A Set Covering Approach with Column-Generation for Parsimony Haplotyping

Giuseppe Lancia* Paolo Serafini†

Abstract

We introduce an exact algorithm, based on Integer Linear Programming, for the parsimony haplotyping problem (PHP). The PHP uses molecular data and is aimed at the determination of a smallest set of *haplotypes* that explain a given set of *genotypes*. Our approach is based on a Set Covering formulation of the problem, solved by branch and bound with both column- and row- generation. Existing ILP methods for the PHP suffer from the large size of the solution space, when the genotypes are long and with many *heterozygous* sites. Our approach, on the other hand, is based on an effective implicit representation of the solution space, and allows the solution of both real-data and simulated instances which are very hard to solve for other ILPs.

1 Introduction

A *single nucleotide polymorphism* (SNP, pronounced "snip") is a site of the human genome (i.e., the position of a specific nucleotide) showing a statistically significant variability within a population. Besides very rare exceptions, at each SNP site only two nucleotides (out of `A`, `T`, `C` and `G`) are observed, and they are called the SNP *alleles*. The recent completion of the sequencing phase of the Human Genome Project [19, 28] has shown that the genomes of any two individuals are more than 99% identical, and that most polymorphisms (i.e., differences at genomic level) are in fact SNPs, occurring, on average, one every thousand bases. Being the predominant form of human polymorphism, the importance of SNPs can hardly be overestimated. Nowadays, SNPs are widely used in therapeutic, diagnostic, and forensic applications. Several projects are currently devoted to comparing the genetic sequences of different individuals to identify chromosomal regions where genetic variance are shared. Among these, we recall the international consortium HapMap [8, 9] and the SeattleSNPs [2].

Humans are *diploid* organisms, i.e., their DNA is organized in pairs of chromosomes. For each pair of chromosomes, one chromosome copy is inherited from the father and the other copy is inherited from the mother. For a given SNP, an individual can be either *homozygous* (i.e., possess the same allele on both chromosomes) or *heterozygous* (i.e., possess two different alleles). The values of a set of SNPs on a particular chromosome copy define a *haplotype*.

*Dip. di Matematica e Informatica, University of Udine, Italy
†Dip. di Matematica e Informatica, University of Udine, Italy

Hapl. 1, paternal: taggtccCtatttCccaggcgcCgtatacttcgacgggTctata
Hapl. 1, maternal: taggtccGtatttAccaggcgcGgtatacttcgacgggTctata

Hapl. 2, paternal: taggtccCtatttAccaggcgcGgtatacttcgacgggTctata
Hapl. 2, maternal: taggtccGtatttCccaggcgcGgtatacttcgacgggCctata

Hapl. 3, paternal: taggtccCtatttAccaggcgcGgtatacttcgacgggTctata
Hapl. 3, maternal: taggtccGtatttAccaggcgcCgtatacttcgacgggCctata

Figure 1: The haplotypes of 3 individuals, with 4 SNPs.

In Fig. 1, we illustrate a simplistic example of three individuals and four SNPs. The alleles for SNP 1, in this example, are C and G. Individual 1, in this example, is heterozygous for SNPs 1, 2 and 3, and homozygous for SNP 4. His haplotypes are `CCCT` and `GAGT`.

Haplotyping an individual consists of determining his two haplotypes, for a given chromosome. With the larger availability in SNP genomic data, recent years have seen the birth of many new computational problems related to haplotyping (see [14] for a survey on haplotyping). These problems are motivated by the fact that it is economically infeasible to determine the haplotypes experimentally. On the other hand, there is a cheap experiment which can determine the (less informative and often ambiguous) genotypes, from which the haplotypes must then be retrieved computationally.

A *genotype* provides information about the multiplicity of each SNP allele: i.e., for each SNP site, the genotype specifies if the individual is heterozygous or homozygous (in the latter case, it also specifies the allele). The ambiguity comes from heterozygous sites, since, to retrieve the haplotypes, one has to decide how to distribute the two allele values on the two chromosome copies. *Resolving* a genotype requires to determine two haplotypes such that, if they are assumed to be the two chromosome copies, then the multiplicity of each SNP allele yields exactly that genotype. Note that, for a genotype with k heterozygous sites, there are 2^{k-1} pairs of distinct haplotypes that could resolve the genotype. Given a set of genotypes, the general *haplotyping* problem requires to determine a set of haplotypes such that each genotype is resolved by two haplotypes. For its importance (as we said, haplotyping from genotype data is nowadays the only viable way) the haplotyping problem has been extensively studied, under many objective functions, among which: *Perfect phylogeny* [4, 10], *Clark's rule* [11, 12] and *Parsimony* [13, 22, 5]. Each model and objective function has specific biological motivations, which are discussed in the cited references. Furthermore, there exist several popular statistically-based programs such as PHASE and FastPhase [26, 27], Haplotyper [25] and Gerbil [21].

In this paper we pursue the Parsimony Haplotyping Problem (PHP), i.e., we are interested in finding a smallest-size set of resolving haplotypes. Among the several objective functions for haplotyping, parsimony is a model whose importance is now being recognized as crucial also in the solution of more complex haplotyping problems.

The computational study of the problem has been first investigated by Gusfield [13], who credited Earl Hubbel for proposing the model and showing its NP-hardness. Gusfield

adopted an Integer Programming formulation for its practical solution. The problem is also APX-hard [22], i.e., there exists a constant $\delta > 1$ such that finding a δ-approximate solution is already NP-hard. On the other hand, there are special cases of the problem that are polynomially solvable [15, 23]. In the last few years, many optimization approaches have been tried for the solution of the PHP. In particular, there have been

- *Integer programming formulations of worst-case exponential size.* The first such formulation was given by Gusfield [13], and has $O(2^n)$ variables and $O(m2^n)$ constraints in worst-case. Gusfield managed to employ some preprocessing reduction rules, to get rid of some variables that can be proved to be non-essential in the formulation. Although practically useful, the rule still leaves an exponential model, whose size grows quite quickly with respect to the instance size. In [13] the experimental results show that this model can be used to tackle problems with up to 50 genotypes, over 30 SNPs, with relatively small levels of heterozygosity. No column-generation technique is used, and none seem suitable for the proposed model.

- *Integer programming formulations of polynomial size and hybrid formulations.* Many authors (most prominently, Brown and Harrower [5], but see also [16, 22]) have independently proposed polynomially-sized integer programming formulations for the PHP. The Linear Programming relaxation of these formulations is quite weak, i.e., it usually yields a lower bound much smaller than the optimal solution. The addition of some valid cuts, as proposed in [5] improves the quality of the bound, but the optimality gap remains large, especially if compared to that of the exponential formulation. In [6, 7], Brown and Harrower propose an hybrid model, in which a fixed set of some (but not all) haplotypes are explicitly present (as in the exponential formulation of Gusfield), while the others are implicitly represented by a polynomial set of variables and constraints. The quality of the bound depends on the choice of the explicit haplotypes, but still is inferior to that of the exponential model. The polynomial/hybrid formulations were successfully used for the solution of problems of similar size as the exponential model. Furthermore, some tests were conducted on slightly larger problems, on which the exponential formulation could not be applied successfully due to the IP size.

- *Quadratic, semi-definite programming approaches, of exponential size.* A quadratic formulation, solved by semi-definite programming, was proposed by Kalpakis and Namjoshi [20]. Similarly to the exponential IP, the formulation has a variable for each possible haplotype (i.e., it has $O(2^n)$ variables) and hence it cannot be used to tackle instances for which the set of possible haplotypes is too large. The size of the problems solved is comparable to the other methods. Based on a similar formulation, an (exponential-time) approximation algorithm is presented in [17].

- *Combinatorial branch-and-bound approaches.* In [29], Wang and Xu propose a simple combinatorial branch-and-bound approach. The solution is built by enumerating all possible resolutions for each of the genotypes in turn. The lower bound is the number of haplotypes used so far. Since the search space is exponential, and the bound is weak, the method is not able to solve instances of size comparable to the other approaches.

Even the solution for 20 genotypes over 20 SNPs can sometimes take an extremely long time to be found.

- *SAT approaches.* In [24], Lynce and Marques-Silva explore the use of a SAT approach for haplotype resolution. The approach is based on formulating the haplotyping problem as a polynomial-size set of boolean formulas (somewhat similar to the ILP model in [5]) which, for a given target value t, is satisfiable if and only if there exists a resolution of at most t haplotypes. To find the optimal solution, a MinSAT solver is called iteratively until the smallest feasible target value is reached. Experimental results show good performance of this method over instances of size comparable to the above approaches.

Generally speaking, most of the above mentioned models run into troubles when trying to solve "large" problems (where the most critical parameter is the number of heterozygous sites per genotype). The exponential IP models imply the creation of too many variables and/or constraints for obtaining a solution within a reasonable time (and, sometimes, the model turns out to be too big to be even input into the solver). The polynomial IP and combinatorial models, on the other hand, employ quite weak lower bounds, so that closing the gap and terminating the branch-and-bound search is again impossible within a reasonable time. Given that the exponential IP formulation turns out to be very tight, ideally we would like to be able to attain a similarly tight bound while avoiding the presence of the exponentially many variables and/or constraints. This goal can be achieved via an approach that relies only a "small" subset of variables and constraints, and that sparingly adds variables and constraints only when they are in fact needed. In this paper we describe such an approach.

Here, we propose an Integer Programming formulation for the PHP with an exponential number of variables and constraints, but in which variables and cuts are added dynamically when needed. The approach creates haplotypes at run-time, through a standard *pricing* procedure, i.e., depending on their reduced costs. These, in turn, depend on the dual variables of the current LP relaxation.

Our approach is based on formulating the PHP as a particular *Set Covering* (SC) problem, in which each possible haplotype corresponds to a set. The Set Covering formulation has an exponential number of variables and constraints. However, we describe pricing and separation procedures that generate variables and violated inequalities at run-time. This way, we are able to tackle instances that cannot even be input to the software based on the best previous formulations. We were able to solve to proven optimality instances for which the total number of haplotypes is in the order of billions. The good performance of the approach is due to two main reasons: on one hand, the LP relaxation bound of the SC model is quite strong; on the other hand, we use an effective heuristic, based on the LP solution, to find good PHP feasible solutions.

Paper organization. The paper is organized as follows. In Section 2 we introduce the basic notation and definitions. In Section 3 we describe the SC formulation and we give a high-level description of the Branch-and-Price-and-Cut approach. In Section 4 we describe

Haplotype 1, paternal:	0 1 0 1	
		2 2 2 1 Genotype 1
Haplotype 1, maternal:	1 0 1 1	
Haplotype 2, paternal:	0 0 1 1	
		2 2 1 2 Genotype 2
Haplotype 2, maternal:	1 1 1 0	
Haplotype 3, paternal:	0 0 1 1	
		2 0 2 2 Genotype 3
Haplotype 3, maternal:	1 0 0 0	

Figure 2: Haplotypes and corresponding genotypes.

the data structures needed by our algorithm. In Section 5 we describe the pricing and separation strategies. In Section 6 we describe the heuristic used to generate feasible solutions. In Section 7 we report on our computational experiments. Some conclusions are drawn in Section 8.

2 Notation and definitions

Given a set of n SNPs, fix arbitrarily a binary encoding of the two alleles for each SNP (i.e., call one of the two alleles '0' and the other '1'). Once the encoding has been fixed, each haplotype corresponds to (with a slight abuse of terminology, hereafter, we will say that each haplotype *is*) a binary vector of length n.

For a haplotype h, we denote by h_i the value of its i-th component, with $i = 1, \ldots, n$. Given two haplotypes h' and h'', their sum is a vector $h' \oplus h''$, where the binary operator $\oplus$ is defined, component-wise, as

$$(h' \oplus h'')_i := \begin{cases} 0 & \text{if } h'_i = h''_i = 0 \\ 1 & \text{if } h'_i = h''_i = 1 \\ 2 & \text{if } h'_i \neq h''_i. \end{cases}$$

A vector $g = h' \oplus h''$ is called a *genotype*. In general, we call genotype any vector $g \in \{0, 1, 2\}^n$.

Definition 1 (**Ambiguity**). *Let g be a genotype. Each position i such that $g_i = 2$ is called an* ambiguous position. *By $A(g)$ we denote the set of ambiguous positions of g. A genotype is* ambiguous *if it has more than one resolution, i.e., if $|A(g)| \geq 2$.*

In the biological interpretation, genotype entries with value 0 or 1 correspond to homozygous SNP sites, while ambiguous positions correspond to heterozygous sites. In Fig. 2, we illustrate a case of three individuals, showing their haplotypes and genotypes.

Definition 2 (**Resolution**). *For g a genotype, a pair of haplotypes $\{h', h''\}$ such that $g = h' \oplus h''$ is a* resolution *of g. The haplotypes h' and h'' are said to* resolve *g. Let G be a set*

of genotypes and H be a set of haplotypes such that each g has a resolution in H. Then H resolves *G, and is called a* resolving set *for G.*

The *parsimony haplotyping problem* (PHP) studied in this paper, is formally defined as follows:

INSTANCE: A set G of m genotypes of length n each.

PROBLEM: Find a resolving set $\hat{H}$ for G of minimum cardinality.

Definition 3 (**Compatibility).** *A haplotype h is* compatible *with a genotype g if $g_i = h_i$ whenever $g_i \neq 2$. Two genotypes g and g' are* compatible *if $g_i = g'_i$ whenever both $g_i \neq 2$ and $g'_i \neq 2$. Notice that two genotypes are compatible if and only if they share at least one compatible haplotype.*

Clearly, a genotype can be resolved only by compatible haplotypes. The notion of compatibility between genotypes can be extended to a whole set of genotypes.

Definition 4 (**Compatibility Graph).** *Given a set G of genotypes, the* compatibility graph *over G is a graph which has vertex set G and an edge between each pair of compatible genotypes.*

Definition 5 (**Cliques and stable sets of genotypes).** *A set K of genotypes is a* clique of genotypes *(or, for short, a* clique*) if its compatibility graph is a clique. A set S of genotypes is a* stable set of genotypes *(or, for short, a* stable set*) if its compatibility graph is a stable set.*

The definition implies that K is a clique if and only if there exists at least one haplotype compatible with each g in K. If we represent K as a $|K| \times n$ matrix A over $\{0, 1, 2\}$, where rows correspond to elements of K, then K is a clique if and only if no column of A has both a 0 and a 1 on two rows. From the definition it follows that it is impossible to resolve a stable set S with less than $2\,|S|$ haplotypes. It also follows that twice the stability number of the compatibility graph is a lower bound for the PHP.

For a genotype g, we denote by $H(g)$ the set of haplotypes that are compatible with g. Given a set of genotypes G, let $H_G = \bigcup_{g \in G} H(g)$. Conversely, for a haplotype h, we denote by $G(h)$ the set of all genotypes $g \in G$ such that g is compatible with h.

Definition 6 (**s-cliques).** *Given a set G of genotypes, a set $K \subseteq G$ is a* selectable clique *(or, shortly, an* s-clique*) if there exists a haplotype h such that $K = G(h)$. Such an h is called a* selector *of K. We denote by $H(K) := \{h \,|\, G(h) = K\}$.*

Note that $G(h)$ is a clique, but not all cliques are selectable. Given a set G of genotypes, we denote by

$$\mathcal{K} := \{G(h) \,|\, h \in H_G\}$$

the family of all s-cliques. The family $\mathcal{K}$ induces a partition of H_G into the following sets:

$$\{H(K) \,|\, K \in \mathcal{K}\}$$

i.e., H_G is partitioned into the sets of selectors of the same s-clique.

Example 1. Let $G = \{g^1, g^2, g^3, g^4, g^5, g^6\} = \{22021, 12222, 12211, 20120, 00102, 02120\}$. The compatibility graph has edges (g^1, g^2), (g^1, g^3), (g^2, g^3), (g^2, g^4), (g^4, g^5), (g^4, g^6) and (g^5, g^6). The previous definitions identify the following sets:

$$H(g^1) = \{00001, 00011, 01001, 01011, 10001, 10011, 11001, 11011\} = \{h^1, h^2, h^3, h^4, h^5, h^6, h^7, h^8\},$$

$$\begin{aligned} H(g^2) &= \{10000, 10001, 10010, 10011, 10100, 10101, 10110, 10111, \\ &\qquad 11000, 11001, 11010, 11011, 11100, 11101, 11110, 11111\} \\ &= \{h^9, h^5, h^{10}, h^6, h^{11}, h^{12}, h^{13}, h^{14}, h^{15}, h^7, h^{16}, h^8, h^{17}, h^{18}, h^{19}, h^{20}\}, \end{aligned}$$

$$H(g^3) = \{10011, 10111, 11011, 11111\} = \{h^6, h^{14}, h^8, h^{20}\},$$

$$H(g^4) = \{00100, 00110, 10100, 10110\} = \{h^{21}, h^{22}, h^{11}, h^{13}\},$$

$$H(g^5) = \{00100, 00101\} = \{h^{21}, h^{23}\},$$

$$H(g^6) = \{00100, 00110, 01100, 01110\} = \{h^{21}, h^{22}, h^{24}, h^{25}\}.$$

The s-cliques are

$$K_1 = \{g^1\}, \quad K_2 = \{g^1, g^2\}, \quad K_3 = \{g^1, g^2, g^3\}, \quad K_4 = \{g^2\}, \quad K_5 = \{g^2, g^3\},$$

$$K_6 = \{g^2, g^4\}, \quad K_7 = \{g^4, g^6\}, \quad K_8 = \{g^4, g^5, g^6\}, \quad K_9 = \{g^5\}, \quad K_{10} = \{g^6\}.$$

The partition of H_G induced by the s-cliques is:

$$H(K_1) = \{h^1, h^2, h^3, h^4\}, \quad H(K_2) = \{h^5, h^7\}, \quad H(K_3) = \{h^6, h^8\},$$

$$H(K_4) = \{h^9, h^{10}, h^{12}, h^{15}, h^{16}, h^{17}, h^{18}, h^{19}\}, \quad H(K_5) = \{h^{14}, h^{20}\}, \quad H(K_6) = \{h^{11}, h^{13}\},$$

$$H(K_7) = \{h^{22}\}, \quad H(K_8) = \{h^{21}\}, \quad H(K_9) = \{h^{23}\}, \quad H(K_{10}) = \{h^{24}, h^{25}\}.$$

An optimal resolving set is $\{h^1, h^8, h^{11}, h^{13}, h^{14}, h^{21}, h^{23}, h^{25}\}$.

In the remainder of the paper, we will assume that all genotypes in G are ambiguous. This can be done without loss of generality, as shown by the following reasoning. When a genotype is non-ambiguous, it is itself a haplotype which could resolve other genotypes. If $\bar{g}$ is a non-ambiguous genotype, then the haplotyping problem can be reformulated by adding a $(n+1)$-th SNP to each genotype, such that $\bar{g}_{n+1} = 2$ and $g_{n+1} = 1$ for each $g \neq \bar{g}$. It is easy to see that there is a one-to-one correspondence between a resolving set H of the original problem and a resolving set $\bar{H}$ of the reformulated problem such that $|H| + 1 = |\bar{H}|$. Hence solving PHP for one problem is equivalent to solving it for the other (with difference one in cardinality). If the non-ambiguous genotypes are more than one, the procedure is simply iterated.

3 The Set Covering model

In this section we describe a Set Covering formulation for the PHP problem, and a Branch-and-Cut-and-Price approach for its solution.

The main idea on which we base our approach relies on the following observation: if H is a set of haplotypes resolving G, then a strong condition is implied for the haplotypes in H, namely,

> **Covering condition:** For each genotype g, position $i \in A(g)$ and value $a \in \{0,1\}$, there is a haplotype $h \in H \cap H(g)$, such that $h_i = a$.

This condition is only necessary, but not sufficient, for H to be a feasible solution of the PHP. Consider, for example, the following "diagonal" instance,

$$G = \begin{bmatrix} 1222 \\ 2122 \\ 2212 \\ 2221 \end{bmatrix} \tag{1}$$

for which the set $\{0111, 1011, 1101, 1110\}$ satisfies the covering condition but does not resolve G.

We call this condition a *covering* condition because we may define a ground set as the set of all triples (g, i, a) such that $g \in G$, $i \in A(g)$ and $a \in \{0,1\}$. Then we may identify a haplotype h with a subset of the ground set, namely

$$h \leftrightarrow \{(g,i,a) \,|\, h \in H \cap H(g)\,,\; h_i = a\}$$

and the covering condition requires that H covers the ground set. It then makes sense to solve this set covering problem optimally, i.e., to minimize the number of haplotypes needed to satisfy the covering condition. This problem is in fact a relaxation of the PHP. It is possible that its optimal solution resolves G. If not, we may still obtain a good feasible PHP solution from the optimal cover by adding only a small number of haplotypes. Furthermore, we can find a PHP optimal solution by strengthening the covering conditions, as we later show.

Notice that in a PHP solution two haplotypes are enough to cover all ambiguous positions of an ambiguous genotype. On the other hand, it is impossible to satisfy the covering conditions of an ambiguous genotype with less than two haplotypes. Hence, the intuition is that good set covering solutions tend to be good for the PHP problem as well.

Let g be a genotype and $i \in A(g)$ be an ambiguous position of g. For $a = 0, 1$, we define the set $H_i^a(g)$ to be the haplotypes compatible with g that have the value a in position i. Then, the sets $H_i^0(g)$ and $H_i^1(g)$ are a partition of $H(g)$. If g has $k \geq 1$ ambiguous positions, then $|H_i^0(g)| = |H_i^1(g)| = 2^{k-1}$.

For each $h \in H_G$ we introduce a 0-1 variable x_h, where $x_h = 1$ if h is taken in the solution. Since variables x_h and haplotypes h are in 1-to-1 correspondence, we will sometimes refer to variables as haplotypes and vice-versa. The following is the basic set covering problem that we will be considering. It calls for finding a minimum set of haplotypes that satisfy the covering conditions:

$$z_{SC} := \min \sum_{h \in H_G} x_h \tag{2}$$

subject to

$$\sum_{h \in H_i^a(g)} x_h \geq 1 \qquad \forall g \in G, i \in A(g),\ a \in \{0,1\} \tag{3}$$

$$x \in \{0,1\}^{H_G}. \tag{4}$$

In worst case, the formulation has $\Theta(2^n)$ variables and $O(mn)$ constraints. In order to optimize its LP-relaxation, we will describe how to generate variables and constraints at run-time.

In the datasets that we examined, we saw that the lower bound z_{SC} is strong. However, there may be cases for which the bound is weaker, such as the instance (1), for which the optimal set covering solution is $\{0111, 1011, 1101, 1110\}$, while an optimal PHP solution is $\{1000, 0100, 0010, 0001, 1111\}$, requiring five haplotypes.

We now discuss how to strengthen the formulation in order to eliminate infeasible integer solutions of the set covering problem. We use some extra inequalities (cuts) which are violated by infeasible integer solutions. Notice that this is not customary, as, usually, cuts are employed to eliminate fractional solutions, while integer solutions are expected to be feasible.

Let us call a set H' of haplotypes *insufficient* if H' does not resolve G. For H' an insufficient set, let $U(H')$ be the set of unresolved genotypes, i.e., genotypes which have no resolution in H'. Let $g \in U(H')$ be an unresolved genotype, and let $C(g, H') := H(g) - H'$. An insufficient set H' and an unresolved genotype $g \in U(H')$ then give rise to the following cut:

$$x(C(g, H')) \geq 1. \tag{5}$$

Let us call $\mathcal{N}'$ the set of all pairs (g, H') with H' an insufficient set of haplotypes and $g \in U(H')$. By using the above cuts, the PHP can be formulated as:

SC model

$$\min \sum_{h \in H_G} x_h \tag{6}$$

subject to

$$\sum_{h \in H_i^0(g)} x_h \geq 1 \qquad \forall g \in G, i \in A(g) \tag{7}$$

$$\sum_{h \in H_i^1(g)} x_h \geq 1 \qquad \forall g \in G, i \in A(g) \tag{8}$$

$$x(C(g, H')) \geq 1 \qquad \forall (g, H') \in \mathcal{N}'. \tag{9}$$

$$x \in \{0,1\}^{H_G}. \tag{10}$$

In this paper we adopt the formulation (6)-(10) for the solution of the PHP. The formulation has an exponential number of variables and constraints. In order to optimize its LP-relaxation, we will describe how to generate variables and constraints at run-time. In particular, we keep one global set $\mathcal{X} \subset H_G$ of generated variables and one global set $\mathcal{N} \subset \mathcal{N}'$ of generated cuts (from the family of constraints (9), while constraints (7)-(8) are always present in the model). Therefore, at run-time, the constraints (9) are replaced by

$$x(C(g,H')) \geq 1 \qquad \forall (g,H') \in \mathcal{N}. \tag{11}$$

During the solution process, the sets of generated variables and cuts are updated only by adding, when needed, new variables or new cuts.

To solve the model (6)-(10), we use branch-and-bound, where each branching decision either forces or forbids the use of a certain haplotype. Therefore, at a generic node P of the search tree, there will be a set $I(P) \subset H_G$ of *included* haplotypes and a set $E(P) \subset H_G$ of *excluded* haplotypes (corresponding to haplotypes whose use has been forced or forbidden, respectively). Then, at P, the following *branching constraints* must be enforced:

$$x_h = 1 \quad \forall h \in I(P), \qquad\qquad x_h = 0 \quad \forall h \in E(P). \tag{12}$$

Assume the existence of a "black-box" procedure $\texttt{LP}(P) \rightarrow (x^*, v^*, \tilde{H})$ which, at a given node P of the search tree, returns a solution $x^* \geq 0$, its value, $v^* := \sum_h x^*_h$, and a resolving set $\tilde{H}$ of haplotypes (whose size provides an upper bound to the PHP). Furthermore, x^* and v^* are such that:

(i) Either x^* is integer, and it is an optimal solution to (6)–(9), (which implies x^* is feasible for the PHP), under the branching constraints (12),

(ii) Or x^* is fractional, and it is an optimal solution to (6)–(8), under the branching constraints (12) and the constraints (11),

(iii) Or the problem (6)–(9), under the branching constraints (12), is infeasible and in that case $v^* := +\infty$.

Furthermore, $\texttt{LP}(P)$ has the side-effect of updating the sets $\mathcal{X}$ and/or $\mathcal{N}$. In Section 5 we show how the black box computes x^* and v^*, while in section 6 we describe how the set $\tilde{H}$ is obtained. Given the existence of the above black-box, we can set up the branch-and-bound strategy described in Algorithm 3 for the solution of the PHP.

In the next two sections we discuss how to solve the LP relaxation of SC. First, we describe a representation of the s-cliques selectors by means of a suitable data structure. Next, we use this representation in our pricing procedure.

Algorithm 1 BRANCH-AND-BOUND(PHP)

Let $\bar{x}$ be any feasible solution of the PHP.
Initialize $\mathcal{X} := \{h \in H_G \,|\, \bar{x}_h = 1\}$ and $\mathcal{N} := \emptyset$. Let $u := |\mathcal{X}|$.
Initialize a list of problems $\mathcal{L} := \{P_0\}$, with $I(P_0), E(P_0) := \emptyset$.
while $\mathcal{L} \neq \emptyset$ **do**
  Extract a problem P from $\mathcal{L}$
  Call LP(P), obtaining $(x^*, v^*, \tilde{H})$ [the call may update $\mathcal{X}$ and $\mathcal{N}$].
  if $u > |\tilde{H}|$ then $u := |\tilde{H}|$ and $\bar{x} :=$ incidence vector of $\tilde{H}$
  if $v^* \geq u$ [x^* can be integer or fractional, or P infeasible]
    do nothing [fathoms the node]
  else if x^* is integer [found feasible sol for PHP with $v^* < u$]
    update $u := v^*$ and $\bar{x} := x^*$.
  else [x^* is fractional and $v^* < u$]
    Let h be such that $|x^*_h - 1/2|$ is minimum.
    Create problems P^0 and P^1 s.t.:
    $E(P^0) = E(P) \cup \{h\}$, $I(P^0) = I(P)$
    $E(P^1) = E(P)$, $I(P^1) = I(P) \cup \{h\}$.
    Add P^0 and P^1 to the list $\mathcal{L}$.
  endif
endwhile
return optimal solution $\bar{x}$ of value u.

4 Identifying the s-cliques

4.1 Finding all s-cliques

In order to find all the s-cliques we use a rooted binary tree, called the *s-clique tree*. In the s-clique tree, each vertex is labeled by a "virtual" genotype and is associated to a subset of genotypes, namely, the genotypes in G compatible with the virtual genotype. Each arc of the s-clique tree is associated to a SNP. All left arcs are labeled 0, and all right arcs are labeled 1. The tree is built, recursively, as follows. We label the root with the virtual genotype $22\ldots2$ (and, consequently, we associate the root to all genotypes in G), and we *elaborate* the root. Elaborating a generic node v means the following.

Let $g(v)$ be the virtual genotype labeling v, and G_v be the set of all genotypes compatible with $g(v)$. Let I_v be the set of SNPs associated to the arcs of the unique path from the root to v. If for each $j \in \{1, \ldots, n\} - I_v$ all genotypes in G_v show the same symbol at position j, then we set these positions in the virtual genotype as $g_j(v) := g_j$ for $j \in \{1, \ldots, n\} - I_v$ and any $g \in G_v$, we declare v a leaf and we stop elaborating v. Otherwise, v is an internal node: we pick a (branching) position $t \in \{1, \ldots, n\} - I_v$ and consider the two genotypes g^0 and g^1 defined as

$$g^0_i = g^1_i = g_i(v), \qquad i \neq t, \qquad g^0_t = 0, \quad g^1_t = 1.$$

Let G^0 and G^1 be the set of genotypes compatible with g^0 and g^1 respectively. If G^0 is not

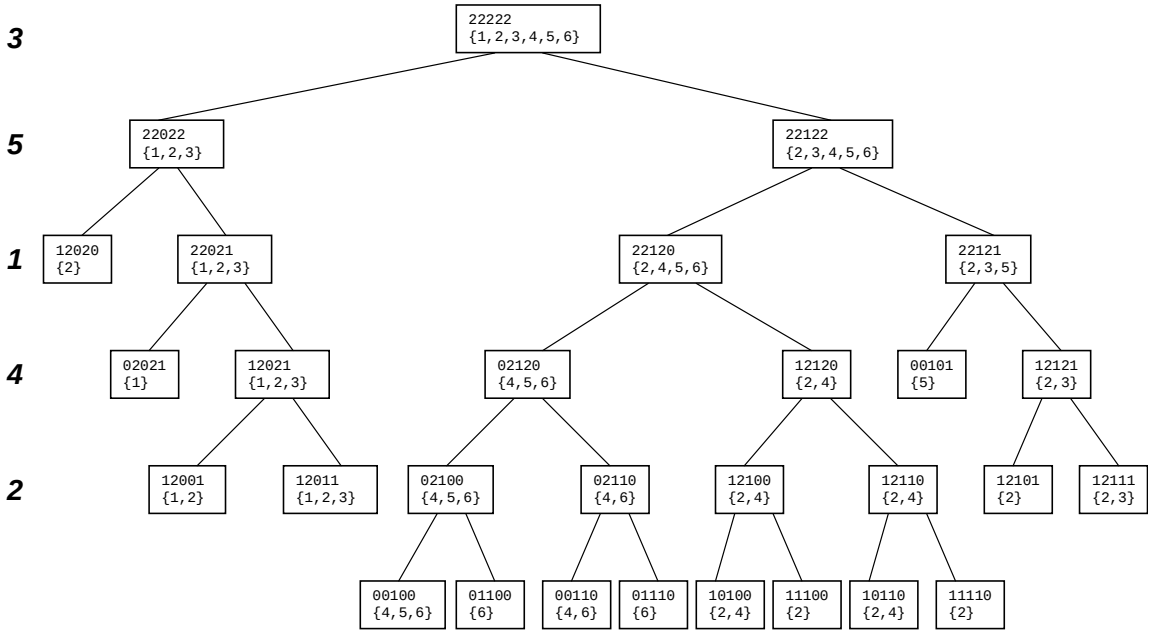

Figure 3: Tree of all s-cliques.

empty, we create a left son of v, called v_0, with $g(v_0) := g^0$ and $G_{v_0} := G^0$, and if G^1 is not empty, we create a right son of v, called v_1, with $g(v1) := g^1$ and $G_{v_1} := G^1$. We then elaborate v_0 and v_1. In Fig. 3 we show the s-clique tree obtained from Example 1, where for all nodes of the same level the same branching position is used. In particular, the tree was obtained with the following sequence of branching positions: $3, 5, 1, 4, 2$. In the figure, the tree nodes are labeled by their virtual genotype. The set G_v is also shown below each leaf v.

From the above construction a few observations follow:

- By construction, G_v contains all and only those genotypes which are compatible with at least one haplotype in $H(g(v))$ (not necessarily the same).
- For any two nodes v and v' the sets $H(g(v))$ and $H(g(v'))$ are disjoint.
- If a node v is a leaf, then *all* haplotypes in $H(g(v))$ are compatible with *all* genotypes in G_v.
- Different nodes can be associated to the same set of genotypes. In particular, the same s-clique can appear multiple times in the tree.

- The leaves can appear at different levels of the tree, and the total number of nodes in the tree can vary depending on how we choose the SNPs when branching from a node.

The most important property of the s-clique tree is the following:

Lemma 1. *A node v is a leaf if and only if it is associated to an s-clique.*

Proof: (if) Let v be a leaf. Let $h \in H(g(v))$. Each $g \in G_v$ is compatible with h, so that $g \in G(h)$. Furthermore, any $g \notin G_v$ must be incompatible with $g(v)$. Since incompatible genotypes cannot share a haplotype, g is not compatible with h, and so $g \notin G(h)$. (only if) Let $G(h)$ be an s-clique. Starting at the root, follow h in the tree until a leaf v is found. By construction, each genotype g compatible with h must be in G_v so that $G(h) \subseteq G_v$. Furthermore, $g \in G_v$ and $g \notin G(h)$ would imply that there exists a position $t \notin I_v$ such that $g_t \neq 2$ and $g_t \neq h_t$. On the other hand, for any $g' \in G(h)$ it is either $g'_t = 2$ or $g'_t = h_t$. Since both g' and g are in G_v, and $g'_t \neq g_t$, by construction, v should not be a leaf. It follows that there cannot exist $g \in G_v - G(h)$. $\diamond$

4.2 Representing the s-cliques selectors

The relevant information we need is concerned with the selectors of an s-clique. The leaves of the binary tree previously described give a list of all s-cliques, with possible repetitions, and, at the same time, of their selectors: for an s-clique K associated to the leaves $v^1, v^2, \ldots, v^q$, the set of selectors $H(K)$ is given by

$$H(K) = \bigcup_{i=1}^{q} H(g(v^i)).$$

In order to represent the set $H(K)$, we introduce a structure which we call the *pattern table*, which can represent any nonempty subset of $\{0,1\}^n$.

A *pattern* p is an n-string over $\{0, 1, \texttt{-}\}$. In a natural way, a pattern represents the hypercube $Q(p) := \{b \in \{0,1\}^n \mid b_i = p_i \, \forall i : p_i \neq \texttt{-}\}$, i.e., the set of 0-1 vectors that can be obtained by replacing each "-" of p with 0 or 1 in all possible ways.

We say that a pattern p *contains* a pattern q if $Q(q) \subseteq Q(p)$. It is easy to see that p contains q if and only if, for all $i = 1, \ldots, n$, $p_i \in \{0,1\}$ implies $p_i = q_i$. For instance, `01-1--` contains `01-10-` and `010100`, while it does not contain `-1-1-0`. We say that two patterns p and q *intersect* if $Q(p) \cap Q(q) \neq \emptyset$. Notice that two patterns p and q intersect if and only if there is no position i for which $p_i, q_i \in \{0,1\}$ and $p_i \neq q_i$. For instance, `01-1--` and `-1-1-0` intersect. We say that two patterns that do not intersect are *disjoint*. Let p and q be two intersecting patterns. Then the pattern l such that $Q(l) = Q(p) \cap Q(q)$ can be easily computed, component-wise, as follows:

$$l_i := \begin{cases} 0 & \text{if } (p_i = 0) \vee (q_i = 0) \\ 1 & \text{if } (p_i = 1) \vee (q_i = 1) \\ \texttt{-} & \text{if } (p_i = \texttt{-}) \wedge (q_i = \texttt{-}) \end{cases}$$

When l is computed as above, we write $l := p \cap q$.

Lemma 2. *Let p and q be two patterns such that p contains q. Let r be the number of positions where $p_i \neq q_i$. Then there exist disjoint patterns $s^1, \ldots, s^r$ such that $Q(p) - Q(q) = Q(s^1) \cup Q(s^2) \ldots \cup Q(s^r)$.*

Proof: Without loss of generality, we can assume that $p_i = q_i$ for $1 \leq i \leq n - r$, and, for $n - r < i \leq n$, $p_i =$ `-`, $p_i \neq q_i$. Then, we define s^j (with $1 \leq j \leq r$), component-wise, as follows:

$$s_i^j := \begin{cases} q_i & \text{for } 1 \leq i \leq n - r + (j-1) \\ 1 - q_i & \text{for } i = n - r + j \\ \texttt{-} & \text{for } n - r + j < i \leq n \end{cases}$$

For instance, if $p =$ `--10010---` and $q =$ `--10010010` it is $s^1 =$ `--100101--`, $s^2 =$ `--1001000-`, $s^3 =$ `--10010011`.

Notice that each s^j is disjoint from q and is contained in p. Therefore, $Q(s^1) \cup \ldots \cup Q(s^r) \subseteq Q(p) - Q(q)$. Conversely, assume $b \in Q(p) - Q(q)$. Since b is disjoint from q there must exist at least a position $i > n - r$ such that $b_i \neq q_i$. Let i^* be the smallest such position. Then $b \in Q(s^{i^*})$ and so $Q(s^1) \cup \ldots \cup Q(s^r) \supseteq Q(p) - Q(q)$. $\diamond$

A *pattern table* is a list of patterns $p^1, p^2, \ldots, p^r$, which represents the set $Q(p^1) \cup Q(p^2) \cup \ldots \cup Q(p^r)$. The set $H(K)$ can possibly be represented by more than one pattern table. One representation is induced by the tree generating all s-cliques. If K is associated to the leaves $v^1, v^2, \ldots, v^q$, then the induced pattern table consists of patterns $p^1, \ldots, p^q$, where each pattern p^i is obtained from the virtual genotype $g(v^i)$ by replacing each 2 with a "`-`", i.e.,

$$p_j^i = \begin{cases} g_j(v^i) & \text{if } g_j(v^i) \neq 2 \\ - & \text{otherwise} \end{cases} \qquad i = 1, \ldots q, \quad j = 1, \ldots, n.$$

For instance, the pattern tables induced by the tree in Figure 3 for the s-cliques $K_4 = \{g_2\}$ and $K_6 = \{g_2, g_4\}$ of Example 1 are

$K_4 \rightarrow$	`1-0-0`	$K_6 \rightarrow$	`10100`
	`11100`		`10110`
	`11110`		
	`1-101`		

The size of the pattern table thus obtained can be large with respect to n. The size can be greatly reduced by putting the table into *standard form*. A pattern table is in standard form, or *irreducible*, if the following conditions hold:

1. (Non-containment:) For each pair p^i and p^j, neither p^i contains p_j nor p_j contains p^i.
2. (Irreducibility:) For each pair p^i and p^j, there are at least two positions a, b such that $p_a^i \neq p_a^j$ and $p_b^i \neq p_b^j$.

Our interest in pattern tables lies in the fact that they can give compact representations of (possibly large) subsets of $\{0,1\}^n$. A compact representation is one using a small number of patterns. If a table is not in standard form it is immediate to reduce its size and still represent the same set. For instance, if property 1 is not satisfied, we can get rid of a pattern contained in another. Moreover, if property 1 is satisfied but property 2 is not, then there must exist two patterns which are identical in all positions but one, where one of them is 0 while the other is 1. Then they can be replaced by their merging, i.e., a new pattern with a `-` in that position. By applying the above rules, a pattern table which is not in standard form can be put in standard form. Two different orders of application of the rules may yield different final tables.

The above tables, once put into standard form, become

$K_4 \rightarrow$ `1-0-0` `111-0` `1-101` $\qquad$ $K_6 \rightarrow$ `101-0`

From our computational experiments, we have observed that, even for instances where the number of possible haplotypes ($O(2^n)$) and the number of subsets of genotypes ($O(2^m)$) are both large, the the number of patterns (and, a fortiori, the number of s-cliques) is usually orders of magnitude smaller. In particular, as reported in Table 4 and Table 5 of Section 7, the ratio between the total number of haplotypes and of patterns ranges from 10 to 10^6.

5 Solving the LP relaxation of SC

In this section we discuss the implementation of the "black-box" $\texttt{LP}(P)$, i.e., we describe how to solve the LP relaxation of the SC model at a node P of the branch-and-bound tree.

We will use both column-generation and row-generation. $\texttt{LP}(P)$ requires the solution of a sequence of LPs, where each LP solved is defined only over a subset of the variables and a subset of the constraints. At each iteration, either new variables are added, or new constraints are added, or a termination condition is met and the optimization stops.

5.1 The pricing problem

The generic LP relaxation to be solved at node P requires to minimize $\sum_h x_h$ under the constraints (7), (8), (11), (12), and the non-negativity $x \geq 0$.

In our pricing procedure, we keep an implicit representation of haplotypes that have not been generated yet. By using this representation when we price-in new haplotypes, we are guaranteed that it is impossible to generate the same haplotype twice. The representation is also used to ensure that the pricing scheme is not affected by the presence of branching constraints (those fixing haplotypes to either 0 or 1).

Let $\alpha_{gi} \geq 0$ and $\beta_{gi} \geq 0$ be the dual variables corresponding to constraints (7) and (8) respectively, and $\gamma_{gH} \geq 0$ be the dual variables corresponding to constraints (11).

The dual constraint corresponding to a haplotype $h \in H_G$ is

$$\sum_{g\in G(h)}\sum_{i\in A(g)}\left((1-h_i)\,\alpha_{gi}+h_i\,\beta_{gi}\right)+\sum_{\substack{(g,H)\in\mathcal{N}:\\ g\in G(h)}}\gamma_{gH}\leq 1. \tag{13}$$

The pricing problem corresponds to finding an inequality (13) violated by the current dual optimal variables $(\alpha^*,\beta^*,\gamma^*)$. This task can be carried out by maximizing the expression

$$\sum_{g\in G(h)}\sum_{i\in A(g)}\left((1-h_i)\,\alpha^*_{gi}+h_i\,\beta^*_{gi}\right)+\sum_{\substack{(g,H)\in\mathcal{N}:\\ g\in G(h)}}\gamma^*_{gH} \tag{14}$$

over all h, and checking if the maximum thus found is larger than 1.

In order to compute the maximum of (14), we first rewrite (14) as follows (where, for p a proposition, $[p]=0$ if p is false, and $[p]=1$ if p is true):

$$\sum_i\left((1-h_i)\sum_{g\in G(h)}[i\in A(g)]\,\alpha^*_{gi}+h_i\sum_{g\in G(h)}[i\in A(g)]\,\beta^*_{gi}\right)+\sum_{\substack{(g,H)\in\mathcal{N}:\\ g\in G(h)}}\gamma^*_{gH}. \tag{15}$$

We want to compute the maximum of (15) over all h without explicitly enumerating them. Towards this goal, we consider a partition of the haplotypes into subsets for which the inner sums for α^* and β^*, and the sum for γ^*, do not depend on h. Notice that the partition of H_G into the s-cliques selectors, i.e., $\{H(K)\,|\,K\in\mathcal{K}\}$, is such that $G(h)$ is invariant for $h\in H(K)$.

For an s-clique $K\in\mathcal{K}$, define

$$\begin{array}{rcll}\alpha(K,i) & := & \sum_{g\in K}[i\in A(g)]\,\alpha^*_{gi} & i=1,\ldots,n\\ \beta(K,i) & := & \sum_{g\in K}[i\in A(g)]\,\beta^*_{gi} & i=1,\ldots,n\\ \gamma(K) & := & \sum_{(g,H)\in\mathcal{N}:g\in K}\gamma^*_{gH}. & \end{array}$$

By using the above definitions, we can rewrite (15) as

$$\lambda(K,h):=\sum_i\left((1-h_i)\,\alpha(K,i)+h_i\,\beta(K,i)\right)+\gamma(K).$$

Furthermore, denote by $\lambda(K)$ the value

$$\lambda(K)=\max_{h\in H(K)-\mathcal{X}}\lambda(K,h). \tag{16}$$

With this notation, the pricing problem can be rephrased as: find $\hat{h}\notin\mathcal{X}$ such that $\lambda(G(\hat{h}))$ is maximum. If $\lambda(G(\hat{h}))>1$, then $\hat{h}$ is a variable with negative reduced cost, that should be added to the current LP. Otherwise, there are no variables that should be added to the current LP.

The haplotype $\hat{h}$ can be found by computing $\max\{\lambda(K)\,|\,K\in\mathcal{K}\}$, which can be accomplished by going through all s-cliques one at a time. In Section 4.1 we have described a method for determining all s-cliques of G.

For each s-clique K we consider the pattern table representing the set of selectors $H(K)$ derived from the s-clique tree and put into standard form (see Section 4.2). Assume this pattern table consists of patterns $p^1, \ldots, p^q$. Then, $H(K) = Q(p^1) \cup \ldots \cup Q(p^q)$. For each $j = 1, \ldots, q$, we maximize $\lambda(K, h)$ over $h \in Q(p^j)$ and we compute the corresponding optimal haplotype $\tilde{h}^j$ as follows:

$$\tilde{h}_i^j := \begin{cases} 0 & \text{if } (p_i^j = 0) \\ 1 & \text{if } (p_i^j = 1) \\ 0 & \text{if } (p_i^j = \texttt{-}) \wedge (\alpha(K,i) \geq \beta(K,i)) \\ 1 & \text{if } (p_i^j = \texttt{-}) \wedge (\alpha(K,i) < \beta(K,i)) \end{cases} \qquad i = 1, \ldots, n. \tag{17}$$

The first two cases correspond to components of the haplotype whose value is forced by the pattern. On the other hand, if the component of the pattern is `-`, it is possible to set the haplotype component to either 0 or 1. In the former case, the contribution to the sum is $\alpha(K, i)$ while in the latter case it is $\beta(K, i)$. Therefore, the best choice is the one for which the contribution is maximum.

Let $\tilde{h}(K) = \tilde{h}^k$, with $k = \text{argmax}\{\lambda(K, \tilde{h}^j) \,|\, j = 1, \ldots, q\}$. Then, $\hat{h} = \tilde{h}(\hat{K})$, with $\hat{K} = \text{argmax}\{\lambda(K, \tilde{h}(K)) \,|\, K \in \mathcal{K}\}$.

For an example, let us consider the s-cliques $K_4 = \{g^2\}$ and $K_6 = \{g^2, g^4\}$ of Example 1. Assume we are at the root node and $\mathcal{N} = \emptyset$, so that $\gamma^* = 0$, while the dual variables for the genotypes $g^2 = 12222$ and $g^4 = 20120$ and their ambiguous positions are as follows:

	α^*_{gi}					β^*_{gi}				
g^2		.2	0	.2	.1		.3	.1	.1	.2
g^4	.1			.1		.2			.4	

Let us consider K_4, whose pattern table consists of patterns $p^1 =$ `1-0-0`, $p^2 =$ `111-0` and $p^3 =$ `1-101`. Then the computation (17) applied to p^1 goes as follows:

- Since $p_1^1 = 1$, we set $\tilde{h}_1^1 = 1$.
- Since $p_2^1 =$ `-` and $\alpha(K_4, 2) = .2 < \beta(K_4, 2) = .3$, we set $\tilde{h}_2^1 = 1$.
- Since $p_3^1 = 0$, we set $\tilde{h}_3^1 = 0$.
- Since $p_4^1 =$ `-` and $\alpha(K_4, 4) = .2 > \beta(K_4, 4) = .1$, we set $\tilde{h}_4^1 = 0$.
- Since $p_5^1 = 0$, we set $\tilde{h}_5^1 = 0$.

Therefore, it is $\tilde{h}^1 = 11000$ and $\lambda(K_4, \tilde{h}^1) = 0 + .3 + 0 + .2 + .1 = .6$. Similarly, one obtains $\tilde{h}^2 = 11100$ with $\lambda(K_4, \tilde{h}^2) = 0 + .3 + .1 + .2 + .1 = .7$, and $\tilde{h}^3 = 11101$ with $\lambda(K_4, \tilde{h}^3) = 0 + .3 + .1 + .2 + .2 = .8$. From these values we get $\tilde{h}(K_4) = \tilde{h}^3$ and $\lambda(K_4) = .8$.

Let us now consider the s-clique $K_6 = \{g^2, g^4\}$, with pattern table consisting of the single patterns $p^1 =$ `101-0`. Then the computation (17) applied to p^1 goes as follows:

- Since $p_1^1 = 1$, we set $\tilde{h}_1^1 = 1$.

- Since $p_2^1 = 0$, we set $\tilde{h}_2^1 = 0$.

- Since $p_3^1 = 1$, we set $\tilde{h}_3^1 = 1$.

- Since $p_4^1 =$ - and $\alpha(K_6, 4) = .2 + .1 = .3 < \beta(K_6, 4) = .1 + .4 = .5$, we set $\tilde{h}_4^1 = 1$.

- Since $p_5^1 = 0$, we set $\tilde{h}_5^1 = 0$.

Therefore, it is $\tilde{h}(K_6) = 10110 =: \tilde{h}$ and $\lambda(K_6, \tilde{h}) = \lambda(K_6) = .2+.2+.1+(.1+.4)+.1 = 1.1$.

Note that, since $\lambda(K_6, \tilde{h}) > 1$, the haplotype $\tilde{h}$ can be profitably generated and inserted in the current LP.

In a similar way, one obtains $\tilde{h}(K_i)$ and $\lambda(K_i)$ for the remaining s-cliques, from which the haplotype $\hat{h}$ maximizing $\lambda(K_i)$ can be derived.

The computational cost of solving the pricing problem for one s-clique K whose pattern table has q patterns is $O(\sum_{g \in K} |A(g)| + q\, n + |\mathcal{N}|)$. The first term accounts for the time needed to compute $\alpha(K, i)$ and $\beta(K, i)$ over all i. This computation is done only once at the beginning of each pricing phase. The second term is due to the fact that there are q patterns, and for each pattern the time needed to find the best haplotype by (17) is $O(n)$. The last term is the cost of computing $\gamma(K)$.

5.2 Excluding haplotypes in the pricing scheme

In order to explicitly exclude haplotypes from pricing, we describe a procedure which, given a pattern p and a haplotype $h \in Q(p)$, computes a set of patterns $p^1, \ldots, p^r$ such that $Q(p^1) \cup \ldots \cup Q(p^r) = Q(p) - \{h\}$.

Lemma 3. *Let p be a pattern and h be a haplotype such that $h \in Q(p)$. Let r be the number of positions where $p_i =$ -. Then there exist disjoint patterns $p^1, \ldots, p^r$ such that $Q(p) - \{h\} = Q(p^1) \cup Q(p^2) \ldots \cup Q(p^r)$.*

Proof: Without loss of generality, we can assume that $p_i \neq$ - for $i = 1, \ldots, n - r$. Then, we define p^j (with $1 \leq j \leq r$), component-wise, as follows:

$$p_i^j := \begin{cases} h_i & \text{for } 1 \leq i \leq n - r + (j - 1) \\ 1 - h_i & \text{for } i = n - r + j \\ \text{-} & \text{for } n - r + j < i \leq n. \end{cases}$$

For instance, if $p =$ `10010---` and $h = 10010010$ it is $p^1 =$ `100101--`, $p^2 =$ `1001000-`, $p^3 =$ `10010011`.

Notice that each $Q(p^j)$ does not contain h and is contained in $Q(p)$. Therefore, $Q(p^1) \cup \ldots \cup Q(p^r) \subseteq Q(p) - \{h\}$. Conversely, assume $b \in Q(p) - \{h\}$. Since $b \neq h$ there must exist at least a position $i > n - r$ such that $b_i \neq h_i$. Let i^* be the smallest such position. Then $b \in Q(p^{i^*})$ and so $Q(p^1) \cup \ldots \cup Q(p^r) \supseteq Q(p) - \{h\}$. ◇

The above procedure can be used to explicitly forbid a haplotype $\hat{h}$ from being generated again. Since the pattern table is kept in standard form, there exists exactly one pattern p

such that $\hat{h} \in Q(p)$. We then replace p with the patterns $p^1, \ldots, p^r$ as in Lemma 3 (and reduce the new pattern table so that it is still in standard form) and we iterate the pricing procedure.

The forbidding of a haplotype should occur immediately after it has been generated by the pricing procedure. Alternatively, one could chose not to forbid the haplotype immediately, but to wait until, if ever, the haplotype is generated again by the pricing procedure. In our computational experiments, we have found the latter alternative to be preferable.

6 Computing good feasible solutions

In this section we describe how the solutions of the SC model are exploited to obtain "good" resolving sets $\tilde{H}$ of haplotypes. These in turn are used to provide a tight upper bound to the optimum, and henceforth to speed-up the branch-and-bound search.

We obtain such a set $\tilde{H}$ from each solution H_{SC} which is feasible for the SC model but not feasible for the PHP. The basic idea is to add the minimum number of haplotypes to H_{SC}, so as to obtain a PHP solution. In order to describe our strategy, we need some definitions. Let H' be a set of haplotypes. We define

- OPT(H') a set $H^* \subset H'$ such that H^* resolves G and the size of H^* is minimum.

Notice that the computation of OPT(H') is itself a PHP problem, under the restriction that only haplotypes in H' can be used in any solution. Furthermore, we define

- UNS to be the set of genotypes in G which do not have a resolution within H_{SC}

and, for G' a set of genotypes, we define

- $\mathrm{CO}(G') := \cup_{g \in G'} \{h : \exists h' \in H_{SC} : g = h \oplus h'\}$.

The set CO(G') is built by taking, for each $h \in H_{SC}$ and each genotype $g \in G'$ compatible with h, the complement of h with respect to g.

Given H_{SC}, a trivial way to obtain a feasible solution $\tilde{H}$ for the PHP is to add to H_{SC} at most one haplotype for each unresolved genotype. This is because each genotype $g \in$ UNS has at least one haplotype $h \in H_{SC}$ compatible with it and hence g can be solved by adding the complement of h with respect to g. This yields an upper bound to the PHP equal to $|\tilde{H}| \leq |H_{SC}| + |\mathrm{UNS}|$. To improve over this trivial upper bound, we can consider

$$\tilde{H} = \mathrm{OPT}\big(H_{SC} \cup \mathrm{CO}\big(\mathrm{UNS}\big)\big). \tag{18}$$

In the above expression we first introduce, for each unresolved genotype g, *all* haplotypes that are the complement with respect to g of some haplotype in H_{SC}. This way, each genotype has at least one resolution in $H_{SC} \cup$ CO(UNS). Within this set, we find the optimal resolving subset $\tilde{H}$ of haplotypes.

As already remarked, each time we compute $\tilde{H}$ as in (18) we need to solve a PHP problem over a restricted set of haplotypes. Since the number of resolutions of each genotype within

the restricted set of allowed haplotypes is rather small (as observed in our computational experiments), we are able to quickly solve this problem by means of a combinatorial branch-and-bound technique.

The combinatorial branch-and-bound which we use for the solution of each $\mathrm{OPT}(H')$ problem works as follows. A global incumbent is provided by the smallest $\tilde{H}$ among those computed thus far. To each node v of the search tree a pair (G_v, H_v) is associated, where $G_v \subset G$ is the set of genotypes already solved, $H_v \subset H'$ is a resolving set for G_v and each genotype in $G - G_v$ has more than one resolution in H' but no resolution in H_v. A lower bound to the solution value at the node is given by $|H_v|$ plus the size of an independent set in the compatibility graph of $G - G_v$ (as the compatibility graph is fairly small, we are in fact able to quickly find the largest independent set). If the lower bound exceeds the incumbent, the node is pruned. Otherwise, we consider a genotype g in $G - G_v$ which has the smallest number of resolutions in H'. Notice that each resolution uses at least one haplotype not in H_v. For each resolution of g, say $g = h_1 \oplus h_2$, we branch from node v to a new node w, setting $G_w := G_v \cup \{g\}$ and $H_w := H_v \cup \{h_1, h_2\}$.

The solution given by (18) can be improved as follows:

$$\tilde{H} = \mathrm{OPT}\big(H_{SC} \cup \mathrm{CO}(G)\big). \tag{19}$$

The time required to solve the above OPT() problem depends on the quality of the available incumbent. Moreover, solving (19) is computationally more demanding than (18) because it involves a larger set of allowed haplotypes. Hence, we have adopted the following strategy. We have introduced a parameter p, and, for the first p times that the procedure is called, $\tilde{H}$ is computed from (18) (the first p calls are intended to quickly lower the incumbent, so that the next calls can take advantage of a tight upper bound). Starting from call number $p+1$, $\tilde{H}$ is computed from (19). From our computational experiments, we have found that $p = 3$ is a suitable value for the parameter p.

An overview of $\mathtt{LP}(P)$

Having described how the column-generation problem is solved, and how to find good feasible solutions, we can now give an overall description of the procedure $\mathtt{LP}(P)$, whose behavior was anticipated in Section 3.

1. First, an LP over $\mathcal{X}, \mathcal{N}$ is solved, where $\mathcal{X}$ and $\mathcal{N}$ are the global sets of variables and cuts generated up to now. If the LP is infeasible, **return** $v^* = +\infty$.

2. *Column-generation phase.* Given the optimal solution to the last LP, the pricing problem is solved. Possibly, new variables are added to $\mathcal{X}$ and a new LP is solved. This process is iterated until the pricing problem does not find any variable to add to $\mathcal{X}$. Let x^* be the optimal solution of the last LP solved, and v^* its value.

3. *Termination.* At the end of the column-generation phase, the optimal solution x^* may be integer or fractional. In both cases, let $\bar{x}$ denote the rounding-up of x^*. If $\bar{x}$ is feasible for the PHP then **return** x^* and v^*.

Gene name	Description	n. SNPs
ABCE1	ATP-binding cassette	117
BDKRB2	Bradykinin receptor	76
IGF1	Insulin-like growth factor	155
IL22	Interleukin	53
LIPE	Lipase, hormone-sensitive	120
MGP	Matrix Gla protein	28
NFKBIA	Nuclear factor of kappa light polypeptide	49
PROC	Protein C	58
RELA	V-rel reticuloendotheliosis viral oncogene	22
SELE	Selectin	99
THBD	Thrombomodulin	30
TNFRSF1A	Tumor necrosis factor receptor	64
TYK2	Tyrosine kinase	183
USF1	Upstream transcription factor	46
ZNF202	Zinc finger protein	46

Table 1: The 15 genes from SeattleSNPs

4. *Feasibility cuts.* Otherwise, the heuristic procedure for generating feasible solutions is called, starting from $\bar{x}$, and yields a set $\tilde{H}$. Furthermore, the following cuts are generated. Let $H^1 := \{h : \bar{x}_h = 1\}$. Then H^1 is an insufficient set, and the inequalities (9) for all unresolved genotypes $g \in U(H^1)$ are added to $\mathcal{N}$. After the addition of the cuts, **goto** step 2.

7 Computational Experiments

The program was implemented in C, and run on a 1.2 GHz Centrino PC, with 512MB of RAM. The LP solver used was GLPK [3]. In order to compare our approach with some of the existing methods, we have implemented the ILP formulations proposed in [5] and [13]. The former formulation turned out to be too weak for our test-instances (none of the generated instances could be solved within the time threshold that we set). As far as the latter formulation is concerned, the (conceptual) model in [13] has a variable for each $h \in H(G)$ and for each resolution of a genotype, while the number of constraints is proportional to the total number of resolutions. In [13], Gusfield describes also an improvement of the conceptual model, called RTIP, in which many variables are not present as they are not needed by some optimal solution. Hence, we have used RTIP for our tests.

In our experiments, we have set a time timit of two hours per instance. Furthermore, these are the memory limits which most affect our tests: the maximum size of any LP which can be solved in our memory (roughly, $200,000$ variables by $200,000$ constraints); the maximum number of nodes of the s-clique tree (which we set to $150,000$).

Real-data instances. In a first set of computational experiments, we have tested our approach on some real-data datasets, taken from the *SeattleSNPs* database [2]. The SeattleSNPs database is focused on identifying, genotyping, and modeling the associations be-

Gene	m'	n'	avg2	max2	tot_h	time	opt
ABCE1.row	3	117	5.33	9	576	1	5
ABCE1.col	47	41	3.09	11	3272	1	29
BDKRB2.row	9	76	7.22	12	8144	1	12
BDKRB2.col	47	33	5.15	8	1787	111	30
IGF1.row	8	155	8	15	33816	1	10
IGF1.col	47	45	4.96	11	7775	1	31
IL22.row	22	53	9.38	14	45111	12	22
IL22.col	47	27	4.93	9	1248	157	28
LIPE.row	6	120	2.5	5	48	1	7
LIPE.col	47	49	5.03	13	21810	2	27
MGP.row	14	28	3.42	6	148	1	10
MGP.col	47	10	3.42	6	46	1	13
NFKBIA.row	4	49	6	10	2054	1	7
NFKBIA.col	47	15	2.75	6	122	1	15
PROC.row	6	58	16.4	18	401152	1	7
PROC.col	47	15	3.2	6	137	1	20
RELA.row	27	22	3.94	7	319	1	13
RELA.col	47	14	2.93	6	130	1	12
SELE.row	5	99	2.8	7	138	1	8
SELE.col	47	33	5.86	13	28244	196	27
THBD.row	26	30	3.5	7	258	1	17
THBD.col	47	6	1	1	7	1	7
TNFRSF1A.row	7	64	5.2	14	16900	1	9
TNFRSF1A.col	47	30	5.65	8	1913	70	28
TYK2.row	4	183	13.2	28	2.68×10^8	1	5
TYK2.col	47	77	8.13	22	9.28×10^6	81	37
USF1.row	34	46	8.6	18	302386	43	23
USF1.col	47	10	4.6	11	2654	1	20
ZNF202.row	16	46	4.42	11	2111	1	11
ZNF202.col	47	10	3.2	5	72	1	8

Table 2: Results for real-data instances

tween SNPs in candidate genes and pathways underlying inflammatory responses in humans. The database contains data for 47 individuals (24 African-American and 23 European subjects) and about 100 genes. For each gene, 47 genotypes are reported. The genotypes are incomplete, i.e., a genotype may cover only a subset of the SNPs, due to missing data and/or experimental errors. Each instance can thus be represented by a $47 \times n$ matrix over $\{0, 1, 2, 3\}$. Each row is a genotype over n SNPs. Entries 0,1,2 have the usual meaning, while each entry "3" means that the alleles are not known at the specific site. We have selected 15 genes at random. They are described in Table 1.

In order to remove uncertainty from the data and recover "normal" instances of the PHP we have proceeded as follows. From each instance X, of size $47 \times n$ we have obtained two instances X.row and X.col. X.row has size $m' \times n$, and is obtained from X by removing all rows (genotypes) that contain an entry "3". Similarly, X.col, of size $47 \times n'$, is obtained from X by removing all columns (SNPs) that contain an entry "3". In table 2 we report the computational results of our algorithms for the 30 instances obtained from the 15 selected genes. Each row corresponds to an instance. The columns labeled m' and n' report the size

Gene	time	Gene	time
BDKRB2.col	0.12 s	PROC.row	1 h 50 m
IL22.col	0.15 s	TYK2.row	exceeds available memory
SELE.col	4.96 s	TYK2.col	217 s
TNFRSF1A.col	0.29 s	USF1.row	1 h 40 m

Table 3: Results on a more powerful computer

of the instance. The columns labeled "avg2" and "max2" report the average and maximum number of 2s in the instance. The column labeled "tot_h" reports the total number of haplotypes compatible with some genotype in the instance (i.e., $|H_G|$). The column labeled "time" reports the running time, in CPU seconds (rounded-up to the closest integer). Finally, the column labeled "opt" reports the value of the optimal solution.

From the table it can be seen that all instances were solved, within a maximum running time of 196 seconds. The number of branch-and-bound nodes (not reported) ranged from a minimum of 1 to a maximum of 126, with an average of 10.

With some exceptions, these instances are relatively small, and so RTIP was able to tackle them effectively. However, the following instances were too large and could not be solved by RTIP: PROC.row, USF1.row, TYK2.col, and TYK2.row. All the remaining instances were solved, with running times ranging from 1 second to a maximum of 202 seconds.

We recall that the RTIP model requires an exponential number of rows and columns. Hence, using a machine with larger memory and faster CPU than ours can only marginally increase the maximum size of the solvable instances. This is confirmed by the empirical results kindly provided by one of the referees. He/she considered the four instances for which our program took the largest computing time (BDKRB3.col, IL22.col, SELE.col, TNFRSF1A.col) and the four instances which RTIP could not solve on our machine (PROC.row, TYK2.row, TYK2.col, USF1.row). RTIP was then tried on these instances on a computer with 2GHz CPU, 4GB RAM and using CPLEX 10. Table 3 reports the obtained results.

Simulated instances. In a second set of experiments, we have tested our approach on instances generated by *ms*, a program developed by R. Hudson [1, 18]. This program has become the standard code for simulating haplotype populations and has been used in [13, 5, 6] for testing the proposed approaches. The haplotypes are generated according to coalescent theory, i.e., representing the evolution of the sampled haplotypes. A population generated by *ms* contains several repetitions of the same haplotype. This is consistent with what we observe in nature.

We have generated two sets of 6 classes of instances. In each set, each class is characterized by the number of SNPs, set to either 30 or 50, and by a target number of genotypes, set to either 30, 40 or 50. We generated 10 instances for each class. Each single instance was generated as follows. Let m be the target number of genotypes and n be the number of SNPs. We have created a population of $3\,m$ haplotypes (as previously remarked, not necessarily distinct) of length n each, by using *ms* with recombination parameter $\rho = 0$ for the first set, and $\rho = 16$ for the second. Then, we have randomly paired each haplotype with exactly another haplotype thereby obtaining $3/2\,m$ random genotypes. After removing

m		30		40		50	
n		30	50	30	50	30	50
unsolved		0 (1)	4 (10)	0 (1)	3 (6)	0 (0)	0 (10)
genotypes	min	27	29	35	39	45	45
	avg	29.5	31	36.9	42	47.1	50.3
	max	33	33	44	44	51	55
amb sites	min	1	1	1	1	1	1
	avg	5.99	9.48	5.31	7.87	5.02	9.79
	max	12.4	22.6	10.1	16	11	20.9
tot hap	min	514	7.76×10^4	428	892	1390	3.48×10^5
	avg	8×10^4	7.94×10^8	2.27×10^4	3.81×10^6	1.18×10^4	3.5×10^7
	max	6.7×10^5	3.89×10^9	2.04×10^5	2.55×10^7	5.85×10^4	1.76×10^8
patterns	min	78	262	102	144	105	443
	avg	373	2174	167	5957	183	3192
	max	2301	7805	243	36.9×10^4	215	16.4×10^4
LP cols	min	31 (783)	236 (-)	30 (672)	41 (38098)	40 (1625)	63 (-)
	avg	111 (20276)	1731 (-)	64.3 (8338)	1185 (113683)	117 (16238)	316 (-)
	max	536 (47346)	4416 (-)	172 (44338)	7513 (157063)	584 (85257)	1270 (-)
LP rows	min	234 (799)	264 (-)	232 (691)	390 (33893)	330 (1674)	533 (-)
	avg	355 (19561)	593 (-)	405 (8333)	658 (105720)	493 (16522)	990 (-)
	max	468 (42782)	908 (-)	624 (44655)	843 (151738)	757 (86370)	1264 (-)
tot time	min	0 (0)	1 (-)	0 (1)	0 (230)	0 (2)	1 (-)
	avg	0.9 (152)	19.4 (-)	0.63 (53.55)	17.4 (3672.2)	1.88 (233.6)	10.4 (-)
	max	5 (575)	6.8 (-)	3 (436)	100 (7106)	13 (1925)	28 (-)
time to opt	min	0	1	0	0	0	1
	avg	0.8	5.4	0.63	9.57	1.77	10
	max	5	16	3	46	13	28

Table 4: Results for simulated instances, $\rho = 0$

the non-ambiguos and the duplicated genotyopes, we have accepted the instance if the final number of genotypes obtained was in the interval $[m - 5, m + 4]$.

The results for $\rho = 0$ are reported in Table 4, and those for $\rho = 16$ are in Table 5. Each column refers to a class of 10 instances. Some instances could not be solved within our time and memory constraints. The number of unsolved instances within each class is reported in the row labeled "unsolved".

Next, each block of rows with the same label reports the minimum, average and maximum value of some parameter over all solved instances in the class. The rows labeled "genotypes" refer to the number of genotypes. The rows labeled "amb sites" refer to the average number of ambiguous positions per instance. The rows labeled "tot hap" refer to the number of compatible haplotypes. The rows labeled "patterns" refer to the number of patterns for the starting pattern table. The rows labeled "LP cols" (respectively, "LP rows"), refer to the number of variables (respectively, constraints) generated during the computation. The rows labeled "tot time" refer to the total running time of the algorithm in CPU seconds. The rows labeled "time to opt" refer to the time to compute the optimal solution (while the rest of the running time is spent for proving optimality).

In the blocks named "unsolved", "LP cols", "LP rows" and "tot time" we report, in parentheses, also the values relative to the performance of RTIP. As it can be observed, the

m		30		40		50	
n		30	50	30	50	30	50
unsolved		0 (2)	4 (6)	0 (1)	5 (9)	0 (4)	5 (10)
genotypes	min	28	31	35	40	45	45
	avg	32	33	39.4	42	48.3	51
	max	34	34	42	44	53	54
amb sites	min	1.1	1.33	1.1	1.2	1	1
	avg	6.44	8.29	6.9	8.4	6.34	8.04
	max	13.2	16.5	13.6	17.6	13	18.8
tot hap	min	1.95×10^3	2.71×10^4	2.7×10^3	1.62×10^5	1.9×10^3	1.42×10^5
	avg	7.26×10^4	1.07×10^6	8.22×10^4	6.14×10^5	4.14×10^4	9.68×10^6
	max	2.83×10^5	4.2×10^6	5.09×10^5	1.1×10^6	1.85×10^5	4.61×10^7
patterns	min	134	302	203	470	263	546
	avg	1142	4862	846	6214	1955	11110
	max	3930	21522	2905	10871	13028	17878
LP cols	min	87 (1268)	124 (3098)	88 (3066)	79 (43480)	76 (3069)	65 (-)
	avg	368 (19490)	2030 (15825)	437 (24497)	855 (43480)	323 (16497)	1318 (-)
	max	1443 (64060)	6670 (36937)	1144 (48924)	2750 (43480)	1140 (41740)	3500 (-)
LP rows	min	248 (1144)	413 (2965)	409 (2970)	512 (37223)	532 (3180)	628 (-)
	avg	436 (17820)	576 (14363)	584 (22723)	738 (37223)	665 (16100)	852 (-)
	max	566 (60139)	793 (33230)	805 (44525)	944 (37223)	971 (39279)	1138 (-)
tot time	min	0 (1)	1 (7)	0 (3)	4 (280)	0 (3)	3 (-)
	avg	4.5 (225.5)	34.66 (102.7)	49.2 (187)	56 (280)	25.4 (96.6)	61.2 (-)
	max	14 (963)	125 (314)	159 (487)	198 (280)	152 (278)	189 (-)
time to opt	min	0	1	0	3	0	2
	avg	2.3	9.16	22.4	17.8	1.8	17.2
	max	11	18	83	11	3	23

Table 5: Results for simulated instances, $\rho = 16$

number of compatible haplotypes is fairly large for some instances, and the RTIP model suffers from the large size of the corresponding IP. The size of the LP models at the root node for our approach can be up to two orders of magnitude smaller than for RTIP. This translates into running times that are up to two orders of magnitude smaller as well.

8 Conclusions

In this paper we have presented a new IP formulation for the pure parsimony haplotyping problem. Our formulation, relying on an implicit representation of the set of all haplotypes, and using both column- and row- generation, can be used to solve larger instances (with respect to the total size of $H(G)$) than other ILP approaches in the literature.

One of the key data structures we use is the pattern table, and the performance of our algorithm can benefit from a reduction of the size of the pattern table. This yields an interesting theoretical new problem: given a set H of haplotypes, represented by a pattern table T in standard form, what is the *smallest* (with respect to the number of patterns) pattern table which still represents H? This problem deserves further study in the future.

Acknowledgments

GL thanks Paola Bertolazzi and Leonardo Tininini for many helpful discussions. Part of this work was supported through MIUR's grant P.R.I.N. "Algoritmi di ottimizzazione per l'analisi comparativa di dati genomici di grandi dimensioni ".

References

[1] `http://home.uchicago.edu/~rhudson1`. the World Wide Web.

[2] `http://pga.gs.washington.edu`. the World Wide Web.

[3] `http://www.gnu.org/software/glpk/`. the World Wide Web.

[4] V. Bafna, D. Gusfield, G. Lancia, and S. Yooseph. Haplotyping as perfect phylogeny: a direct approach. *Journal of Computational Biology*, 10(3-4):323–340, 2003.

[5] D. G. Brown and I. M. Harrower. A new integer programming formulation for the pure parsimony problem in haplotype analysis. In *Proceedings of Annual Workshop on Algorithms in Bioinformatics (WABI)*, Lecture Notes in Computer Science, pages 254–265. Springer, 2004.

[6] D. G. Brown and I. M. Harrower. A new formulation for haplotype inference by pure parsimony. Technical Report CS-2005-03, University of Waterloo, Department of Computer Science, Waterloo, Canada, 2005.

[7] D.G. Brown and I.M. Harrower. Integer programming approaches to haplotype inference by pure parsimony. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3:348–359, 2006.

[8] International HapMap Consortium. The HapMap project. *Nature*, 426:789–796, 2003.

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

[10] E. Eskin, E. Halperin, and R. Karp. Efficient reconstruction of haplotype structure via perfect phylogeny. *Journal of Bioinformatics and Computational Biology*, 1(1):1–20, 2003.

[11] D. Gusfield. A practical algorithm for optimal inference of haplotypes from diploid populations. In R. Altman, T.L. Bailey, P. Bourne, M. Gribskov, T. Lengauer, I.N. Shindyalov, L.F. Ten Eyck, and H. Weissig, editors, *Proceedings of the Annual International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 183–189, Menlo Park, CA, 2000. AAAI Press.

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

[13] D. Gusfield. Haplotype inference by pure parsimony. In *Proceedings of the Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 2676 of *Lecture Notes in Computer Science*, pages 144–155. Springer, 2003.

[14] D. Gusfield and S. H. Orzack. Haplotype inference. In *Handbook of Computational Molecular Biology*, pages 1–28. Champman and Hall/CRC-press, 2005.

[15] B. Halldorsson, S. Istrail, and R. Sharan. Islands of tractability for parsimony haplotyping. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, in press, 2007.

[16] B. V. Halldrsson, V. Bafna, N. Edwards, R. Lippert, S. Yooseph, and S. Istrail. A survey of computational methods for determining haplotypes. In *Computational methods for SNP and haplotype inference: DIMACS/RECOMB satellite workshop*, number 2983 in Lecture Notes in Computer Science, pages 26–47. Springer, 2004.

[17] Y.T. Huang, K.M. Chao, and T. Chen. An approximation algorithm for haplotype inference by maximum parsimony. In *ACM Symposium on Applied Computing (SAC)*, pages 146–150, 2005.

[18] R. Hudson. Generating samples under the wright-fisher neutral model of genetic variation. *Bioinformatics*, 18(2):337–338, 2002.

[19] International Human Genome Sequencing Consortium. Initial sequencing and analysis of the human genome. *Nature*, 409:860–921, 2001.

[20] K. Kalpakis and P. Namjoshi. Haplotype phasing using semidefinite programming. *technical report*, 2004.

[21] G. Kimmel and R. Shamir. Gerbil: Genotype resolution and block identification using likelihood. *Proceedings of the National Academy of Sciences*, 102:158–162, 2005.

[22] G. Lancia, C. Pinotti, and R. Rizzi. Haplotyping populations by pure parsimony: Complexity, exact and approximation algorithms. *INFORMS Journal on Computing*, 16(4):17–29, 2004.

[23] G. Lancia and R. Rizzi. A polynomial solution to a special case of the parsimony haplotyping problem. *Operations Research Letters*, 34(3):289–295, 2006.

[24] I. Lynce and J. Marques-Silva. SAT in bioinformatics: Making the case with haplotype inference. In *Theory and Application of Satisfiability Testing - SAT06*, volume 4121 of *Lecture Notes in Computer Science*, pages 136–141. Springer, 2006.

[25] T. Niu, Z. S. Qin, X. Xu, and J. S. Liu. Bayesian haplotype inference for multiple linked single-nucleotide polymorphisms. *American Journal of Human Genetics*, 70:157–169, 2002.

[26] M. Stephens and P. Donnelly. A comparison of bayesian methods for haplotype reconstruction from population genotype data. *American Journal of Human Genetics*, 73:1162–1169, 2003.

[27] M. Stephens and P. Scheet. Accounting for decay of linkage disequilibrium in haplotype inference and missing-data imputation. *American Journal of Human Genetics*, 76:449–462, 2005.

[28] J.C. Venter *et al.* The sequence of the human genome. *Science*, 291:1304–1351, 2001.

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