Universidade Técnica de Lisboa

Instituto Superior Técnico

Departamento de Matemática

Haplotype Inference by Pure Parsimony using Pseudo-Boolean Optimization

Ana Sofia Graça

Diploma Thesis

Applied Mathematics and Computation

Supervisor:

Prof. Inês Lynce (INESC-ID: SAT)

Co-Supervisors:

Prof. Arlindo Oliveira (INESC-ID: ALGOS/KD-BIO)

Prof. Amílcar Sernadas (IST: DM)

September 2006

Abstract

In the field of human genomics, the identification of genetic variations among human beings has emerged as a critical issue. Identifying those variants represents an important step towards improving prevention, diagnosis and treatment of disease.

Single Nucleotide Polymorphisms are the most common variations between human beings. Hence, building a full haplotype map (which identify SNPs) of the human genome has become an important goal in genomics. Since, in practice, genotype data rather than haplotype data is available, a key computational problem has been the inference of haplotypes from genotypes.

Some studies have shown that Haplotype Inference by Pure Parsimony (HIPP), i.e. trying to minimize the number of required haplotypes to explain a given set of genotypes, is an accurate approach to the problem. Several authors have studied the HIPP problem. In particular, some integer programming (IP) approaches have been developed. In this work, these IP formulations have been modified in order to be solved using pseudo-boolean ILP solvers. Experimental results have shown that PBO approaches are not only much more efficient than theirs IP counterparts, but also competitive with SAT-based models that represent the state of the art.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank my supervisors, Professor Inês Lynce and Professor Arlindo Oliveira, for all their help and guidance with this work, for all the time they disposed with this project and most of all for giving me the valuable opportunity of really doing research.

I also thank Professor Amílcar Sernadas for his orientation at the beginning of my final project, for all his patience with my academic indecisions and for having suggested this topic for my diploma thesis.

I must thank Ana Casimiro, António Morgado, Paulo Matos, Ruben Martins and Emanuel Santos for all their advices, help with Latex and other software tools in which I felt really lost.

At last but not at least I would like to thank my family and friends for all their support and constant motivation.

Chapter 1

Introduction

Genetic differences between individuals have been an important object of study. For genes associated with diseases, understanding human genetic variations can revolutionize medicine with an early detection and treatment of illnesses, such as diabetes, cancer, stroke, heart disease, depression and asthma.

Single Nucleotide Polymorphisms (SNPs), sites in the genome where DNA sequences of many individuals differ by a single base, are the most common variations between human beings. Hence, a fundamental task is to identify and study haplotypes, the set of SNPs found to be statistically associated. However, due to technological limitations, haplotypes are difficult to obtain. For a DNA region, each person has two haplotypes, each one inherited from one parent. The conflated data of these two inherited haplotypes is called genotype and is, in practice, simpler to obtain than the haplotypes themselves.

Haplotype Inference (HI), the process of going from genotypes to haplotypes, is the object of this work. In particular, we focus in the *Pure Parsimony* approach to the HI problem (HIPP), under the assumption that the number of haplotypes in a population is much smaller than the number of possible genotypes.

Several authors have developed work using the Haplotype Inference by Pure Parsimony approach. In 2003, Gusfield [11] proposed an integer linear programming (IP) formulation called RTIP. However, this formulation grows exponentially on the size of the problem. Although it solves very spidly small-sized instances, due to memory limitations, it is incapable of solving real problem instances, which could be very large. Lancia et al. [17] proved that the HIPP problem is APX-hard and proposed a polynomial-sized integer linear program for the HIPP problem, but did not test it. Still in 2003, a branch and bound algorithm, Hapar, was proposed by Wang and Lu [31] . In 2004, Brown and Harrower [2] proposed a new integer linear programming formulation, PolyIP, that is polynomial on the size of the instance. Also Brown and Harrower [3], in 2006, suggested an integer programming which is a hybrid between RTIP and PolyIP and so called HybridIP. Recently, in 2006, a very competitive SAT-based approach, proposed by Lynce and Marques-Silva, named SHIPs [20], has been created. In this project, we emerged with a new idea to solve the HIPP problem, which shows itself to be very competitive and robust.

1.1 Contributions

In this thesis, three new approaches to Haplotype Inference by Pure Parsimony are introduced. Based on existing integer linear programming formulations (RTIP, PolyIP and HybridIP), we formulate three Pseudo-Boolean Optimization (PBO) models to the HIPP problem (RTPB, PolyPB and HybridPB). The major innovation here, is that, instead of using IP tools to solve the problem, we apply modern Pseudo-Boolean solvers. The most used solver (MINISAT+) translate problem instances into SAT and then solve them.

The goal of this project is twofold. The first is to improve the efficiency of RTIP, PolyIP and HybridIP. The second is to verify whether the success of SHIPs is due to the use of SAT techniques or to the formulation itself. The first target was reached with success: although the PBO solvers used seem to have some memory problems with the exponential-sized RTIP, the pseudo-boolean approach is more appropriate to solve at least Poly and Hybrid formulations. The efficiency of these two last models is remarkable. Not only can we improve the required resolution time but also are we able to solve much more instances in a reasonable time. PBO models were also tested against the branch-and-bound approach, Hapar. Again, we show that our models are more efficient. Finally, PolyPB and HybridPB are compared with SHIPs. Experimental results show that PBO approaches are competitive with SHIPs. In fact, although PolyPB and HybridPB need more time to solve most part of the instances, PBO approaches are able to solve a larger set of hard instances in a reasonable time. This means that PBO models are more robust than SHIPs. As a result, we conclude that SAT-based modern solvers fit very well the resolution of the HIPP problem. In particular, PBO approaches are promising, pointing for the need for further research in this field.

This final degree project involves a deep understanding of the Haplotype Inference problem, and, in particular, of the Pure Parsimony approach. In order to understand state-of-the-art formulations to the problem, several optimization methods and solvers have been studied. A number of techniques are directly relevant. Among these, Integer Programming, Satisfiability and, more deeply, Pseudo-Boolean Optimization have been explored. The three IP formulations were implemented in order to be applied to PBO solvers. Finally, all experimental results, data treatment, analysis and comparison of the PBO approaches against the existing solvers were obtained during the course of this project.

1.2 Organization

This thesis compiles previous work about the Haplotype Inference by Pure Parsimony (HIPP) and introduces an innovation which gives very competitive results.

In Chapter 2 we make an overview of celular biology, introduce genetic polymorphisms, haplotypes and, finally, explain the Haplotype Inference (HI) problem. Chapter 3 abstracts from biology, as much as possible, to explain the mathematical formulation of the HI problem and the algorithmic models proposed to solve the problem. In particular, we focus on the HIPP formulation and give some important mathematical results of the problem.

HIPP formulations presented in this project use some techniques of discrete optimization which represent rich fields of research because of their importance in several contexts. Hence, Chapter 4 is a brief overview of Integer Linear Programming (IP), Boolean Satisfiability (SAT) and, a hybrid between these, Pseudo-Boolean Optimization (PBO). We also present briefly some of the software packages used to solve these problems.

In Chapter 5 we describe all existent Haplotype Inference formulations using the Pure Parsimony criterion. We review three IP models: RTIP, PolyIP and HybridIP. We also describe the branch and bound algorithm, HAPAR, and the SAT-based approach, SHIPs. Finally, we introduce the three new PBO approaches based on the IP models.

Finally, in Chapter 6 we conduct an evaluation of the PBO, IP and SAT approaches. Experiments show that, the use of PBO in this problem, represents a great improvement over IP models and, moreover, is also competitive against the SAT-based approach SHIPs.

Chapter 2

Biological Problem

2.1 DNA

DNA was discovered in 1869 by Johann Miescher when he isolated the substance from the nucleus of white blood cells. However, for a long time, little attention was paid to DNA; biologists did not believe that genes could be hidden in such a simple molecule. DNA, *deoxyribonucleic acid*, is a molecule consisting of a sugar, a phosfate group and one of four nitrogenous bases: A(adenine), T(thymine), G(guanine), or C(cytosine). Only in 1944, Oswald Avery [1] and colleagues proved that indeed DNA is the stand of genetic information and the interest in this molecule grew. DNA has a double-helical structure as conjectured in 1953 by Watson and Crick [32]. DNA has the autoreplication property and, furthermore, DNA is responsible for the protein synthesis.

The genetic information of a human being define his traits, such as hair color, eye color, susceptibility to diseases. These traits are caused by variations in genes. Despite the great similarity between our genes, no two individuals are quite the same. The human genome has, roughly, 3 billion nucleotides but about $99,9\%$ of them are equal between all human beings. However, this still leaves room to $4^{3.000.000}$ different genomes, which is enough for a great diversity. In particular, these differences explain the genetic variations that influence how people differ in their risk of disease or their response to drugs.

2.2 Genetic Polymorphisms

We call genetic polymorphism to a difference in DNA sequence among individuals, groups or populations. Single Nucleotide Polymorphisms, sequence repeats, insertions, deletions and recombinations are kinds of polymorphisms in genes and give rise to the different human traits. Genetic polymorphisms may be the result of chance processes, or may have been induced by external agents, such as radiations or viruses. In particular, a genetic mutation is a polymorphism which is not present in most individuals and either has been associated with diseases or has resulted from external agents.

2.2.1 SNPs

In this work, we will focus on *single nucleotide polymorphisms*, SNPs ("snips"), which are single nucleotide sites of the human genome that show a significant variability within a population. About 90% of all human polymorphisms are of this simple form.

There are two different types of nucleotide mutations resulting in SNPs: a **transition** substitution which occurs between purines (A, G) (see Figure 2.1) or between pyrimidines (C, T); and a **transversion** substitution between a purine and a pyrimidine.

Figure 2.1: A part of two chromosomes showing a SNP, result of a transition substitution. Both the A and G alleles are shown.

The probability of a given nucleotide site to be a SNP is approximately 1/1350 (see Figure 2.2). However, SNPs are not uniformly distributed over the entire human genome, neither over all chromosomes and neither within a single chromosome: coding regions have three times more SNPs than non-coding regions; sex chromosomes have much lower variations; within a single chromosome, SNPs can be concentrated about a specific region, usually implying a region of medical or research interest.

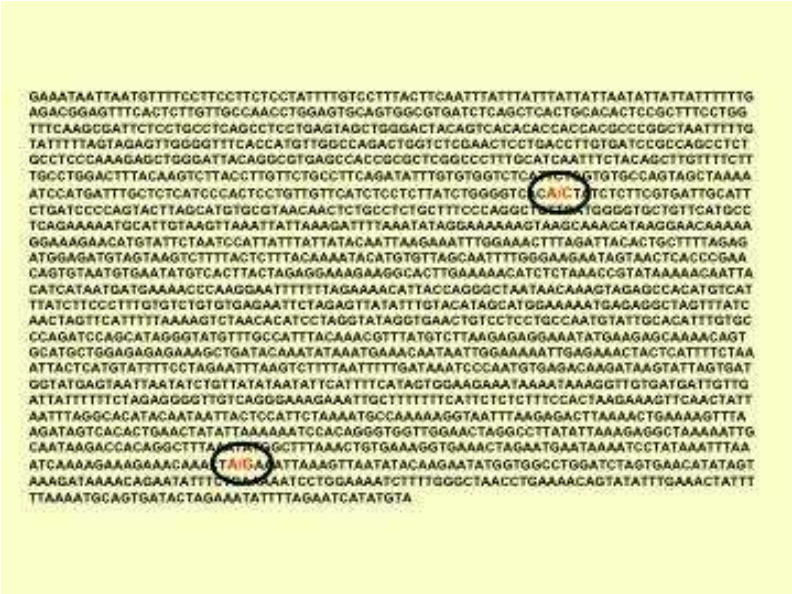

Figure 2.2: When DNA sequences on a part of chromosome 7 from two random individuals are compared, then two single nucleotide polymorphisms (SNPs) occur in about 2.200 nucleotides

A SNP in a coding region may be classified into two classes depending on the different effects on the resulting protein. A **synonymous** is a silent mutation where the exchange of the aminoacid does not change the protein it produces; this happens due to redundancy in the genetic code. On the other hand, a **non-synonymous** substitution results in an

alteration of the encoded aminoacid. A missense mutation changes the protein by causing a change of codon; a nonsense mutation results in a misplaced termination codon. One half of all coding sequence SNPs result in non-synonymous codon changes.

As long as genomic variation is responsible for the diversity in the human species, the study of SNPs is of great importance in discovering genes responsible for genetic diseases and helping to prevent them. There are, already, over one million SNPs identified using some biochemical reactions that isolate the precise location of a suspected SNP and determine its identity, with the help of the DNA polymerase enzyme.

2.3 Haplotypes

There is some ambiguity in the definition of a haplotype. A haplotype can be the genetic constitution of an individual chromosome, can refer to only one locus or to the entire genome. Or, by the definition preferred in this work, a haplotype is the set of single nucleotide polymorphisms found to be statistically associated on a single chromatin and that tend to be inherit together, i.e. not easily separable by recombination. Nonetheless, by the context, it is easy to understand the corresponding definition, since they are closely related.

Table 2.1 gives twelve haplotypes (sets of SNPs) of the β_2AR genes, a real example from the β_2-Adrenergic receptors gene [6]. The nucleotide number refers to the position of the site, relative to the first nucleotide of the starting codon (position 1). The allele represents the two nucleotide possibilities at each SNP site: the first nucleotide is the wild type, while the second one is the mutant.

Table 2.1: Haplotypes of the β_2AR genes.

Nucleotide	-1023	-709	-654	-468	-406	-367	-47	-20	46	79	252	491	523
Alleles	G/A	C/A	G/A	C/G	C/T	T/C	T/C	T/C	G/A	C/G	G/A	C/T	C/A
h_1	A	C	G	C	C	T	T	T	A	C	G	C	C
h_2	A	C	G	G	C	C	C	C	G	G	G	C	C
h_3	G	A	A	C	C	T	T	T	A	C	G	C	C
h_4	G	C	A	C	C	T	T	T	A	C	G	C	C
h_5	G	C	A	C	C	T	T	T	G	C	G	C	C
h_6	G	C	G	C	C	T	T	T	G	C	A	C	A
h_7	G	C	G	C	C	T	T	T	G	C	A	T	A
h_8	G	C	A	C	C	T	T	T	A	C	A	C	A
h_9	A	C	G	C	T	T	T	T	A	C	G	C	C
h_{10}	G	C	G	C	C	T	T	T	G	C	A	C	C
h_{11}	G	C	G	C	C	T	T	T	G	C	G	C	C
h_{12}	A	C	G	G	C	T	T	T	A	C	G	C	C

2.3.1 The Origins of Haplotypes

Humans are *diploid* organisms, i.e. our genome contains pairs of chromosomes, with one inherited from the father and the other inherited from the mother. However, chromosomes

do not pass from generation to generation as identical copies. Rather, chromosomes pairs suffer a process of *recombination*. This means that the two chromosomes of a pair exchange pieces to build another chromosome which is passed on to the next generation. Although segments of chromosomes are recombined over the course of many generations, there are DNA sequences which are shared by multiple individuals (see Figure 2.3). These segments which have not been broken up by recombination are the haplotypes.

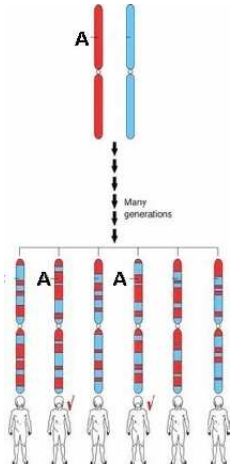

Figure 2.3: This diagram shows two ancestral chromosomes being recombined over many generations to yield new chromosomes. If the genetic variant marked by A increases the risk of a certain disease, the two individuals who have inherit this variant are in risk. Adjacent to the variant A, at the same haplotype, there are many SNPs that can be used to identify the location of the variant.

Genetic and archaeological evidences indicate that humans are descendant of ancestors who lived in Africa about 150.000 years ago. As we are a relatively young species, most of the variation on any current human population comes from the ancestral human population. As humans migrated out of Africa, they carried with them most but not all of the genetic variation that existed in the ancestral population. Because recombination has occured more times in Africa than outside Africa, African haplotypes tend to be longer than the other haplotypes. As humans spread all over the world, random chance, natural selection and other genetic mechanisms have given rise to different frequencies of haplotypes in different populations, mostly in those that are widely separated and unlikely to exchange DNA through mating. On the other hand, mutations have created new haplotypes which have not had enough time to spread widely beyond the population, and so appear more frequently in the originating region. Hence, isolated populations, like some Indian tribes, can represent an important object of study.

2.3.2 Linkage Disequilibrium

Haplotypes may contain alleles in **Linkage Disequilibrium** (LD), i.e. it may exist a non-random association between alleles at different loci, not necessarily on the same chromosome. LD describes a situation in which some combinations of alleles or genetic markers occur more or less frequently in a population than would be expected from a

random formation of haplotypes based on alleles frequencies. More exactly, we have LD when there are dependence between allele frequencies at separate loci. In the extreme case, an allele found at one locus predicts which allele will be found at the other. In fact, a chromosome region may contain many SNPs, but only a few "tag" SNPs are sufficient to identify (tag) each of the haplotypes in question (see Figure 2.4). These SNPs suffice to provide most of the information on the pattern of genetic variation in the region.

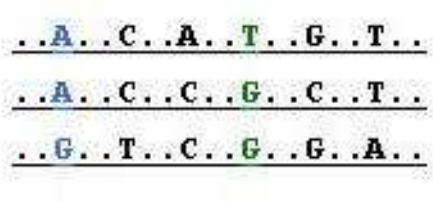

Figure 2.4: A chromosome region with only the SNPs shown. Three haplotypes are shown. The two SNPs in color are sufficient to identify (tag) each of the three haplotypes. For example, if a chromosome has alleles A and T at these two tag SNPs, then it has the first haplotype.

With increasing meiotic events, recombination between loci should lead alleles to equilibrium. However, this will take longer for alleles at the same haplotype, due to reduced recombination.

2.4 Haplotype Inference

The Human genome is constituted by 23 pairs of chromosomes, with one element of each pair inherited from each parent. The conflated data of both chromosomes on a pair is the genotype, while the genetic information of a single chromosome is the haplotype. This means that the genotype is the mixed data of two haplotypes. A genotype is not always equal to the respective haplotypes due to SNPs. A locus is homozygous (or non-ambiguous) if the two haplotypes have the same allele at this site and so does the genotype. However, if a locus is heterozygous, then the two haplotypes are different at this position and the genotype is ambiguous.

To understand the genetic contribution to diseases and their origins, it is often more informative to have haplotype information rather than genotype data. However, it is not easy to examine separately copies of chromosomes, i.e. to get the haplotype data, since only genotype information is obtained when DNA is analyzed using currently available techniques. The challenge is to infer haplotype data from genotype data. This process is called haplotype inference.

Usually, and from now on in this thesis, we will consider that the underlying data that forms a haplotype is reduced to the single nucleotide polymorphisms in the region. For a SNP, the minority of the population has one nucleotide at this site (the least common allele) while the majority of the population has another nucleotide (the most frequent allele). Rarely occur more than two nucleotides in one site and this case will not be considered in this work. This simplification is justified by the "infinite sites" hypotesis, analysed in Section 2.5.

2.4.1 International HapMap Project

The International HapMap Project (`www.hapmap.org`) results from a collaboration among scientists from Japan, UK, Canada, China, Nigeria and USA, with the goal of developing a haplotype map of the human genome, the HapMap. This Project, which started in 2002, has the purpose of identifying which 200.000 to 1 million tag SNPs (see Figure 2.5) provides almost as much mapping information as the total 10 million SNPs. This substantial reduction of the problem, will help researchers finding the haplotypes responsible for diseases [30].

Although the majority of common haplotypes appear in all human populations, the frequencies of occurrence differ. Hence, diversity in the samples of individuals chosen is required. On the other hand, note that if a person is heterozygotic at a position where both his parents are homozygotic but with different alleles, it is trivial to know the origin of each allele in the individual. This last fact could improve the discovery of haplotypes. The HapMap Project is using samples from 270 people: from the Yoruba people in Ibadan, Nigeria (30 both-parent-and-adult-child trios), Japanese in Tokyo (45 unrelated individuals), Han Chinese in Beijing (45 unrelated individuals) and 30 trios from USA (residents with ancestry from Northern and Western Europe).

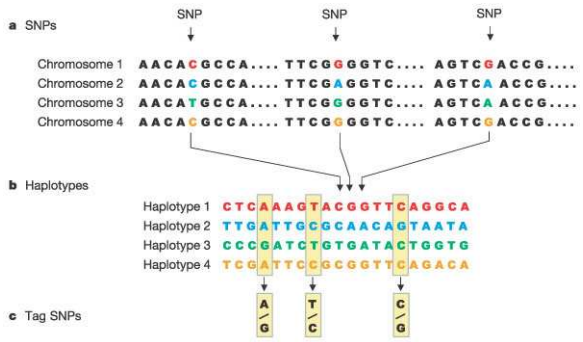

Figure 2.5: The construction of the HapMap project occurs in three steps: a) single nucleotide polymorphisms (SNPs) are identified in DNA samples from multiple individuals, b) adjacent SNPs that are inherit together are compiled into haplotypes, c) tag SNPs, that uniquely identify those haplotypes, are identified. By genotyping the three tag SNPs shown in this figure, researchers can identify which of the four haplotypes shown here are present in each individual.

2.5 Genetic Models to Haplotype Inference

Without any genetic model or assumption it would be impossible to create algorithms to solve the haplotype inference problem or to assess the biological fidelity of any proposed solution. There are two major approaches to solve the inference problem: statistical methods and combinatorial methods.

Statistical methods are usually based on an explicit model of haplotype evolution; the inference problem is then cast as a maximum-likelihood or a Bayesian inference problem. The standard and reasonable assumption one typically consider is that the process of mating among individuals is completely random. Under this assumption, an explicit likelihood function can the derived and the goal is to determine its maximum value. This represents the main problem in practice. Once this value is determined, one can estimate the haplotype frequencies underlying the observed genotype frequencies. With the haplotype frequencies, one can estimate the most probable pair of haplotypes to justify a given genotype. The likelihood approach is, in fact, a special case of the Bayesian-inference problem, which is the process used in well known programs such as Phase and Haplotyper. However, in this project, we will not go deep into the statistical methods; for more information, see [23] or [28], amongst others.

Given some biological assumptions, combinatorial approaches often state an explicit objective function and try to optimize it or use some inference rules in order to obtain a solution to the inference problem. The most well known assumption used is the "infinite sites" model. The "infinite sites" hypothesis states that the number of sites on a genome is so large that the probability of occurring more than one mutation at one site is infinitely small and, therefore, we can assume that it has never occurred in a single species. Although this assumption is almost universally made and empirical data often agrees with this assumption, this is not always true. These cases represent exception and are somewhat difficult to handle.

Several combinatorial models have been proposed to solve the haplotype inference:

- CLARK'S RULE

  In 1990 [4], Clark proposed a common sense approach to haplotyping, which became known as Clark's Rule.

  Under this model, haplotypes in the solution are obtained by successively applying an inference rule, explained in Chapter 3, page 18.

  Clark's method is based in the "infinite sites" genetic model.

- PERFECT PHYLOGENY (or Coalescent Model)

  In 2002, Gusfield [10], introduced the Perfect Phylogeny Haplotyping (PPH) problem.

  He was motivated by studies on the haplotype structure, which show that human DNA can be partitioned into long blocks where genetic recombination has been rare. Indeed, Perfect Phylogeny Haplotyping is based on the "no-recombination in long blocks" assumption which assures that a haplotype of an individual is a copy of one of the haplotypes in one of his parents, and then, the backwards history of a single haplotype is just a path. The "infinite sites" assumption also represents an important role in this model. Under these assumptions, the coalescent model describes the evolutionary history of $2n$ haplotypes as a tree with $2n$ leaves, where each of the m sites labels exactly one edge of the tree (see Chapter 3, page 17).

  The PPH problem can be solved in polynomial time by reducing it to a graph realization problem.

- PURE PARSIMONY

  The Pure Parsimony criteria was first proposed by Earl Hubbell in 2002, who also proved that this problem is NP-hard [12].

  Extending Clark's idea, this approach consists in finding a solution to the haplotype inference problem that minimizes the total number of distinct haplotypes used.

  The Pure Parsimony model is consistent with biological facts. The number of combinatorially possible haplotypes is vastly bigger than the number of distinct haplotypes expected and observed. The fact is explained by the small mutation rate at each site ("infinite sites" assumption) and by the fact that that the recombination rates are low (but not necessarily inexistent as in PPH) on the sequence that constitutes a haplotype. Furthermore, Clark's method seems to give a more realistic solution when it selects less haplotypes [4].

  Pure Parsimony is the core of this project and will be explained in detail in the next chapters.

Chapter 3

Mathematical Formulation of Haplotype Inference

3.1 Basic Definitions

We represent each haplotype by an element of $\{0,1\}^m$, $m \in \mathbb{N}$. Each haplotype is therefore a binary vector where each position represents a SNP. At each site, 0 represents the wild type allele, while 1 represents the mutant allele.

Each genotype is represented by an element of Σ^m where $\Sigma = \{0,1,2\}$, i.e a vector with 0, 1 and 2, where each position represents a SNP. We use 0 (1) when the site is homozygous with value 0 (1), i.e. 0 (1) is inherited from both parents. The value 2 corresponds to a heterozygous site, when different nucleotides were obtained from each parent.

Definition 3.1.1 *Consider the set $\Sigma = \{0,1,2\}$.*

A genotype g is a vector in Σ^m, $m \in IN$.

A haplotype h is a binary vector, i.e. an element of $\{0,1\}^m$, $m \in N$.

A site of g (h) is a component of the vector g (h).

A site of g, $g[j]$ ($1 \leq j \leq m$), is homozygous (or non ambiguous) if $g[j] = 0$ or $g[j] = 1$. Otherwise, $g[j] = 2$, and the site is heterozygous (or ambiguous).

Definition 3.1.2 *Let h_1 and h_2 be two haplotypes of the same size m.*

We define their sum[1] *$g = h_1 \oplus h_2$ as the following genotype with size m:*

$$g[j] = \begin{cases} h_1[j] & \text{if } h_1[j] = h_2[j] \\ 2 & \text{otherwise} \end{cases}, 1 \leq j \leq m$$

We say that a genotype g is resolved (or explained) by a pair of haplotypes $\{h_1, h_2\}$ if $g = h_1 \oplus h_2$. In this case, we say that h_1 is the h_2 conjugate with respect to g. Note that, once you have g and h_1, h_2 can be uniquely determined.

If a genotype g' has at least $m-1$ homozygous sites, then it is said that the genotype is resolved. In this case, there are only two haplotypes h'_1, h'_2 (not necessarily different) such that $g' = h'_1 \oplus h'_2$.

[1] Note that this operation is commutative.

Example 3.1.3 *For instance,* $h_1 = 010011$[2] *and* $h_2 = 110101$ *are two haplotypes with 6 sites.*

Their sum is the 6-site genotype $g = h_1 \oplus h_2 = 210221$.

We can say that g *is resolved (or explained) by* h_1 *and* h_2.

For instance, genotype $g = 02101$ is resolved because the haplotype explaining pair must be 00101/01101. However, usually, genotypes have more than one ambiguous sites and many pairs could possibly explain them. In general, if the genotype has h heterozygous sites, there are 2^{h-1} possible parent pairs that could explain it. For example, genotype 002122 has $2^{3-1} = 4$ possibilities: 000100/001111; 001100/000111; 000110/001101; 001110/000101.

Definition 3.1.4 *Two genotypes,* g_1 *and* g_2*, are compatible if* $g_1[j] = g_2[j]$ *whenever both* $g_1[j]$ *and* $g_2[j]$ *are non ambiguous sites. Alternatively,* g_1 *and* g_2 *are compatible if for every* $j = 1, \ldots, m$ $g_1 + g_2 \neq 1$*. Otherwise, they are incompatible.*

A haplotype h *is compatible with a genotype* g *if* $h[j] = g[j]$ *whenever* $g[j]$ *is a non ambiguous site.*

A *population* is a family $G = \{g_1, \ldots, g_n\}$ of n genotypes, on m sites. Moreover, G is an $n \times m$ matrix, where each row is a m-site genotype. A set H of haplotypes *explains* a population G if for every $g \in G$ there exists $h_1, h_2 \in H$ such that $g = h_1 \oplus h_2$.

The Haplotype Inference problem can be defined as:

Problem 3.1.5 ***Haplotype Inference Problem***

Given a family G *of genotypes, find a set of haplotypes* H *that explains* G *and a relation between elements of* G *and* H*, such that each* $g \in G$ *corresponds to two haplotypes* $h_1, h_2 \in H$ *(not necessarily distinct) and* $g = h_1 \oplus h_2$.

Without any restriction, this problem seems trivial to solve. However, remember that we are trying to solve a biological problem and therefore, to achieve biological relevance, several limitations need to be imposed on the solutions.

[2]Notation: we write 010011 to represent vectors instead of $(0, 1, 0, 0, 1, 1)$.

3.2 Combinatorial Methods for Haplotype Inference

This section will present three combinatorial approaches for the Haplotype Inference problem. The third model, Pure Parsimony, is the main object of this project and will be studied in detail in Chapter 5.

3.2.1 Perfect Phylogeny Haplotyping or Coalescent Model

This model, first presented by Gusfield in 2002 [10], represents the evolutionary history of haplotypes as a directed, acyclic graph, where the lengths of the edges represent the passage in time (in number of generations).

This model is based on two assumptions: "no-recombination in long blocks" [13] and "infinite sites" [29] model. Note that in the absence of recombination, each haplotype sequence has a single ancestor in the previous generation. Under the "no-recombination in long blocks" assumption, each haplotype is passed on through generations intact. Then, the backwards history of a block is not affected by recombination, i.e. a haplotype is just a path through generations. It does not matter whether an individual has two parents, or whether each of his parents has two haplotypes. Another important element in the coalescent model is the "infinite sites" assumption. This means that SNPs are so sparse relative to the mutation rate that, in the human history, at most one mutation has occurred at any site.

Problem 3.2.1 ***The Perfect Phylogeny Haplotype(PPH) Problem***

Given a family G of genotypes, find a solution to the Haplotype Inference Problem such that the set of haplotypes, H, satisfies a perfect phylogeny.

Definition 3.2.2 ***Perfect Phylogeny***

Let $H_{2n\times m}$ be a 0-1 matrix with $2n$ rows and m columns. A perfect phylogeny for H is a rooted tree T with exactly $2n$ leaves that obeys the following properties:

1. *Each of the $2n$ rows labels exactly one leaf of T.*
2. *Each of the m columns labels exactly one edge of T.*
3. *Every interior edge (one not touching a leaf) of T is labeled by at least one column.*
4. *For any row i, the columns that label the edges along the unique path from the root to leaf i specify the columns of H that have a value of one in row i in H. In other words, that path is a compact representation of the row i.*

Theorem 3.2.3 *Classical* ***Theorem of Perfect Phylogeny***

A binary matrix H has a perfect phylogeny if and only if for each pair of columns, there are no three rows with values 0,1; 1,0; and 1,1 in those two columns. Moreover, in this case, the perfect phylogeny for H is unique if and only if the columns of H are distinct.

Example 3.2.4 *Consider the family of genotypes $G = \{22, 02, 10\}$. We have two solutions for the haplotype inference problem $H_1 = \{10, 01, 01, 00, 10, 10\}$ or $H_2 = \{00, 11, 00, 01, 10, 10\}$. However, only H_1 satisfies a perfect phylogeny.*

3.2.2 Clark's Method

Under this model, haplotypes in the solution are obtained successively applying the following algorithm.

First identify all genotypes in G with zero or one ambiguous sites and resolve them in the only way possible, building the initial determined haplotypes, and create the set of determined haplotypes, H, with them. Genotypes with zero or one ambiguous sites are considered resolved and removed from G. Then, successively, apply the inference rule:

Definition 3.2.5 ***Clark's Inference Rule***

If you can resolve a genotype $g \in G$ using a haplotype pair $\{h, h'\}$, where $h \in H$ (determined haplotype) and h' is its conjugate with respect to g, then consider g resolved with h and h', add h' to the set H of determined haplotypes and remove g from G.

The algorithm finishes when all genotypes have been resolved or no further genotypes can be resolved by applying the Inference Rule. In this last case, we say that the algorithm left orphan genotypes.

Clark's method is a non-completely specified algorithm to solve the HI problem: for an ambiguous genotype we could have several choices for the determined haplotype and, consequently, for the explaining haplotype pair. These choices can constrain future choices as the set of determined haplotypes become different. Then, different choices lead to different solutions. Furthermore, while a choice can resolve all genotypes, another may leave ambiguous genotypes that cannot be resolved (orphans).

Some studies have been made in order to decide which are the right choices [24]. Clark suggested to run the algorithm several times with different genotype orders and to choose the solution that resolve more genotypes (local inference method). However, in general, only a tiny number of possibilities can be tried. This leads to the Maximum Resolution Problem: given a set of genotypes, select the execution that maximizes the number of ambiguous genotypes that can be resolved by successive application of Clark's Inference Rule. This problem is shown to be NP-hard [9].

Several alternative variations were analysed [24] and the consensus solution was proposed as a good method to find the best resolution. After running the algorithm several times, the authors suggested to select solutions using the fewest or close to the fewest number of distinct haplotypes. From this set of solutions, record the haplotype pair that was most commonly used to explain each genotype g. The set of those haplotypes is the consensus solution, which seems to have higher accuracy than other solutions.

3.2.3 Pure Parsimony

The Haplotype Inference by Pure Parsimony (HIPP) problem can be mathematically formulated as follows:

Problem 3.2.6 *Given a family G of genotypes, find a minimum-cardinality set of haplotypes H that explain G.*

Note that the solution to the HIPP problem may not be unique.

Example 3.2.7 *Consider the set of genotypes* $G = \{02120, 22110, 20120\}$. *There are HI solutions for this set that use five haplotypes, but* $H_1 = \{00100, 01110, 10110\}$ *and* $H_2 = \{00110, 01100, 10100\}$ *are two HIPP solutions with only three distinct haplotypes.*

Complexity

Theorem 3.2.8 *The Haplotype Inference by Pure Parsimony problem is APX-hard [17].*

Definition 3.2.9 *A problem is APX-hard if it is NP-hard and there is a constant value* δ *below which it is impossible to approximate the problem in polynomial time, unless* $P = NP$ [3].

The proof of the APX-hardness of the HIPP [17] is done by reducing of the NODE-COVER [4] problem, which is known to be APX-hard, to the HIPP problem.

[3] Proving that $P = NP$ or that $P \neq NP$ is an open problem, although it is widely believed that it is unlikely that $P = NP$

[4] Given as input an undirected graph $G = (V, E)$, we are asked to find a node cover $X \subseteq V$ having the smallest possible cardinality. A node cover is a vertex-set $X \subseteq V$ such that every edge in E has at least one endpoint in X.

3.3 Lower and Upper Bounds to Pure Parsimony

The Pure Parsimony approach searches a solution to the HI problem that minimizes the number of haplotypes used to explain the given genotypes. In this section, we investigate upper and lower bounds to this number, denoted by OPT (number of haplotypes on the optimal solution).

The following results, published by Lancia et al. and Lynce et al. [17, 20], are not only theoretically important, but also useful in practice, when applied to some HIPP resolution methods.

In the following, let G be the set of genotypes we want to explain and $n = |G|$.

Trivial lower and upper bounds to OPT are, respectively, 1 and $2n$.

Proposition 3.3.1 *Let G be a nonempty population. Then, there exists H that explains G with $1 \leq |H| \leq 2|G|$, i.e.*

$$1 \leq OPT \leq 2n.$$

Proof 3.3.2 *For a m-site genotype $g \in G$, let h_0 be the haplotype whose sites are 1 whenever g is 1 and 0 in all other positions. Let $h_1[j]$ be 0 whenever $g[j] = 0$ and 1 otherwise. Hence, $g = h_0 \oplus h_1$. It is straightforward that at the most $2n$ distinct haplotypes are required to explain g.*

Proposition 3.3.3 *Assume that G is a population such that $|G| = n \geq 2$, and each g is unique in G. Suppose H explains G. Then $|H| \geq \sqrt{2n}$. In particular, $OPT \geq \sqrt{2n}$.*

Proof 3.3.4 *Suppose H explains G and each g is unique in G. Then, for each $g \in G$ we can associate a different pair of haplotypes. Hence, $n \leq \binom{|H|}{2} = (|H|\,(|H|-1))/2$, which implies $|H| \geq \sqrt{2n}$.*

3.3.1 Clique-Based Bounding

The latest bounds considered are still very loose in general. We know that the HIPP problem is APX-hard, hence it is not possible to find bounds that are always tight. Anyway, better bounds were studied, because even though they are not very tight, some approaches (e.g. SHIPs) work very well with them in practice.
We begin presenting some definitions of graph theory.

Definition 3.3.5 *Let $G = (V, E)$ be a graph. A clique in G is a subgraph $G' = (V', E')$ of G such that if $v_1, v_2 \in V'$ then $v_1 v_2 \in E'$.*

A maximal clique of G is a clique of G with a maximum number of vertices.

Definition 3.3.6 *Given a family G of genotypes, we define the incompatibility graph I as the graph whose vertices are the genotypes and has an edge between two vertices if the respective genotypes are incompatible, i.e. $I = (V_I, E_I)$ where $V_I = G$ and $E_I = \{g_i g_j : g_i, g_j \in G$ are incompatible$\}$.*

Proposition 3.3.7 *Suppose I is an incompatibility graph with a clique with size k (i.e. with k vertices). Then, the minimum number of required haplotypes, OPT, is at least $2k - \sigma$, where σ is the number of genotypes in the clique that are homozygous in all sites.*

Proof 3.3.8 *First we prove the following lemma:*

Lemma 3.3.9 *Let H_1 and H_2 be the sets of haplotypes compatible with g_1 and g_2, respectively. If g_1 and g_2 are incompatible, then $H_1 \cap H_2 = \phi$.*

Proof of the lemma: As g_1 and g_2 are incompatible, there exists k: $1 \leq k \leq m$, non ambiguous position for both genotypes, such that $g_1[k] \neq g_2[k]$. Suppose $h \in H_1$. Then $h[j] = g_1[j]$ whenever $g_1[j]$ is a non ambiguous site. In particular, $h[k] = g_1[k] \neq g_2[k]$. Hence, h is not compatible with g_2, i.e. $h \notin H_2$. Similarly, if $h \in H_2$ then $h \notin H_1$.

Let G_I be the clique in I with size k and $\{g_i\}_{i=1,\ldots,k}$ be its vertices. Then, $g_1, \ldots, g_k$ are incompatible and from the latest lemma $H_1, \ldots, H_k$ are disjunct. As a result, a solution to the HI problem must have two distinct haplotypes for each genotype $g_i \in G_I$, unless g_i has no heterozygous sites, in which case only one haplotype in necessary. Hence, $OPT \geq 2k - \sigma$.

The lower bound calculated in the previous proposition can be improved if we make a better analysis of genotype structures. Note that, if a genotype has a heterozygous site where all genotypes which are compatible with him, and used in lower bound purposes, have the same homozygous value, then a different haplotype is required and the lower bound can be increased by 1.

Algorithm 1 Lower Bound Algorithm

ImproveLB(G_C)

1. $lb \leftarrow 2\,|G_C| - \sigma$
2. Sort genotypes by increasing number of heterozygotic sites
3. Create set G_{SET} with genotypes in clique G_C
4. **for** each $g \in G$ of non-clique genotypes **do**
   Let S be the subset of genotypes in G_{SET} compatible with g
   if g has a heterozygous site j and every $s \in S$ has the homozygous site j **then**
   $lb \leftarrow lb + 1$
   $ng \leftarrow MergeGenotypes(S \cup g)$
   $G_{SET} \leftarrow (G_{SET} - S) \cup ng$
   end if
5. **end for**
6. **return** lb

Algorithm 1 formalizes this idea. The lower bound is initialized with the value calculated in Proposition 3.3.7 to guarantee never to get a bound inferior to the clique-based

bound of Proposition 3.3.7. Then, we search genotypes, not in the clique, that require one more explaining haplotype. Procedure $MergeGenotypes$ creates a new genotype from a set of genotypes such that any heterozygous site or site with genotypes having both 0 and 1 becomes heterozygous. If all genotypes have the same homozygous site, then the resulting genotype keeps the same value at the respective position (Equation 3.1).

$$MergeGenotypes(S)[j] = \begin{cases} 0 & \text{if} \quad \text{if } g[j] = 0 \text{ for every } g \in S \\ 1 & \text{if} \quad \text{if } g[j] = 1 \text{ for every } g \in S \\ 2 & \text{if} \quad \text{otherwise} \end{cases}, \; 1 \leq j \leq m; \tag{3.1}$$

The output is the new lower bound. A simple analysis shows that this algorithm runs in $\mathcal{O}(n^2 m)$.

Chapter 4

Discrete Optimization Problems

The approaches to HIPP problem presented in this work use some important techniques which are able to find application problems in several fields. In this chapter, we look into those techniques: Integer Linear Programming Optimization (IP), Pseudo-Boolean Optimization (PBO) and Boolean Satisfiability (SAT).

4.1 Integer Linear Programming

In the following chapter, three Integer Linear Programming formulations to solve Haplotype Inference by Pure Parsimony will be presented.

Integer Linear Programming (IP) is a special case of Linear Programming (LP). LP is a field of research rather studied and with several important results. We only provide a brief overview, essential to have a good understanding of the models used.

Many practical optimization problems can be expressed using Linear Programming formulations.

The goal of Linear Programming is to optimize (minimize or maximize) a linear function, subject to a finite set of linear constraints (equalities or inequalities). The general formulation is shown in Table 4.1[1]. 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 LP application consists in maximizing the business profit subject to constraints evolving the cost of production scheme.

Table 4.1: Linear Programming Optimization

minimize (or maximize):	$f(x_1, \ldots, x_n) = \sum_{j=1}^{n} c_j x_j;$
subject to:	
	$g_i(x_1, \ldots, x_n) = \sum_{j=1}^{n} a_{ij} x_j \leq b_i;$
	$c_j, a_{ij} \in \mathbb{R},\ 1 \leq i \leq m,\ 1 \leq j \leq n$

[1]Note that $\geq$-constraints can be changed into $\leq$-constraints by multiplying by -1. An equality can be converted into two inequalities.

In 1947, Dantzig developed the simplex method to solve LPs. Although the complexity of simplex is exponential in the worst case, this algorithm is efficient in most cases and is very useful in practice. Although some methods have been developed to solve LPs in polynomial time, simplex is still commonly used. CPLEX (`www.ilog.com/products/cplex`) is the most well known commercial tool used to solve LPs applying the simplex procedure.

Although many problems can be naturally formulated as LPs, there are others which require further restrictions. For instance, there are many problems that only admit variables with integer values. These problems, formulated as LPs where all variables take only integer values, are known as integer linear programming (IP) formulations. In this case, the problems become harder and the methods used to solve LPs do not work. In fact, solving a generic IP is an NP-hard problem. However, several techniques have been studied in order to solve IPs reasonably quickly, in most practical situations.

Usually, to solve an IP problem, one solves the respective LP relaxation, obtained removing the integrality constraints on the variables, and then tries to find the optimal integer solution. This last step can be taken applying different techniques.

One method uses the fact that the solution of an LP minimization problem (a maximization formulation can be easily conversed into a minimization), constitutes a lower bound to the IP problem because all integer solutions are also solutions of the LP. In order to push the solution from the LP domain to the IP domain, one can add constraints, called valid cuts, which are valid for all feasible IP solutions but are violated by the current LP solution. Then, once again, we solve the LP relaxation and obtain a new solution. We repeat this process until we reach an integral solution, the one we are looking for.

Another method used to solve an IP minimization is the branch-and-bound algorithm. First, calculate an upper bound to the value of any feasible solution. Let v be a variable which could take $n < +\infty$ distinct integer values and branch the problem to the n IPs obtained by setting v to each possible value. The lowest optimal IP solution among this set is the optimal solution of the original IP. Then, as long as there are variables that admit only a limited set of values, one could repeat the proceedure and explore the entire solution space. However, on each branch, if the LP solution is greater than the upper bound, then we can discard this branch and stop exploring it.

A branch-and-cut algorithm is a branch-and-bound proceeding where, at each step, we add valid cuts to improve the bounds. In general, this is an efficient method to solve IP problems. Given the NP-hardness of IP, any of these procedures has exponential complexity on the worst case.

4.2 Boolean Satisfiability

We begin this section by introducing some basic definitions of propositional logic.

4.2.1 Propositional Logic

Let $X = \{x_i\}_{i \in \mathbb{N}}$ be the numerable set of propositional variables, which can be assigned to 0 (false) or 1 (true).

Definition 4.2.1 ***Propositional Formulas***
The set Φ of propositional formulas is defined inductively as follows:

- $x_i \in \Phi$, *for each* $i \in N$.
- $(\neg\varphi \in \Phi)$, *whenever* $\varphi \in \Phi$.
- $(\varphi_1 \vee \varphi_2) \in \Phi$, *whenever* $\varphi_1, \varphi_2 \in \Phi$.
- $(\varphi_1 \wedge \varphi_2) \in \Phi$, *whenever* $\varphi_1, \varphi_2 \in \Phi$.

Implication, $(\varphi_1 \Rightarrow \varphi_2)$, is also usually introduced but it is not used in this section. However it could be seen as an abbreviation of $(\neg\varphi_1 \wedge \varphi_2)$.

Some particular formulas, called CNF formulas, have great importance in Satisfiability (SAT) theory.

Definition 4.2.2 ***Literal, Clause and CNF Formulas***

- *A literal l is either a variable x_i or its complement $\neg x_i$.*
- *A clause w is a disjunction ($\vee$) of literals.*
- *A CNF (conjunctive normal form) formula φ is a conjunction ($\wedge$) of clauses.*

Definition 4.2.3 ***Satisfaction***
Let ρ be an assignment, i.e. $\rho : X \to \{0, 1\}$.
The satisfaction of a formula φ by an assignment ρ is defined inductively as follows:

- $\rho \models x_i$ *if* $\rho(x_i) = 1$;
- $\rho \models (\neg\varphi)$ *if* $\rho \not\models \varphi$;

- $\rho \models (\varphi_1 \vee \varphi_2)$ *if* $\rho \models \varphi_1$ *or* $\rho \models \varphi_2$;

- $\rho \models (\varphi_1 \wedge \varphi_2)$ *if* $\rho \models \varphi_1$ *and* $\rho \models \varphi_2$;

By the definition of satisfaction, we conclude that, given an assignment ρ, a CNF formula is satisfied by ρ if all its clauses are satisfied. A clause is satisfied when at least one of its literals is satisfied. Finally, a positive (negative) literal x_i ($\neg x_i$) is satisfied by ρ if $\rho(x_i) = 1$ ($\rho(x_i) = 0$).

Moreover, each formula φ denotes a unique n-variable Boolean function $f(x_1, \ldots, x_n)$ ($f : \{0,1\}^n \to \{0,1\}$), where $x_1, \ldots, x_n$ are the variables of φ.

We say simply that a formula φ is satisfiable if there exists an assignment which satisfies φ.

Theorem 4.2.4 *For each propositional formula there is an equivalent CNF formula.*

We say that two formulas are equivalent if both denote the same Boolean function.

4.2.2 Satisfiability Problem

The Boolean Satisfiability problem (SAT) consists of, given a formula φ, deciding whether there exists an assignment to the variables such that φ becomes satisfied. This important problem was the first proved to be NP-complete.

SAT instances contain a large set of problems and in several contexts; for instance, large design and analysis problems from the field of Electronic Design Automation (EDA) can be cast as SAT instances. Due to its great importance, many algorithms have been studied to increase the range of applicability of SAT. The most well known procedures are backtracking search algorithms which at each node of the search tree, elects an assignment and prunes subsequent search by iteratively applying the unit clause and the pure literal rules [25]. The unit clause rule is the following: if a formula φ contains a unit clause (clause with only one literal), assign the value true to this literal. Iterated application of the unit clause is called the Boolean Constraint Propagation.

SAT can be seen as a particular case of IP. In fact, any SAT problem is an IP instance with an objective function which is constant. Each clause on the SAT problem is equivalent to a constraint of the respective IP problem. For instance, the clause $\neg\alpha \vee \beta \vee \delta$ is satisfied if and only if $(1 - \alpha) + \beta + \delta \geq 1$.

MINISAT [7], SATZ [18] and GRASP [22] are examples of some efficient SAT solvers.

4.3 Pseudo-Boolean Optimization

Although growing interest in SAT has lead to the construction of powerful SAT-solvers, several problems cannot be easily formulated directly into SAT, and less restricted solvers were created.

Pseudo-Boolean Optimization (PBO) problems, also known as 0-1 Integer Programming problems, can be considered intermediates between IPs and SAT. A PBO problem consists in finding a satisfying assignment to a set of PB-constraints that minimizes a given objective function. A PB-constraint is an inequality $\sum_{j=1}^{n} a_j l_j \geq b$, where, for all j, l_j is a literal and a_j, b are integer coefficients. The left-hand side will be abbreviated by LHS, and the right-hand constant b referred to as RHS. A PB-constraint is said to be satisfied under an assignment if the sum of its activated coefficients exceeds or is equal to the right-hand side constant b.

The differences between IP and PBO are that, in PBO, variables are restricted to boolean values and all coefficients must be integers. On the other hand, a SAT problem is a PBO problem without an objective function. However, in general, a PB constraint is equivalent to a large, potentially exponential, number of CNF clauses. Only a PB-constraint where all coefficients are equal to one is equivalent to a single CNF clause.

Every PB-constraint can be rewritten in a normal form where all coefficients are non-negative integers. Hence, any instance of a linear Pseudo-Boolean Optimization problem can be cast as follows:

Table 4.2: Pseudo-Boolean Optimization

minimize	$\sum_{j=1}^{n} c_j x_j$;
subject to	$\sum_{j=1}^{n} a_{ij} l_j \geq b_i$;
	$x_j \in \{0, 1\}$;
	$a_{ij}, b_i, c_j \in N_0$; $1 \leq i \leq m$, $1 \leq j \leq n$

In this project we use two Pseudo-Boolean solvers that have shown to perform well on the PB-evaluation 2005 [21]: MINISAT+ [8] and Pueblo [27].

4.3.1 MINISAT+

MINISAT+ [8] solves PBO problems by converting PB constraints to equivalent systems of CNF clauses and submitting them to a SAT solver (MINISAT).

Three different techniques can be used in MINISAT+ for translating pseudo-boolean constraints into clauses: BDDs, adders or sorters. The first technique consists in converting the PB constraint into the BDD representation and then translating the BDD into CNF clauses. Another method is to convert the PB constraint to a network of adders, where the sum at the LHS of the PB constraint is produced as a binary number and the circuit representing the comparison of this sum against the RHS is created. Then, the CNF representation of this circuit is obtained. Finally, converting the PB constraint to a network of sorters, where numbers are represented in unary instead of binary, again the

circuit is obtained and translated to CNF clauses. For more details on these techniques, the original reference [8] should be consulted.

To handle the objective function $f(x)$, MINISAT+ iteratively runs the SAT solver until it finds an assignment that minimizes $f(x)$. First, the PBO solver runs the SAT solver on the set of constraints (without considering the objective function) to get an initial solution $\overrightarrow{x_0}$ such that $f(\overrightarrow{x_0}) = k$. Then, the constraint $f(\overrightarrow{x}) < k$ is added to the problem and the SAT solver is run again. The process is iterated until the problem is unsatisfiable. At this point, we know that we have the optimal solution, and its cost is k. Note that, each constraint added subsumes all previous ones. So, at each step, the optimization inequality can be replaced with the new one, in order to have control on the number of constraints of the problem. However, in MINISAT+ it is not so easy because as the constraint has been converted into a number of clauses, usually new variables have been introduced; removing clauses containing those variables will vastly reduce the pruning power of the learned clauses. As a result, MINISAT+ implements the naive optimization loop.

4.3.2 Pueblo

Pueblo [27] is a hybrid pseudo-boolean SAT solver. Cutting plane techniques (used usually by IP solvers) and implication graph analysis (used in modern SAT and PB solvers) for conflict-based learning are combined in order to produce a more powerful solver. Pueblo creates both a PB constraint and a CNF clause after a conflict is detected. Constraints and clauses are treated as separate constraint classes, in order to benefit from the best of both.

Chapter 5

Haplotype Inference by Pure Parsimony Approaches

Over the last few years, several methods for solving the Haplotype Inference by Pure Parsimony problem have been proposed. Gusfield [11], Brown and Harrower [2, 3] suggested Integer Linear Programming approaches to the problem, Wand and Xu [31] proposed a branch-and-bound algorithm and, more recently, Lynce and Marques-Silva [19] came up with a Satisfiability approach. Finally, inspired by IPs and SAT models, this work suggests a new HIPP approach based on Pseudo-Boolean Optimization.

5.1 Integer Linear Programming Approaches

5.1.1 RTIP

In 2003, Dan Gusfield [11] introduced the first Integer Linear Programming formulation to solve the HIPP problem. First, he created a conceptual formulation called TIP, described below. This formulation, and thus the running time required to create it, increases exponentially with the problem size, and therefore it is impractical to use it without additional improvements. Then, Gusfield introduced a simplification that improves TIP significantly.

We begin by describing the conceptual formulation of TIP. Let g_i denote the i^{th} genotype and suppose it has h_i ambiguous sites (i.e. sites with value 2). Then, there are 2^{h_i-1} pairs of haplotypes that can explain genotype g_i. Expand all these possible explaining pairs, enumerate them and attribute a variable $y_{i,j}$ to each pair j ($1 \leq j \leq 2^{h_i-1}$). At the same time, enumerate the corresponding haplotypes in such a way that equal haplotypes have the same enumeration number. Whenever a haplotype that has not been seen before is enumerated, we create a new variable x_k which represents this new haplotype. Let H be the set of different haplotypes in this expansion (and $|H|$ its cardinality). Then, the formulation TIP is the following:

$$\begin{aligned} \text{minimize:} \quad & \sum_{k=1}^{|H|} x_k \\ \text{subject to:} \quad & \\ & \sum_{j=1}^{2^{h_i-1}} y_{i,j} = 1 \end{aligned} \tag{5.1}$$

$$y_{i,j} - x_{k_{i,j_1}} \leq 0 \tag{5.2}$$
$$y_{i,j} - x_{k_{i,j_2}} \leq 0 \tag{5.3}$$
$$x_k, y_{i,j} \in \{0,1\}$$

$$1 \leq i \leq n; 1 \leq j \leq 2^{h_i - 1}; 1 \leq k \leq 2^{h_i - 1}$$

Constraint (5.1) means that each genotype must be explained by one and only one pair of haplotypes. Only one variable $y_{i,j}$, for each i, must be assigned to 1, the one representing the chosen pair. Constraints (5.2) and (5.3) say that if one haplotype pair is the solution to explain genotype g_i, then the associated haplotypes must also be selected. Note that if $y_{i,j}$ is assigned to 1, then we should also set $x_{k_{i,j_1}}$ and $x_{k_{i,j_2}}$ to 1, 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 objective function ensures that the number of x_k variables set to 1 is the minimum possible. Remember that we want a minimum number of haplotypes to be selected.

For example, suppose we have two 4-site genotypes: $g_1 = 2102$ and $g_2 = 0201$. For genotype g_1, we have two possible explaining pairs, $y_{1,1} = \{1101; 0100\}$ and $y_{1,2} = \{1100; 0101\}$, while g_2 has only one parent pair $y_{2,1} = \{0101; 0001\}$. One possible enumeration for the haplotypes is: $x_1 = 1101$, $x_2 = 0100$, $x_3 = 1100$, $x_4 = 0101$ and $x_5 = 0001$. The formulation for this example will be:

$$\begin{aligned} y_{1,1} + y_{1,2} &= 1; \\ y_{2,1} &= 1; \\ y_{1,1} - x_1 &\leq 0; \\ y_{1,1} - x_2 &\leq 0; \\ y_{1,2} - x_3 &\leq 0; \\ y_{1,2} - x_4 &\leq 0; \\ y_{2,1} - x_4 &\leq 0; \\ y_{2,1} - x_5 &\leq 0; \end{aligned}$$

Although this formulation is correct and useful theoretically, observe that it is exponential in the number of heterozygous sites. Indeed, this formulation would be generally impractical on problems of current biological interest. Consequently, Gusfield proposed a simplification which turns the model, in general, considerably smaller. Instead of considering all possible parents for each genotype, the RTIP (reduced TIP) approach only expands pairs where at least one haplotype could partly explain another genotype. Note that the solution with less haplotypes must discard these parents, so this simplification will not change the correctness of the TIP formulation. For instance, in the example, pair $y_{1,1}$ would be discarded from all equations. If for one genotype all possible parents were discarded, then no matter which possible pair we choose we get anyway a parsimonious solution.

The success of this idea is helped by the fact that population DNA sequences have suffered many recombinations. As the level of recombination rises, haplotypes become more differentiated. Consequently, also genotypes become more different, and we have more pairs that can be discarded in the RTIP approach and, as a result, this reduced model becomes much smaller than TIP.

However, if we needed to expand all pairs of haplotypes before removing variables, the profits would be insignificant. Nonetheless this is not necessary. Let H_1 an H_2 be the set of haplotypes that can partly explain genotype g_1 and g_2 respectively. Note that the intersection of such sets, i.e. the set of haplotypes that can explain partially both genotypes can be determined in $\mathcal{O}(m \mid H_1 \cap H_2 \mid)$, where m is the number of sites of g_1 and g_2. Simply, cover g_1 and g_2 from left to right: if for a given site, one has value 1 and the other has value 0 then $H_1 \cap H_2 = \emptyset$; if a site occurs with value 2 for one genotype and 0 or 1 for the other, then haplotypes in the intersection must have this site with the same value of the later.

Even with these optimizations, RTIP remains an exponential-sized model in the worst case.

5.1.2 PolyIP

In 2004, Brown and Harrower [2] discovered a polynomial-sized formulation for HIPP. PolyIP is another Integer Linear Programming (IP) formulation for Pure Parsimony.

Let genotype g_i be explained by haplotypes h_{2i-1} and h_{2i}. Then, create a variable for each site of h_{2i-1} and h_{2i}. In addition, denote by $y_{i,k}$ the variable representing site k of haplotype h_i, in such a way that $y_{i,k} = h_i[k]$ (note that $h_i[k] \in \{0, 1\}$). Then, PolyIP can be formulated as follows:

$$\begin{aligned}
\text{minimize:} \quad & \textstyle\sum_{i=1}^{2n} x_i \\
\text{subject to:} \quad & \\
& y_{2i'-1,k} + y_{2i',k} = \begin{cases} 0 & \text{if} \quad g_{i'}[k] = 0 \\ 2 & \text{if} \quad g_{i'}[k] = 1 \\ 1 & \text{if} \quad g_{i'}[k] = 2 \end{cases} && (5.4) \\
& d_{i,j} \geq y_{i,k} - y_{j,k} && (5.5) \\
& d_{i,j} \geq y_{j,k} - y_{i,k} && (5.6) \\
& x_i \geq 2 - i + \textstyle\sum_{j=1}^{i-1} d_{j,i} && (5.7) \\
& x_i, y_{i,k}, d_{i,j} \in \{0, 1\} \\
& \\
& 1 \leq i' \leq n; 1 \leq k \leq m; 1 \leq i \leq j \leq 2n
\end{aligned}$$

Constraint (5.4) ensures that h_{2i-1} and h_{2i} really explain g_i, $1 \leq i \leq n$. In order to count the number of different haplotypes used, we introduce new variables $d_{i,j}$ ($1 \leq i \leq j \leq 2n$) such that $d_{i,j} = 1$ if (but not only if) $h_i \neq h_j$. If $h_i \neq h_j$, there must exist k, $1 \leq k \leq m$, such that $y_{i,k} \neq y_{j,k}$; consequently, constraints (5.5) and (5.6) force $d_{i,j}$ to be 1.

Next, create variables x_i for $1 \leq i \leq 2n$, such that x_i is 1 if (but not only if) it is different from all haplotypes that had appeared before. This is the meaning of constraint (5.7). Note that $\sum_{j=1}^{i-1} d_{j,i} = i - 1$ if h_i is distinct from the previous haplotypes, by the definition of $d_{i,j}$.

To understand the objective function, remember that we want to minimize the number of distinct haplotypes used, i.e. minimize of variables x_i set to 1.

Both the number of variables and the number of constraints in this formulation are $\mathcal{O}(nm+n^2)$. So, in contrast with TIP and RTIP formulations, which, as we have seen, are exponential-sized formulations, PolyIP can be formulated in polynomial time and space. However, remember that solving an IP is NP-hard and so the performance of PolyIP applied to a IP solver, just like it has been formulated, may be poor. This happens because the PolyIP formulation does not behave well under LP relaxation. Consequently, many cuts and simplifications were done in PolyIP in order to help the IP solver to find the right solution [3]. For instance, the authors have changed the definition of $d_{i,j}$ so that it is assigned to 1 only when h_i and h_j differ. Also, they have augmented the objective function and added some cuts that depend on input data patterns. These changes had the objective of reducing the gap between the optimal solutions of the LP relaxation and the IP, and improve the efficiency of PolyIP solver.

Perhaps surprisingly, the exponential-sized RTIP does not always run slower than the polymomial-sized PolyIP. In fact, RTIP is able to solve, in seconds, many problem instances for which PolyIP needs hours (although the opposite also happens). Perhaps the reason is that the smaller formulation has to computationally discover necessary features of the optimal solution (such as the candidate pairs) that are explicitly specified in the larger formulation.

5.1.3 HybridIP

In 2006, Brown and Harrower [3] presented a hybrid between RTIP and PolyIP, which joins the strength of both formulations. Although RTIP is impractical to solve big instances, it is very fast solving small problems. This happens because when we are able to formulate RTIP, the formulation of the problem is described in detail and, most of the times, it can be easily solved by the IP solver.

With HybridIP, the authors intended to create a formulation with both practical size and reasonable runtimes. In order to reach this goal, HybridIP, inspired by RTIP, expands some haplotype pairs and then, formulates the problem similarly to PolyIP. The HybridIP formulation is described below.

$$\begin{aligned}
&\text{minimize:} && \textstyle\sum_{i=1}^{|H_e|} x'_i + \sum_{i=1}^{2n} x_i \\
&\text{subject to:} &&
\end{aligned}$$

$$\sum_{w_{i,(j,k)} \in W_i} w_{i,(j,k)} + u_i = 1, \qquad 1 \le i \le n \tag{5.8}$$

$$x'_j \ge w_{i,(j,k)}, \qquad 1 \le i \le n; w_{i,(j,k)} \in W_i \tag{5.9}$$

$$x'_k \ge w_{i,(j,k)}, \qquad 1 \le i \le n; w_{i,(j,k)} \in W_i; k \ne * \tag{5.10}$$

$$y_{2i-1,k} + y_{2i,k} = \begin{cases} 0 & \text{if} \quad g_i[k] = 0 \\ 2 & \text{if} \quad g_i[k] = 1 \\ 1 & \text{if} \quad g_i[k] = 2 \end{cases}, \qquad 1 \le i \le n; 1 \le k \le m \tag{5.11}$$

$$d_{i,j} \ge y_{i,k} - y_{j,k}, \qquad 1 \le i < j \le 2n; 1 \le k \le m \tag{5.12}$$

$$d_{i,j} \ge y_{j,k} - y_{i,k}, \qquad 1 \le i < j \le 2n; 1 \le k \le m \tag{5.13}$$

$$y_{j,k} \le d'_{i,j} \text{ if } h'_i[k] = 0, \qquad 1 \le i \le |H_e|; 1 \le j \le 2n; 1 \le k \le m \tag{5.14}$$

$$1 - y_{j,k} \le d'_{i,j} \text{ if } h'_i[k] = 1, \qquad 1 \le i \le |H_e|; 1 \le j \le 2n; 1 \le k \le m \tag{5.15}$$

$$d'_{j,2i-1} \ge u_i, \qquad 1 \le j \le |H_e|; 1 \le i \le n \tag{5.16}$$

$$d'_{j,2i} \ge u_i, \qquad 1 \le j \le |H_e|; 1 \le i \le n \tag{5.17}$$

$$d'_{j,2i-1} \le 1 - w_{i,(j,k)}, \qquad 1 \le i \le n \text{ such that } w_{i,(j,k)} \in W_i \tag{5.18}$$

$$d'_{j,2i} \le 1 - w_{i,(j,k)}, \qquad 1 \le i \le n \text{ such that } w_{i,(j,k)} \in W_i, k \ne * \tag{5.19}$$

$$x_i \ge 2 - (K + i) + \sum_{j=1}^{K} d'_{j,i} + \sum_{j=1}^{i-1} d_{j,i}, \qquad 1 \le i \le 2n \tag{5.20}$$

$$x_i, x'_i, y_{i,k}, d_{i,j}, d'_{i,j}, w_{i,(j,k)}, u_i \in \{0, 1\}, \qquad \text{for all } i, j \text{ and } k \tag{5.21}$$

In the HybridIP formulation, we expand only some explaining parents, choosing a set H_e of K possible explaining haplotypes. Each genotype g_i can be explained by zero, one or two haplotypes from H_e. For every (h_j, h_k) haplotype pair that explains genotype g_i: if both h_j and h_k are in H_e, then we create a variable $w_{i,(j,k)}$; if only h_j (respectively, h_k) is in H_e, then $w_{i,(j,*)}$ (respectively, $w_{i,(k,*)}$) is created; otherwise, neither haplotype is in H_e and we create a variable u_i. Each of these variables will be set to 1 if the respective haplotype pair is the one chosen to explain g_i.

Constraint (5.8) guarantees that, for each genotype, exactly one of these variables is set to one because we require one and only one explaining pair for each genotype.

As in RTIP, we have a variable x'_j that should be set to 1 when haplotype $h_j \in H_e$ is chosen to explain any genotype (constraints (5.9) and (5.10)).

On the other hand, we have constraints similar to those in PolyIP: constraint (5.11) ensures that the haplotype pair chosen to explain g_i really explains it (remember that $y_{i,k}$ is valued to 1 (respectively, to 0) when $h_i[k] = 1$ (respectively, $h_i[k] = 0$)); constraint (5.12) and constraint (5.13) introduce variables $d_{i,j}$ that are set to 1 when h_i and h_j are distinct.

Variables $d'_{i,j}$ are introduced to set the differences between haplotypes chosen and haplotypes in H_e. Then, $d'_{i,j}$ should be 1 when the selected haplotype is different from all haplotypes in H_e (constraints (5.14) and (5.15)). If we choose an explaining pair where neither haplotype is in H_e, then we must also mark this difference setting $d'_{i,j}$ to 1 (constraints (5.16) and (5.17)). However, if the pair has haplotypes in H_e, then $d'_{i,j}$ must be set to 0 (constraints (5.18) and (5.19)).

Finally, create variables x_i for each haplotype in the solution, as in the PolyIP, that mark the difference between a haplotype and the previous (constraint (5.20)).

The objective function is to minimize the number of distinct haplotypes used that are in or out H_e.

If the number of haplotypes in set H_e, K, is constant, then HybridIP is a polynomial-sized model. In particular, if $K = 0$, i.e. we do not explicitly expand any haplotype, then this formulation is essentially PolyIP. On the other hand, if we expand all possible parents, then we get a formulation behaving like RTIP. Some experiments were made in [3] in order to choose the best value for the parameter K. Indeed, HybridIP seems to be very sensitive to that choice. Although the authors could not find a clear best choice for the K parameter, the value of 24 was chosen, as being the smallest value that performed well in their tests. However, how should we choose this set H_e of K haplotypes? The best answer to this question seems to be: expand haplotypes that partially explain genotypes with less heterozygous sites until you reach the cutoff K. In fact, this implies that the expanded set is guaranteed to contain many of the haplotypes found in the optimal solution.

All the valid cuts and modifications applied to PolyIP were also applied to HybridIP in order to improve the speed of solving the HIPP problem using integer programming.

5.1.4 Structural Simplifications

Several simplifications could be done to the set of genotypes without loss of information. These simplifications consist in using the structural properties of genotypes with the purpose of reducing the search space [3].

If in the original instance, we have two equal genotypes, clearly one could be discarded assuming that the two genotypes will be explained identically. If for a given site all genotypes have the same homozygous value, then this site can also be discarded, because we already know that all haplotypes must have this value at the same position. Moreover, for pairs of sites with *complemented* values (i.e. whenever one site has value 1, the other has value 0 and vice versa) in each genotype, one of the sites can also be discarded,

because one can reproduces the eliminated site from the other site.

If one keeps the information of which sites were discarded and why, it is straightforward to construct the solution to the problem, once discovered one solution to the simplified set.

These structural simplifications were first proposed by Brown and Harrower [3], who apply them to the IP HIPP solvers. They were also used by the SHIPs model [19].

5.2 HAPAR

HAPAR was proposed by Wang and Xu in 2003 [31]. HAPAR is a branch-and-bound algorithm, different from the previous integer linear programming approaches, but also designed to find a parsimony solution.

Let the *coverage of a haplotype* be the number of genotypes in the sample that the haplotype can resolve, and the *coverage of a resolution* be the sum of the coverage of an explaining pair of haplotypes. The efficiency of any branch-and-bound algorithm depends on the quality of the initial solution. To create an initial solution to this branch-and-bound algorithm, the authors suggest a greedy algorithm that for each genotype gives the explaining pair with maximum coverage. The solution of this greedy algorithm is often close to the optimum solution. The branch and bound algorithm sketched in Algorithm 2 receives a set of n genotypes and outputs the solution to the HIPP problem. Basically, this algorithm searches all possible solutions and finds the best one cutting off the pruning space.

The complexity of the algorithm is $\mathcal{O}(2^{nm})$. However, this number is dependent of the size of the lists of explaining haplotypes. In order to reduce the size of these lists, some improvements were made. In practice, this approach has better performance than the IP models described in the previous section. These results are described in Chapter 6.

HAPAR algorithm is the following:

Algorithm 2 Illustration of the Branch-and-bound Algorithm: HAPAR

Input $\mathcal{M}_{n\times m}$,
for each genotype g_i **do**
List all possible explaining pairs of haplotypes on $Array(i)$
$s_i \leftarrow$ length of $Array(i)$
end for
$S \leftarrow \phi$; (S is the set of resolutions)
$f(S)$ is the number of distinct haplotypes in S
Use the greedy algorithm to get a solution S^* and set $f^*(S)$ to be the size of the solution.
Search for an optimal solution as follows:
1.
for $j_1 = 1$ to s_1 **do**
$S = Array(1)[j_1]$
if $f(S) > f^*(S)$ **then**
go to 1 and try next j_1
end if
2.
for $j_2 = 1$ to s_2 **do**
$S = \{Array(1)[j_1], Array(2)[j_2]\}$.
if $f(S) > f^*(S)$ **then**
go to 2 and try next j_2;
end if
.....
n.
for $j_n = 1$ to s_n **do**
$S = \{Array(1)[j_1], Array(2)[j_2], \ldots, Array(n)[j_n]\}$
if $f(S) < f^*(S)$ **then**
$f^*(S) = f(S)$;
end if
end for
.....
end for
end for

5.3 SHIPs

SHIPs is a recent and very interesting contribution to the HIPP problem.

In February 2006, Lynce and Marques-Silva defined a new method for solving the HIPP problem using Boolean satisfiability (SAT) [19] which is by far more efficient than the IP models and branch-and-bound algorithm described in previous sections.

Given the set G of genotypes, the SAT-based HIPP model (SHIPs) searches for a set of haplotypes H with cardinality r such that H explains G. The SHIPs algorithm considers increasing values of r from a lower bound lb to an upper bound ub until all the equations below become satisfied. Note that if r coincides with the number of necessary haplotypes, then the equations described above must be satisfiable; otherwise, they are unsatisfiable. Hence, when the program terminates we have a solution H with r haplotypes.

The HIPP solution is given by an assignment that satisfies the following CNF formulas, where r is the smallest positive integer such that this assignment exists:

$$(\neg h_{kj} \vee \neg s^a_{ki}) \wedge (\neg h_{kj} \vee \neg s^b_{ki}) \quad \text{if } g_i[j] = 0 \tag{5.22}$$

$$(h_{kj} \vee \neg s^a_{ki}) \wedge (h_{kj} \vee \neg s^b_{ki}) \quad \text{if } g_i[j] = 1 \tag{5.23}$$

$$(g^a_{ij} \vee g^b_{ij}) \wedge (\neg g^a_{ij} \vee \neg g^b_{ij}) \quad \text{if } g_i[j] = 2 \tag{5.24}$$

$$(h_{kj} \vee \neg g^a_{ij} \vee \neg s^a_{ki}) \wedge (\neg h_{kj} \vee g^a_{ij} \vee \neg s^a_{ki}) \wedge$$
$$(h_{kj} \vee \neg g^b_{ij} \vee \neg s^b_{ki}) \wedge (\neg h_{kj} \vee g^b_{ij} \vee \neg s^b_{ki}) \quad \text{if } g_i[j] = 2 \tag{5.25}$$

$$(\sum_{k=1}^{r} s^a_{ki} = 1) \wedge (\sum_{k=1}^{r} s^b_{ki} = 1) \tag{5.26}$$

$$1 \leq i \leq n; 1 \leq j \leq m; 1 \leq k \leq r$$

Suppose r is the smallest integer value for which there exists ρ, an assignment that satisfies the formulas. Let n be the number of genotypes and m be the number of sites on each genotype. First, we describe the propositional variables. For $k = 1, ..., r$ and $j = 1, ..., m$, let h_{kj} represent the j^{th} site of haplotype k, i.e., $\rho(h_{kj}) = h_k[j]$ (note that $h_k[j]$ is either 0 or 1). The model also uses the so called *selector* variables, s^a_{ki} and s^b_{ki} ($1 \leq k \leq r$, $1 \leq i \leq n$), 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} if and only if $\rho(s^a_{k_1 i}) = 1$ and $\rho(s^b_{k_2 i}) = 1$.

Formula (5.22) ensures that if $g_i[j] = 0$, then the haplotypes selected to explain g_i must have value 0 at this site, i.e. if h_k represents one haplotype chosen to explain g_i then $\rho(h_{kj}) = 0$. Similarly, if $g_i[j] = 1$ and h_k is chosen, by s^a_{ki} or s^b_{ki}, to explain g_i then $h_k[j]$ must be assigned value 1 (Formula (5.23)). Otherwise, when $g_i[j] = 2$, we require the chosen haplotypes to have opposing values at site j (Formula (5.25)). This is done by creating two boolean variables, g^a_{ij}, g^b_{ij}, such that we must have $\rho(g^a_{ij}) \neq \rho(g^b_{ij})$ (Formula (5.24)). Last condition (Formula (5.26)), formalizes the idea that each genotype g_i must be explained by exactly one pair of haplotypes, and therefore, there exists one and only one k_1 and one and only one k_2 such that $\rho(s^a_{k_1 i}) = 1$ and $\rho(s^b_{k_2 i}) = 1$.

In fact, this formulation has several symmetrical solutions that can be removed. For instance, selector variables, s^a_{ki} and s^b_{ki}, can exchange roles; also variables h_{kj} have some symmetries. Some reductions on the number of g variables and some constraints affecting

s variables described in [20] are other improvements that could be done to reduce the size of SHIPs formulation.

The complexity of the SHIPs formulation is $\mathcal{O}(rnm)$. Since $r = \mathcal{O}(n)$, the creation of the SAT-based model runs in $\mathcal{O}(n^2m)$. Afterwards, MINISAT, SATZ or other SAT solver can be applied to the formulation. Our experiments have been run using MINISAT which has shown to be the best solver for these problems.

The lower bound represents an essential role in SHIPs' performance. Indeed, tighter lower bounds on the value of r improve significantly SHIPs efficiency. The lower bounds applied by this SAT-based approach are the clique-based boundings of Section 3.3.1. In fact, the SHIPs version evaluated in this project uses the lower bound from Proposition 3.3.7. However, SHIPs with the improved bound from Algorithm 1 is significantly more efficient.

5.4 Pseudo-Boolean Optimization Approaches

Two facts inspired the creation of this work. On each of the three integer programming formulations (RTIP, PolyIP, HybridIP), variables can only take values in $\{0, 1\}$. Moreover, all coefficients appearing on the constraints and on the cost function are integers. These facts approximate the three formulations to SAT problems. Using a linear programming solver, as CPLEX, we are not taking advantage of these facts. On the other hand, SHIPs is a SAT-based approach, which performs very well in practice.

These results inspired the formulation of three Pseudo-Boolean Optimization models, similar to RTIP, PolyIP and HybridIP, which were we called RTPB, PolyPB and HybridPB, respectively. However, instead of applying ILP solvers (CPLEX) to find a solution to these formulations, we use modern Pseudo-Boolean solvers (MINISAT+ [8] and Pueblo [27]), which apply SAT techniques to solve the pseudo-boolean optimization problems.

The formulations T, RT, Poly and Hybrid were implemented in C code. The output of these four programs is a file with the respective cost function and constraints, in the format accepted by PBO solvers accept.

5.4.1 RTPB

The RTPB formulation is identical to RTIP, which is described in Section 5.1.1. Remember the TIP formulation, where we have two types of variables: x_k, representing each distinct haplotype h_k; and $y_{i,j}$, representing each haplotype pair j which can explain genotype g_i. Variable x_k is assigned value 1 when haplotype h_k is selected to partially explain any genotype. Otherwise, x_k is assigned value 0. Variable $y_{i,j}$ is set to 1 when the respective pair is chosen to explain genotype g_i and 0 otherwise.

$$\begin{aligned}
\text{minimize:} \quad & \textstyle\sum_{k=1}^{|H|} x_k \\
\text{subject to:} \quad & \\
& \textstyle\sum_{j=1}^{2^{h_i-1}} y_{i,j} = 1 && (5.27)\\
& y_{i,j} - x_{k_{i,j_1}} \leq 0 && (5.28)\\
& y_{i,j} - x_{k_{i,j_2}} \leq 0 && (5.29)\\
& x_k, y_{i,j} \in \{0, 1\}
\end{aligned}$$

$$1 \leq i \leq n; 1 \leq j \leq 2^{h_i-1}; 1 \leq k \leq 2^{h_i-1}$$

As a result, all variables are 0-1 integers and, clearly, this formulation constitutes a PBO problem. Moreover, all coefficients (either on constraints and on the cost function) are not only integers but also binary integers. In fact, this formulation is not directly a SAT problem only because there exists a cost function.

Although, the T formulation was implemented trivially from the mathematical model, the implementation of RT was not so easy. The reduced formulation of TIP uses reduction of discarding haplotypes that appear only once, which is very effective in practice. However, if this idea was not implemented in the best way, there would be no time/memory

benefits. In order to solve this problem, two hashtables were used on the program: one to store the pairs of haplotypes and the other to keep the distinct haplotypes.

Finally the result of this program is a file with the RT formulation (objective function and constraints) that is given as input to the PBO solver (MINISAT+), which outputs the variable attribution, i.e the solution of our problem. Joining the two steps, we obtain RTPB. If we use Pueblo instead of MINISAT+, we obtain a formulation named RTPueblo.

5.4.2 PolyPB

PolyPB has a simpler formulation than PolyIP. Although the basic formulation is the same, Brown and Harrower [3] implement several cuts and modifications in order to strengthen the formulation so that it would increase the efficiency of the IP solver. These modifications were required to solve LP relaxation efficiently. However, PolyPB does not need relaxations because it uses solvers which only admit boolean variables. As a result, modifications are not needed and the basic Poly formulation is sufficient.

Indeed, Poly is clearly a pseudo-boolean optimization problem. Note that, not only all variables are boolean, but also all coefficients are integers. In fact, all coefficients on the left-hand side have boolean values but some coefficients on the right-hand site may take values in the set $\{0, 1, \ldots, i-2\}$ (because of constraint (5.33)).

$$\begin{aligned}
&\text{minimize:} && \textstyle\sum_{i=1}^{2n} x_i \\
&\text{subject to:} && \\
& && y_{2i'-1,k} + y_{2i',k} = \begin{cases} 0 & \text{if} \quad g_{i'}[k] = 0 \\ 2 & \text{if} \quad g_{i'}[k] = 1 \\ 1 & \text{if} \quad g_{i'}[k] = 2 \end{cases} && (5.30) \\
& && d_{i,j} \geq y_{i,k} - y_{j,k} && (5.31) \\
& && d_{i,j} \geq y_{j,k} - y_{i,k} && (5.32) \\
& && x_i \geq 2 - i + \textstyle\sum_{j=1}^{i-1} d_{j,i} && (5.33) \\
& && x_i, y_{i,k}, d_{i,j} \in \{0,1\} && \\
& && 1 \leq i' \leq n; 1 \leq k \leq m; 1 \leq i \leq j \leq 2n && \\
& && && (5.34)
\end{aligned}$$

The PolyPB C-program directly encodes this formulation. The result of this program, a file with the Poly formulation (objective function and constraints), is given as input to a PBO solver, MINISAT+, which outputs the variable attribution, i.e the solution of our problem. This procedure is called PolyPB. If we use Pueblo instead of MINISAT+, we obtain PolyPueblo.

5.4.3 HybridPB

The HybridPB formulation is the same of HybridIP of page 33, except, again, that the introduction of cuts and modifications that is because unnecessary on HybridPB.

Again all variables of Hybrid take values on {0,1} and in all inequations coefficients are integers. Moreover, only inequations (5.11) and (5.20) could have non-boolean coefficients.

We enumerate genotypes by increasing number of heterozygous sites. Then, we expand haplotype parents of genotypes with less heterozygous sites, until we reach k distinct haplotypes. Let H_e be the set of these k haplotypes. We tested k values equal to 24 and 32. The model seems to be sensitive to this parameter; in fact, expanding only 24 haplotypes, the solver is significantly faster. The value chosen by Brown and Harrower was also $k = 24$. As a result, we have chosen to evaluate HybridPB with $k = 24$ in all experiments of Chapter 6.

After creating the Hybrid formulation, we apply it to MINISAT+ so we could get a solution to the HIPP problem. Joining the two steps, we get HybridPB. We also test Hybrid formulation with Pueblo, which has been called HybridPueblo.

Chapter 6

Experimental Results

6.1 Implementation

All results shown were obtained on a 1.9 GHz AMD Athlon XP with 1GB of RAM running RedHat Linux. To solve PBO instances, we use *MiniSat v1.13+* BigNum version [8]. Unless otherwise stated, the default conversion from PB-constraints to clauses was used. The *Pueblo version 1.4* [27] is used. For the ILP-based HIPP solvers, the ILP package used was CPLEX version 7.5.

6.2 Instances

Experimental results are obtained using both synthetic and real data. Problem instances can be organized into four groups. Two of those are synthetic and were obtained by simulation [3] and the other two are real biological data [3, 5, 6, 15, 16, 26]. Characteristics of the four classes are synthesized in Table 6.1, where, for each class, the number of instances, the minimum and maximum number of SNPs and the minimum and maximum number of genotypes are discriminated.

Class	#Inst	*min*SNP	*max*SNP	*min*Gen	*max*Gen
uniform	245	10	100	30	100
nonuniform	135	10	100	30	100
hapmap	24	30	75	7	68
biological	450	13	103	5	50
Total	854	10	103	5	100

Table 6.1: Classes of instances used

6.2.1 Synthetic Data

Synthetic instances were obtained from Brown and Harrower [3], and extended with additional, more complex problem instances, using Hudson's program, *ms* [14]. This program generates haplotype samples under a variety of assumptions about migration, recombination rate and population size. Then, randomly, those haplotypes are paired to generate

genotypes, using two different methods. In one case, distinct haplotypes are uniformly paired - uniform instances - so, all haplotypes are sampled with the same frequency. On the other case, the collection of haplotypes generated by Hudson's program are sampled non-uniformly; in this collection haplotypes may not be unique, so some haplotypes have higher probability of being chosen than others - nonuniform instances.

6.2.2 Biological Data

Part of the real data (24 HapMap instances) is used in [3] and was obtained from the HapMap project described in Chapter 2. Classes of instances from chromosomes 10 and 21 over all four HapMap populations (CEU, HCB, JPT, YRI) are considered. Sequences of lengths 30, 50, and 75, having SNPs from regions with small amounts of recombination, are used. Each of these 24 instances was reduced to contain a unique set of genotypes, so the actual number of genotypes in each sample varies significantly, generally increasing with sequence length (range from 7 to 68 genotypes).

Our Pseudo-Boolean approach is also tested in the biological data of some important studied diseases. These instances are grouped in five sets of important genes: ABCD, ACE, β_2AR, CF and IBD.

The first data set is from the 74 kb gene ABCD, responsible for P-glycoprotein [16]. This data set came from the University of California-San Francisco membrane transporter gene study, where 247 individuals were studied on 48 SNPs. In the samples we use, different number of individuals were tested (from 5 to 50 genotypes) and only 27 SNPs were considered. The maximum value for the number of haplotypes needed in this instances is 31, although this number is very unstable in general.

Angiotensin converting enzyme, also known as ACE [26], catalysis the conversion of angiotensin I to the physiologically active peptide angiotensin II. Due to its key function in the renin-angiotensin system, many association studies have been performed with DCP1. DCP1 is a gene encoding ACE and stretches over a genomic region of length 24 kb. The genomic sequences for this gene of 11 individuals were studied: 52 out of 78 SNPs were non-unique polymorphic sites and 13 distinct haplotypes were found. In this project, several samples were used, with the size of the population varying between 5 and 50 individuals for these 52 SNPs. As the size of the population grows, the number of haplotypes required by the pure parsimony approach increases. However, this number is never bigger than 13.

The β_2-adrenergic receptors (β_2AR) are G protein-coupled receptors that mediate the actions of catecholamines in multiple tissues [6]. Localization of SNPs and identification of haplotypes of the β_2AR gene are described in Table 2.1 in Chapter 2. For β_2AR gene, we tested instances with the number of genotypes varying between 5 and 50, always for the 13 SNPs.

The Cystic Fibrosis Gene [15], also known as CF, under some mutations, may confer residual pancreatic exocrine function in a subgroup of patients who are pancreatic sufficient. The ability to detect such mutations in the cystic fibrosis gene at the DNA level has important implications for genetic diagnosis. For this gene we tested several samples with a population varying between 5 and 50 individuals and in a total of 23 SNPs.

Finally, the most difficult data set from those five, IBD [5], is gene corresponding to an inflammatory bowel disease studied in father-mother-child trios. IBD extends over 500 kb

of the genome sequence and contain 103 SNPs typed in 387 individuals. In this analysis, we work with samples containing from 5 to 50 genotypes and 103 SNPs.

6.3 Experiments

6.3.1 Comparing Existing Approaches

First of all we will compare existing approaches to solve the HIPP problem. From Figure 6.1, we can derive some conclusions about state-of-the-art HIPP solvers. This graphic compares the performance, on the universe of 854 instances (not simplified), in 1000 seconds, of RTIP [11], PolyIP [2], HybridIP [3], HAPAR [31] and SHIPs [20]. We can conclude that IP approaches are very inefficient, solving only less than 50% of the instances. The branch and bound algorithm, HAPAR, is a little better as it is able to solve almost 60% of the problem instances. Finally, SHIPs shows to be a very efficient approach for the HIPP problem. In fact, it solves around 97% of the instances tested and is several orders of magnitude faster than integer linear programming and branch and bound solutions. However, SHIPs is not capable of solving some biological instances and further research should be done to improve this SAT-based technique.

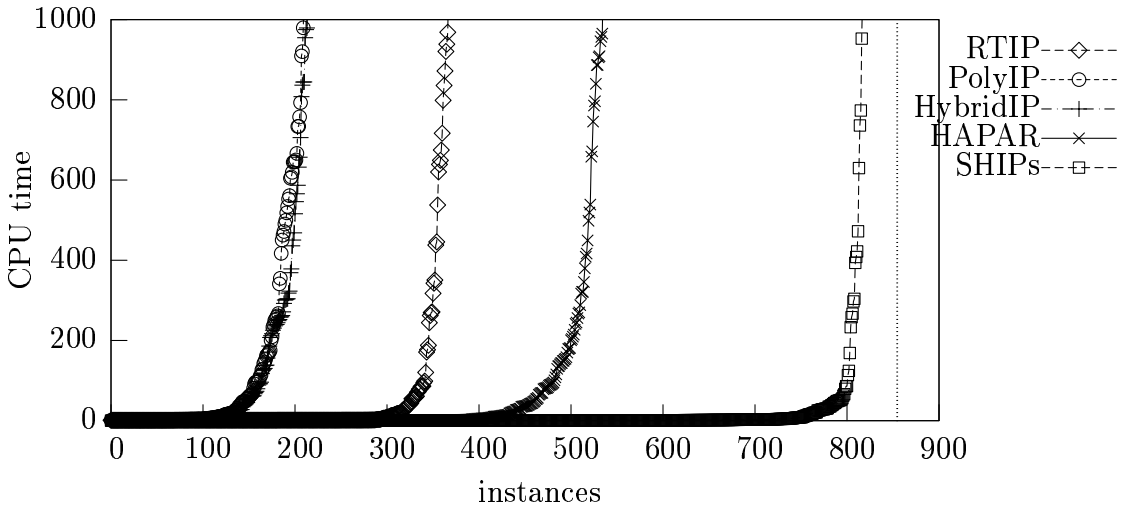

Figure 6.1: Relative performance of state-of-the-art HIPP solvers (1000s). For each solver, instances are ordered by increasing degree of complexity.

6.3.2 Analysing Effects of Simplification

All instances used from now on in this thesis were previously submitted to structural simplifications related in [3] and also used by SHIPs. The objective of this subsection is to show that these simplifications improve solvers performance. Consequently, applying simplifications before using any state-of-the-art solver improves efficiency and therefore it makes no sense to evaluate instances without being simplified.

As described in Section 5.1.4, page 34, these simplifications consist in removing duplicate genotypes, duplicate sites and complemented sites. It is straightforward to get a solution to the original instance once one has a solution to the simplified instance and to simplify an instance takes negligible time. Furthermore, note that the number of haplotypes used to explain a simplified or not simplified instance is the same. All methods, in general, perform better with simplified instances. Note that, in particular, the dimension of the problem decreases. In some solvers the efficiency improves very significantly. Figure 6.2 provides a scatter plot with the run time for RTIP (from Gusfield) applied to simplified instances and not simplified (original) instances, for a timeout of 1000 seconds. Points in the 10^3 line represent problem instances which exceed 1000 seconds without being solved. We could see that except for two instances, the performance of RTIP is

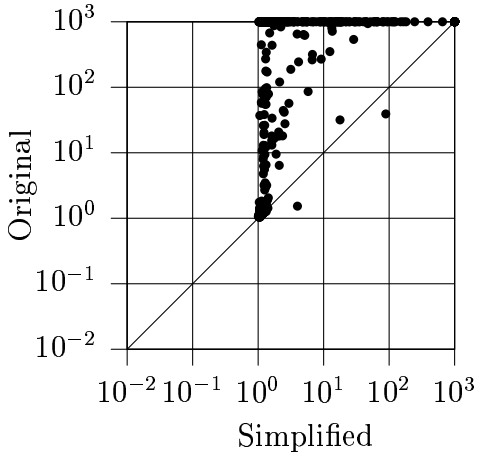

Figure 6.2: Performance of RTIP using simplified vs original instances (1000s)

really better on simplified instances. Moreover, when we simplify instances, RTIP is able to solve a larger number of instances. Indeed, in original instances, RTIP aborts in 487 out of 854 problem instances (about 57%) and in simplified instances RTIP aborts only 84 out of 854 instances (less than 10%), which is a significant difference. This big improvement happens in special with RTIP because of the reduction after TIP. Observe that the formulation RTIP, described in Chapter 5, does not expand pairs where both haplotypes do not partially explain any other genotype. The point is, that, in general, the number of these cases increases significantly when simplifications are considered.

Figure 6.3 compares the results for the same instances but applying PolyIP. Although it is not so evident as in RTIP, the performance, in general, improves with simplified instances. In fact, in original instances, PolyIP aborts more than 75% of the problem instances but in, simplified instances, PolyIP aborts less than 60%. The results of HybridIP are similar for those of PolyIP.

6.3.3 Solving HIPP using PBO

In this section we will make a detailed evaluation of our three PBO approaches, RTPB, PolyPB and HybridPB, described in Section 5.4, against state-of-the-art solvers: RTIP, PolyIP, HybridIP, HAPAR and SHIPs. Experimental results show that although RTPB

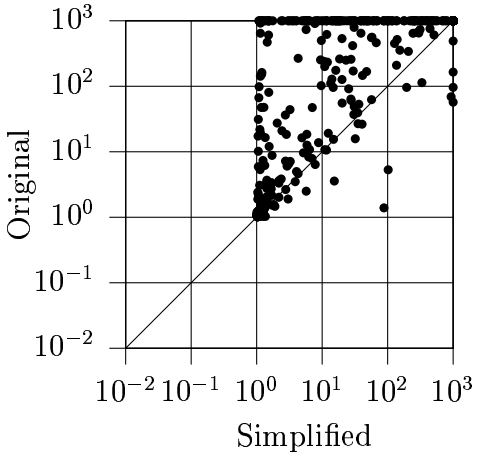

Figure 6.3: Performance of PolyIP using simplified vs original instances (1000s)

has a poor performance, PolyPB and HybridPB are two promising methods, being not only much better than the existing PolyIP and HybridIP, but also competitive with SHIPs.

First we evaluate each PBO method against the correspondent IP model.

RTPB versus RTIP

Comparing RTPB with RTIP, we conclude that, the PBO approach is very fast on solving small instances, but it is incapable of solving a large set of instances which RTIP is able to solve. Figure 6.4 provides a scatter plot comparing the CPU time required by RTIP

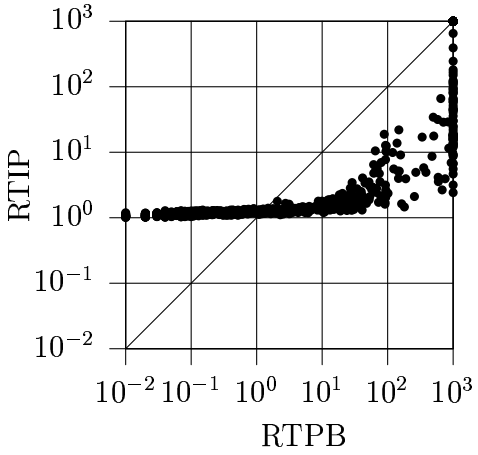

Figure 6.4: Performance of RTPB, using MiniSat+ option sorters, vs RTIP (1000s)

and RTPB (using MiniSat+, option sorters) for solving the 854 instances from Table 6.1, for a timeout of 1000 seconds. Each point represents an instance. Points on the 10^3 lines mean that the solvers are not able to solve the respective instance in 1000 seconds for one of two different reasons: it can exceed 1000 seconds without giving any solution or it can exceed available memory before 1000 CPU seconds. Note that when an instance exceeds

the memory limit, we have a more serious problem because there is no hope for instance will be solved by this solver in this machine. However, if the solver is still running after 1000 CPU seconds, we may wonder whether the instance would be solved with only one more second. Apart from these differences, all these instances are represented as a dot on the 10^3 lines. However, these CPU memory problems only happen with RTIP and RTPB formulations due to their exponential size.

From this plot we conclude that, despite the good performance of RTPB with the easier instances, RTIP performs really better on more complicated benchmarks. Moreover, RTIP can solve a bigger number of instances than RTPB. Comparing RTIP with RTPB, we can see that the PBO solver solves significantly less instances than RTIP. Indeed, RTIP is able to solve 770 (about 90%) of the problem instances, while RTPB is capable of solving only 706 instances (about 83%). In order to further understand the poor performance of

Table 6.2: Performance of RTPB (RT formulation using MiniSat+, option sorters) on the different classes of benchmarks (timeout 1000s). F&R groups the instances which are able to be formulated and resolved. F&NR (Time) and F&NR (Memory) group instances that can be formulated but abort in the resolution due to time or memory limitations, respectively. Finally, NF (Time/Memory) represents benchmarks which can not be formulated, either by time or memory limitations.

Benchmarks		F&R	F&NR(Time)	F&NR(Memory)	NF(Time/Memory)	Total
Uniform		242	0	3	0	245
Non-Uniform		135	0	0	0	135
Hapmap		17	0	5	2	24
Bio	ABCD	90	0	0	0	90
	ACE	69	0	21	0	90
	β_2AR	90	0	0	0	90
	CF	52	21	17	0	90
	IBD	11	0	7	72	90
TOTAL		706	21	53	74	854

RTPB, Table 6.2 describes the data in a more detailed way. In this table, benchmarks are separated into the classes described before. The F&R column refers to the 706 instances which can be solved by RTPB in less than 1000 seconds. The next two columns list instances which can formulated as PBO instances but cannot be solved due to time or memory limitations. Finally, the sixth column contains instances, which cannot be formulated. The majority of the instances that are solved by RTIP and that RTPB is not able to solve, have problems of memory. In fact, they were aborted by MINISAT+ before the time limit of 1000 seconds because they were exceeding the available machine memory. As we can see in Table 6.2 we have 53 instances aborted by MINISAT+. In fact, we use MINISAT+ with the option sorters because it performs better than other options. We conclude that, for these instances, CPLEX manages memory space better than MINISAT+.

If one uses Pueblo instead of MINISAT+, we get Figure 6.5. Again, we have efficiency on small instances but troubles with small-sized/difficult instances. Indeed, RTIP

is able to solve 575 (about 63%) of the problem instances, while RTPB is capable of solving only 364 instances (about 43%), which is worst than the result obtained by MINISAT+. The advantage of Pueblo is that only one instance aborted because of memory limitations. These results show that the PB approach is not promising when applied to

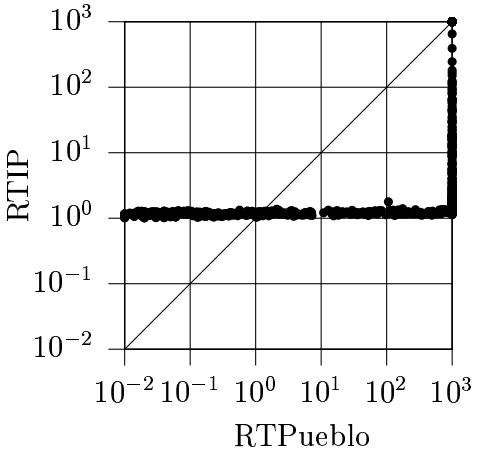

Figure 6.5: Performance of RTPueblo vs RTIP (1000s)

the RT formulation. This, however, does not represent a serious drawback because this exponential-sized formulation is not likely to be competitive in general.

PolyPB versus PolyIP

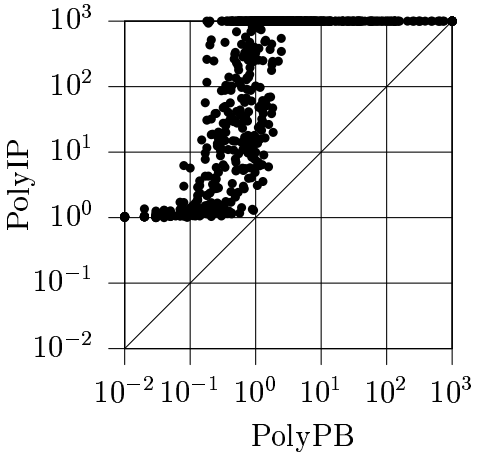

Figure 6.6: Performance of PolyPB vs PolyIP (1000s)

Table 6.3 lists the number of instances solved by each method in less than 1000 seconds. Note that PolyIP is able to solve 345 (out of 854) instances, i.e. about 40%. On the other hand, PolyPB can solve 828 (out of the same 854) instances, that is more than 96%. While the IP method has problems in every class of benchmarks, the PBO approach

Table 6.3: Number of instances, divided by classes, solved by PolyIP and PolyPB (timeout 1000s)

Benchmarks		PolyIP	PolyPB
Uniform		87/245	245/245
Nonuniform		19/135	135/135
HapMap		15/24	23/24
Biological	ABCD	31/90	90/90
	ACE	86/90	90/90
	β_2AR	52/90	90/90
	CF	28/90	87/90
	IBD	27/90	67/90
TOTAL		345/854	828/854

cannot solve only one HapMap instance (which has 68 genotypes with 26 sites, after simplification), three CF-instances (31,45 and 47 genotypes, each with 19 sites), and 23 IBD-instances (29-49 genotypes with 63-77 sites). Regarding the sintetic data, PolyIP solves less than 28% while PolyPB solves 100%. PolyIP can solve less than 51% of the real data, while PolyPB is able to solve more than 94%. As a result, we conclude that not only the performance of PolyPB is better than PolyIP, but also that PolyPB is a promising solution for the HIPP problem.

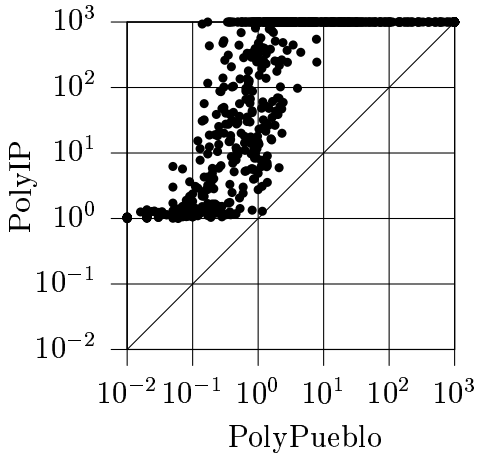

Figure 6.7: Performance of PolyPueblo vs PolyIP (1000s)

Figure 6.7 shows the results obtained if one uses Pueblo instead of MINISAT+. The performance of PolyPueblo is worse than PolyPB, but the point is that, in general, we have better results also using this PBO solver than using CPLEX. Indeed, PolyPueblo solves 87,7% of the problem instances, while PolyIP solves only about 40%. This makes it clear that the PBO formulation is more suitable than the IP formulation in this problem.

HybridPB versus HybridIP

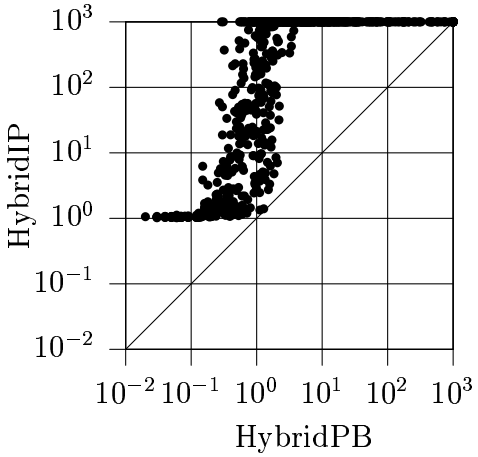

Figure 6.8: Performance of HybridPB vs HybridIP (1000s)

Table 6.4: Number of instances, divided by classes, solved by HybridIP and HybridPB (timeout 1000s)

Benchmarks		HybridIP	HybridPB
Uniform		83/245	245/245
Nonuniform		15/135	135/135
HapMap		15/24	22/24
Biological	ABCD	31/90	83/90
	ACE	86/90	90/90
	β_2AR	53/90	90/90
	CF	29/90	80/90
	IBD	27/90	75/90
TOTAL		339/854	820/854

Similarly to PolyPB, HybridPB also performs much better than the respective IP approach. Figure 6.8 compares the performance of the two Hybrid approaches when using up to 1000 seconds. As one can see HybridPB is faster than HybridIP solving all instances, and in most of them, is orders of magnitude faster. Moreover, HybridPB is able to solve 481 instances which HybridIP is not. Indeed, as Table 6.4 shows, the IP approach can solve less than 40% of the problem instances (339 out of 854), while the PBO method is capable of solving more than 96% (820 out of 854) instances. HybridIP solves less than 26% of the synthetic instances and less than 51% of the real instances, while HybridPB is able to solve all synthetic instances and more than 93% of the real instances.

If Pueblo, instead of MINISAT+, is used, then we obtain the scatter plot of Figure 6.9, for the 854 instances and a timeout of 1000 seconds.

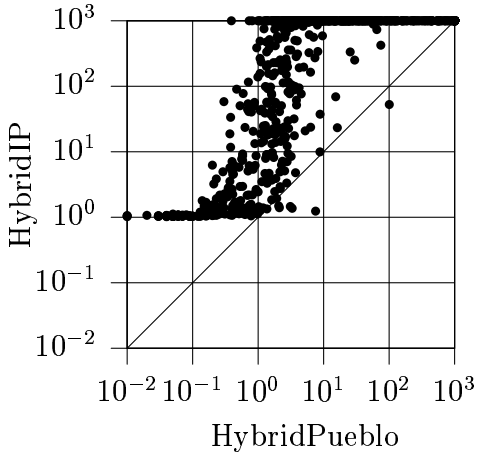

Figure 6.9: Performance of HybridPueblo vs HybridIP (10000s)

The performance of HybridPueblo is worse than HybridPB, but the point is that, in general, we have better results also using a PBO solver than using CPLEX. Indeed, HybridPueblo solves more than 82% of the problem instances, while HybridIP, as we have already seen, solves only about 40%.

PolyPB vs HybridPB

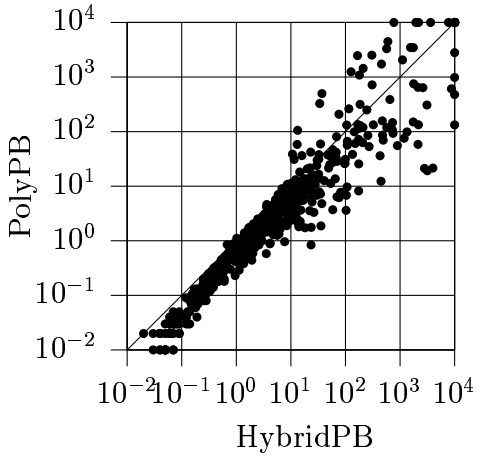

Figure 6.10: Performance of PolyPB vs HybridPB (10000s)

Here, we intend to compare the two promising approaches for the HIPP problem developed in this work. As already mentioned, the Hybrid approach was created in order to improve the Poly approach. However, the comparison is not so easy. Figure 6.10 compares the performance of PolyPB and HybridPB for the 854 instances, within 10000 seconds (we increase the time interval such that one can understand better the behavior of the methods). Although for a big set of instances, which both methods solve in less than 10 seconds, PolyPB seems to perform better than HybridPB, for the difficult instances

Table 6.5: Results: PolyPB vs HybridPB (timeout 10000s)

Benchmarks		PolyPB	HybridPB
Uniform		245/245	245/245
Nonuniform		135/135	135/135
HapMap		23/24	23/24
Biological	ABCD	90/90	88/90
	ACE	90/90	90/90
	β_2AR	90/90	90/90
	CF	90/90	88/90
	IBD	76/90	81/90
TOTAL		839/854	840/854

the two approaches seem to complement each other. Indeed, PolyPB performs better than HybridPB on about 93% of the problem instances, HybridPB surpasses PolyPB on less than 6% of the instances and none can solve 10 instances. Furthermore, PolyPB is able to solve 4 instances that HybridPB is not (2 from ABCD and 2 from CF) but HybridPB is capable of solving 5 instances (from IBD) which PolyPB cannot. Anyway, as the two methods do not differ significantly, we will compare HAPAR and SHIPs only with PolyPB. The results by comparing with HybridPB are similar.

PolyPB vs HAPAR

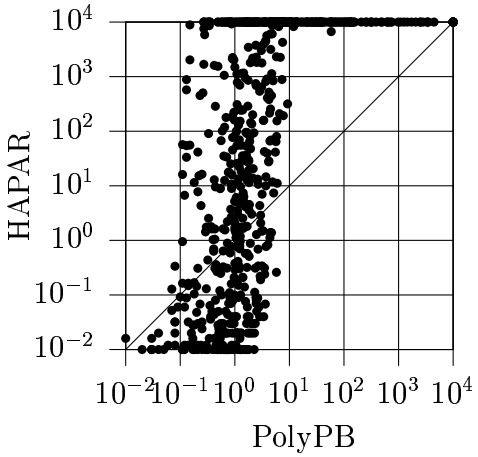

Figure 6.11: Performance of PolyPB vs. HAPAR (10000s)

The results of the comparison between PolyPB and HAPAR [31], for a timeout of 10000 seconds, are shown in Figure 6.11 and Table 6.6. HAPAR is faster than PolyPB on 377 instances out of 854 (about 44%), which both methods can solve in less than 10 seconds. In fact, almost all instances solved by HAPAR, are solved by PolyPB in less than 10 seconds. Moreover, PolyPB is capable of solving 233 instances (about 27%) which

Table 6.6: Number of instances, divided by classes, solved by PolyPB and HAPAR (time-out 10000s)

Benchmarks		HAPAR	PolyPB
Uniform		214/245	245/245
Nonuniform		88/135	135/135
HapMap		19/24	23/24
Biological	ABCD	68/90	90/90
	ACE	89/90	90/90
	β_2AR	90/90	90/90
	CF	25/90	90/90
	IBD	13/90	76/90
TOTAL		606/854	839/854

HAPAR is not. Indeed, HAPAR is not able to solve a big set of both real and synthetic data, almost from all classes.

The comparison between HAPAR and HybridPB is similar. HAPAR is faster than HybridPB on 400 instances (about 47%) but HybridPB can solve 234 instances that HAPAR cannot (more than 27%). Indeed, HybridPB solves 840 instances, while HAPAR only solves 606.

PolyPB vs SHIPs

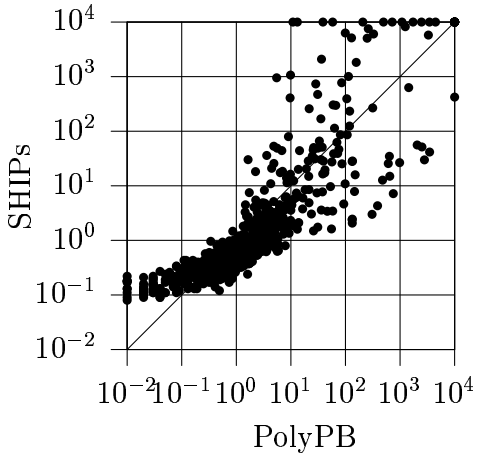

Figure 6.12: Performance of PolyPB vs SHIPs (10000s)

Finally, we compare PolyPB with SHIPs, a very efficient approach recently proposed [20]. Taking a look at Figure 6.12, we can see that, for a big cloud of points, SHIPs is faster than PolyPB. However, as the instance complexity increases, points become sparse in the graphic suggesting that each method complements each other. Then, analysing the points on the 10^4 seconds lines, we conclude that PolyPB are able to solve

Table 6.7: Number of instances, divided by classes, solved by PolyPB and SHIPs (timeout 10000s)

Benchmarks		SHIPs	PolyPB
Uniform		245/245	245/245
Nonuniform		135/135	135/135
HapMap		23/24	23/24
Biological	ABCD	90/90	90/90
	ACE	90/90	90/90
	β_2AR	90/90	90/90
	CF	90/90	90/90
	IBD	65/90	76/90
TOTAL		828/854	839/854

more instances than SHIPs. Indeed, SHIPs is faster than PolyPB on 563 instances (about 66%), but PolyPB is able to solve 12 instances in which SHIPs aborts. SHIPs is only able to solve 1 instance that PolyPB is not. As a result, PolyPB solves 839 out of 854 instances, while SHIPs solves 828. Note that, both methods are able to solve all synthetic data and all ABCD, ACE, β_2AR and CF instances. However, none of them can solve one HapMap instance (the same for both), and a different set of IBD instances.

Comparisons between SHIPs and HybridPB give similar results. In fact, SHIPs is faster than HybridPB in 80% of the benchmarks. However, HybridPB aborts 14 instances while SHIPs aborts 26.

From this analysis, we conclude that, although both SHIPS and PBO methods complement each other, PolyPB and HybridPB seem to be more robust.

Since PolyPB uses MINISAT+, which converts problem instances into SAT, and SHIPs also applies SAT, the most important conclusion of these results is that SAT-based methods are promising to solve the HIPP problem.

6.3.4 Other Experiments

Despite the good performance of PolyPB, some ideas came up to improve this HIPP solver. However, these ideas have not improved the results. Motivated by SHIPs, we tried to introduce some lower or upper bounds in PolyPB. Note that, if we know that the minimum number of haplotypes needed to explain the set of genotypes is superior to LB or inferior to UB, we can enrich the formulation adding the constraint $f(x) \geq LB$ or $f(x) \leq UB$, respectively. One would think that this process would make the solver faster, as it prunes the search space. However, the experiments have shown that, in general, imposing a lower bound brings no benefits and introducing an upper bound degrades the performance of PolyPB.

A new model (Poly+) based on the Poly model was also tested. The difference of this new model resides in the definition of the d_{ij} and x_i variables. In Poly+, $d_{ij} = 1$ if and only if haplotypes h_i and h_j are different, and $x_i = 1$ if and only if h_i is unique with

respect to $h_1, \ldots, h_{i-1}$. Remember that those definitions in Poly (see Section 5.1.2) are implications and not equivalences. This modification was supposed to increase constraint propagation in the PBO solver. However, maybe because the number of variables increases (some auxiliary variables were needed), or maybe because it reduces the search space, this modification proved unsuccessful.

Conclusions

Haplotyping has been a priority in human genomics and pure parsimony has been shown to be an accurate method for solving haplotype inference. This thesis is, first of all, an overview of the work developed in Haplotype Inference by Pure Parsimony, compiling several different methods for solving the problem. None of those is sufficiently efficient for solving all the instances of a set with biological interest, and consequently it is important to pursue research on more capable techniques.

Inspired by the good performance of a SAT-based approach (SHIPs) and based in three integer linear programming approaches (RTIP, PolyIP and HybridIP), three new methods were proposed in this project (respectively, RTPB, PolyPB and HybridPB). These approaches use formulations similar to the respective IP formulations. However, unlike those, the new approaches apply modern pseudo-boolean solvers (based on SAT) in order to find the required solution.

Our experiments show that although RTPB has no significant benefits, PolyPB and HybridPB are very efficient methods to solve the parsimonious problem. RTPB solves easy benchmarks very quickly, although this method leads to exponential-sized problems and therefore it is not able to solve a large set of instances. PolyPB and HybridPB are not only faster on solving all instances tested but also capable of solving a significant bigger number of benchmarks in a reasonable time. Moreover, PolyPB and HybridPB solve significantly more instances than HAPAR and are competitive with SHIPs, two state of the art solutions to this problem. Furthermore, these two PBO approaches are capable of solving several problem instances that no other available HIPP solver manages to solve. From the comparison with SHIPs we are inclined to conclude that the good performance of SHIPs can be explained by the efficiency of modern SAT-solvers. Indeed, SAT-based PBO solvers obtain extremely good results with PolyPB and HybridPB, which are PBO formulations that differ significantly from the SHIPs SAT-based approach.

Glossary

Biology

- Allele: The alternative forms of the genetic character found at a given locus on a chromosome. [Source: Purves et al. Life: The Science of Biology]
- Diploid: The chromosome complement consists of two copies (homologues) of each chromosome. In humans, each chromosome pair is from a different origin (mother, father). [Source: Purves et al. Life: The Science of Biology]
- Genotype: An exact description of the genetic constitution of an individual, with respect to a single trait or a larger set of traits. [Source: Purves et al. Life: The Science of Biology]. The genetic constitution of an organism as revealed by genetic or molecular analysis, i.e. the complete set of genes, both dominant and recessive, possessed by a particular cell or organism.[Source: IUPAC Biotech]
- Haplotype (haploid genotype): A combination of alleles of closely linked loci that are found in a single chromosome and tend to be inherit together. The linear, ordered arrangement of alleles on a chromosome. [Source: Purves et al. Life: The Science of Biology]
- Heterozygous: A diploid organism having different alleles of a given gene on both homologous chromosomes. [Source: Purves et al. Life: The Science of Biology]
- Homozygous: A diploid organism having identical alleles of a given gene on both homologous chromosomes. [Source: Purves et al. Life: The Science of Biology]
- Phenotype: The observable properties of an individual as they have developed under the combined influences of the individual's genotype and the effects of environmental factors. [Source: Purves et al. Life: The Science of Biology]

Bibliography

[1] Oswald T. Avery, Colin M. MacLeod, and Maclyn McCarty. Studies on the chemical nature of the substance inducing transformation of the pneumococcal types. *The Journal of Experimental Medicine*, 79(2):137–158, 1944.

[2] 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)*, pages 254–265.

[3] 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, April-June 2006.

[4] A. G. Clark. Inference of haplotypes from PCR-amplified samples of diploid populations. *Molecular Biology and Evolution*, 7(2):111–122, 1990.

[5] 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.

[6] 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, September 2000.

[7] N. Eén and N. Sörensson. An extensible SAT-solver. In *International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 502–518, 2003.

[8] N. Eén and N. Sörensson. Translating pseudo-Boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–26, 2006.

[9] D. Gusfield. Inference of haplotypes from samples of diploid populations: complexity and algorithms. *Journal of Computational Biology*, 8(3):305–324, August 2001.

[10] D. Gusfield. Haplotyping as perfect phylogeny: conceptual framework and efficient solutions. In *RECOMB*, pages 166–175, 2002.

[11] D. Gusfield. Haplotype inference by pure parsimony. In *14th Annual Symposium on Combinatorial Pattern Matching (CPM'03)*, pages 144–155, 2003.

[12] E. Hubbell, 2000. Personal Communication to Dan Gusfield.

[13] R. Hudson. Gene genealogies and the coalescent process. *Oxford Survey of Evolutionary Biology*, 7:1–44, 1990.

[14] R. R. Hudson. Generating samples under a Wright-Fisher neutral model of genetic variation. *Bioinformatics*, 18(2):337–338, February 2002.

[15] 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.

[16] 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.

[17] 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.

[18] C. M. Li and Anbulagan. Look-ahead versus look-back for satisfiability problems. In *International Conference on Principles and Practice of Constraint Programming (CP)*, pages 341–355, October 1997.

[19] I. Lynce and J. Marques-Silva. Efficient haplotype inference with Boolean satisfiability. In *National Conference on Artificial Intelligence (AAAI)*, July 2006.

[20] 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)*, pages 136–141, 2006.

[21] V. Manquinho and O. Roussel. The first evaluation of Pseudo-boolean solvers (PB'05). *Journal on Satisfiability, Boolean Modeling and Computation*, 2:103–143, 2006.

[22] J. P. Marques-Silva and K. A. Sakallah. Grasp: A new search algorithm for satisfiability. In *ACM/IEEE International Conference on Computer-Aided Design*, pages 220–227, November 1996.

[23] 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.

[24] S. H. Orzack, D. Gusfield, J. Olson, S. Nesbitt, L. Subrahmanyan, and V. P. Stanton. Analysis and exploration of the use of rule-based algorithms and consensus methods for the inferral of haplotypes. *Genetics*, 165:915–928, 2003.

[25] P.W. Purdom and C.A. Brown. The pure literal rule and polynomial average time. *SIAM J. Computing*, 14:943–953, 1985.

[26] 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.

[27] 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.

[28] M. Stephens, N. Smith, and P. Donelly. A new statistical method for haplotype reconstruction. *American Journal of Human Genetics*, 68:978–989, 2001.

[29] S. Tavare. Calibrating the clock: Using stochastic processes to measure the rate of evolution. *Calculating the Secrets of Life. National Academy Press*, 1995.

[30] The International HapMap Consortium. A haplotype map of the human genome. *Nature*, 437:1299–1320, 27 October 2005.

[31] L. Wang and Y. Xu. Haplotype inference by maximum parsimony. *Bioinformatics*, 19(14):1773–1780, 2003.

[32] J. D. Watson and F. H. C. Crick. Molecular structure of nucleic acids. *Nature*, 171:737–738, 1953.