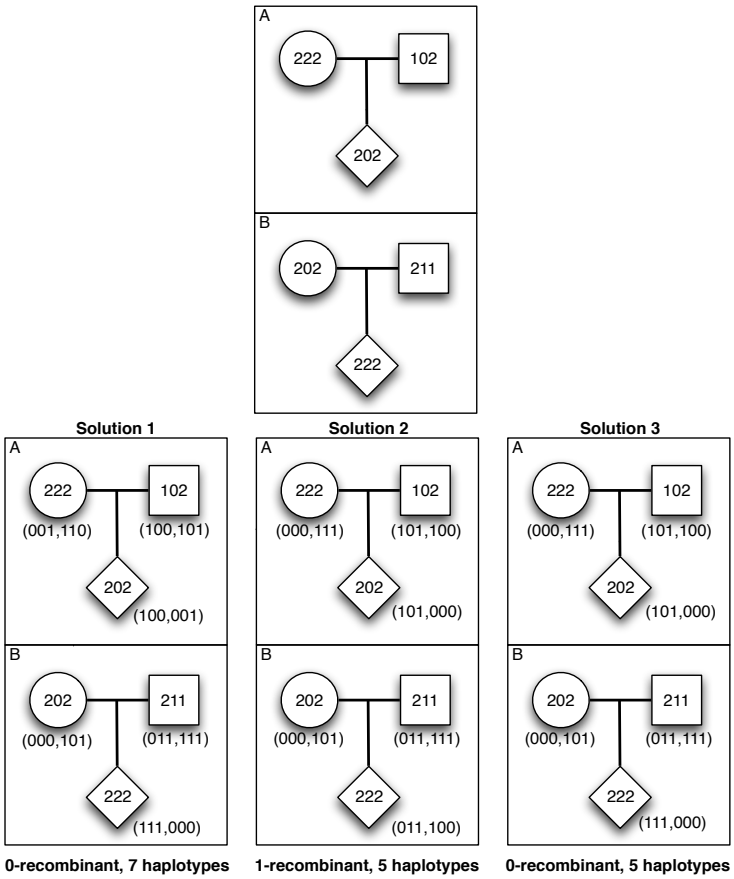
We propose a new approach for haplotype inference, denoted minimum recombinant maximum parsimony (MRMP) [52]. The MRMP approach is a combination of the HIPP and the MRHC approaches. Therefore, given a set of genotypes from individuals grouped in pedigrees, the MRMP problem searches the haplotype inference solution which minimizes the number of recombinants within pedigrees and the number of distinct haplotypes.

Definition 7.1. ***Minimum Recombinant Maximum Parsimony***
Given a set of n genotypes, $\mathcal{G}$, each genotype with m sites, from individuals grouped in K sets of pedigrees from the same population, the minimum recombinant maximum parsimony (MRMP) problem aims at finding a haplotype inference solution which, first, minimizes the number of recombination events within pedigrees and, then, minimizes the number of distinct haplotypes used within all individuals.

Example 7.2. *(Minimum Recombinant Maximum Parsimony) Figure 7.1 illustrates two trios (mother ◯, father □ and child ◇) from two families A and B with the corresponding genotypes. In this context, g_m^{fam} represents the genotype of the mother in trio fam, g_f^{fam} represents the genotype of the father in trio fam and g_c^{fam} represents the genotype of the child in trio fam, where $fam \in \{A, B\}$. Hence, the set of genotypes is $\mathcal{G} = \{g_m^A = 222,\ g_f^A = 102,\ g_c^A = 202,\ g_m^B = 202,\ g_f^B = 211,\ g_c^B = 222\}$. The bottom of the figure includes three haplotype inference solutions. Solution 1 is a 0-recombinant solution with 7 distinct haplotypes $\mathcal{H}_1 = \{100,101,000,111,011,001,110\}$, such that $g_m^A = 001 \oplus 110$, $g_f^A = 100 \oplus 101$, $g_c^A = 100 \oplus 001$, $g_m^B = 000 \oplus 101$, $g_f^B = 011 \oplus 111$ and $g_c^B = 111 \oplus 000$. Solution 2 is a 1-recombinant solution (there must be one recombination event in family B) using 5 distinct haplotypes $\mathcal{H}_2 = \{100, 101, 000, 111, 011\}$, such that $g_m^A = 000 \oplus 111$, $g_f^A = 101 \oplus 100$, $g_c^A = 101 \oplus 000$, $g_m^B = 000 \oplus 101$, $g_f^B = 011 \oplus 111$ and $g_c^B = 011 \oplus 100$. Solution 3 is a 0-recombinant solution using 5 distinct haplotypes $\mathcal{H}_3 = \{100, 101, 000, 111, 011\}$, such that $g_m^A = 000 \oplus 111$, $g_f^A = 101 \oplus 100$, $g_c^A = 101 \oplus 000$, $g_m^B = 000 \oplus 101$, $g_f^B = 011 \oplus 111$ and $g_c^B = 111 \oplus 000$.*

According to the MRMP model, solution 3 is preferred to the other solutions. Solution 3 is both a MRHC and a HIPP solution. Consequently, solution 3 is a Minimum Recombinant Maximum Parsimony solution. If there exists no solution that minimizes both criteria, then preference is given to the MRHC criterion. Hence, the MRHC solution which uses the smallest number of distinct

Figure 7.1: Solutions for haplotype inference with two trios



haplotypes would be chosen.

Theorem 7.3. *(Complexity) To find a solution to the minimum recombinant maximum parsimony (MRMP) is a NP-hard problem.*

Proof. The proof follows because the HIPP problem is NP-hard and can be reduced to the MRMP problem, by considering that each family contains only one individual. □

7.2 The PedRPoly Model

This section describes the PedRPoly model, which is the first approach for solving the minimum recombinant maximum parsimony problem. Note that the PedRPoly model is a combination of the MRHC PedPhase model [100] (Section 3.3.1) and the HIPP RPoly model [55] (Section 5.2).

PedRPoly is a pseudo-Boolean optimization model. The problem variables, the cost function

and the general constraints of the PedRPoly model are described in the following paragraphs and summarized in Table 7.1.

Variables

Following the RPoly model, PedRPoly associates two haplotypes, h_i^a and h_i^b, with each genotype g_i, and these haplotypes are required to explain g_i, $g_i = h_i^a \oplus h_i^b$, where h_i^a represents the haplotype inherited from the father and h_i^b represents the haplotype inherited from the mother. Moreover, PedRPoly associates a variable $t_{i\,j}$ with each heterozygous site $g_{i\,j} = 2$, such that $t_{i\,j} = 1$ indicates that the mutant type allele was inherited from the father and the wild type allele was inherited from the mother, whereas $t_{i\,j} = 0$ indicates that the wild type allele was inherited from the father and the mutant type allele was inherited from the mother, i.e.

$$t_{i\,j} = \begin{cases} 1 & \text{if } h_{i\,j}^a = 1 \wedge h_{i\,j}^b = 0 \\ 0 & \text{if } h_{i\,j}^a = 0 \wedge h_{i\,j}^b = 1 \end{cases} .$$

In addition, PedRPoly associates two variables, $t_{i\,j}^a$ and $t_{i\,j}^b$, with each missing site $g_{i\,j} =?$. Variable $t_{i\,j}^a$ is associated with the paternal haplotype site $h_{i\,j}^a$, whereas variable $t_{i\,j}^b$ is associated with the maternal haplotype site $h_{i\,j}^b$, i.e.

$$t_{i\,j}^a = h_{i\,j}^a$$

and

$$t_{i\,j}^b = h_{i\,j}^b.$$

The values of h_i^a and h_i^b at homozygous sites are implicitly assumed. Homozygous sites on the wild type nucleotide, $g_{i\,j} = 0$, imply that $h_{i\,j}^a = h_{i\,j}^b = 0$, whereas homozygous sites on the mutant type nucleotide, $g_{i\,j} = 1$, imply that $h_{i\,j}^a = h_{i\,j}^b = 1$.

The grandparental origin of each site of the haplotypes must be considered when analyzing recombination events within pedigrees. Following the MRHC PedPhase model, for each non-founder individual i and site j, two variables are defined: $g_{i\,j}^1$ and $g_{i\,j}^2$. The assignment $g_{i\,j}^1 = 0$ means that the paternal allele of individual i at site j (i.e. $h_{i\,j}^a$) comes from the paternal grandfather, and $g_{i\,j}^1 = 1$

means that $h^a_{i\,j}$ comes from the paternal grandmother, i.e.

$$g^1_{i\,j} = \begin{cases} 0 & \text{if} \quad h^a_{i\,j} = h^a_{f(i)\,j} \\ 1 & \text{if} \quad h^a_{i\,j} = h^b_{f(i)\,j} \end{cases},$$

where $f(i)$ corresponds to the father of individual i. In a similar way, $g^2_{i\,j} = 0$ ($g^2_{i\,j} = 1$) means that the maternal allele of individual i at site j comes from the maternal grandfather (grandmother), i.e.

$$g^2_{i\,j} = \begin{cases} 0 & \text{if} \quad h^b_{i\,j} = h^a_{m(i)\,j} \\ 1 & \text{if} \quad h^b_{i\,j} = h^b_{m(i)\,j} \end{cases},$$

where $m(i)$ corresponds to the mother of individual i.

In order to allow counting the number of recombinations, the model defines new variables, $r^1_{i\,j}$ and $r^2_{i\,j}$, for each non-founder individual i and $1 \leq j < m$. Variable $r^1_{i\,j}$ is assigned value 1 if a recombination took place at site j, to create the paternal haplotype of individual i. Similarly, variable $r^2_{i\,j}$ is assigned value 1 if a recombination took place at site j, to create the maternal haplotype of individual i. Thus,

$$r^l_{i\,j} = 1 \text{ if } g^l_{i\,j} \neq g^l_{i\,j+1},$$

for $l \in \{1, 2\}$ and $1 \leq j \leq m - 1$.

Moreover, a simplification to the original MRHC model is considered. Actually, in the PedPhase model, $r^l_{i\,j} = 1$ *if and only if* $g^l_{i\,j} \neq g^l_{i\,j+1}$. Observe that an implication, instead of an equivalence, is sufficient for correctness and allows reducing in half the number of constraints regarding variables r.

In addition, the model defines variables to count the number of distinct haplotypes used. Let $x^{p\,q}_{i\,k}$, with $p, q \in \{a, b\}$ and $1 \leq k < i \leq n$, be assigned value 1 if haplotype p of genotype g_i (h^p_i) and haplotype q of genotype g_k (h^q_k) are different.

Furthermore, the model needs variables u to denote when one of the haplotypes, associated with a given genotype, is different from all previous haplotypes. Hence, u^p_i, with $p \in \{a, b\}$ and $1 \leq i \leq n$, is assigned value 1 if haplotype p of genotype g_i is different from all previous haplotypes.

Table 7.1: The PedRPoly model (Minimum Recombinant Maximum Parsimony) [a]

minimize:	$(2n+1) \times \sum_{non-founder\ i} \sum_{j=1}^{m-1} (r^1_{i\,j} + r^2_{i\,j}) + \sum_{i=1}^{n} (u^a_i + u^b_i)$	(7.1)
subject to:		
Mendelian Laws		(Table 7.2)
$-r^l_{i\,j} + g^l_{i\,j} - g^l_{i\,j+1} \leq 0$	$\forall_{i:\ 1 \leq i \leq n,\ i\ non-founder},\ \forall_{j:\ 1 \leq j < m}$	(7.2)
$-r^l_{i\,j} - g^l_{i\,j} + g^l_{i\,j+1} \leq 0$	$\forall_{i:\ 1 \leq i \leq n,\ i\ non-founder},\ \forall_{j:\ 1 \leq j < m}$	(7.3)
$\neg(R \Leftrightarrow S) \Rightarrow x^{p\,q}_{i\,k}$ (Table 7.3)	$\forall_{i,k:\ 1 \leq k < i \leq n,\ \kappa(i,k)}$	(7.4)
$\sum_{\{k:\ k<i,\ \kappa(i,k)\}} \sum_{q \in \{a,b\}} x^{p\,q}_{i\,k} - u^p_i \leq 2\mathcal{K}_{<i} - 3$	$\forall_{i:\ 1 \leq i \leq n},\ \forall_{p \in \{a,b\}}$	(7.5)

[a] Genotypes must be sorted, starting with the genotypes used in the lower bound. The predicate $\kappa(i,k)$ is defined true iff genotypes g_k and g_i are compatible and $\mathcal{K}_{<i} = |\{g_k \in \mathcal{G} : k < i \wedge \kappa(i,k)\}|$

Cost Function

The cost function consists in minimizing the number of recombination events and the number of distinct haplotypes, which are, respectively, given by the sum of variables r and u,

$$minimize \quad ((2n+1) \times \sum_{(\text{non-founder } i)} \sum_{j=1}^{m-1} (r^1_{i\,j} + r^2_{i\,j})) + \sum_{i=1}^{n} (u^a_i + u^b_i). \tag{7.1}$$

Given that priority is given to the minimum recombinant criterion, a larger weight is associated with the number of recombinations. Note that $2n$ is a trivial upper bound on the number of haplotypes in the solution and, therefore, giving weight $2n+1$ to the number of recombinations implies that a MRHC solution is always preferred. The idea of associating a larger weight with the number of recombinations is biological motivated by the fact that recombination events within haplotypes in a pedigree are rare. Moreover, note that a larger number of recombinants suggests a larger number of haplotypes. In general, a recombination event generates a new haplotype, whereas without recombination, the haplotypes of the child are exact copies of the parents' haplotypes. Nonetheless, different weights w, $1 \leq w < 2n+1$, were also tried but did not lead to improvements neither in accuracy or efficiency.

General Constraints

The PedRPoly model requires constraints which ensure that the Mendelian laws of inheritance are satisfied. In addition, constraints must guarantee that if there is a recombination event between

Table 7.2: Mendelian laws of inheritance rules [a]

Condition	Constraint
$g_{i\,j} = 0 \wedge g_{f(i)\,j} = 2$	$t_{f(i)\,j} \Leftrightarrow g^1_{i\,j}$
$g_{i\,j} = 0 \wedge g_{f(i)\,j} = ?$	$(g^1_{i\,j} \vee \neg t^a_{f(i)\,j}) \wedge (\neg g^1_{i\,j} \vee \neg t^b_{f(i)\,j})$
$g_{i\,j} = 1 \wedge g_{f(i)\,j} = 2$	$t_{f(i)\,j} \Leftrightarrow \neg g^1_{i\,j}$
$g_{i\,j} = 1 \wedge g_{f(i)\,j} = ?$	$(g^1_{i\,j} \vee t^a_{f(i)\,j}) \wedge (\neg g^1_{i\,j} \vee t^b_{f(i)\,j})$
$g_{i\,j} = 2 \wedge g_{f(i)\,j} = 0$	$\neg t_{i\,j}$
$g_{i\,j} = 2 \wedge g_{f(i)\,j} = 1$	$t_{i\,j}$
$g_{i\,j} = 2 \wedge g_{f(i)\,j} = 2$	$(g^1_{i\,j} \vee t_{i\,j} \vee \neg t_{f(i)\,j}) \wedge (g^1_{i\,j} \vee \neg t_{i\,j} \vee t_{f(i)\,j}) \wedge$ $(\neg g^1_{i\,j} \vee t_{i\,j} \vee t_{f(i)\,j}) \wedge (\neg g^1_{i\,j} \vee \neg t_{i\,j} \vee \neg t_{f(i)\,j})$
$g_{i\,j} = 2 \wedge g_{f(i)\,j} = ?$	$(g^1_{i\,j} \vee t_{i\,j} \vee \neg t^a_{f(i)\,j}) \wedge (g^1_{i\,j} \vee \neg t_{i\,j} \vee t^a_{f(i)\,j}) \wedge$ $(\neg g^1_{i\,j} \vee t_{i\,j} \vee \neg t^b_{f(i)\,j}) \wedge (\neg g^1_{i\,j} \vee \neg t_{i\,j} \vee t^b_{f(i)\,j})$
$g_{i\,j} = ? \wedge g_{f(i)\,j} = 0$	$\neg t^a_{i\,j}$
$g_{i\,j} = ? \wedge g_{f(i)\,j} = 1$	$t^a_{i\,j}$
$g_{i\,j} = ? \wedge g_{f(i)\,j} = 2$	$(g^1_{i\,j} \vee t^a_{i\,j} \vee \neg t_{f(i)\,j}) \wedge (g^1_{i\,j} \vee \neg t^a_{i\,j} \vee t_{f(i)\,j}) \wedge$ $(\neg g^1_{i\,j} \vee t^a_{i\,j} \vee t_{f(i)\,j}) \wedge (\neg g^1_{i\,j} \vee \neg t^a_{i\,j} \vee \neg t_{f(i)\,j})$
$g_{i\,j} = ? \wedge g_{f(i)\,j} = ?$	$(g^1_{i\,j} \vee t^a_{i\,j} \vee \neg t^a_{f(i)\,j}) \wedge (g^1_{i\,j} \vee \neg t^a_{i\,j} \vee t^a_{f(i)\,j}) \wedge$ $(\neg g^1_{i\,j} \vee t^a_{i\,j} \vee \neg t^b_{f(i)\,j}) \wedge (\neg g^1_{i\,j} \vee \neg t^a_{i\,j} \vee \neg t^b_{f(i)\,j})$

[a] These constraints refer to variables $g^1_{i\,j}$. The constraints involving variables $g^2_{i\,j}$ are defined similarly. $f(i)$ corresponds to the father of i. $1 \leq i \leq n$, i non-founder, $1 \leq j \leq m$.

two positions, then the respective r variable is assigned value 1. The other constraints, related with variables x and u, are adopted from the RPoly model, and aim at counting the number of distinct haplotypes.

Constraints to ensure that the Mendelian laws of inheritance are satisfied are defined in Table 7.2. Note that PedRPoly only associates variables with heterozygous and missing sites (similar to RPoly), while PedPhase also associates variables with homozygous sites. The new definition of variables associated with sites requires the redefinition of the constraints related with the Mendelian laws of inheritance. For instance, consider the first constraint of Table 7.2, $t_{f(i)\,j} \Leftrightarrow g^1_{i\,j}$, for the case $g_{i\,j} = 0$ and $g_{f(i)\,j} = 2$. Clearly, if $t_{f(i)\,j} = 1$ (representing that individual $f(i)$ has inherited value 1

Table 7.3: Definition of predicates R and S

Condition	Predicate	
	R	S
$g_{i\,j} \neq 2 \wedge g_{k\,j} = 2$	$(g_{i\,j} \Leftrightarrow (q \Leftrightarrow a))$	$t_{k\,j}$
$g_{k\,j} \neq 2 \wedge g_{i\,j} = 2$	$(g_{k\,j} \Leftrightarrow (p \Leftrightarrow a))$	$t_{i\,j}$
$g_{i\,j} = 2 \wedge g_{k\,j} = 2$	$(p \Leftrightarrow q)$	$(t_{i\,j} \Leftrightarrow t_{k\,j})$
$g_{i\,j} =? \wedge g_{k\,j} \notin \{2, ?\}$	$t^p_{i\,j}$	$g_{k\,j}$
$g_{k\,j} =? \wedge g_{i\,j} \notin \{2, ?\}$	$t^q_{k\,j}$	$g_{i\,j}$
$g_{i\,j} =? \wedge g_{k\,j} = 2$	$(q \Leftrightarrow a)$	$(t^p_{i\,j} \Leftrightarrow t_{k\,j})$
$g_{k\,j} =? \wedge g_{i\,j} = 2$	$(p \Leftrightarrow a)$	$(t^q_{k\,j} \Leftrightarrow t_{i\,j})$
$g_{i\,j} =? \wedge g_{k\,j} =?$	$t^p_{i\,j}$	$t^q_{k\,j}$

from his father and value 0 from his mother) then $g^1_{i\,j} = 1$ (representing that individual i must have inherited the value 0 from his paternal grandmother). The contrary also holds.

Constraints which ensure that $r^l_{i\,j} = 1$ if $g^l_{i\,j} \neq g^l_{i\,j+1}$ are the following:

$$-r^l_{i\,j} + g^l_{i\,j} - g^l_{i\,j+1} \leq 0 \tag{7.2}$$

and

$$-r^l_{i\,j} - g^l_{i\,j} + g^l_{i\,j+1} \leq 0, \tag{7.3}$$

for $1 \leq i \leq n$, i non-founder, $1 \leq j < m$.

Based on the RPoly model, the conditions on the $x^{p\,q}_{i\,k}$ variables are related to the values of variables $t_{i\,j}$ and $t_{k\,j}$ for heterozygous sites and of variables $t^a_{i\,j}$, $t^b_{i\,j}$, $t^a_{k\,j}$ and $t^b_{k\,j}$ for missing sites, and are described by equations

$$\neg(R \Leftrightarrow S) \Rightarrow x^{p\,q}_{i\,k}, \tag{7.4}$$

where predicates R and S are described in Table 7.3, for $1 \leq k < i \leq n$ with $\kappa(i,k)$ and $p, q \in \{a, b\}$. Note that $x^{p\,q}_{i\,k} = 1$ if genotypes g_i and g_k are incompatible, so it is only required to introduce constraints for g_i and g_k such that $\kappa(i,k)$ is true.

Then, the conditions on the u^p_i variables are based on the conditions for the $x^{p\,q}_{i\,k}$ variables, with

$1 \leq k < i$ and $q \in \{a, b\}$. These conditions are described by

$$\sum_{\{k:\ k<i,\ \kappa(i,k)\}} \ \sum_{q \in \{a,b\}} x_{i\,k}^{p\,q} - u_i^p \leq 2\mathcal{K}_{<i} - 3, \tag{7.5}$$

where $\kappa(i, k)$ is a predicate which is true if and only if g_i and g_k are compatible and $\mathcal{K}_{<i}$ is the cardinality of the set $\{g_k \in \mathcal{G} : k < i \wedge \kappa(i, k)\}$.

7.3 Optimized PedRPoly

This section describes three improvements on the original PedRPoly model, which contribute for an efficient haplotype inference method. In addition, the practical contribution of each technique to improving the efficiency of PedRPoly is detailed. In what follows, we used PedRPoly with the Boolean multilevel optimization (BMO) MaxSAT solver, MSUnCore [7]. Although we consider PedRPoly to be a PBO model, PBO and MaxSAT are equivalent formalisms (Section 2.4.2), and, therefore, the use of MSUnCore is straightforward.

7.3.1 Integrating Lower Bounds

The FurtherImprovedLowerBound procedure (Section 4.1.2) provides a list of genotypes with an indication of the contribution of each genotype to the lower bound. Each genotype either contributes with +2, indicating that 2 new haplotypes will be required for explaining this genotype, or with +1, indicating that 1 new haplotype will be required.

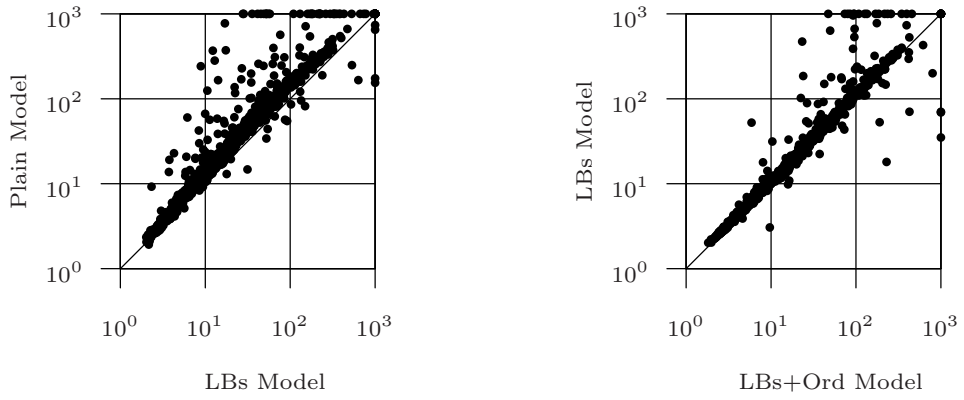
This lower bound has been included in the PedRPoly model. s As in RPoly, the use of lower bounds allows the variables u associated with haplotypes affected by the lower bound to be fixed and, consequently, the clauses used for constraining the value *need not* to be generated. Indeed, if g_i is a genotype contributing with +2 to the lower bound, then $u_i^a = 1$ and $u_i^b = 1$. Moreover, if g_i is a genotype contributing with +1 to the lower bound, then either u_i^a or u_i^b can be assigned value 1. The new model with integrating lower bounds will be named *PedRPoly-LB*.

Practical Impact

The dataset used in the experimental results consists of 945 challenging instances, which will be described in detail in Section 8.1.

Figure 7.2 (left) provides a scatter plot which compares the performance of the plain PedRPoly model with PedRPoly implementing the identification of lower bounds, within a timeout of 1000

Figure 7.2: CPU time comparison between models: plain PedRPoly model *vs* PedRPoly-LB model and PedRPoly-LB model *vs* PedRPoly-LB-Ord model



seconds.

PedRPoly-LB reduces in half the number of instances aborted by the plain PedRPoly model. The plain model aborts 59 instances while PedRPoly-LB is not able to solve 26 instances. PedRPoly-LB solves 37 instances which the plain PedRPoly aborts, although being able to solve 4 instances which PedRPoly-LB aborts. Moreover, PedRPoly-LB is faster than plain PedRPoly for more than 94% of the problem instances.

7.3.2 Sorting Genotypes

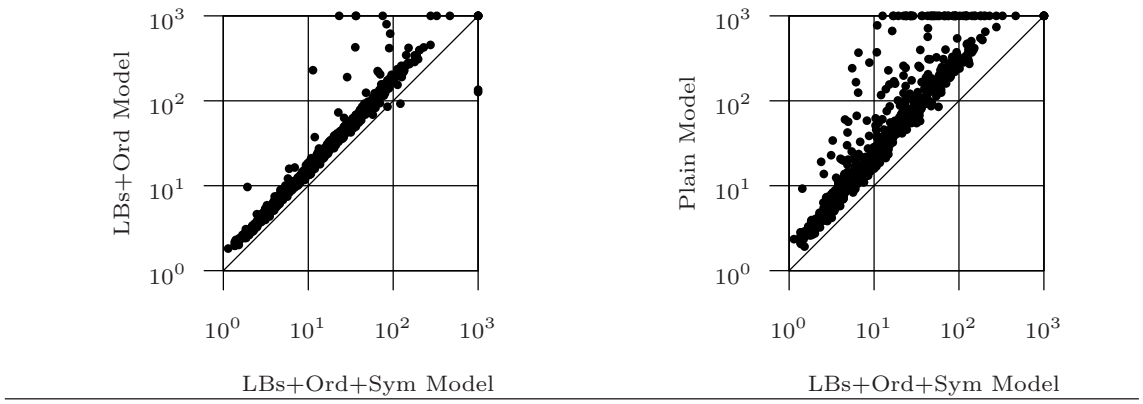
The order in which the genotypes are organized, before the model being generated, can have an important impact on the efficiency of the solver. In particular, note that variables u designate whether a haplotype associated with a genotype is different from all previous haplotypes. The lexicographic order in the genotypes is used as an heuristic defined by a total order in the genotype sites where $0 < 1 < 2 <\ ?$, i.e.

$$g_{i\,j} < g_{l\,j} \wedge (\forall_{\{k:\ k<j\}}\ g_{i\,k} = g_{l\,k}) \Rightarrow i < l, \tag{7.6}$$

for $1 \leq i, l \leq n$ and $1 \leq k < j \leq m$.

The new model, which integrates lower bounds and where the genotypes are sorted according to the lexicographic order is named *PedRPoly-LB-Ord*.

Figure 7.3: CPU time comparison between models: PedRPoly-LB-Ord model *vs* PedRPoly-LB-Ord-Sym model and plain PedRPoly model *vs* PedRPoly-LB-Ord-Sym model



Practical Impact

Figure 7.2 (right) compares the performance of PedRPoly-LB with PedRPoly-LB-Ord. PedRPoly-LB-Ord is able to solve 17 instances which PedRPoly-LB aborts. However, there are 3 instances which PedRPoly-LB solves and PedRPoly-LB-Ord is not able to solve. Moreover, PedRPoly-LB-Ord is faster than PedRPoly-LB for 86% of the instances.

7.3.3 Eliminating Key Symmetries

Symmetry breaking is a well-known technique for pruning the search space and, therefore, is expected to contribute to the efficiency of a model. Note that, in general, in the haplotype inference problem, if a genotype g is explained by the haplotype pair (h^a, h^b), then g is also explained by the haplotype pair (h^b, h^a). Within pedigrees, this symmetry in pairs of haplotypes does not exist for every individual. For non-founders, symmetry is already broken by imposing that the first haplotype comes from the father and the second haplotype comes from the mother. Nonetheless, symmetry can be broken in founders. This symmetry is broken by introducing a new constraint for each heterozygous founder i, imposing that the first heterozygous site $g_{i\,j}$ is explained with $h^a_{i\,j} = 1$ and $h^b_{i\,j} = 0$, i.e.

$$g_{i\,j} = 2 \wedge (\forall_{\{k:\ k<j\}}\ g_{i\,k} \neq 2) \Rightarrow t_{i\,j} = 1, \tag{7.7}$$

for $1 \leq i \leq n$ and $1 \leq k < j \leq m$.

The new model which includes breaking symmetry in founders is named *PedRPoly-LB-Ord-Sym*.

Table 7.4: PedRPoly: comparison between models (timeout 1000 sec; memory limit 3.5 GB)

Solver	# Solved inst.	% Solved inst.	Avg run time (sec)
PedRPoly	886/945	93.76%	62.50
PedRPoly-LB	919/945	97.25%	47.30
PedRPoly-LB-Ord	933/945	98.73%	41.64
PedRPoly-LB-Ord-Sym	938/945	99.26%	24.08

Practical Impact

Figure 7.3 (left) compares the performance of PedRPoly-LB-Ord with PedRPoly-LB-Ord-Sym. The final model is able to solve 938 out of 945 instances. PedRPoly-LB-Ord-Sym solves 7 instances which PedRPoly-LB-Ord aborts and aborts 2 instances which PedRPoly-LB-Ord solves. Moreover, PedRPoly-LB-Ord-Sym is faster than PedRPoly-LB-Ord for 99% of the instances.

Figure 7.3 (right) compares the performance of plain PedRPoly and PedRPoly-LB-Ord-Sym. The later is faster than the former for *all* instances, and solves 52 instances which the plain model aborts. These facts illustrate the importance of the improved model in the efficiency of PedRPoly.

Table 7.4 summarizes the improvement achieved by combining modeling techniques. Overall, PedRPoly-LB-Ord-Sym outperforms all other models, being capable of solving 99.26% of the instances within 1000 seconds, and using an average run time of 24 seconds. In the remainder of the paper, PedRPoly-LB-Ord-Sym will be denoted simply by PedRPoly.

7.4 Additional Remarks

Solving the PedRPoly model is a binate covering problem, i.e. each constraint can be translated into SAT using a single clause. This fact suggests that optimization solvers based on SAT may be adequate for solving the PedRPoly model.

Indeed, PedRPoly is a Boolean optimization model which can be solved using any PBO or MaxSAT solver.

Regarding the size of the model, the following space complexity theorem may be formulated.

Theorem 7.4. *(Space Complexity) Let n be the number of genotypes in the pedigree and m the number of positions of each genotype. Then, the number of variables of the PedRPoly model is $\mathcal{O}(n^2 + nm)$ and, in addition, the number of constraints of the PedRPoly model is $\mathcal{O}(n^2 m)$.*

Table 7.5: PedRPoly: comparison using different solvers (timeout 1000 sec; memory limit 3.5 GB)

Solver	# Solved inst.	% Solved inst.	Avg run time (sec)
MaxSat_bmo	938/945	99.26%	24.08
WPM1	911/945	96.40%	14.75
CPLEX	553/945	58.52%	84.73
Scip	455/945	48.15%	139.07
MiniSat+	260/945	27.51%	238.45
IncWMaxSatz	221/945	23.39%	71.04
Bsolo	160/945	16.93%	170.26
WMaxSatz	16/945	1.69%	113.16

Finally, we would like to point out that not every key feature used in the RPoly model can be integrated in the PedRPoly model. In particular, structural simplifications cannot be applied. Note, for example, that two equal genotypes may have to be explained differently according to the genetic data of their relatives.

7.5 Impact of the Constraint Solver

A key issue for the efficiency of the haplotype inference solver is to select an adequate underlying optimization solver. In this section, eight different optimization solvers were tested for solving the final version of the PedRPoly model. Integer linear programming, pseudo-Boolean optimization and also weighted MaxSAT solvers were considered. Scip [2] (version 1.2.0) combines constraint programming and mixed integer programming methodologies. CPLEX (version 12.1) is an IBM/ILOG commercial linear programming optimization tool. Weighted MaxSAT solvers were also tested: MaxSat_bmo [7], WPM1 [5], WMaxSatz [96] (version 2.5), and IncWMaxSatz [106]. MiniSat+ [38] and Bsolo [119] (version 3.5) are pseudo-Boolean optimization solvers, also known as 0-1 ILP solvers.

Table 7.5 summarizes the performance of the different solvers. Clearly, the solver which is able to solve a larger number of instances is MaxSat_bmo, which solves 99.26% of the instances.

The second best performing solver is WPM1 which solves 96.40% of the instances. The third and fourth best performing solvers are the integer programming solvers. CPLEX solves 58.52% and Scip

solves 48.15% of the instances, followed by MiniSat+ which solves 27.51% and IncWMaxSatz which solves 23.39% of the problem instances. Bsolo solves 16.93% and WMaxSatz solves 1.69% of the instances. Most of the instances aborted by IncWMaxSatz and WMaxSatz were due to limitations in the internal data structures used by these solvers.

The fact that MaxSat_bmo is the best solver for solving the PedRPoly model is justified by the fact that MaxSat_bmo performs Boolean multilevel optimization, and therefore is specialized for solving Boolean problems with more than one cost function.

7.6 Conclusions

This chapter proposes a new approach for the haplotype inference problem. The new approach combines two well-known combinatorial approaches: the pure parsimony [55] and the minimum recombinant [100] approaches. Hence, the new approach is named minimum recombinant maximum parsimony (MRMP) approach.

Given a set of pedigrees from the same population, the MRMP approach aims at finding a solution to the haplotype inference problem which minimizes the number of recombination events within families and, between those solutions, choose a solution which uses the minimum number of distinct haplotypes [51, 52].

A model for solving the MRMP problem is proposed. The new model, named PedRPoly, is based on the PedPhase and on the RPoly models. The plain model needs to be enhanced with several constraint modeling techniques, such as introducing lower bounds, breaking symmetries and including heuristic sorting techniques, in order to be practical to use. These modeling techniques increase the number of instances solved by 6%. In addition, the most adequate constraint solver must be chosen. MaxSat_bmo [7] increases the number of instances solved by 260% when compared with PedRPoly using MiniSat+ [38]. MaxSat_bmo and WPM1 [5] are the most robust solvers at tackling the PedRPoly model, each solving, respectively, 99.3% and 96.4% of the problem instances.

To summarize, the PedRPoly is a new model for haplotype inference in pedigrees, which is efficient and practical for challenging instances.

8

Haplotype Inference in Pedigrees: Experimental Results

This chapter has the purpose of studying the accuracy of PedRPoly, a method for haplotype inference in pedigrees whose formulation was proposed in the previous chapter.

First of all, the experimental setup, including the used datasets and haplotype inference algorithms, are presented. Second, the accuracies of PedRPoly and PedPhase are compared. Third, the accuracy of PedRPoly is compared with the accuracy of statistical pedigree-based haplotype inference methods. Finally, the conclusions are presented.

8.1 Experimental Setup

8.1.1 Datasets

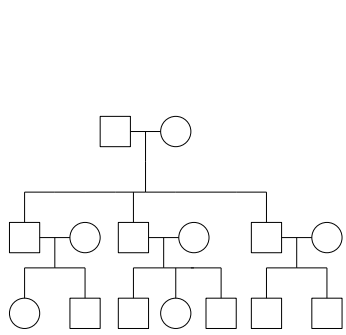
The experimental data was simulated using the SimPed software [1] [95]. Given the pedigree structure, as well as the frequencies of haplotypes for founder individuals, SimPed generates haplotypes for all individuals in the pedigree.

The haplotypes for founders and their frequencies were obtained from 7 real datasets of experimentally identified haplotypes [4, 133], and correspond to the A-G datasets already used in other haplotyping studies [27]. The number of SNPs ranges from 5 to 47. Note that haplotyping regions with tens of SNPs are still relevant in several association studies. Moreover, larger regions can always be partitioned into small blocks [172].

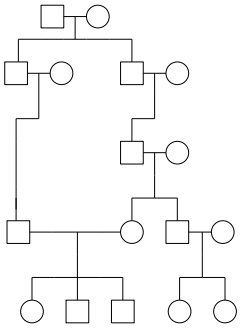
The same three pedigree structures used in the PedPhase paper [100] were considered: pedigree 1 with 15 individuals, pedigree 2 with 29 individuals and pedigree 3 with 17 individuals. Pedigree 3

[1] *http://www.hgsc.bcm.tmc.edu/cascade-tech-software_simped-ti.hgsc*

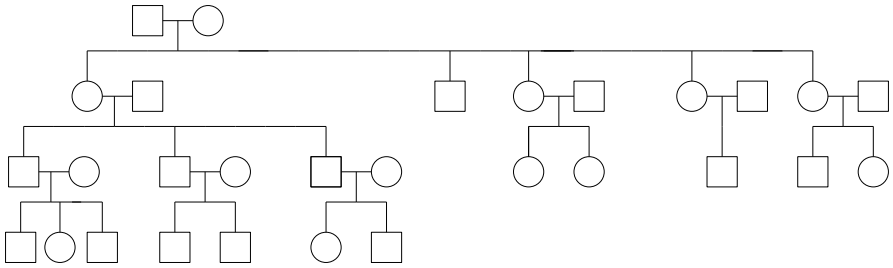
Figure 8.1: Pedigree structures: pedigree 1 has 15 individuals, pedigree 2 has 29 individuals and pedigree 3 has 17 individuals and a mating loop



Pedigree 1



Pedigree 3



Pedigree 2

contains a mating loop, which means that two mating individuals have a common ancestor in the pedigree. Pedigree structures are represented in Figure 8.1.

Each simulated instance consists of 10 replicates of the given pedigree, simulating 10 different families from the same population. Hence, the number of genotypes per instance may be 150, 290 or 170. Recombination events are uniformly distributed between SNPs with intermarker recombination fractions of 0.1%, 0.5% and 1%. Three variations in missing rates were considered: 1%, 10% and 20%. For each combination of parameters, 5 independent replicates were selected, resulting in a total of 945 ($= 7 \times 3^3 \times 5$) input trials. Table 8.1 summarizes the details of the used instances.

The assumed intermarker recombination rate of 0.1-1% is quite high. Nonetheless, the evaluation performed is interesting in regions of high recombination rates (recombination hotspots) where the SNPs are sparsely sampled.

Genotyping errors have not been simulated. Nonetheless, genotyping errors do not represent a significant limitation because they can be minimized by previously applying an appropriate error

Table 8.1: Details of the dataset

# Instances	945
Number of SNPs	5-47
Number of Genotypes	150-290
Size of Pedigrees	15-29
Intermarker Recombination Fractions	0.1%-1%
Missing Rate	1%-20%

detection software [141].

8.1.2 Setting-up

All results were obtained on a Intel Xeon 5160 server (3.0GHz, 1333Mhz, 4GB) running Red Hat Enterprise Linux WS4. PedPhase was run on Windows because the algorithm of this software that solves the minimum recombinant haplotype configuration problem is not available for Linux. Results are presented for a timeout of 1000 seconds and a memory limit of 3.5 GB.

8.1.3 Phasing Algorithms

The accuracy of four methods for haplotype inference in pedigrees is compared in this chapter. PedRPoly, described in the previous chapter, is compared with PedPhase, Superlink and PhyloPed.

For PedPhase [2] [100], version 2.1 is used. When we mention the PedPhase method, we always refer to the integer linear programming method that solves the minimum recombinant haplotype configuration problem. Indeed, PedPhase is a more general tool, which implements five algorithms for haplotype inference in pedigrees: the ILP algorithm, the block-extension algorithm, the constraint-finding algorithm for 0-recombinant data, the locus- and the member-based dynamic programming algorithms.

In addition, two statistical tools: Superlink [3] [42] (version 1.7) and PhyloPed [4] [88] (version 0.4) were used. Superlink performs a maximum likelihood calculation and PhyloPed uses blocked Gibbs sampling, applying the perfect phylogeny model if there is little evidence of ancestral recombinations

[2] *http://vorlon.case.edu/~jxl175/haplotyping.html*
[3] *http://cbl-fog.cs.technion.ac.il/superlink*
[4] *http://phyloped.icsi.berkeley.edu/phyloped*

or recurrent mutations in the founder haplotypes.

8.2 Comparing Accuracies

Two different commonly used error rates were considered as performance metrics. The *switch error rate* measures the percentage of possible switches in haplotype orientation, used to recover the correct phase in an individual [107]. Missing alleles are not considered for computing the switch error. The *missing error rate* (or *genotype inference error rate*) is the percentage of incorrectly inferred missing data [121].

8.2.1 PedRPoly *vs* PedPhase

This section analyzes the gains in accuracy resulting from adding the maximum parsimony criterion to the minimum recombinant criterion, i.e. the accuracy of PedRPoly, which integrates both HIPP with MRHC, is compared with the one of PedPhase, which uses only the MRHC approach.

PedRPoly aims at being an improvement on the PedPhase approach. Indeed, PedRPoly uses the minimum recombinant maximum parsimony approach, whose set of solutions is contained in the set of solutions of the MRHC approach.

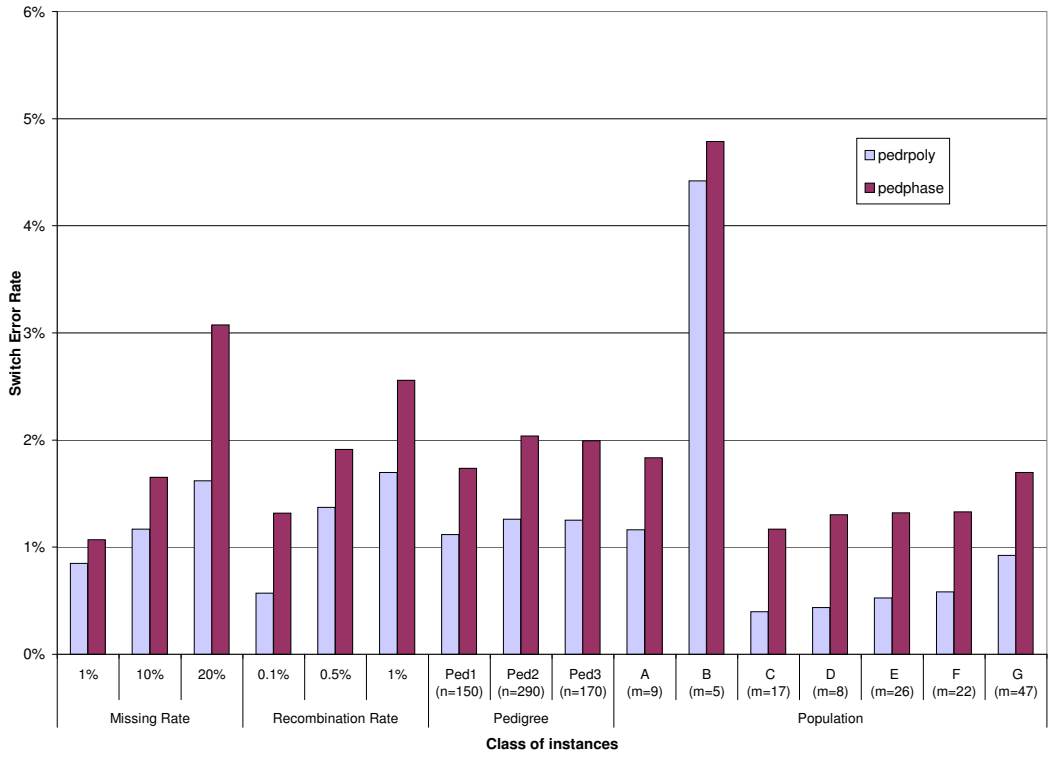
Note that instances for which at least one of the two solvers is unable to give a solution within the timeout have been removed from the comparison. PedPhase is able to solve 99.8% of the instances, whereas PedRPoly is able to solve 99.3%. As a result, 9 out of 945 instances have been left out.

Figure 8.2 presents a bar graph comparing the switch error rate of PedRPoly with the switch error rate of PedPhase. Results have been organized by parameter value: missing rate, recombination rate, pedigree and population. Each value is the average of the error rate for the instances generated with the corresponding parameter value. PedRPoly is more accurate than PedPhase for 67.09% of the instances. The two solvers have equal error rates for 19.55% of the instances. For 13.35% of the instances, PedRPoly is less accurate than PedPhase.

Considering all instances that both solvers are able to solve, the average switch error rate of PedRPoly is 1.21% and the average switch error rate of PedPhase is 1.92%, i.e. the switch error rate of PedRPoly is 37% less than the one of PedPhase.

In addition, a few interesting observations can be stated. As one might expect, the error rate increases as the missing and the recombination rates increases. Moreover, it is clear that the instances of population B are associated with higher error rates. This fact is explained by the instances of this class having a small number of SNPs, more exactly five SNPs. Hence, a small

Figure 8.2: Switch error: comparing PedRPoly and PedPhase



number of errors implies a large proportion of errors because the number of heterozygous sites per genotype is small.

Figure 8.3 presents a bar graph for evaluating the missing error rate of the two tools. PedRPoly is more accurate than PedPhase for 73.93% of the instances. The two solvers have equal error rate for 12.39% of the instances. For 13.68% of the instances, PedRPoly is less accurate than PedPhase. Moreover, considering all instances that both solvers are able to solve, the average missing error rate of PedRPoly is 2.51% and the average missing error rate of PedPhase is 4.10%, i.e. the missing error rate of PedRPoly is 39% less than the one of PedPhase.

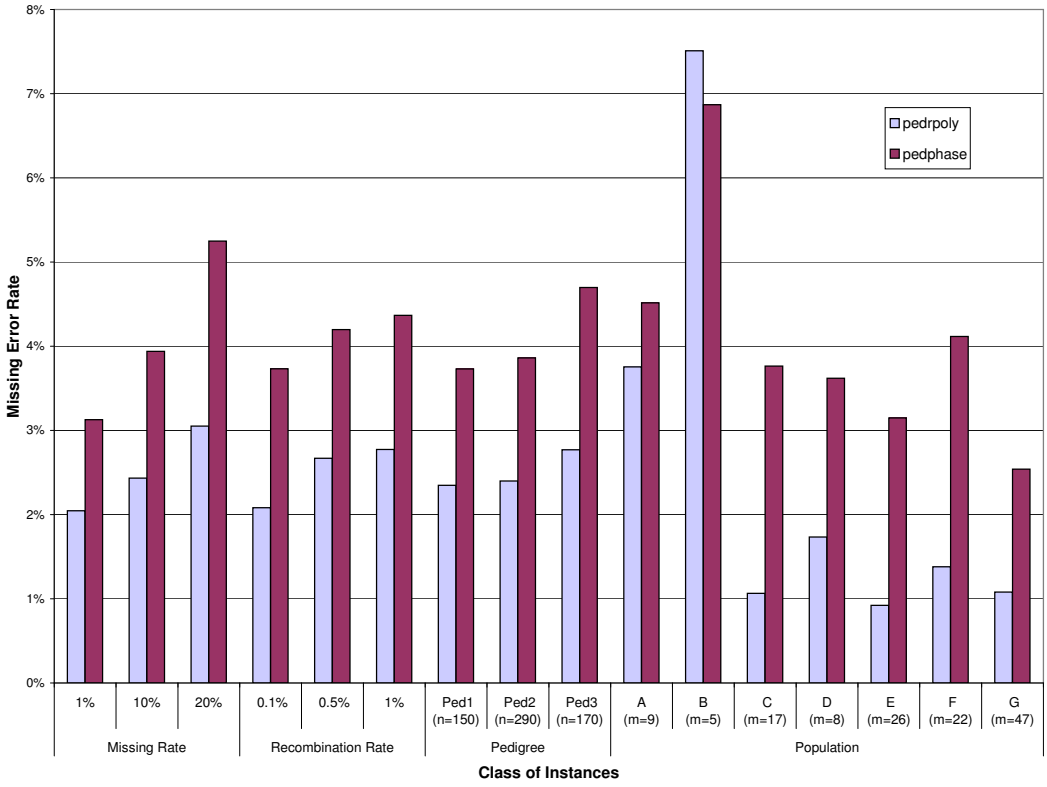
The population information included by the PedRPoly model is shown to be extremely important for haplotyping and inferring missing data. Overall, we conclude that PedRPoly consistently outperforms PedPhase in terms of accuracy.

Comparing the Number of Haplotypes

The numbers of distinct haplotypes in the PedRPoly and in PedPhase solutions are compared with the number of distinct haplotypes in the real solution.

Figure 8.4 (left) provides the distribution of the difference between the number of distinct haplo-

Figure 8.3: Missing error: comparing PedRPoly and PedPhase

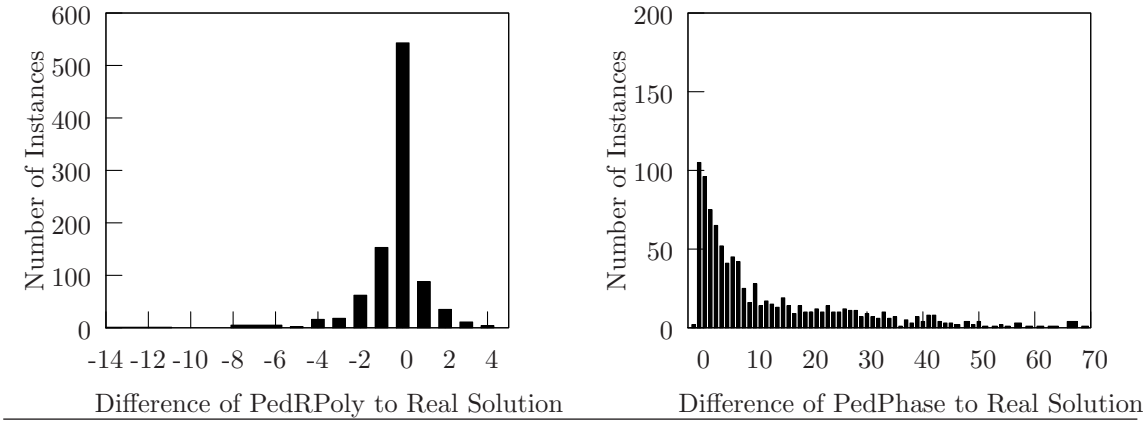


types in the PedRPoly solution and in the real solution. The number of haplotypes in the PedRPoly solution is exactly the same as in the real solution for 60% of the instances, and for more than 99.6% of the problem instances, the number of haplotypes in the PedRPoly solution differs from the number of haplotypes in the real solution by less than 5 haplotypes. Note that there are 138 instances, corresponding to the bars with positive values: 1, 2, 3 and 4, for which the number of haplotypes in the real solution is lower than the number of haplotypes in the PedRPoly solution. This fact indicates that the number of recombinants is underestimated and the number of haplotypes is overestimated, i.e. the real solution is not the one which minimizes the number of recombinants.

Figure 8.4 (right) provides the distribution of the difference between the number of haplotypes in the PedPhase solution and in the real solution. Clearly, PedPhase solutions are less similar to the real solutions with respect to the number of haplotypes. For the same set of instances, the PedPhase solution has the same number of haplotypes as the real solution for 12.3% of the instances, and differs from the real solution by less than 5 haplotypes for 45.8% of the instances.

A study was performed to analyze any particularity of the instances where PedRPoly is less accurate than PedPhase. The only fact concluded is that, in general, those are instances where the

Figure 8.4: Difference between the number of distinct haplotypes in the solution provided by PedRPoly/PedPhase and in the real solution



error rate of PedPhase is inferior to the average error rate of PedPhase, even though the difference in the number of haplotypes to the real solution can be large.

8.2.2 PedRPoly *vs* Statistical Approaches

Although the main goal of this work is to show that MRHC combined with HIPP has better accuracy than MRHC alone, two distinct statistical methods for haplotype inference within pedigrees are also evaluated. These methods are Superlink [42] and PhyloPed [88].

We would like to point out that some other methods for haplotype inference in pedigrees, such as HAPLORE [173] and ZAPLO [132], could not be used in these experiments because these algorithms explicitly assume no recombination between SNPs, which our datasets contain.

PhyloPed does not work properly for very long blocks, as it runs out of memory for the algorithms employed. For this reason, in this experiment we have used PhyloPed with blocks of size 5. Hence, the number of blocks in the instances of populations A, B, C, D, E, F and G is 2, 1, 4, 2, 6, 5 and 10, respectively. In addition, the current version of PhyloPed does not allow mating loops. Hence, PhyloPed is not used in instances containing pedigree 3. Moreover, PhyloPed does not allow individuals containing only missing sites and, therefore, one additional instance with this feature has to be removed. At the end, PhyloPed can be run in 629 instances from the considered dataset.

In addition, note that Superlink is used with information in the recombination rates, information which is not given to the other methods.

Table 8.2: Switch error rate

		PedRPoly	PedPhase	Superlink	PhyloPed
Missing Rate	1%	**0.83%**	1.07%	1.13%	15.90%
	10%	**1.17%**	1.56%	1.95%	16.72%
	20%	**1.83%**	3.22%	3.84%	15.49%
Recombination Rate	0.1%	**0.57%**	1.31%	1.49%	12.02%
	0.5%	**1.56%**	1.92%	2.39%	16.13%
	1%	**1.59%**	2.36%	2.73%	20.01%
Pedigree	Ped1 (n=150)	**1.12%**	1.72%	2.04%	15.00%
	Ped2 (n=290)	**1.39%**	2.05%	2.41%	17.39%
Population	A (m=9)	**1.10%**	1.86%	2.44%	18.36%
	B (m=5)	**4.58%**	4.72%	5.43%	8.93%
	C (m=17)	**0.36%**	1.04%	1.31%	18.61%
	D (m=8)	**0.37%**	1.22%	1.41%	8.60%
	E (m=26)	**0.39%**	1.12%	1.31%	22.03%
	F (m=22)	**0.52%**	1.13%	1.34%	18.82%
	G (m=47)	**0.95%**	1.57%	1.64%	20.24%
TOTAL	All	**1.24%**	1.87%	2.20%	16.06%

Instances for which at least one of the methods is unable to give a solution have been removed from the comparison. PedPhase is unable to solve 2 instances, PedRPoly is unable to solve 7 instances, Superlink does not solve 69 instances and PhyloPed cannot be run in 316 instances. As a result, there are 559 out of 945 instances for which all methods can provide a solution within the timeout.

Table 8.2 summarizes the switch error rate of the four haplotype inference in pedigrees methods. One can observe that, for each class of instances, PedRPoly is consistently the most accurate, exhibiting the lower error rate. PedPhase is the second most accurate method, followed by Superlink. PhyloPed exhibits the most higher error rates. Considering all the 559 instances, the average switch error rate of PedRPoly is 1.24%, the average switch error rate of PedPhase is 1.87%, the average switch error rate of Superlink is 2.20% and the average switch error rate of PhyloPed is 16.06%.

Table 8.3 summarizes the missing error rate of the four haplotype inference in pedigrees methods.

Table 8.3: Missing error rate

		PedRPoly	PedPhase	Superlink	PhyloPed
Missing Rate	1%	**1.95%**	2.74%	8.37%	11.96%
	10%	**2.40%**	3.77%	10.11%	13.21%
	20%	**3.33%**	5.18%	12.62%	14.46%
Recombination Rate	0.1%	**2.19%**	3.43%	9.83%	11.97%
	0.5%	**2.68%**	3.90%	10.42%	13.49%
	1%	**2.65%**	4.10%	10.37%	13.89%
Pedigree	Ped1(n=150)	**2.36%**	3.73%	10.27%	12.87%
	Ped2(n=290)	**2.69%**	3.91%	10.12%	13.44%
Population	A(m=9)	**3.48%**	3.84%	9.98%	15.01%
	B(m=5)	7.31%	**6.94%**	13.92%	10.80%
	C(m=17)	**0.96%**	3.27%	9.05%	13.01%
	D(m=8)	**1.63%**	3.46%	12.07%	12.47%
	E(m=26)	**0.83%**	2.83%	9.10%	13.95%
	F(m=22)	**1.26%**	3.52%	7.78%	14.30%
	G(m=47)	**0.99%**	1.98%	8.12%	12.53%
TOTAL	All	**2.51%**	3.81%	10.21%	13.12%

PedRPoly exhibits the smaller missing error rate for all classes of instances, except for population B, where PedPhase shows to be more accurate. Superlink and PhyloPed exhibit higher error rates. Considering all the 559 instances, the average missing error rate of PedRPoly is 2.51%, the average missing error rate of PedPhase is 3.81%, the average missing error rate of Superlink is 10.21% and the average missing error rate of PhyloPed is 13.12%. PedRPoly, PedPhase and Superlink have higher missing error rates than switch error rate, whereas PhyloPed has smaller missing error rates than switch error rate.

One explanation for the high error rates of PhyloPed is the partition in small blocks. Unfortunately, PhyloPed does not handle large blocks, neither uses an expeditious ligation method. Hence, PhyloPed tend to be more accurate in instances with a small number of SNPs. Indeed, for the class of population B, PhyloPed presents smaller error rates than for the average instance, whereas the other three methods drastically increase the error rate.

Regarding the efficiency of the methods, PedPhase and PhyloPed are the most efficient, solving each instance in a few seconds. PedRPoly is the third most efficient solver, being able to solve 938 out of the 945 instances within 1000 seconds. PedRPoly efficiency is analyzed in detailed in Chapter 7. Superlink is the less efficient method, being unable to solve 69 instances within 1000 seconds. Setting the timeout to 10,000 seconds, Superlink is still not able to solve 57 instances.

8.3 Conclusions

The proposed method for haplotype inference in pedigrees, PedRPoly, was tested on a set of 945 simulated instances, with different populations, pedigree structures, recombination and missing rates. The accuracy of PedRPoly was compared with the accuracies of other methods for haplotype inference in pedigrees, among which are PedPhase [100], Superlink [42] and PhyloPed [88].

Experimental results show that the PedRPoly method is considerably more accurate than PedPhase, exhibiting smaller switch and missing error rates. This fact confirms that the population information included by the PedRPoly model is important for haplotyping and inferring missing data. Moreover, PedRPoly significantly outperforms Superlink and PhyloPed in the analyzed data.

Overall, we conclude that PedRPoly is an accurate method for haplotype inference in pedigrees and is a credible alternative to the state of the art approaches.

9

Conclusions

This chapter concludes the dissertation by summarizing the achievements and pointing out directions for future research work.

9.1 Summary of Achievements

In the past years, there has been an enormous progress in developing new computational methods for solving the haplotype inference problem. In particular, the haplotype inference by pure parsimony approach has deserved a considerable attention in the last decade, with several HIPP methods being proposed. However, the HIPP problem is NP-hard and, therefore, efficient methods for HIPP are of interest.

Boolean satisfiability has been successfully applied in different fields. The application of SAT-based methodologies in haplotype inference has been shown to produce very efficient results when compared to alternative methods.

This PhD work contributes with new SAT-based algorithms for haplotype inference, which are very competitive, both in terms of efficiency and accuracy. This dissertation describes three main contributions of the research developed in this PhD, which are summarized in the following paragraphs.

The first achievement of this work is a new method for solving the HIPP problem. The new method, named RPoly, is considerable more efficient than the other methods developed for solving the HIPP problem. RPoly is inspired by previous HIPP methods, but includes several new optimizations. First, RPoly is a compact model, with a reduced number of variables and constraints. Moreover, RPoly integrates the computation of lower bounds, that allows further reducing the size of the model. RPoly also includes cardinality constraints, that allow pruning the search space. In

addition, the use of the pseudo-Boolean optimization solver MiniSat+ is shown to have a significant impact in improving the run times of the haplotype inference method. Furthermore, RPoly is able to handle missing sites. All these features contribute for a method which is significantly efficient and practical and, therefore, is currently the state of the art for solving the HIPP problem.

In addition, the RPoly method is shown to be competitive with other methods in terms of accuracy. RPoly presents an accuracy which is comparable to the best statistical methods in some classes of instances, in particular, in more recent populations which have less diversity in haplotypes. For example, the accuracy of RPoly is comparable with the ones of Beagle and FastPHASE.

Moreover, having a competitive HIPP solver allows to extend the pure parsimony approach with some ideas which contribute for solving similar problems and improving the accuracy of haplotype inference approaches. Following this direction, the second main achievement of this thesis has emerged.

The second contribution of this dissertation is a new Boolean optimization model for haplotype inference in groups of pedigrees. The suggested approach, named minimum recombinant maximum parsimony (MRMP), is based on the combination of two well-known combinatorial approaches: MRHC and HIPP, taking into consideration both pedigree information and population information. Given sets of pedigrees from the same population, the MRMP approach aims at finding a haplotype inference solution which minimizes the number of recombination events within pedigrees and the number of distinct haplotypes within all individuals. The method for MRMP, named PedRPoly, can be viewed as a special case of multi-objective optimization. The use of an appropriate constraint solver and the integration of modeling techniques contribute to a robust haplotype inference method. Experimental results show that the PedRPoly method is competitive in terms of efficiency, and more accurate than the existing methods for haplotype inference in pedigrees.

We consider that the third contribution of this dissertation is to systematize the HIPP approach, contributing to better understanding the problem by providing a vast comparison between all HIPP methods, testing the lower and upper bounds for HIPP, and analyzing the accuracy of the approach. We concluded that the SAT-based techniques are the most adequate for solving the HIPP problem, representing approaches which are considerable more efficient than the remaining HIPP algorithms.

Furthermore, an additional contribution was given, which is mainly related with the use of SAT. First, a number of constraint solvers, including ILP, PBO and MaxSAT solvers have been used. The main conclusion is that the SAT-based PB and weighted MaxSAT solvers are the most suitable for solving the RPoly model. In addition, the models presented in this dissertation provide an interesting new set of challenging Boolean optimization benchmarks, some of which cannot yet be

solved by any PBO and MaxSAT solvers.

A final remark is that computational methods for haplotype inference would still be relevant in case it turned up to be technologically cost-effective to obtain the haplotype data directly. Indeed, although the haplotypes of some individuals may be available, there may exist other individuals whose DNA is unavailable. This is particularly true for pedigrees where the haplotypes of the individuals are strongly related. Algorithms such as PedRPoly are also important for associating consistent haplotypes with unsampled pedigree members.

9.2 Directions of Future Work

The methods proposed in this dissertation have a number of limitations which could be explored in future work.

Genome-wide association studies require a haplotype inference method which could deal with very long blocks of SNPs. This requirement suggests that integrating a partition-ligation procedure in the RPoly and PedRPoly methods would be of interest.

In addition, RPoly could be improved in order to handle polyallelic and polyploid data. SATlotyper, the HIPP method which is able to deal with polyploid species and polyallelic SNPs, is inefficient. Integrating these features in the robust RPoly method would contribute for an efficient polyploid and polyallelic HIPP method. The idea could also be extended to pedigree-based approaches, as PedRPoly.

Moreover, haplotype inference should also be evaluated for genotypes coming from different populations (with a minor overlap) to check whether the inference mechanism is able to distinguish individuals from different populations.

Another future work direction is to integrate additional criteria in the RPoly and PedRPoly models with the goal of further improving the accuracy of the methods. Although some attempts have been done in this direction during this PhD work, some other criteria remain to be evaluated. One hypothesis is to minimize the entropy of the solution which is suggested by the assumption that the most common haplotypes are evenly distributed in the population.

Another interesting idea for future work is to develop a portfolio of haplotype inference algorithms, pursuing the goal of obtaining more accurate solutions. Indeed, as presented in Section 6.4, there is no haplotype inference method which is consistently the most accurate. These results suggest that a portfolio of haplotype inference methods could be of interest. The final solution would be chosen based on the votes of each haplotype inference method, probably giving higher importance

to methods which are believed to have better accuracy.

Regarding the PedRPoly model, a number of questions still remain without an answer. First, it would be interesting to compare the sizes of the sets of solutions for the minimum recombinant maximum parsimony and the minimum recombinant haplotype configuration problems. This work implies developing a dedicated solver which efficiently counts all the solutions. Second, the PedRPoly method should be tested in larger and real instances. In addition, we believe that there is still room for improving the efficiency of PedRPoly, in particular with the advance of better multi-objective SAT-based solvers.

Although a research effort can still be performed to the development of new population-based haplotype inference methods, this subject comes to a saturated point where it becomes extremely difficult to produce new interesting results. However, there are several other computational biology related problems which can be tackled in the future using SAT techniques. These problems include imputation, i.e. inferring unobserved genotypes based on a set of known haplotypes; estimating haplotype frequencies from blocks of consecutive SNPs using pooled DNA; or developing partition-ligation methods in order to deal with long genotypes/haplotypes.

Finally, there are a large number of hard real world problems where SAT-based algorithms can be applied, and the modeling techniques presented in this dissertation could be an inspiration to different models in relevant applications.

Bibliography

[1] G. R. Abecasis, S. S. Cherny, W. O. Cookson, and L. R. Cardon. LR: Merlin-rapid analysis of dense genetic maps using sparse gene flow trees. *Nature Genetics*, 30:97–101, 2002.

[2] T. Achterberg, T. Berthold, T. Koch, and K. Wolter. Constraint integer programming: a new approach to integrate CP and MIP. In *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'08)*, volume 5015 of *LNCS*, pages 6–20, 2008.

[3] F. Aloul, A. Ramadi, I. Markov, and K. Sakallah. Generic ILP versus specialized 0-1 ILP: an update. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD'02)*, pages 450–457, 2002.

[4] A. M. Andrés, A. G. Clark, L. Shimmin, E. Boerwinkle, C. F. Sing, and J. E. Hixson. Understanding the accuracy of statistical haplotype inference with sequence data of known phase. *Genetic Epidemiology*, 31(7):659–671, 2007.

[5] C. Ansótegui, M. L. Bonet, and J. Levy. Solving (weighted) partial MaxSAT through satisfiability testing. In *International Conference on Theory and Applications of Satisfiability Testing (SAT'09)*, volume 5584 of *LNCS*, pages 427–440, 2009.

[6] J. Argelich, C. M. Li, and F. Manyà. An improved exact solver for partial Max-SAT. In *International Conference on Nonconvex Programming: Local & Global Approaches (NCP'07)*, pages 230–231, 2007.

[7] J. Argelich, I. Lynce, and J. Marques-Silva. On solving Boolean multilevel optimization problems. In *International Joint Conference on Artificial Intelligence (IJCAI'09)*, pages 393–398, 2009.

[8] O. Bailleux, Y. Boufkhad, and O. Roussel. A translation of pseudo-Boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:191–200, 2006.

[9] O. Bailleux, Y. Boufkhad, and O. Roussel. New encodings of pseudo-Boolean constraints into CNF. In *International Conference on Theory and Applications of Satisfiability Testing (SAT'09)*, volume 5584 of *LNCS*, pages 181–194, 2009.

[10] R. Bayardo and R. Schrag. Using CSP look-back techniques to solve real world SAT instances. In *AAAI Conference on Artificial Intelligence (AAAI'97)*, pages 203–208, 1997.

[11] D. L. Berre. SAT4J library. www.sat4j.org.

[12] P. Bertolazzi, A. Godi, M. Labbé, and L. Tininini. Solving haplotyping inference parsimony problem using a new basic polynomial formulation. *Computers and Mathematics with Applications*, 55(5):900–911, 2008.

[13] A. Biere. PicoSAT essentials. *Journal on Satisfiability, Boolean Modeling and Computation*, 4:75–97, 2008.

[14] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'99)*, volume 1579 of *LNCS*, pages 193 – 207, 1999.

[15] P. Blain, C. Davis, A. Holder, J. Silva, and C. Vinzant. *Clustering Challenges in Biological Networks*, chapter Diversity graphs, pages 129–150. World Scientific, 2009.

[16] D. Brown and I. Harrower. A new integer programming formulation for the pure parsimony problem in haplotype analysis. In *Workshop on Algorithms in Bioinformatics (WABI'04)*, volume 3240 of *LNCS*, pages 254–265, 2004.

[17] D. Brown and I. Harrower. Integer programming approaches to haplotype inference by pure parsimony. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(2):141–154, 2006.

[18] B. Browning and S. Browning. Haplotypic analysis of wellcome trust case control consortium data. *Human Genetics*, 123(3):273–280, 2008.

[19] S. Browning and B. Browning. Rapid and accurate haplotype phasing and missing data inference for whole genome association studies using localized haplotype clustering. *American Journal of Human Genetics*, 81(5):1084–1097, 2007.

[20] J. T. Burdick, W. Chen, G. R. Abecasis, and V. G. Cheung. In silico method for inferring genotypes in pedigrees. *Nature Genetics*, 38:1002–1004, 2006.

[21] C. Burgtorf, P. Kepper, M. Hoehe, C. Schmitt, R. Reinhardt, H. Lehrach, and S. Sauer. Clone-based systematic haplotyping (CSH): a procedure for physical haplotyping of whole genomes. *Genome Research*, 13(12):2717–2724, 2003.

[22] M. Buro and H. K. Büning. Report on a SAT competition. *Bulletin of the European Association for Theoretical Computer Science*, 49:143–151, 1993.

[23] D. Catanzaro, A. Godi, and M. Labbé. A class representative model for pure parsimony haplotyping. *INFORMS Journal on Computing*, 22(2):195–209, 2010.

[24] D. Catanzaro and M. Labbé. The pure parsimony haplotyping problem: overview and computational advances. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 16:561–584, 2009.

[25] W.-M. Chen and G. R. Abecasis. Family-based association tests for genome wide associations scans. *American Journal of Human Genetics*, 81(5):913–926, 2007.

[26] A. G. Clark. Inference of haplotypes from PCR-amplified samples of diploid populations. *Molecular Biology and Evolution*, 7(2):111–122, 1990.

[27] S. Climer, G. Jäger, A. R. Templeton, and W. Zhang. How frugal is mother nature with haplotypes? *Bioinformatics*, 25(1):68–74, 2009.

[28] S. A. Cook. The complexity of theorem-proving procedures. In *ACM symposium on Theory of computing (STOC'71)*, pages 151–158, 1971.

[29] O. Coudert. On solving covering problems. In *Design Automation Conference (DAC'96)*, pages 197–202, 1996.

[30] D. C. Crawford and D. A. Nickerson. Definition and clinical importance of haplotypes. *Annual Review of Medicine*, 56:303–320, 2005.

[31] M. J. Daly, J. D. Rioux, S. F. Schaffner, T. J. Hudson, and E. S. Lander. High-resolution haplotype structure in the human genome. *Nature Genetics*, 29:229–232, 2001.

[32] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the Association for Computing Machinery*, 5(7):394–397, 1962.

[33] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, 7:201–215, 1960.

[34] O. Delaneau, C. Coulonges, and J. F. Zagury. Shape-IT: new rapid an accurate algorithm for haplotype inference. *Bioinformatics*, 9:540, 2008.

[35] Z. Ding, V. Filkov, and D. Gusfield. A linear-time algorithm for the perfect phylogeny haplotyping (PPH) problem. In *International Conference on Research in Computational Molecular Biology (RECOMB'05)*, pages 585–600, 2005.

[36] C. M. Drysdale, D. W. McGraw, C. B. Stack, J. C. Stephens, R. S. Judson, K. Nandabalan, K. Arnold, G. Ruano, and S. B. Liggett. Complex promoter and coding region β_2-adrenergic receptor haplotypes alter receptor expression and predict in vivo responsiveness. In *National Academy of Sciences*, volume 97, pages 10483–10488, 2000.

[37] N. Eén and N. Sörensson. An extensible SAT-solver. In *International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, volume 2919 of *LNCS*, pages 502–518, 2003.

[38] N. Eén and N. Sörensson. Translating pseudo-Boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–26, 2006.

[39] E. Erdem and F. Ture. Efficient haplotype inference with answer set programming. In *AAAI Conference on Artificial Intelligence (AAAI'08)*, pages 434–441, 2008.

[40] E. Eskin, E. Halperin, and R.M. Karp. Large scale reconstruction of haplotypes from genotype data. In *International Conference on Research in Computational Molecular Biology (RECOMB'03)*, pages 104–113, 2003.

[41] L. Excoffier and M. Slatkin. Maximum likelihood estimation of molecular haplotype frequencies in a diploid population. *Molecular Biology and Evolution*, 12(5):921–927, 1995.

[42] M. Fishelson, N. Dovgolevsky, and D. Geiger. Maximum likelihood haplotyping for general pedigrees. *Human Heredity*, 59(1):41–60, 2005.

[43] Z. Fu and S. Malik. On solving the partial MAX-SAT problem. In *International Conference on Theory and Applications of Satisfiability Testing (SAT'06)*, volume 4121 of *LNCS*, pages 252–265, 2006.

[44] G. Gao, D. B. Allison, and I. Hoeschele. Haplotyping methods for pedigrees. *Human Heredity*, 67(4):248266, 2009.

[45] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[46] L. Gaspero and A. Roli. Stochastic local search for large-scale instances of the haplotype inference problem by pure parsimony. *Journal of Algorithms*, 63(1-3):55–69, 2008.

[47] E. Giunchiglia, Y. Lierler, and M. Maratea. Answer set programming based on propositional satisfiability. *Journal of Automated Reasoning*, 36(4):345–377, 2006.

[48] D. B. Goldstein, S. K. Tate, and S. M. Sisodiya. Pharmacogenetics goes genomic. *Nature Reviews Genetics*, 4(12):937–947, 2003.

[49] R. Gomory. An algorithm for integer solutions to linear programs. In *Recent Advances in Mathematical Programming*, pages 269–302, 1963.

[50] A. Graça, I. Lynce, J. Marques-Silva, and A. Oliveira. Generic ILP vs specialized 0-1 ILP for haplotype inference. In *Workshop on Constraint Based Methods for Bioinformatics (WCB'08)*, 2008.

[51] A. Graça, I. Lynce, J. Marques-Silva, and A. Oliveira. Haplotype inference combining pedigrees and unrelated individuals. In *Workshop on Constraint Based Methods for Bioinformatics (WCB'09)*, 2009.

[52] A. Graça, I. Lynce, J. Marques-Silva, and A. Oliveira. Efficient and accurate haplotype inference by combining parsimony and pedigree information. In *Algebraic Numeric Biology (ANB'10)*, volume 6479 of *LNCS*, 2010.

[53] A. Graça, I. Lynce, J. Marques-Silva, and A. Oliveira. Haplotype inference by pure parsimony: a survey. *Journal of Computational Biology*, 17(8):969–992, 2010.

[54] A. Graça, J. Marques-Silva, and I. Lynce. *Mathematical Approaches to Polymer Sequence Analysis*, chapter Haplotype Inference using Propositional Satisfiability. Springer Verlag, 2011.

[55] A. Graça, J. Marques-Silva, I. Lynce, and A. Oliveira. Efficient haplotype inference with pseudo-Boolean optimization. In *Algebraic Biology (AB'07)*, volume 4545 of *LNCS*, pages 125–139, 2007.

[56] A. Graça, J. Marques-Silva, I. Lynce, and A. Oliveira. Efficient haplotype inference with combined CP and OR techniques. In *International Conference on Integration of AI and OR Tech-*

niques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'08), volume 5015 of *LNCS*, pages 308–312, 2008.

[57] A. Graça, J. Marques-Silva, I. Lynce, and A. Oliveira. Haplotype inference with pseudo-Boolean optimization. *Annals of Operations Research*, 2011. doi: 10.1007/s10479-009-0675-4. In Press.

[58] F. Gudbjartsson, K. Jonasson, M. L. Frigge, and A. Kong. Allegro, a new computer program for multipoint linkage analysis. *Nature Genetics*, 25:12–13, 2000.

[59] D. Gusfield. Efficient algorithms for inferring evolutionary trees. *Networks*, 21:19–28, 1991.

[60] D. Gusfield. Inference of haplotypes from samples of diploid populations: complexity and algorithms. *Journal of Computational Biology*, 8(3):305–324, 2001.

[61] D. Gusfield. Haplotyping as perfect phylogeny: conceptual framework and efficient solutions. In *International Conference on Research in Computational Molecular Biology (RECOMB'02)*, pages 166–175, 2002.

[62] D. Gusfield. Haplotype inference by pure parsimony. In *Symposium on Combinatorial Pattern Matching (CPM'03)*, pages 144–155, 2003.

[63] D. Gusfield and S. H. Orzach. *Handbook on Computational Molecular Biology*, volume 9, chapter Haplotype Inference. CRC Press, 2005.

[64] J. L. Haines. Chromlook: an interactive program for error detection and mapping in reference linkage data. *Genomics*, 14(2):517–519, 1992.

[65] B.V. Halldórsson, V. Bafna, N. Edwards, R. Lippert, S. Yooseph, and S. Istrail. A survey of computational methods for determining haplotypes. In *DIMACS/RECOMB Satellite Workshop on Computational Methods for SNPs and Haplotype Inference*, volume 2983 of *LNCS*, pages 26–47, 2004.

[66] E. Halperin and E. Eskin. Haplotype reconstruction from genotype data using imperfect phylogeny. *Bioinformatics*, 20(12):1842–1849, 2004.

[67] P. L. Hammer and S. Rudeanu. *Boolean Methods in Operations Research and Related Areas*. Springer Verlag, 1968.

[68] P. Hansen and B. Jaumard. Algorithms for the maximum satisfiability problem. *Computing*, 44(4):279–303, 1990.

[69] M. E. Hawley and K. K. Kidd. HAPLO: a program using the EM algorithm to estimate the frequencies of multi-site haplotypes. *Journal Hereditary*, 86(5):409–411, 1995.

[70] F. Heras, J. Larrosa, and A. Oliveras. MiniMaxSat: a new weighted Max-SAT solver. In *International Conference on Theory and Applications of Satisfiability Testing (SAT'07)*, volume 4501 of *LNCS*, pages 41–55, 2007.

[71] F. Heras, J. Larrosa, and A. Oliveras. MiniMaxSat: An efficient weighted Max-SAT solver. *Journal of Artificial Intelligence Research*, 31:1–32, 2008.

[72] Y-T. Huang, K-M. Chao, and T. Chen. An approximation algorithm for haplotype inference by maximum parsimony. *Journal of Computational Biology*, 12(10):1261–1274, 2005.

[73] E. Hubbell. Finding a parsimony solution to haplotype phase is NP-hard. Personal communication to Dan Gusfield, 2000.

[74] R. R. Hudson. Gene genealogies and the coalescent process. *Oxford Survey of Evolutionary Biology*, 7:1–44, 1990.

[75] R. R. Hudson. Generating samples under a Wright-Fisher neutral model of genetic variation. *Bioinformatics*, 18(2):337–338, 2002.

[76] IBM ILOG. Cplex optimizer. www.cplex.com.

[77] F. Ivančić, Z. Yang, M. K. Ganai, A. Gupta, and P. Ashar. Efficient SAT-based bounded model checking for software verification. *Theoretical Computer Science*, 404(3):256–274, 2008.

[78] G. Jäger, S. Climer, and W. Zhang. Complete parsimony haplotype inference problem and algorithms. In *European Symposium on Algorithms (ESA'09)*, volume 5757 of *LNCS*, pages 337–348, 2009.

[79] R. Jeroslow and J. Wang. Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, 1:167–187, 1990.

[80] G. Johnson, L. Esposito, B. Barratt, A. Smith, J. Heward, G. Genova, H. Ueda, H. Cordell, I. Eaves, F. Dudbridge, R. Twells, F. Payne, W. Hughes, S. Nutland, H. Stevens, P. Carr, E. Tuomilehto-Wolf, J. Tuomilehto, S. Gough, D. Clayton, and J. Todd. Haplotype tagging for the identification of common disease genes. *Nature*, 29:233–237, 2001.

[81] H.-Y. Jung, Y.-J. Parkb, Y.-J. Kimb, J.-S. Parkb, K. Kimmb, and I. Koh. New methods for imputation of missing genotype using linkage disequilibrium and haplotype information. *Information Sciences: an International Journal*, 177(3):804–814, 2007.

[82] K. Kalpakis and P. Namjoshi. Haplotype phasing using semidefinite programming. In *IEEE Symposium on Bioinformatics and Bioengineering (BIBE'05)*, pages 145–152, 2005.

[83] L. V. Kantorovich. Mathematical methods of organizing and planning production. *Management Science*, 6(4):366–422, 1960. English translation of the article originally published in Russian, in 1939.

[84] E. Kelly, F. Sievers, and R. McManus. Haplotype frequency estimation error analysis in the presence of missing genotype data. *BMC Bioinformatics*, 5:188, 2004.

[85] B. Kerem, J. Rommens, J. Buchanan, D. Markiewicz, T. Cox, A. Chakravarti, M. Buchwald, and L. C. Tsui. Identification of the cystic fibrosis gene: Genetic analysis. *Science*, 245:1073–1080, 1989.

[86] L. G. Khachiyan. Polynomial algorithm in linear programming. *U.R.S.S. Computational Mathematics and Mathematical Physics*, 20:53–72, 1980.

[87] M. Kimura. The number of heterozygous nucleotide sites maintained in a finite population due to steady flux of mutations. *Genetics*, 61:893–903, 1969.

[88] B. Kirkpatrick, E. Halperin, and R. M. Karp. Haplotype inference in complex pedigrees. *Journal of Computational Biology*, 17(3):269–280, 2010.

[89] D. L. Kroetz, C. Pauli-Magnus, L. M. Hodges, C. C. Huang, M. Kawamoto, S. J. Johns, D. Stryke, T. E. Ferrin, J. DeYoung, T. Taylor, E. J. Carlson, I. Herskowitz, K. M. Giacomini, and A. G. Clark. Sequence diversity and haplotype structure in the human ABCD1 (MDR1, multidrug resistance transporter). *Pharmacogenetics*, 13:481–494, 2003.

[90] L. Kruglyak, M. J. Daly, M. P. Reeve-Daly, and E. S. Lander. Parametric and nonparametric linkage analysis: a unified multipoint approach. *American Journal of Human Genetics*, 58(6):1347–1363, 1996.

[91] L. Kruglyak and E. S. Lander. Faster multipoint linkage analysis using fourier transforms. *Journal of Computational Biology*, 5(1):1–7, 1998.

[92] G. Lancia, C. M. Pinotti, and R. Rizzi. Haplotyping populations by pure parsimony: complexity of exact and approximation algorithms. *INFORMS Journal on Computing*, 16(4):348–359, 2004.

[93] G. Lancia and R. Rizzi. A polynomial case of the parsimony haplotyping problem. *Operation Research Letters*, 34(3):289–295, 2006.

[94] G. Lancia and P. Serafini. A set-covering approach with column generation for parsimony haplotyping. *INFORMS Journal on Computing*, 21(1):151–166, 2009.

[95] S. M. Leal, K. Yan, and B. Müller-Myhsok. SimPed: A simulation program to generate haplotype and genotype data for pedigree structures. *Human Heredity*, 60(2):119–122, 2005.

[96] C. M. Li, F. Manyà, N. O. Mohamedou, and J. Planes. Exploiting cycle structures in Max-SAT. In *International Conference on Theory and Applications of Satisfiability Testing (SAT'09)*, volume 5584 of *LNCS*, pages 467–480, 2009.

[97] C. M. Li, F. Manyà, and J. Planes. Exploiting unit propagation to compute lower bounds in branch and bound Max-SAT solvers. In *International Conference on Principles and Practice of Constraint Programming (CP'05)*, volume 3709 of *LNCS*, pages 403–414, 2005.

[98] C. M. Li, F. Manyà, and J. Planes. New inference rules for Max-SAT. *Journal of Artificial Intelligence Research*, 30:321–359, 2007.

[99] J. Li and T. Jiang. Efficient inference of haplotypes from genotypes on a pedigree. *Journal of Bioinformatics and Computational Biology*, 1(1):41–69, 2003.

[100] J. Li and T. Jiang. Computing the minimum recombinant haplotype configuration from incomplete genotype data on a pedigree by integer linear programming. *Journal of Computational Biology*, 12(6):719–739, 2005.

[101] X. Li and J. Li. Comparison of haplotyping methods using families and unrelated individuals on simulated rheumatoid arthritis data. In *BMC Proceedings*, pages S1–S55, 2006.

[102] X. Li and J. Li. Efficient haplotype inference from pedigree with missing data using linear systems with disjoint-set data structures. In *International Conference on Computational Systems Bioinformatics (CSB'08)*, pages 297–307, 2008.

[103] X. Li and J. Li. Efficient haplotype inference from pedigrees with missing data using linear systems with disjoint-set data structures. In *Conference on Computational Systems Biology (ISB'08)*, page 297308, 2008.

[104] X. Li and J. Li. An almost linear time algorithm for a general haplotype solution on tree pedigrees with no recombination and its extensions. *Journal of Bioinformatics and Computational Biology*, 3(7):521545, 2009.

[105] Z. Li, W. Zhou, X-S. Zhang, and L. Chen. A parsimonious tree-grow method for haplotype inference. *Bioinformatics*, 21(17):3475–3481, 2005.

[106] H. Lin, K. Su, and C. M. Li. Within-problem learning for efficient lower bound computation in Max-SAT solving. In *AAAI Conference on Artificial Intelligence (AAAI'06)*, pages 351–356, 2008.

[107] S. Lin, A. Chakravarti, and D. J. Cutler. Haplotype and missing data inference in nuclear families. *Genome Research*, 14(8):1624–1632, 2004.

[108] L. Liu, X. Chen, J. Xiao, and T. Jiang. Complexity and approximation of the minimum recombination haplotype configuration problem. In *International Symposium on Algorithms and Computation*, pages 370–379, 2005.

[109] L. Liu and T. Jiang. Linear-time reconstruction of zero-recombinant Mendelian inheritance on pedigrees without mating loops. In *Genome Informatics Workshop (GIW'07)*, pages 95–106, 2007.

[110] L. Liu, C. Xi, J. Xiao, and T. Jiang. Complexity and approximation of the minimum recombinant haplotype configuration problem. *Theoretical Computer Science*, 378(3):316–330, 2007.

[111] Y. Liu and C.-Q. Zhang. A linear solution for haplotype perfect phylogeny problem. In *International Conference on Bioinformatics and its Applications (ICBA)*, 2004.

[112] K. E. Lohmueller, C. D. Bustamante, and A. G. Clark. Methods for human demographic inference using haplotype patterns from genomewide single-nucleotide polymorphism data. *Genetics*, 182:217–231, 2009.

[113] I. Lynce, A. Graça, J. Marques-Silva, and A. Oliveira. Haplotype inference with Boolean constraint solving: an overview. In *IEEE International Conference on Tools with Artificial Intelligence (ICTAI'08)*, volume I, pages 92–100. IEEE Computer Society, 2008.

[114] I. Lynce and J. Marques-Silva. Efficient haplotype inference with Boolean satisfiability. In *AAAI Conference on Artificial Intelligence (AAAI'06)*, pages 104–109, 2006.

[115] I. Lynce and J. Marques-Silva. SAT in bioinformatics: Making the case with haplotype inference. In *International Conference on Theory and Applications of Satisfiability Testing (SAT'06)*, number 4121 in LNCS, pages 136–141, 2006.

[116] I. Lynce and J. Marques-Silva. Haplotype inference with Boolean satisfiability. *International Journal on Artificial Intelligence Tools*, 17(2):355–387, 2008.

[117] I. Lynce, J. Marques-Silva, and S. Prestwich. Boosting haplotype inference with local search. *Constraints*, 13(1):155–179, 2008.

[118] F. Chin S. Fung M. Kao M. Y. Chan, W. Chan. Linear-time haplotype inference on pedigrees without recombinations. In *Workshop on Algorithms in Bioinformatics (WABI'06)*, pages 56–67, 2006.

[119] V. Manquinho and J. Marques-Silva. Effective lower bounding techniques for pseudo-Boolean optimization. In *Design, Automation and Test in Europe Conference (DATE'05)*, pages 660–665, 2005.

[120] V. Manquinho, J. Marques-Silva, and J. Planes. Algorithms for weighted Boolean optimization. In *International Conference on Theory and Applications of Satisfiability Testing (SAT'09)*, volume 5584 of *LNCS*, pages 495–508, 2009.

[121] J. Marchini, D. Cutler, N. Patterson, M. Stephens, E. Eskin, E. Halperin, S. Lin, Z. Qin, H. Munro, G. Abecassis, P. Donnelly, and International HapMap Consortium. A comparison of phasing algorithms for trios and unrelated individuals. *American Journal of Human Genetics*, 78(3):437–450, 2006.

[122] J. Marques-Silva. The impact of branching heuristic in propositional satisfiability algorithms. In *Portuguese Conference on Artificial Intelligence (EPIA'99)*, volume 1695 of *LNAI*, pages 62–74, 1999.

[123] J. Marques-Silva. Boolean satisfiability in electronic design automation. In *IEEE/ACM Design Automation Conference (DAC'00)*, pages 675–680, 2000.

[124] J. Marques-Silva, J. Argelich, A. Graça, and I. Lynce. Boolean lexicographic optimization. In *International Workshop on Experimental evaluation of algorithms for solving problems with combinatorial explosion (RCRA'10)*, 2010.

[125] J. Marques-Silva, I. Lynce, A. Graça, and A. Oliveira. Efficient and tight upper bounds for haplotype inference by pure parsimony using delayed haplotype selection. In *Portuguese Conference on Artificial Intelligence (EPIA'07)*, volume 4874 of *LNAI*, pages 621–632, 2007.

[126] J. Marques-Silva and J. Planes. Algorithms for maximum satisfiability using unsatisfiable cores. In *Design, Automation and Test in Europe Conference (DATE'08)*, pages 408–413, 2008.

[127] J. Marques-Silva and K. Sakallah. GRASP: A new search algorithm for satisfiability. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD'96)*, 1996.

[128] J. McCluskey and C. A. Peh. The human leucocyte antigens and clinical medicine: an overview. *Reviews in Immunogenetics*, 1(1):3–20, 1999.

[129] M. Moskewics, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Design Automation Conference (DAC'01)*, pages 530–535, 2001.

[130] J. Neigenfind, G. Gyetvai, R. Basekow, S. Diehl, U. Achenbach, C. Gebhardt, J. Selbig, and B. Kersten. Haplotype inference from unphased SNP data in heterozygous polyploids based on SAT. *BMC Genomics*, 9:356, 2008.

[131] T. Niu, Z. Qin, X. Xu, and J. Liu. Bayesian haplotype inference for multiple linked single-nucleotide polymorphisms. *American Journal of Human Genetics*, 70:157–169, 2002.

[132] J. R. O'Connell. Zero-recombinant haplotyping: applications to fine mapping using SNPs. *Genetic Epidemiology*, 19(1):S64–S70, 2000.

[133] S. H. Orzack, D. Gusfield, J. Olson, S. Nesbitt, L. Subrahmanyan, and Jr. V. P. Stanton. Analysis and exploration of the use of rule-based algorithms and consensus methods for the inferral of haplotypes. *Genetics*, 165:915–928, 2003.

[134] X. Pan. Haplotype inference by pure parsimony with constraint programming. Master Thesis in Information Technology. Faculty of Science and Technology. Uppsala Universitet. Sweden, 2009.

[135] C. H. Papadimitriou. On the complexity of integer programming. *Journal of the Association for Computing Machinery*, 28(4):765 – 768, 1981.

[136] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991.

[137] K. Pipatsrisawat, A. Palyan, M. Chavira, A. Choi, and A. Darwiche. Solving weighted Max-SAT problems in a reduced search space: A performance analysis. *Journal on Satisfiability Boolean Modeling and Computation*, 4:191–217, 2008.

[138] D. Qian and L. Beckmann. Minimum-recombinant haplotyping in pedigrees. *American Journal of Human Genetics*, 70(6):1434–1445, 2002.

[139] M. Ramírez and H. Geffner. Structural relaxations by variable renaming and their compilation for solving MinCostSAT. In *International Conference on Principles and Practice of Constraint Programming*, pages 605–619, 2007.

[140] M. J. Rieder, S. T. Taylor, A. G. Clark, and D. A. Nickerson. Sequence variation in the human angiotensin converting enzyme. *Nature Genetics*, 22:481–494, 2001.

[141] M. Sánchez, S. Givry, and T. Schiex. Mendelian error detection in complex pedigrees using weighted constraint satisfaction techniques. *Constraints*, 13(1-2):130–154, 2008.

[142] S. F. Schaffner, C. Foo, S. Gabriel, D. Reich, M. J. Daly, and D. Altshuler. Calibrating a coalescent simulation of human genome sequence variation. *Genome Research*, 15:1576–1583, 2005.

[143] P. Scheet and M. Stephens. A fast and flexible statistical model for large-scale population genotype data: applications to inferring missing genotypes and haplotypic phase. *American Journal of Human Genetics*, 78:629–644, 2006.

[144] R. Sharan, B. V. Halldórsson, and S. Istrail. Islands of tractability for parsimony haplotyping. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(3):303–311, 2006.

[145] H. M. Sheini and K. A. Sakallah. Pueblo: A hybrid pseudo-Boolean SAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:165–189, 2006.

[146] S. T. Sherry, M. H. Ward, M. Kholodov, J. Baker, L. Phan, E. M. Smigielski, and K. Sirotkin. dbSNP: the NCBI database of genetic variation. *Nucleic Acids Research*, 29:308–311, 2001.

[147] P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1-2):181–234, 2002.

[148] E. Sobel and K. Lange. Descent graphs in pedigree analysis: applications to haplotyping, location scores, and marker-sharing statistics. *American Journal of Human Genetics*, 58(6):1323–1337, 1996.

[149] M. Stephens and P. Donelly. Inference in molecular genetics. *Journal of the Royal Statistical Society: Series B*, 62:605–655, 2000.

[150] M. Stephens and P. Donnelly. A comparison of Bayesian methods for haplotype reconstruction from population genotype data. *American Journal of Human Genetics*, 165(4):2213–2233, 2003.

[151] M. Stephens, N. Smith, and P. Donelly. A new statistical method for haplotype reconstruction. *American Journal of Human Genetics*, 68:978–989, 2001.

[152] D. M. Strickland, E. Barnes, and J. S. Sokol. Optimal protein structure alignment using maximum cliques. *Operations Research*, 53(3):389–402, 2005.

[153] The International HapMap Consortium. The international hapmap project. *Nature*, 426:789–796, 2003.

[154] The International HapMap Consortium. A haplotype map of the human genome. *Nature*, 437:1299–1320, 2005.

[155] The International HapMap Consortium. A second generation human haplotype map over 3.1 million SNPs. *Nature*, 449:851–861, 2007.

[156] L. Tininini, P. Bertolazzi, A. Godi, and G. Lancia. CollHaps: A heuristic approach to haplotype inference by parsimony. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2008.

[157] G. S. Tseitin. On the complexity of derivation in propositional calculus. *Studies in Constructive Mathematics and Mathematical Logic, Part II*, pages 115–125, 1968.

[158] G. Lancia V. Bafna, D. Gusfield and S. Yooseph. Haplotyping as perfect phylogeny: a direct approach. Technical Report CSE-2002-21, Department of Computer Science, The University of California at Davis, 2002.

[159] M. Vasquez and J. Hao. A logic-constrained knapsack formulation and a tabu algorithm for the daily photograph scheduling of an earth observation satellite. *Journal of Computational Optimization and Applications*, 20(2):137–157, 2001.

[160] R. VijayaSatya and A. Mukherjee. An efficient algorithm for perfect phylogeny haplotyping. In *IEEE Computational Systems Bioinformatics Conference (CSB'05)*, pages 103–110, 2005.

[161] R. VijayaSatya and A. Mukherjee. An optimal algorithm for perfect phylogeny haplotyping. *Journal of Computational Biology*, 13:897–928, 2006.

[162] T. Villa, T. Kam, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Explicit and implicit algorithms for binate covering problems. In *IEEE Transactions on Computer-Aided Design*, volume 16, pages 677–691, 1997.

[163] I.-L. Wang and H.-E.Yang. Computational experiments on algorithms for haplotype inference problems by pure parsimony. In *Joint Conference on Information Sciences (JCIS'06)*, pages 824–827, 2006.

[164] L. Wang and Y. Xu. Haplotype inference by maximum parsimony. *Bioinformatics*, 19(14):1773–1780, 2003.

[165] R.-S. Wang, X.-S. Zhang, and L. Sheng. Haplotype inference by pure parsimony via genetic algorithm. In *International Symposium Operations Research and Its Applications (ISORA'05)*, volume 5 of *LNOR*, pages 308–318, 2005.

[166] D. E. Weeks, E. Sobel, J. R. O'Connell, and K. Lange. Computer programs for multilocus haplotyping of general pedigrees. *American Journal of Human Genetics*, 56(6):1506–1507, 1995.

[167] E. M. Wijsman. A deductive method of haplotype analysis in pedigrees. *American Journal of Human Genetics*, 41(3):356–373, 1987.

[168] 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 *ACM-SIAM Symoposium on Discrete Algorithms (SODA'07)*, pages 655–664, 2007.

[169] E. P. Xing, M. I. Jordan, and R. Sharan. Bayesian haplotype inference via the dirichlet process. *Journal of Computational Biology*, 14(3):267–284, 2007.

[170] H. Xu, R. A. Rutenbar, and K. Sakallah. sub-SAT: A formulation for related Boolean satisfiability with applications in routing. In *International Symposium on Physical Design (ISPD'02)*, volume 22, pages 814–820, 2002.

[171] R. Zabih and D. McAllester. A rearrangement search strategy for determining propositional satisfiability. In *AAAI Conference on Artificial Intelligence (AAAI'88)*, pages 155–160, 1988.

[172] K. Zhang, Z. Qin, T. Chen, J. S. Liu, M. S. Waterman, and F. Sun. HapBlock: haplotype block partitioning and tag SNP selection software using a set of dynamic programming algorithms. *Bioinformatics*, 21(1):131–134, 2005.

[173] K. Zhang, F. Sun, and H. Zhao. HAPLORE: a program for haplotype reconstruction in general pedigrees without recombination. *Bioinformatics*, 21(1):90–103, 2005.