International Journal on Artificial Intelligence Tools

Haplotype Inference with Boolean Satisfiability

INÊS LYNCE
Instituto Superior Técnico/INESC-ID, Technical University of Lisbon
Rua Alves Redol, 9, 1000-029 Lisboa, Portugal
ines@sat.inesc-id.pt

JOÃO MARQUES-SILVA
School of Electronics and Computer Science, University of Southampton
Highfield, Southampton SO17 1BJ, United Kingdom
jpms@ecs.soton.ac.uk

Received (5 February 2007)

Mutation in DNA is the principal cause for differences among human beings, and Single Nucleotide Polymorphisms (SNPs) are the most common mutations. Hence, a fundamental task is to complete a map of haplotypes (which identify SNPs) in the human population. Associated with this effort, a key computational problem is the inference of haplotype data from genotype data, since in practice genotype data rather than haplotype data is usually obtained. Different haplotype inference approaches have been proposed, including the utilization of statistical methods and the utilization of the pure parsimony criterion. The problem of haplotype inference by pure parsimony (HIPP) is interesting not only because of its application to haplotype inference, but also because it is a challenging NP-hard problem, being APX-hard. Recent work has shown that a SAT-based approach is the most efficient approach for the problem of haplotype inference by pure parsimony (HIPP), being several orders of magnitude faster than existing integer linear programming and branch and bound solutions. This paper provides a detailed description of SHIPs, a SAT-based approach for the HIPP problem, and presents comprehensive experimental results comparing SHIPs with all other exact approaches for the HIPP problem. These results confirm that SHIPs is currently the most effective approach for the HIPP problem.

Keywords: Boolean Satisfiability; Haplotype Inference, Bioinformatics

1. Introduction

Over the last few years, an emphasis in human genomics has been on identifying genetic variations among different people. A comprehensive search for genetic influences on disease involves examining all genetic differences in a large number of affected individuals. This allows to systematically test common genetic variants for their role in disease; such variants explain much of the genetic diversity in our species. A particular focus has been put on the identification of Single Nucleotide Polymorphisms (SNPs), point mutations found in the population, most often with only two possible values, and tracking their inheritance. However, this process is

in practice very difficult due to technological limitations. At a genomic position for which an individual inherited two different values, it is currently difficult to identify from which parent each value was inherited. Instead, researchers can only identify whether the individual is heterozygous at that position, i.e. whether the values inherited from both parents are different. This process of going from genotypes (which include the ambiguity at heterozygous positions) to haplotypes (where we know from which parent each SNP is inherited) is called *haplotype inference*.

The next high priority phase of human genomics involves the development of a full haplotype map of the human genome. The HapMap project [31] represents the best known effort to develop a public resource that will help researchers to find genes associated with human disease. The HapMap resource can guide the design and analysis of genetic association studies, shed light on structural variation and recombination, and identify sites that may have been subject to natural selection during human evolution. The achievement of these goals depends on an efficient method that performs inference of haplotypes from genotypes. A well-known approach to the haplotype inference problem is called Haplotype Inference by Pure Parsimony (HIPP).[1,12,14,16,18,21,34] The problem of finding such solutions is APX-hard (and, therefore, NP-hard).[21] The HIPP problem consists of finding a solution to the haplotype inference problem that minimizes the total number of distinct haplotypes used. Current methods for solving the haplotype inference problem by pure parsimony utilize Integer Linear Programming (ILP)[12,1,2] and branch and bound algorithms.[34]

Recent work [23,24] has proposed the utilization of SAT for the HIPP problem. Preliminary results are significant: on existing well-known problem instances, the SAT-based HIPP solution (SHIPs) is the most efficient approach to the HIPP problem, being orders of magnitude faster than any other alternative exact approach for the HIPP problem. Moreover, SHIPs is the only exact approach for the HIPP problem capable of solving a large number of problem instances from different sources.

This paper provides a unified description of the SAT-based model proposed in two conference papers.[23,24] In addition, the paper proposes an alternative lower bound procedure, which improves the one used in previous versions of SHIPs.[23,24] Moreover, the paper provides detailed experimental results comparing the performance of all existing solutions to the HIPP problem. The results confirm that the SAT-based approach is currently the most efficient solution to the HIPP problem.[23,24] The promising experimental results also make SHIPs competitive with statistical methods.[30,26] One potential advantage of HIPP is result reproducibility.[13] This is in contrast with statistical methods, which are unable to ensure reproducibility of results.[13] Hence, in settings where result reproducibility is a key requirement, SHIPs is posed as a promising solution for the haplotype inference problem.

The paper also compares two different SAT solvers to be used within SHIPs. Currently, the available SAT solvers are MiniSAT [7] and SATZ.[22] The results clearly suggest that MiniSAT, and consequently modern SAT techniques (including clause learning, search restarts and lazy data structures) are essential for solving the HIPP

problem efficiently.

Moreover, the practical effectiveness of SHIPs allows considering the SAT-based implementation of criteria other than pure parsimony. Examples of alternative criteria are outlined in Section 6.

Finally, the different SHIPs models provide a new, relevant, and essentially endless, source of challenging SAT problem instances, most of which can only be solved by SAT solvers using the most effective SAT techniques, and many of which cannot yet be solved by a state of the art SAT solver.

The paper is organized as follows. Section 2 reviews the problem of haplotype inference, and formalizes haplotype inference by pure parsimony. Afterwards, Section 3 reviews related work, focusing on solutions based on integer linear programming. Section 4 provides a detailed description of the SAT-based approach for the HIPP problem. Section 5 provides experimental results, comparing SHIPs with all existing exact solutions to the HIPP problem, and Section 6 concludes the paper.

2. Haplotype Inference

The *genome* is the whole hereditary information of an organism that is encoded in the *DNA*. The DNA is normally organized in the form of a set of large macromolecules called *chromosomes*. A chromosome contains, minimally, a very long, continuous piece of DNA, regulatory elements and other intervening elements. In diploid organisms, chromosomes are grouped in sets of two, where one chromosome is inherited from the father and the other from the mother. In what follows we will only consider diploid organisms.

The DNA is a double-stranded molecule held together by weak bonds between base pairs of *nucleotides*. The position of a specific nucleotide is called a *site* or *locus*. There are four different types of nucleotides in DNA, which can be distinguished by the bases they contain. These bases are adenine (A), guanine (G), cytosine (C), and thymine (T). Base pairs are formed only between A and T and between G and C; thus the base sequence of each single strand can be deduced from that of the other strand in the DNA.

Replication is performed by first splitting the DNA double strand, and afterwards recreating each one of the two new strands with the corresponding bases. Since each of the bases can only combine with one other base, the bases on the old strand dictate which bases will be on the new strand. Each of the two double strands obtained at the end will end up as a complete replica of the original DNA, unless a mutation occurs.

A *mutation* is an imperfection in the replication process, leading to DNA sequence variations: a base is accidentally skipped, inserted, or incorrectly copied. Once propagated to the next generation, a mutation may lead to variations within a population.

A *Single Nucleotide Polymorphism* or *SNP* is a DNA sequence variation, occurring when a single nucleotide is altered. For example, a SNP might change the

nucleotide sequence TTACCGT to TCACCGT. (A variation must occur in at least 1% of the population to be considered a SNP.) SNPs make up 90% of all human genetic variations, and occur every 100 to 300 bases along the human genome. It has been observed that around two of every three SNPs substitute C with T. Variations in the DNA sequences of humans can affect how humans respond to diseases and treatments.

A *gene* is an ordered sequence of nucleotides located in a particular position that encodes a specific function. The variants of a single gene are named *alleles*. Different alleles give rise to differences in traits.

Different alleles may be explained in terms of SNPs. Depending on the number of possible alleles, a SNP site can be *biallelic* (two different alleles) or *multiallelic* (more than two different alleles). Almost always, there are only two possible alleles for a SNP site among the individuals in a population. In what follows we will only consider biallelic SNPs.

A *haplotype* is the genetic constitution of an individual chromosome. The underlying data that forms a haplotype can be the full DNA sequence in the region, or more commonly the SNPs in that region. Diploid organisms pair homologous chromosomes, and thus contain two haplotypes, one inherited from each parent. The *genotype* describes the conflated data of the two haplotypes. In other words, an *explanation* for a genotype is a pair of haplotypes. Conversely, this pair of haplotypes explains the genotype. If for a given site both copies of the haplotype have the same value, then the genotype is said to be *homozygous* at that site; otherwise is said to be *heterozygous*.

Given a set $\mathcal{G}$ of n genotypes, each of length m, the haplotype inference problem consists in finding a set $\mathcal{H}$ of $2 \cdot n$ haplotypes, not necessarily different, such that for each genotype $g_i \in G$ there is at least one pair of haplotypes (h_j, h_k), with h_j and $h_k \in \mathcal{H}$ such that the pair (h_j, h_k) explains g_i. The variable n denotes the number of individuals in the sample, and m denotes the number of SNP sites. g_i denotes a specific genotype, with $1 \leq i \leq n$. (Furthermore, g_{ij} denotes a specific site j in genotype g_i, with $1 \leq j \leq m$.)

Without loss of generality, we may assume that the values of the two possible alleles of each SNP are always 0 or 1. Value 0 represents the wild type and value 1 represents the mutant. A haplotype is then a string over the alphabet {0,1}. Moreover, genotypes may be represented by extending the alphabet used for representing haplotypes to {0,1,2}. Homozygous sites are represented by values 0 or 1, depending on whether both haplotypes have value 0 or 1 at that site, respectively. Heterozygous sites are represented by value 2.

Table 1 gives twelve haplotypes of the β_2AR genes.[a] Each haplotype has 13 sites. Each site corresponds to a specific nucleotide in a gene where a mutation occurred. Each nucleotide is characterized by the position in the sequence. For each nucleotide, a pair of possible alleles is given: the first allele corresponds to the wild

[a]Data made available by Drysdale et. al.[6]

Table 1. Haplotypes of the β_2AR genes.

Nucl.	-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	1000000010000
h_2	A	C	G	G	C	C	C	C	G	G	G	C	C	1001011101000
h_3	G	A	A	C	C	T	T	T	A	C	G	C	C	0110000010000
h_4	G	C	A	C	C	T	T	T	A	C	G	C	C	0010000010000
h_5	G	C	A	C	C	T	T	T	G	C	G	C	C	0010000000000
h_6	G	C	G	C	C	T	T	T	G	C	A	C	A	0000000000101
h_7	G	C	G	C	C	T	T	T	G	C	A	T	A	0000000000111
h_8	G	C	A	C	C	T	T	T	A	C	A	C	A	0010000010101
h_9	A	C	G	C	T	T	T	T	A	C	G	C	C	1000100010000
h_{10}	G	C	G	C	C	T	T	T	G	C	A	C	C	0000000000100
h_{11}	G	C	G	C	C	T	T	T	G	C	G	C	C	0000000000000
h_{12}	A	C	G	G	C	T	T	T	A	C	G	C	C	1001000010000

type and the second to the mutant. The last column of each haplotype contains the representation of that haplotype. Different genotypes may result from this set of haplotypes: for example, the pair of haplotypes (h_1, h_7) explains the genotype 200000020222, whereas the pair (h_7, h_8) explains the genotype 0020000020121.

One of the approaches to the haplotype inference problem is called Haplotype Inference by Pure Parsimony (HIPP). A solution to this problem minimizes the total number of distinct haplotypes used. The HIPP problem is APX-hard (see Refs. 12, 21 for proofs and historical perspective). Experimental results provide support for this method: the number of haplotypes in a large population is typically very small, although genotypes exhibit a great diversity. For example, consider the set of genotypes: 2120, 2102, and 1221. There are solutions for this example that use six distinct haplotypes, but the solution 0100/1110, 0100/1101, 1011/1101 uses only four distinct haplotypes. Moreover, existing empirical evaluations provide strong evidence that the accuracy of HIPP is comparable with alternative approaches.[12,16,34]

3. Related Work

The problem of haplotype inference is an active area of research. The most widespread approaches are based on statistical methods.[30,26] An alternative is haplotype inference by pure parsimony.[13] A number of solutions exist for the HIPP problem, based on Integer Linear Programming (ILP) and branch-and-bound, which are reviewed below. Heuristic approximation algorithms include the one introduced by Huang et al.[16] One additional approach for the HIPP problem has been proposed by Kalpakis and Namjoshi,[18] using relaxations of a semidefinite programming model. Moreover, there has been work on solving restricted cases of haplotype inference,[15,5] some of which based on 2SAT.[15]

3.1. *Exponential-Size Integer Linear Programming Models*

Over the last few years, a number of authors have proposed optimization models for the HIPP problem. With a few notable exceptions,[34] the majority of the proposed models are based on Integer Linear Programming (ILP).[12,14,1,2]

The original ILP model, *RTIP*, has linear space complexity on the number of candidate haplotypes,[12,13] and so it is exponential on the number of given genotypes. For each genotype, all candidate pairs of haplotypes for explaining the genotype are enumerated. For example, given genotype 02122, the candidate pairs of haplotypes for explaining it are: (00100,01111), (01100,00111), (00110,01101) and (00101,01110). More generally, each genotype having k heterozygous sites is explained by 2^{k-1} pairs of haplotypes. Hence, the space complexity is $\mathcal{O}(2^m)$ where m is the number of sites, which represents the maximum number of heterozygous sites per genotype.

A Boolean variable $y_{i,u}$ is associated with each pair u of haplotypes that can explain a given genotype g_i, and denotes whether this pair of haplotypes is used for explaining g_i. A cardinality constraint,

$$\sum_r y_{i,u} = 1 \tag{1}$$

requires that exactly one pair of haplotypes must be used for explaining each genotype, among all pairs that can explain the genotype. Each candidate haplotype is associated with a dedicated variable x_v, such that $x_v = 1$ if the haplotype is used. The utilization of a specific pair of haplotypes for explaining a genotype (i.e. $y_{i,u} = 1$) implies the respective x_v variable, $y_{i,u} \rightarrow x_v$, for each haplotype in the pair. The cost function consists of minimizing the number of haplotypes used,

$$\text{minimize} \sum x_v \tag{2}$$

This model corresponds to the *TIP* model.[12] The *RTIP* model introduces one essential simplification. If genotype g_i can be explained by pair of haplotypes (h_a, h_b), such that both h_a and h_b cannot explain any other genotype, then pair of haplotypes (h_a, h_b) needs not be considered for explaining g_i. If all pairs are discarded for a genotype g_i, then it suffices to pick any pair for explaining g_i.

3.2. *Polynomial-Size Integer Linear Programming Models*

A more recent ILP model, *PolyIP*, is polynomial in the number of sites m and population size n, [14,1] with a number of constraints and variables, respectively, in $\Theta(n^2 m)$ and $\Theta(n^2 + n\,m)$. The PolyIP model represents the $2\,n$ candidate haplotypes as sequences of Boolean variables, and then establishes conditions for the haplotypes to explain the corresponding genotypes, such that the total number of distinct haplotypes is minimized. Haplotypes are represented with Boolean variables $y_{i\,j}$, $1 \leq i \leq 2\,n$ and $1 \leq j \leq m$, i.e. m variables for each of the $2\,n$ candidate haplotypes.

First, the PolyIP model defines conditions on the sites, with $1 \leq i \leq n$ and $1 \leq j \leq m$:

$$\begin{aligned} y_{2i-1\,j} = 0 \text{ and } y_{2i\,j} = 0, \text{ if } g_{ij} = 0, \\ y_{2i-1\,j} = 1 \text{ and } y_{2i\,j} = 1, \text{ if } g_{ij} = 1, \\ y_{2i-1\,j} + y_{2i\,j} = 1 \text{ if } g_{ij} = 2 \end{aligned} \tag{3}$$

where $g_{ij} \in \{0, 1, 2\}$ denotes the possible values at each site. Second, the PolyIP model defines conditions for identifying different haplotypes, with $1 \leq i, l \leq 2n$ and $1 \leq j \leq m$. Boolean variable $d_{i\,l}$ is defined such that $d_{i\,l} = 1$ if $h_i \neq h_l$. The resulting conditions become:

$$\begin{aligned} y_{i\,j} - y_{l\,j} \leq d_{i\,l} \\ y_{l\,j} - y_{i\,j} \leq d_{i\,l} \end{aligned} \tag{4}$$

If at least one site of h_i and h_l differs, then $d_{i,l}$ needs to be assigned value 1.

Third, the model introduces the x_i variables denoting whether h_i is different from all previous haplotypes h_l, where $1 \leq l < i$, and defines conditions on these variables. Boolean variable x_i is defined such that $x_i = 1$ if h_i is unique with respect to the previous haplotypes. Thus, if h_i is unique, then $\sum_{l=1}^{i-1} d_{l\,i} = i - 1$; otherwise $\sum_{l=1}^{i-1} d_{l\,i} < i - 1$. As a result, the condition on variable x_i becomes:

$$x_i \geq 2 - i + \sum_{l=1}^{i-1} d_{l\,i} \tag{5}$$

Finally, the cost function consists of minimizing the number of different haplotypes:

$$\text{minimize } \sum_{i=1}^{2n} x_i \tag{6}$$

A number of optimizations have been proposed to the basic PolyIP model,[1] with the purpose of pruning the search space to be handled by the ILP solver.

More recently, Brown and Harrower introduced a new polynomial-size formulation, *HybridIP*, representing a hybrid of the RTIP and PolyIP formulations.[2] Nevertheless, our experimental results (see Section 5) suggest that the performance of the two polynomial models does not differ significantly.

3.3. *Branch-and-Bound Solutions*

The RTIP model [12] inspired a branch-and-bound algorithm, *Hapar*, to the HIPP problem.[34] Similarly to RTIP, Hapar considers all pairs of haplotypes explaining each genotype. A greedy procedure is used for computing an initial set of haplotypes explaining all genotypes, which serves as the initial upper bound. Afterwards standard branch-and-bound search is performed. The existing upper bound is used for pruning portions of the search tree where a solution smaller than the current upper bound cannot be identified. Moreover, one significant optimization consists of eliminating pairs of haplotypes that are guaranteed not to yield solutions better than the solutions produced by other pairs of haplotypes.[34]

3.4. *Simplifications to the Input Genotypes*

A key technique for tackling the HIPP problem consists of using the structural properties of genotypes with the purpose of reducing the search space.[b] Observe that, in the presence of two equal genotypes, one can be discarded, assuming the two genotypes are to be explained identically. Hence, the solution for the remaining genotype is also the solution for the discarded genotype. Duplicate sites can also be discarded, i.e. pairs of sites for which every genotype has equal values. Moreover, complemented sites can also be discarded, where two sites are *complemented* if the homozygous sites have complemented values.

For example, consider the following set of genotypes:

$$\begin{array}{c} 00102 \\ 11021 \\ 22210 \\ 00102 \end{array} \tag{7}$$

For this set of genotypes, the last genotype equals the first genotype, and so can be discarded. Moreover, the second site is a duplicate of the first site, and so the second site can be discarded. Finally, the third site is the complement of the first site, and so the third site can also be discarded. The resulting simplified instance becomes:

$$\begin{array}{c} 002 \\ 121 \\ 210 \end{array} \tag{8}$$

In practice, the simplification of the input genotypes is an essential step for any approach to the HIPP problem, allowing significant reductions in the number of genotypes and sites.[2,23]

4. SAT-Based Haplotype Inference

This section presents the SAT-based approach for the HIPP problem (SHIPs). The section is organized in three main parts. First the top-level SHIPs algorithm is described. Afterwards, Section 4.2 presents the *core model*, which contains the key ideas of the SAT-based model for the HIPP problem. The core model is ineffective in practice.[23,24] Hence, a number of key optimizations are detailed in Section 4.3. The resulting (complete) SHIPs model is extremely effective in practice.

4.1. *The SHIPs Algorithm*

Algorithm 1 summarizes the top-level operation of SHIPs. The algorithm accepts a set of genotypes $\mathcal{G}$ and a lower bound on the number of haplotypes lb necessary to explain the set of genotypes. A trivial value for lb is 1. The algorithm searches for

[b]These techniques have previously been proposed for ILP approaches to the HIPP problem.[2]

Algorithm 1 Top-level SHIPs algorithm

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

the least value r such that there exists a set $\mathcal{H}$ of haplotypes, with $r = |\mathcal{H}|$, which explain all genotypes in $\mathcal{G}$. Observe that the value of r is guaranteed to be such that $lb \leq r \leq 2\,n$. Clearly, a solution with $2\,n$ haplotypes is guaranteed to exist. For each value of r considered, a CNF formula φ^r is created, and a SAT solver is invoked (identified by the function call SAT(φ^r)).

The search for the minimum number of haplotypes proceeds through increasing values of r, starting with $r = lb$ and terminating when the resulting instance of SAT is satisfiable. The last value of r is returned.

Other organizations of the top-level SHIPs algorithm could be considered. Examples include searching down from an upper bound or performing a binary search between a lower and an upper bound. The motivation for searching up from a lower bound is to ensure that the generated CNF formulas are the *most* compact. With the proposed approach, the largest CNF formula is the one generated for the number of haplotypes corresponding to the target solution. Alternative approaches would generate necessarily larger CNF formulas.

The simplification techniques described in Section 3.4 are implemented via the function call ApplySimplifications($\mathcal{G}$) in line 1 of Algorithm 1.

4.2. *The Core Model*

In what follows we assume n genotypes each with m sites. The same indexes will be used throughout: i ranges over the genotypes and j over the sites, with $1 \leq i \leq n$ and $1 \leq j \leq m$. In addition, r candidate haplotypes are considered, each with m sites, and with $1 \leq r \leq 2\,n$. An additional index k is associated with haplotypes, $1 \leq k \leq r$. As a result, $h_{kj} \in \{0, 1\}$ denotes the j^{th} site of haplotype k. Moreover, a haplotype h_k, is viewed as a m-bit word, $h_{k\,1} \ldots h_{k\,m}$. A valuation $v : \{h_{k\,1}, \ldots, h_{k\,m}\} \rightarrow \{0, 1\}$ to the bits of h_k is denoted by h_k^v. Observe that valuations can be extended to other sets of variables.

For a given value of r, the model considers r haplotypes and seeks to associate two haplotypes (which can possibly correspond to the same haplotype) with each genotype g_i, $1 \leq i \leq n$. The process of selecting two haplotypes for explaining each

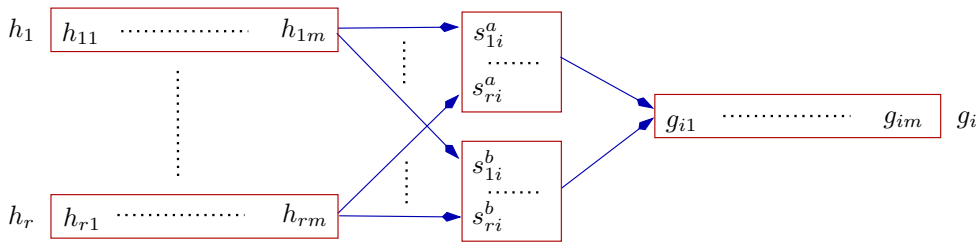

Fig. 1. Associating haplotypes with each genotype

genotype is illustrated in Figure 1.

For each genotype g_i, the model uses *selector* variables for selecting which haplotypes are used for explaining g_i. Since the genotype is to be explained by *two* haplotypes, the model uses two sets, a and b, of r selector variables, respectively s^a_{ki} and s^b_{ki}, with $k = 1, \ldots, r$. Hence, genotype g_i is explained by haplotypes h_{k_1} and h_{k_2} if $s^a_{k_1 i} = 1$ and $s^b_{k_2 i} = 1$. Clearly, g_i is also explained by the same haplotypes if $s^a_{k_2 i} = 1$ and $s^b_{k_1 i} = 1$.

Observe that the operation of the SHIPs model can be described by the matrix formulation $G = S^a \cdot H \oplus S^b \cdot H$, where G is a $n \times m$ matrix describing the genotypes, H is a $r \times m$ matrix of haplotype variables, S^a and S^b are $n \times r$ matrices of selector variables, and $\oplus$ is the explanation operation.[c]

We can now derive the conditions for the SHIPs model. If a site g_{ij} of a genotype g_i is either 0 or 1, then this is the value required at this site and so this information is used by the model.

If a site g_{ij} is 0, then the model requires, for $k = 1, \ldots, r$:

$$(\neg h_{kj} \vee \neg s^a_{ki}) \wedge (\neg h_{kj} \vee \neg s^b_{ki}) \tag{9}$$

Hence, if haplotype k is selected for explaining genotype i, either by the a or the b representative, then the value of haplotype k at site j *must* be 0.

If a site g_{ij} is 1, then the model requires, for $k = 1, \ldots, r$:

$$(h_{kj} \vee \neg s^a_{ki}) \wedge (h_{kj} \vee \neg s^b_{ki}) \tag{10}$$

Hence, if haplotype k is selected for explaining genotype i, either by the a or the b representatives, then the value of haplotype k at site j *must* be 1.

Otherwise, one requires that the haplotypes explaining the genotype g_i have opposing values at site i. This is done by creating one variable $t_{ij} \in \{0, 1\}$, such that site j of the haplotype selected by the a representative selector assumes the same value as t_{ij}, and site j of the haplotype selected by the b representative selector

[c]An alternative matrix-based model for capturing the process of selecting two haplotypes for describing each genotype has been considered by other authors in the context of using semidefinite programming for the HIPP problem.[18]

assumes the complemented value of t_{ij}. As a result, the model requires, for $k = 1, \ldots, r$:

$$\begin{array}{l}(h_{kj} \vee \neg t_{ij} \vee \neg s^a_{ki}) \wedge (\neg h_{kj} \vee t_{ij} \vee \neg s^a_{ki}) \wedge \\ (h_{kj} \vee t_{ij} \vee \neg s^b_{ki}) \wedge (\neg h_{kj} \vee \neg t_{ij} \vee \neg s^b_{ki})\end{array} \tag{11}$$

Observe that h_{kj} equals t_{ij} if $s^a_{ki} = 1$ and h_{kj} equals $\neg t_{ij}$ if $s^b_{ki} = 1$.

Clearly, for each i, and for a or b, it is necessary that exactly one haplotype is used, and so exactly one selector variable can be assigned value 1. This can be captured with the following cardinality constraints:

$$\left(\sum_{k=1}^{r} s^a_{ki} = 1\right) \wedge \left(\sum_{k=1}^{r} s^b_{ki} = 1\right) \tag{12}$$

The traditional way of encoding cardinality constraints into SAT is by explicitly representing clauses to exclude all possible combinations of variables being simultaneously true, thus being quadratic on the number of variables [33]. Other encodings typically require a linear number of clauses at the cost of introducing additional variables [11,29]. The general idea of these improved encodings is to build a count-and-compare hardware circuit and then translate this circuit to CNF.

Given that the cardinality constraints required represent the constraint = 1, we have used a CNF representation of a simplified adder circuit requiring the output of the adder to be equal to 1. The encoding in CNF of the adder circuit requires the utilization of additional variables v^a_{ki} and v^b_{ki}. These variables are set to 1 if the partial sum is equal to 1. We consider the case for the a variables; for the b variables the model is exactly the same. The v^a_{ki} variables are defined as follows, with $1 \leq k \leq r$:

$$\begin{array}{ll}(\neg v^a_{1\,i}) & \wedge \\ (\neg v^a_{ki} \vee \neg s^a_{ki}) & \wedge \\ v^a_{k+1\,i} \leftrightarrow (s^a_{ki} \vee v^a_{ki}) & \wedge \\ (v^a_{r+1\,i}) & \end{array} \tag{13}$$

The CNF clauses for these constraints are straightforward to generate. Observe that the size of the above formula is linear in the number of s variables, and that the encoding is decided by unit propagation, i.e. it ensures arc-consistency.[11] Hence, the proposed encoding is optimal in the number of clauses produced.[29]

The core SHIPs model is summarized in Table 2. Next, we analyze the correctness and the space complexity of the model. Given a set $\mathcal{G}$ of genotypes, let φ^r be the CNF formula associated with r candidate haplotypes. Using equations (9), (10), (11), (12), (13) and taking into consideration that $1 \leq i \leq n$, $1 \leq j \leq m$ and $1 \leq k \leq r_f$, where r_f is the final value of r, we can establish the following results.

Theorem 4.1. φ^r is satisfiable iff the set $\mathcal{G}$ of genotypes can be explained with r or fewer haplotypes.

Proof.

Table 2. The Core SHIPs Model.

Condition	Equation	Clauses	Indexes
$g_{ij} = 0$	(9)	$(\neg h_{kj} \vee \neg s^a_{ki}) \wedge (\neg h_{kj} \vee \neg s^b_{ki})$	$1 \leq k \leq r$ $1 \leq i \leq n$ $1 \leq j \leq m$
$g_{ij} = 1$	(10)	$(h_{kj} \vee \neg s^a_{ki}) \wedge (h_{kj} \vee \neg s^b_{ki})$	$1 \leq k \leq r$ $1 \leq i \leq n$ $1 \leq j \leq m$
$g_{ij} = 2$	(11)	$(h_{kj} \vee \neg t_{ij} \vee \neg s^a_{ki}) \wedge (\neg h_{kj} \vee t_{ij} \vee \neg s^a_{ki}) \wedge$ $(h_{kj} \vee t_{ij} \vee \neg s^b_{ki}) \wedge (\neg h_{kj} \vee \neg t_{ij} \vee \neg s^b_{ki})$	$1 \leq k \leq r$ $1 \leq i \leq n$ $1 \leq j \leq m$
(12)	(13)	$(\neg v^a_{1\,i}) \wedge (\neg v^a_{k\,i} \vee \neg s^a_{k\,i}) \wedge (v^a_{r+1\,i}) \wedge$ $(\neg s^a_{k\,i} \vee v^a_{k+1\,i}) \wedge (\neg v^a_{k\,i} \vee v^a_{k+1\,i}) \wedge (s^a_{k\,i} \vee v^a_{k\,i} \vee \neg v^a_{k+1\,i})$ $(\neg v^b_{1\,i}) \wedge (\neg v^b_{k\,i} \vee \neg s^b_{k\,i}) \wedge (v^b_{r+1\,i}) \wedge$ $(\neg s^b_{k\,i} \vee v^b_{k+1\,i}) \wedge (\neg v^b_{k\,i} \vee v^b_{k+1\,i}) \wedge (s^b_{k\,i} \vee v^b_{k\,i} \vee \neg v^b_{k+1\,i})$	$1 \leq k \leq r$ $1 \leq i \leq n$

$\rightarrow$ Let us assume φ^r is satisfiable. From (12) and (13) is is clear that, for each i, $1 \leq i \leq n$, only one a representative selector variable and only one b representative selector variable can be assigned value 1. Without loss of generality let us consider a fixed site j of a fixed genotype i. Exactly one a representative selector variable and one b representative selector variable can be assigned value 1, and so the selector variables do not trivially satisfy the clauses created with equations (9), (10) and (11). Let k_1 and k_2 be the indexes for the a and b representatives, respectively, i.e. $s^a_{k_1 i} = 1$ and $s^b_{k_2 i} = 1$. Now depending on the value of g_{ij} one of (9), (10) and (11) has been used. If (9) was used, then the values of $h_{k_1 j}$ and $h_{k_2 j}$ must be 0, otherwise the formula would not be satisfiable. If (10) was used, then the values of $h_{k_1 j}$ and $h_{k_2 j}$ must be 1, otherwise the formula would not be satisfiable. Finally, if (11) was used, then the values of $h_{k_1 j}$ and $h_{k_2 j}$ must differ, i.e. $h_{k_1 j} = t_{i_j}$ and $h_{k_2 j} = \neg t_{i_j}$; otherwise the formula would not be satisfiable. Hence, genotype g_i is explained by the haplotypes h_{k_1} and h_{k_2}. Observe that the number of haplotypes used for explaining the m genotypes can be equal to or less than r.

$\leftarrow$ Without loss of generality consider a genotype g_i which is explained by two haplotypes h_{k_1} and h_{k_2}. Moreover, consider the formula φ^r. The selector variables can be assigned such that h_{k_1} and h_{k_2} are used for explaining g_i, e.g. by setting $s^a_{k_1 i} = 1$ and $s^a_{k_2 i} = 1$. All other selector variables associated with genotype g_i can be assigned value 0. Hence (12) is satisfied. Now, since h_{k_1} and h_{k_2} explain g_i, for each site j one of following three cases can take place:

(1) The value of g_{ij} is 0, in which case the model uses (9) and the clause is satisfied, since by hypothesis the haplotypes h_{k_1} and h_{k_2} have value 0 at site j.
(2) The value of g_{ij} is 1, in which case the model uses (10) and the clause

is satisfied, since by hypothesis the haplotypes h_{k_1} and h_{k_2} have value 1 at site j.

(3) The value of g_{ij} is 2, in which case the model uses (11) and the associated clauses are satisfied, since by hypothesis the haplotypes have different values at site j and we can assign the Boolean variable t_{ij} so as to satisfy the clauses of (11).

Observe that the number of haplotypes used for explaining the m genotypes can be equal to or less than r.

Hence the results follows. □

Corollary 4.1. *The value returned by Algorithm 1 represents the smallest number of haplotypes that explain all genotypes in set $\mathcal{G}$.*

Proof. Suppose there could exist a smaller value $q < r$ such that the genotypes in $\mathcal{G}$ could be explained with q haplotypes. Since Algorithm 1 considers the values of r in increasing order, then by Theorem 4.1, φ^q would be satisfiable, and so the value returned by the algorithm would be q; a contradiction. □

Theorem 4.2. If a solution is found with r_f haplotypes, then the number of constraints of the SAT model is $\mathcal{O}(r_f\, n\, m)$, which is $\mathcal{O}(n^2\, m)$. In addition, the number of variables is $\mathcal{O}(n\, m + r_f\, m + r_f\, n)$, which is $\mathcal{O}(n^2 + n\, m)$.

Proof. Assume a solution is found with r_f haplotypes. Clearly, the number of t variables is $\mathcal{O}(n\, m)$, the number of h variables is $\mathcal{O}(r_f\, m)$, and the number of s and v variables is $\mathcal{O}(r_f\, n)$. Hence, the number of variables is $\mathcal{O}(n\, m + r_f\, m + r_f\, n)$, which is $\mathcal{O}(n^2 + n\, m)$, since $r_f = \mathcal{O}(n)$.

The number of clauses generated by (9), (10) and (11) is $\mathcal{O}(r_f\, n\, m)$, taking into consideration the ranges for the i, j and k indexes. Finally, the number of clauses generated by (13) is $\mathcal{O}(r_f\, n)$. Hence, the total number of clauses is $\mathcal{O}(r_f\, n\, m)$, which is $\mathcal{O}(n^2\, m)$, since $r_f = \mathcal{O}(n)$. It is also straightforward to conclude that the number of literals is also $\mathcal{O}(r_f\, n\, m)$. □

Finally, observe that the number of constraints and variables of the PolyIP model [1] are, respectively, in $\Theta(n^2\, m)$ and $\Theta(n^2 + n\, m)$, hence exhibiting the same worst-case complexity as the SHIPs model. Nevertheless, r_f is in practice usually much smaller than n, and so the SAT-based model yields significantly more compact representations than other IP models.[1,2]

4.3. *The Complete Model*

As mentioned before, the core SHIPs model is not effective in practice. As a result, several key improvements have been developed, which are essential for obtaining significant performance gains over existing approaches. These improvements are described next.

4.3.1. *Breaking symmetries on the h variables*

A key technique for pruning the search space is motivated by observing the existence of symmetries in the problem formulation. Consider two haplotypes h_{k_1} and h_{k_2}, and the selector variables $s^a_{k_1 i}$, $s^a_{k_2 i}$, $s^b_{k_1 i}$ and $s^b_{k_2 i}$. Furthermore, consider Boolean valuations v_x and v_y to the sites of haplotypes h_{k_1} and h_{k_2}. Then, $h^{v_x}_{k_1}$ and $h^{v_y}_{k_2}$, with $s^a_{k_1 i} s^a_{k_2 i} s^b_{k_1 i} s^b_{k_2 i} = 1001$, correspond to $h^{v_y}_{k_1}$ and $h^{v_x}_{k_2}$, with $s^a_{k_1 i} s^a_{k_2 i} s^b_{k_1 i} s^b_{k_2 i} = 0110$, and one of the assignments can be eliminated. Elimination of redundant assignments can be achieved by enforcing an ordering of the Boolean valuations to the haplotypes.[d] Hence, for any valuation v to the problem variables we require:

$$h^v_1 < h^v_2 < \ldots < h^v_r \tag{14}$$

Observe that the matrix model proposed earlier in Section 4.2, $G = S^a \cdot H \oplus S^b \cdot H$, allows applying symmetry breaking by lexicographically ordering the rows of matrix $H = [h_1 h_2 \ldots h_r]^T$.[8]

It is straightforward to enforce each sorting constraint between two haplotypes in linear size on the number of sites. This is done by representing in CNF a Boolean comparator circuit between h_k and h_{k+1}, with $1 \leq k < r$, and requiring $h_k < h_{k+1}$. One possible solution consists of using additional variables lt_{kj} and ltt_{kj}, and the comparison constraints become, with $1 \leq k \leq r$:

$$\begin{array}{ll} (\neg lt_{k\,0}) & \wedge \\ ltt_{k\,j} \leftrightarrow (\neg h_{k\,j} \wedge h_{k+1\,j}) & \wedge \\ lt_{k\,j} \leftrightarrow (ltt_{k\,j} \vee lt_{k\,j-1}) & \wedge \\ (lt_{k\,m}) & \wedge \\ (\neg h_{k\,j} \vee h_{k+1\,j} \vee lt_{k\,j-1}) & \end{array} \tag{15}$$

which are straightforward to represent in CNF. The number of added clauses is $\mathcal{O}(r\,m)$, hence not affecting the asymptotic complexity of the number of clauses.

4.3.2. *Breaking symmetries on the s variables*

Besides the symmetries associated with the h variables, it is also possible to eliminate symmetries on the s variables. Observe that the model consists of selecting a candidate haplotype for the a representative and another haplotype for the b representative, such that each genotype is explained by the a and b representatives. Given a set of r candidate haplotypes, let h_{k_1} and h_{k_2}, with $k_1, k_2 \leq r$, be two haplotypes which explain a genotype g_i. This means that g_i can be explained by the assignments $s^a_{k_1 i} s^a_{k_2 i} s^b_{k_1 i} s^b_{k_2 i} = 1001$, but also by the assignments $s^a_{k_1 i} s^a_{k_2 i} s^b_{k_1 i} s^b_{k_2 i} = 0110$.

This symmetry can be eliminated by requiring that only one arrangement of the s variables can be used to explain each genotype g_i. One solution is to require that the haplotype selected by the s^a_{ki} variables always has an index *smaller* than

[d] See for example Ref. 9 for a survey of work on the utilization of lexicographic orderings for symmetry breaking.

the haplotype selected by the s^b_{ki} variables. This requirement is captured by the following conditions:

$$\left(s^a_{ki} \rightarrow \bigwedge_{k_2=1}^{k-1} \neg s^b_{k_2 i} \right) \ , \ \left(s^b_{ki} \rightarrow \bigwedge_{k_1=k+1}^{r} \neg s^a_{k_1 i} \right) \tag{16}$$

Clearly, each condition above can be represented by a single clause, for each k_1 (or k_2) and i. Moreover, observe that for genotypes without homozygous sites, the upper limit of the first constraint can be set to k and the lower limit of the second condition can be set to k. Overall, the number of added clauses is $\mathcal{O}(r\,n)$, for r candidate haplotypes, and so the added clauses do not affect the worst-case space complexity of the model.

Observe that the matrix model proposed earlier in Section 4.2, $G = S^a \cdot H \oplus S^b \cdot H$, allows again an alternative representation of symmetry breaking.[8] If $S^a = [s^a_1 \ldots s^a_n]^T$ and $S^b = [s^b_1 \ldots s^b_n]^T$, then we can impose the condition $s^a_i \leq s^b_i$, $1 \leq i \leq n$, i.e. for each genotype i, the strings representing the selector variables a and the selector variables b are lexicographically ordered.

4.3.3. *Constraining the s variables of incompatible genotypes*

The s variables can be further constrained due to the fact that the haplotypes that can explain a given genotype may not be used for explaining another genotype.

Definition 4.1. Two genotypes, g_i and g_l, are declared *incompatible* iff there exists a site for which the value of one genotype is 0 and the other is 1.

For example, $g_1 = 012$ is incompatible with $g_2 = 112$, whereas the genotypes g_1 and $g_3 = 210$ are not incompatible. Incompatible genotypes *cannot* be explained with common haplotypes.

Consider two *incompatible* genotypes, g_{i_1} and g_{i_2}, and a candidate haplotype h_k. Hence, if either $s^a_{ki_1}$ or $s^b_{ki_1}$ is activated, and so h_k is used for explaining genotype g_{i_1}, then haplotype h_k *cannot* be used for explaining g_{i_2}; hence both $s^a_{ki_2}$ and $s^b_{ki_2}$ *must not* be activated.

The implementation of this condition is achieved by adding the following clauses for each pair of incompatible genotypes g_{i_1} and g_{i_2} and for each candidate haplotype h_k:

$$(\neg s^a_{ki_1} \vee \neg s^a_{ki_2}) \wedge (\neg s^a_{ki_1} \vee \neg s^b_{ki_2}) \wedge (\neg s^b_{ki_1} \vee \neg s^a_{ki_2}) \wedge (\neg s^b_{ki_1} \vee \neg s^b_{ki_2}) \tag{17}$$

In the worst case, the number of incompatible pairs of genotypes is quadratic on the number of genotypes. Hence, the number of added clauses is $\mathcal{O}(r\,n^2)$, where r is the number of candidate haplotypes. In practice, this worst-case space complexity was never observed, and the search pruning obtained from the added clauses can be significant.

The complete SHIPs model is summarized in Table 4.3.3, where predicate $\chi(g_a, g_b)$ holds provided genotypes g_a and g_b are incompatible.

Table 3. The Complete SHIPs Model.

Condition	Equation	Clauses	Indexes
$g_{i\,j} = 0$	(9)	$(\neg h_{k\,j} \vee \neg s^a_{k\,i}) \wedge (\neg h_{k\,j} \vee \neg s^b_{k\,i})$	$1 \leq k \leq r$ $1 \leq i \leq n$ $1 \leq j \leq m$
$g_{i\,j} = 1$	(10)	$(h_{k\,j} \vee \neg s^a_{k\,i}) \wedge (h_{k\,j} \vee \neg s^b_{k\,i})$	$1 \leq k \leq r$ $1 \leq i \leq n$ $1 \leq j \leq m$
$g_{i\,j} = 2$	(11)	$(h_{k\,j} \vee \neg t_{i\,j} \vee \neg s^a_{k\,i}) \wedge (\neg h_{k\,j} \vee t_{i\,j} \vee \neg s^a_{k\,i}) \wedge$ $(h_{k\,j} \vee t_{i\,j} \vee \neg s^b_{k\,i}) \wedge (\neg h_{k\,j} \vee \neg t_{i\,j} \vee \neg s^b_{k\,i})$	$1 \leq k \leq r$ $1 \leq i \leq n$ $1 \leq j \leq m$
(12)	(13)	$(\neg v^a_{1\,i}) \wedge (\neg v^a_{k\,i} \vee \neg s^a_{k\,i}) \wedge (v^a_{r+1\,i}) \wedge$ $(\neg s^a_{k\,i} \vee v^a_{k+1\,i}) \wedge (\neg v^a_{k\,i} \vee v^a_{k+1\,i}) \wedge (s^a_{k\,i} \vee v^a_{k\,i} \vee \neg v^a_{k+1\,i})$ $(\neg v^b_{1\,i}) \wedge (\neg v^b_{k\,i} \vee \neg s^b_{k\,i}) \wedge (v^b_{r+1\,i}) \wedge$ $(\neg s^b_{k\,i} \vee v^b_{k+1\,i}) \wedge (\neg v^b_{k\,i} \vee v^b_{k+1\,i}) \wedge (s^b_{k\,i} \vee v^b_{k\,i} \vee \neg v^b_{k+1\,i})$	$1 \leq k \leq r$ $1 \leq i \leq n$
(14)	(15)	$(\neg lt_{k\,0}) \wedge (lt_{k\,m}) \wedge (h_{k+1\,j} \vee \neg ltt_{k\,j}) \wedge$ $(\neg h_{k\,j} \vee \neg ltt_{k\,j}) \wedge (\neg h_{k+1\,j} \vee h_{k\,j} \vee \neg ltt_{k\,j})$ $(\neg lt_{k\,j-1} \vee lt_{k\,j}) \wedge (\neg ltt_{k\,j} \vee lt_{k\,j}) \wedge$ $(lt_{k\,j-1} \vee ltt_{k\,j} \vee \neg lt_{k\,j})(\neg h_{k\,j} \vee h_{k+1\,j} \vee lt_{k\,j-1})$	$1 \leq k < r$ $1 \leq j \leq m$
–	(16)	$\left(\neg s^a_{k\,i} \vee \bigvee_{k_2=1}^{k-1} \neg s^b_{k_2\,i}\right) \wedge \left(\neg s^b_{k\,i} \vee \bigvee_{k_1=k+1}^{r} \neg s^a_{k_1\,i}\right)$	$1 \leq k \leq r$ $1 \leq i \leq n$
$\chi(g_{i_1}, g_{i_2})$	(17)	$(\neg s^a_{k\,i_1} \vee \neg s^a_{k\,i_2}) \wedge (\neg s^a_{k\,i_1} \vee \neg s^b_{k\,i_2}) \wedge$ $(\neg s^b_{k\,i_1} \vee \neg s^a_{k\,i_2}) \wedge (\neg s^b_{k\,i_1} \vee \neg s^b_{k\,i_2})$	$1 \leq k \leq r$ $1 \leq i_1 \leq n$ $1 \leq i_2 \leq n$

4.4. *Using Lower Bounds*

This section presents two approaches for computing lower bounds. The first one was used in earlier published results on SHIPs.[23,24] The second lower bound improves on this earlier approach for computing lower bounds, and is used in the results presented in this paper.

Observe that the HIPP problem is APX-hard.[21] As a result, the computed lower bounds are not guaranteed to be tight. Whereas for ILP-based approaches the usefulness of non-tight lower bounds is not clear, the iterative SHIPs algorithm is likely to gain from using non-trivial lower bounds, even if the lower bounds are not tight. First, the number of iterations of the SHIPs algorithm is reduced. Second, and more importantly, tighter lower bounds allow simplifying the generated instances of SAT.

An existing alternative lower bound approach is based on computing the rank of a matrix representing the genotypes.[18,3] This approach provides only a numeric lower bound, and so it is not clear whether it can be also used for simplifying the SAT model. In addition, our experiments indicate that the lower bounds proposed

in this section are competitive, and most often of better quality, than the rank-based lower bound. Observe that the rank R of the genotype matrix is *at most* R_m, the minimum between the number of genotypes and the number of sites, and most often this value is guaranteed not to be reached. In contrast, the lower bounds described in this section are larger than R_m for a large number of examples considered in Section 5.

4.4.1. *A Clique-Based Lower Bound*

As mentioned earlier, trivial lower and upper bounds on the number of haplotypes are 1 and $2\,n$, respectively. This section describes an approach for computing lower bounds on the number of haplotypes. Lower bounds allow reducing the number of iterations of the top-level SHIPs algorithm, but also allow reducing the number of variables and constraints in the model.

The techniques for computing lower bounds rely on information regarding incompatible genotypes (Definition 4.1 in the previous section). The approach proposed uses a maximal clique for computing a lower bound on the number of required haplotypes. Clearly, for two incompatible genotypes, g_i and g_l, the haplotypes that explain g_i *must* be distinct from the haplotypes that explain g_l. Given the incompatibility relation we can create an *incompatibility* graph I, where each vertex is a genotype, and two vertexes have an edge if they are incompatible. Suppose I has a clique of size k. Then the number of required haplotypes is at least $2 \cdot k - \sigma$, where σ is the number of genotypes in the clique which do not have heterozygous sites.

Consider the following set of genotypes:

$$\begin{array}{c} 0102 \\ 1021 \\ 2210 \\ 1101 \end{array} \tag{18}$$

These genotypes are incompatible among each other. Hence, in the incompatibility graph, the genotypes form a clique with 4 vertexes. The obtained lower bound is $2 \cdot 4 - 1 = 7$, because the last genotype is homozygous, and so can be explained with a single haplotype. Hence, a lower bound on the number of haplotypes for this example is 7.

In order to maximize the computed lower bound, the objective is to identify the maximum clique in I. Since this problem is NP-hard,[10] we use the size of a maximal clique in the incompatibility graph, computed using a simple greedy heuristic. The genotype with the highest number of incompatible genotypes is first selected. At each step, the genotype selected is one that is still incompatible with all the already selected genotypes, and preference is given to the haplotype with the (statically computed) highest number of incompatible genotypes.

Moreover, we note that the information regarding the lower bound can be used for *reducing* the size of the model, and so it can also potentially reduce the search

space. If a genotype g_i is part of the clique and has at least one heterozygous site, then we can associate two *dedicated* haplotypes with g_i. If a genotype g_i is part of the clique and all its sites are homozygous, then we associate only one *dedicated* haplotype with g_i. In addition, when considering the candidate haplotypes for a genotype g_l, which is incompatible with genotype g_i included in the clique, the haplotypes associated with g_i *need not* be considered as candidates for g_l. This eliminates s variables and the corresponding clauses.

Furthermore, it is possible to increase the lower bound obtained with a maximal clique. Suppose a genotype g_i is heterozygous at site j, and further assume that all other genotypes assume the same homozygous value (either 0 or 1) at site j. Then, it is straightforward to conclude that explaining genotype g_i requires one haplotype which cannot be used to explain *any* of the other genotypes. Hence, g_i can be used to increase the lower bound by 1.

4.4.2. *An Improved Lower Bound*

This section describes an alternative approach for computing lower bounds for SHIPs. Similarly to the procedure outlined in the previous section, a maximal clique is computed. In addition, analysis of the structure of the genotypes allows the lower bound to be further increased. The objective of the new procedure is to identify heterozygous sites which require at least one additional haplotype given a set of previously chosen genotypes. The procedure starts from the clique-based lower bound (see previous section) and grows the lower bound by searching for these heterozygous sites among genotypes not yet considered for lower bounding purposes. Since the algorithm starts from the clique-based lower bound, it is guaranteed never to be less than the bound obtained from the computed clique.

Algorithm 2 summarizes the alternative lower bound procedure. The procedure MergeGenotypes 3creates a new genotype from a set of genotypes such that any heterozygous site or a site with genotypes having both 0 and 1 becomes heterozygous. If all genotypes have the same homozygous site, then the merged site keeps the same value. For each genotype g not in the clique, if the genotype has a heterozygous site and all compatible genotypes have the same value at that site (either 0 or 1), then g is guaranteed to require one additional haplotype to be explained. Hence the lower bound can be increased by 1.

The proposed lower bound procedure runs in polynomial time. A straightforward analysis yields a run time complexity in $\mathcal{O}(n^2\, m)$, by observing that each call to the MergeGenotypes function can involve at most $\mathcal{O}(n)$ genotypes and each pairwise merge runs in time $\mathcal{O}(m)$. Finally, observe that the asymptotic time complexity of the alternative lower bound procedure is the same as the asymptotic time complexity for generating the SHIPs model. In practice, the computational overhead of computing the lower bound is negligible.

Algorithm 2 Improved the clique-based LB

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

Algorithm 3 Merging genotypes

```
MergeGenotypes(S, g)
1  ng ← g
2  for each s ∈ S
3        do for j = 1 to m
4              do if (s[j] ≠ ng[j])
5                    then ng[j] ← 2
6  return ng
```

4.5. *Using Lower Bounds*

As mentioned above, lower bounds play a dual and key role in SHIPs. First, by using lower bounds the number of iterations of the SHIPs algorithm is reduced. Second, and more importantly, tighter lower bounds allow simplification of the generated SAT instances. As shown earlier, the SHIPs lower bound procedures described in this section associate one or two haplotypes with specific genotypes. As a result, the resulting SAT model is simplified; genotypes with associated haplotypes need not select from a set of candidate haplotypes. Moreover, when identifying the candidate haplotypes for a given genotype g having no associated haplotype, it is only necessary to consider haplotypes not associated with a specific genotype or haplotypes associated with genotypes that are compatible with g. The simplification of the SAT

model given lower bound information is in practice very beneficial for the efficiency of the SHIPs algorithm.

5. Experimental Results

The model described in the previous section, referred to as SHIPs (Sat-based Haplotype Inference by Pure Parsimony), has been implemented as a Perl script, corresponding to Algorithm 1. The algorithm iteratively generates CNF formulas which are then handed to a SAT solver. Currently, MiniSAT v1.14 [7] or SATZ v215.2 [22] can be used.

This section presents results comparing SHIPs with all existing exact solutions to the HIPP problem, namely the ILP-based approaches (RTIP, PolyIP and HybridIP), and a branch and bound solution (Hapar). The next section outlines the experimental setup and the following sections provide results for different classes of instances, using MiniSAT.[7] The last section evaluates the relevancy of the actual SAT solver used.

5.1. *Experimental Setup*

For evaluating SHIPs, as well as other tools, we have collected a comprehensive number of problem instances. Instances for the haplotype inference problem may be obtained following two different approaches:

- **Generate problem instances:** Synthetic problem instances are typically generated using Hudson's program `ms`.[17] This program generates *haplotypes* following a standard coalescent approach. Given the haplotypes, the genotypes are generated by pairing haplotypes either *uniformly* (repeated haplotypes are removed) or *non-uniformly* (repeated haplotypes are not removed and so have a higher probability of being paired). Moreover, we obtained instances that were generated using a coalescent model (loosely based on Hudson's program) that additionally incorporates variation in recombination rates and demographic events.[28] The parameters of the model were chosen to match aspects of data from a sample of white Americans. These problem instances were recently used to evaluate some *phasing* algorithms.[25]
- **Obtain real problem instances:** The HapMap project (`www.hapmap.org`) [31,32] provides a comprehensive source of *genotype* data over four populations. In addition, haplotypes for small genomic regions have been identified and are available from scientific publications.[19,27,6,4,20]

Accordingly, the instances used for evaluating the different algorithms are organized into four classes shown in Table 5.1. For each class we give the number of instances, and the minimum and maximum number of SNPs and genotypes, respectively. The *ms* class of instances contains the uniform and non-uniform instances used by Brown [2] but extended with additional, more complex

Table 4. Classes of instances used: number of SNPs and genotypes.

Class	# Instances	*min*SNPs	*max*SNPs	*min*GENs	*max*GENs
ms	380	10	100	30	100
phasing	329	14	194	90	90
hapmap	24	30	75	7	68
biological	450	13	103	5	50
Total	1183	10	194	5	100

Table 5. Classes of instances used: number of SNPs and genotypes after simplifications.

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

problem instances. The *phasing* class of instances is described by Schaffner [28] and corresponds to the SU-100kb, SU1, SU2 and SU3 classes available from `http://www.stats.ox.ac.uk/~marchini/phaseoff.html`. The *hapmap* class of instances is the one used by Brown,[2] which was extracted from the HapMap project. Finally, the instances for the *biological* class are generated from data available.[19,27,6,4,20]

All these problem instances were simplified according to the techniques described in 3.4, i.e. duplicated genotypes as well as duplicated and complemented SNPs were removed. The simplification time is negligible (always less than 1 second even for large instances) and therefore will not be taken into account. The size of the resulting instances is summarized in Table 5.1. Again, the number of instances and the minimum and maximum number of SNPs and genotypes is given for each class of problem instances. Clearly, most of the instances are significantly reduced due to these simplifications. For example, problem instances from the *ms* class have been reduced from $\geq$10 to 4 SNPs and from $\geq$30 to 9 genotypes.

More instances could have been generated and used. However, as the results below clearly demonstrate, besides SHIPs, existing approaches are unable to solve the vast majority of problem instances in reasonable time. Hence, we believe that the problem instances considered suffice to provide a comprehensive comparison of SHIPs with existing solutions to the HIPP problem.

All results shown were obtained on a 3.0 GHz Intel Xeon 5160 with 4GB of RAM running RedHat Linux. For the ILP-based HIPP solvers, the ILP package used was CPLEX version 10.2. Given the large number of problem instances and HIPP solvers considered, and taking into consideration that all approaches except SHIPs abort many problem instances, a timeout of 1000 seconds was used.

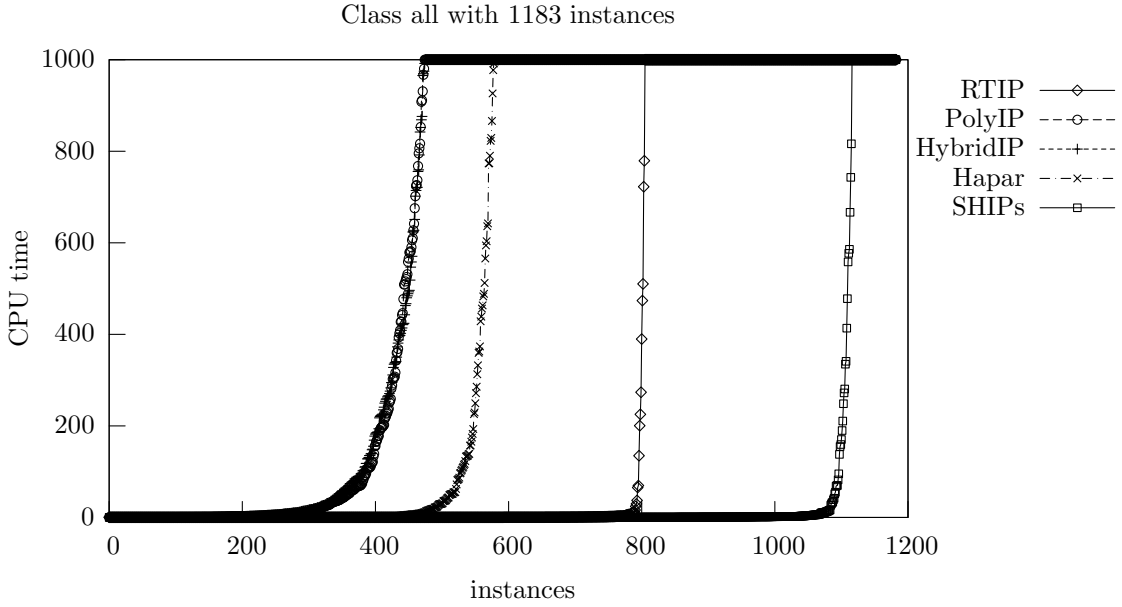

Fig. 2. HIPP solvers on all instances

5.2. *Results on All Instances*

The results on all instances are shown in Figure 2. For each solver, the plot gives the run times for solving each instance. The run times are sorted for each solver. We should note that many instances are solved by all solvers within a few seconds. Indeed, 202 problem instances are solved by any of the HIPP solvers in less than 10 seconds.

As can be concluded, SHIPs is the HIPP tool capable of solving the largest number of problem instances. SHIPs aborts 67 problem instances out of 1183 instances, whereas RTIP aborts 378 instances, Hapar aborts 603 instances, HybridIP aborts 708 instances and PolyIP aborts 709 instances. SHIPs takes on average 7.87 seconds for solving the non-aborted instances. RTIP takes 5.53 seconds, Hapar takes 39.9 seconds, PolyIP takes 82.56 seconds and HybridIP takes 86.66 seconds. The surprising small time required for RTIP is explained due to the fact that usually RTIP either solves a problem instance in a small amount of time or aborts due to memory exhaustion: 96% of the problem instances aborted by RTIP were aborted due to memory exhaustion and on average an instance is aborted due to memory exhaustion in 37.54 seconds. Clearly, RTIP may be competitive for solving some problem instances but it is not a robust solver.

Two scatter plots comparing the run times of RTIP and Hapar with SHIPs on all instances are shown in Figure 3. A log scale is used for both axis. Each point corresponds to one problem instance, where the x-axis represents the run

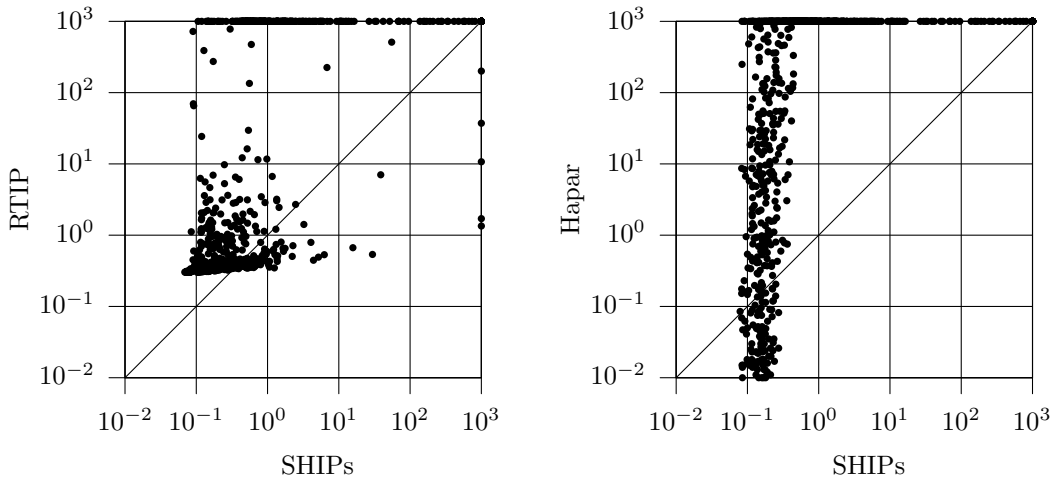

Fig. 3. Run times of SHIPs vs. Hapar and RTIP on all instances

time required by SHIPs and the y-axis represents the run time required by RTIP or Hapar. Points in the top left half part of the plot are instances where SHIPs performed better than the other solvers.

As can be concluded, with the exception of a few outliers, the performance of SHIPs is significantly better than either RTIP or Hapar. Hapar is only more effective for instances where both SHIPs and Hapar take less than 1 second. Also, and with the exception of 8 instances in total, RTIP is only more effective for instances where both SHIPs and RTIP take less than 1 second.

5.3. *MS Instances*

The results for the instances of class ms are shown in Figure 4. SHIPs has the best performance: SHIPs solves all instances, and only requires more than 2 seconds for two instances (which require 2.27 and 4.41 seconds). Besides SHIPs, only RTIP can solve all the problem instances. Indeed, almost every instance in the ms class is also solved by RTIP in less than 2 seconds, except for one instances which is solved in 9.74 seconds. Hapar is the HIPP solver having the 3^{rd} best performance, but it aborts 98 out of 380 problem instances. For the other ILP-based solvers the results are significantly worse: PolyIP and HybridIP perform much worse than RTIP, respectively aborting 250 and 254 instances out of 380. (It is interesting to observe that, for all classes of instances, the results for PolyIP and HybridIP are in general very similar.) From these results it is clear that the best performing HIPP solvers are SHIPs and RTIP, followed by Hapar. Hence we compare the results obtained for each problem instance using SHIPs and using RTIP or Hapar.

Two scatter plots comparing the run times of RTIP and Hapar with SHIPs are shown in Figure 5.

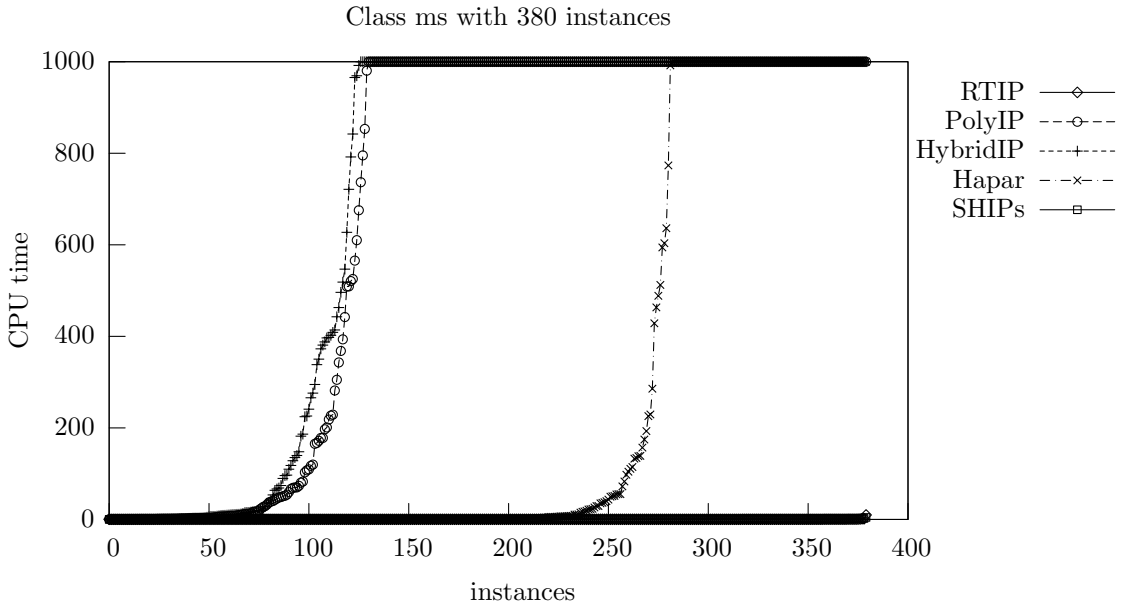

Fig. 4. HIPP solvers on ms instances

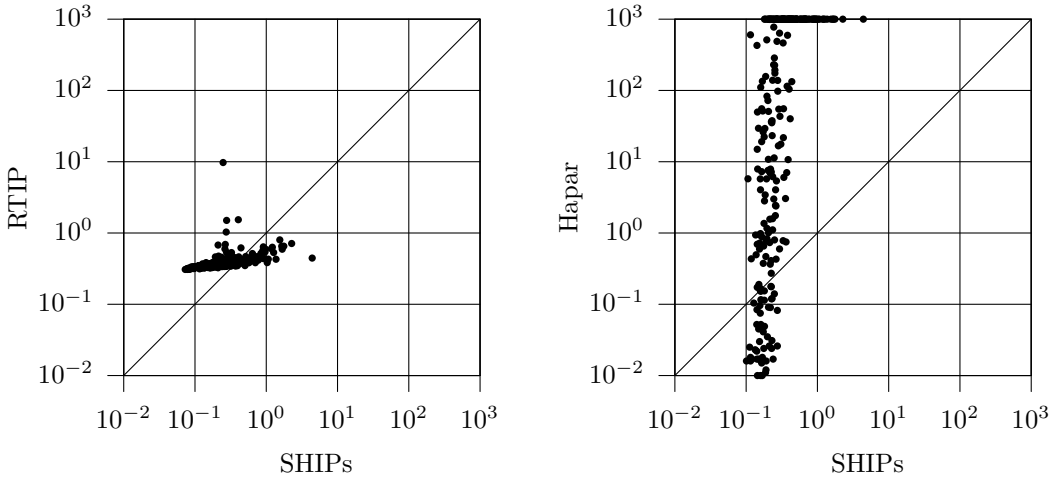

Fig. 5. Run times of SHIPs vs. RTIP and Hapar on ms instances

As can be observed, SHIPs is always faster than RTIP on very easy instances, for which SHIPs requires less than 1 second. This is probably related with the setup time required by CPLEX. Overall, SHIPs is faster than RTIP in 321 out of 380 instances.

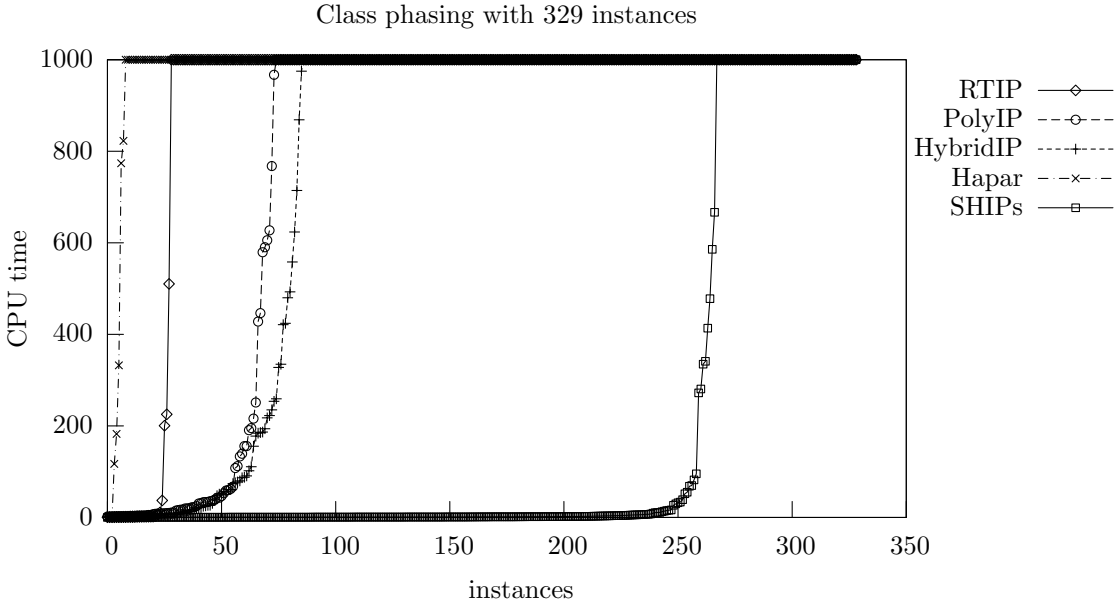

Fig. 6. HIPP solvers on phasing instances

Hapar is in general more efficient for easy instances, i.e. instances for which both Hapar and SHIPs require less than 1 second. However, there is a large number of instances solved by SHIPs which Hapar aborts. For each of the 98 instances for which Hapar aborts after 1000 seconds, SHIPs takes less than 5 seconds.

5.4. *Phasing Instances*

The results for the instances of class phasing are shown in Figure 6. For this class of instances, the results are clear: SHIPs is significantly more efficient than the other approaches. SHIPs aborts 62 instances out of 329, HybridIP 243, PolyIP 255, RTIP 301 and Hapar 321. SHIPs is able to solve 240 instances in less than 10 seconds and 244 instances in less than 100 seconds. Observe that contrary to what happens for all other classes of instances, RTIP is less competitive than the other IP approaches. This can be explained by the memory requirements for this class of instances: these instances are the ones with the highest average number of SNPs and genotypes. Indeed, these are challenging problem instances, which no other HIPP approach can efficiently solve, and which motivate further research work.

Two scatter plots comparing the run times of HybridIP and RTIP with SHIPs are shown in Figure 7. Although PolyIP is the 3^{rd} most competitive approach, after SHIPs and HybridIP, its performance is quite similar to the HybridIP performance. For this reason we compare SHIPs with RTIP instead. These two plots confirm what we have mentioned before: SHIPs not only solves the largest number of problem

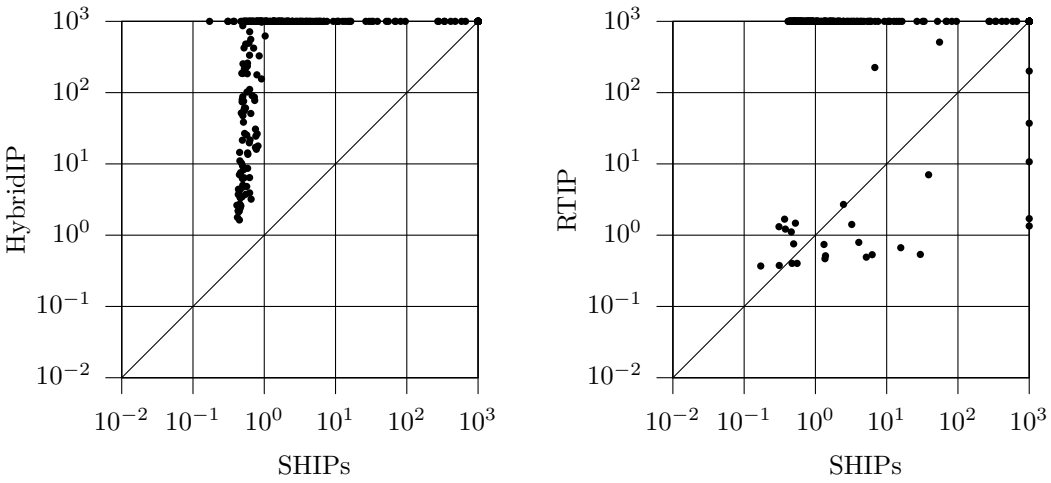

Fig. 7. Run times of SHIPs vs. HybridIP and RTIP on phasing instances

instances but also is in general significantly faster than the other approaches. The only exception if for some instances when compared with RTIP: RTIP requires less time than SHIPs for solving 17 problem instances, of which 5 instances are aborted by SHIPs. The existence of these 5 outliers is interesting, albeit not significant.

5.5. *Hapmap Instances*

The results for the instances of class hapmap are shown in Figure 8. As with the previous classes of instances, the performance difference between SHIPs and the other HIPP solvers is again clear. With the exception of instance *test_chr21_YRI_75*, SHIPs is the only HIPP solver capable of solving every instance in less than 1 second. For instance *test_chr21_YRI_75*, SHIPs takes 69.92 seconds. RTIP is unable to solve 4 out of 24 instances in less than 1000 seconds, Hapar 6, PolyIP 8 and Hybrid 8.

Two scatter plots relating the run times of RTIP and Hapar with SHIPs are shown in Figure 9. As can be concluded, SHIPs is always faster than RTIP on solving the hapmap instances. Also, similarly to the previous classes of instances, Hapar is in general more efficient for easy instances, for which both Hapar and SHIPs require less than 1 second. However, for the harder instances Hapar takes orders of magnitude more time than SHIPs.

5.6. *Biological Instances*

The results for the instances of class biological are shown in Figure 10. As with all previous classes of instances, the performance difference between SHIPs and the other HIPP solvers is clear. From the 450 problem instances, SHIPs is able to solve 407 instances in less than 1 second and 427 instances in less than 10 seconds. SHIPs is unable to solve 5 out of 450 instances in less than 1000 seconds, RTIP 73, Hapar

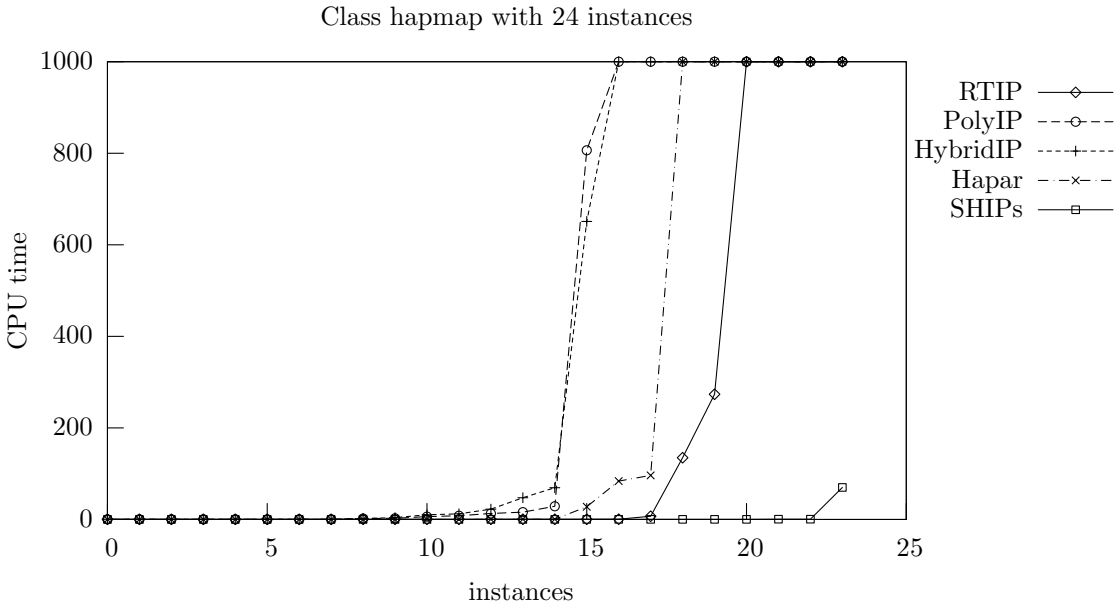

Fig. 8. HIPP solvers on hapmap instances

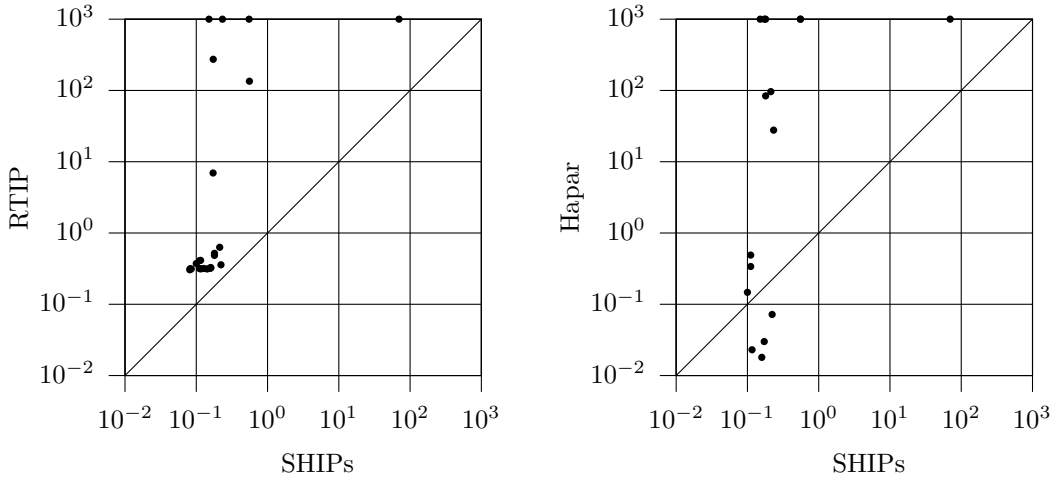

Fig. 9. Run times of SHIPs vs. RTIP and Hapar on hapmap instances

178, PolyIP 196 and HybridIP 203. RTIP performs significantly better than the other approaches probably because the number of candidate haplotype pairs is in general small. For this class of problem instances, RTIP aborts 16% of the problem instances, Hapar 40%, PolyIP 44% and HybridIP 45%.

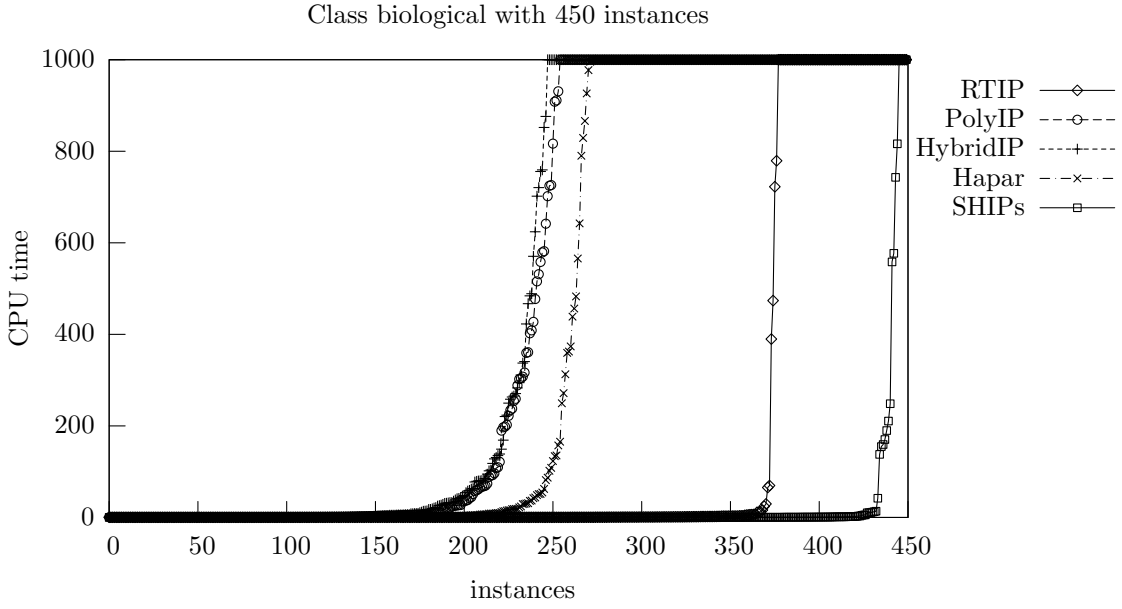

Fig. 10. HIPP solvers on biological instances

Two scatter plots relating the run times of RTIP and Hapar with SHIPs are shown in Figure 11. As can be concluded, SHIPs is almost always faster than RTIP. Also, SHIPs is in most of the cases orders of magnitude faster than RTIP. Similarly to the previous classes of instances, Hapar is in general more efficient for easy instances, i.e. instances for which both Hapar and SHIPs require less than 1 second. However, for all instances for which both Hapar and SHIPs require more than 1 second, Hapar can take orders of magnitude more time than SHIPs.

5.7. *Additional Remarks*

The results presented in the previous sections are consistent and conclusive: the utilization of SAT is the most effective exact approach for the HIPP problem. Indeed, for the large majority of instances, SHIPs is orders of magnitude more efficient than any other existing approach for the HIPP problem, and this is true across a large number of classes of instances. In addition, SHIPs is the only exact approach for the HIPP problem capable of solving a vast number of problem instances, which no other exact approach for the HIPP problem can solve [e]. More interestingly, SHIPs solves most of these instances in a few CPU seconds. Hence, the SHIPs approach allows considering using HIPP as an effective alternative to the more widely used

[e]This of course will depend on the allowed CPU time. In our case the experiments restricted the CPU times to 1000 seconds.

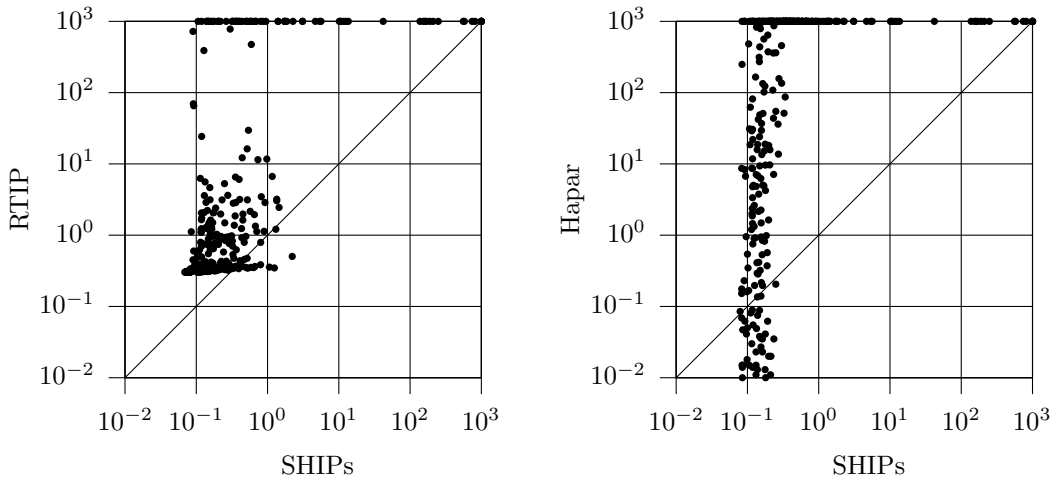

Fig. 11. Run times of SHIPs vs. RTIP and Hapar on biological instances

alternatives based on statistical methods.[30,26]

Despite the very promising results, SHIPs is unable to solve 67 problem instances in less than 1000 seconds, out of the 1183 problem instances considered. These challenging problem instances motivate further research work in SAT-based haplotype inference.

5.8. *SAT Solver Relevancy*

This section evaluates how sensitive the SHIPs model is to techniques in the actual SAT solver used. Two SAT solvers are compared: SATZ [22] and MiniSAT [7]. SATZ implements the basic DPLL algorithm enhanced with a powerful look-ahead heuristic. MiniSAT also implements DPLL but is enhanced with clause learning, lazy data structures, adaptive branching heuristics and search restarts. MiniSAT is expected to search more nodes per second, although learning useless clauses may bring a significant overhead. MiniSAT usually performs better on structured instances, such as industrial problem instances, whereas SATZ performs better on random or handmade instances. The set of instances considered is the same used in the previous sections.

The results are summarized in Figure 12, and are conclusive: with a few exceptions, for the easiest problem instances, for which the run times are similar, MiniSAT outperforms SATZ by a large margin. In addition, a scatter plot comparing SHIPs using MiniSAT and SHIPs using SATZ on all problem instances is shown in Figure 13. Clearly, MiniSAT is faster than SATZ on the vast majority of problem instances. For instances where MiniSAT requires more than 10 CPU seconds, the speedups of MiniSAT over SATZ exceed 2 orders magnitude. Hence, the results support the claim that a modern SAT solver is essential for the performance gains

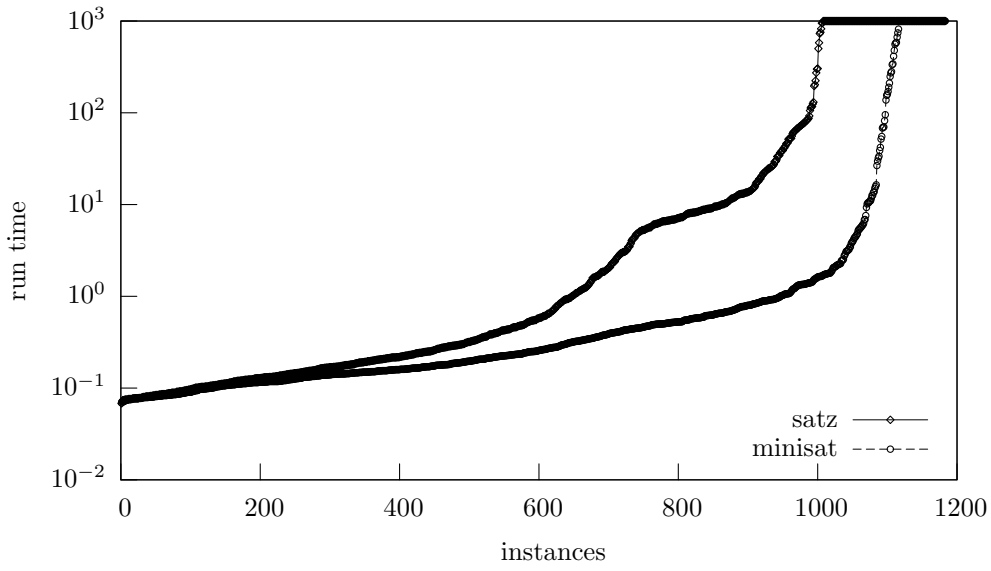

Fig. 12. SATZ vs MiniSAT on all instances

of SHIPs compared to all other exact HIPP solutions. Indeed, the results obtained suggest that the most effective component of SHIPs is the SAT solver used and the search techniques it integrates in order to focus the search on more structured parts of each problem instance, namely clause learning. We believe that haplotype inference problem instances are structured instances for which some parts of the problem are trivially solved whereas some other parts are quite hard to solve. This observation seems straightforward if we consider that for some genotypes it may be trivial to identify an explaining pair of haplotypes whereas from some other genotypes, for which explaining haplotypes may also explain many other genotypes, to identify such haplotypes may become a hard task.

Interestingly, though, even when SATZ is used instead of MiniSAT, SHIPs is still more competitive than all the other existing solvers. SHIPs with SATZ aborts 176 instances, whereas RTIP aborts 378 instances, followed by the other solvers.

6. Conclusions and Future Work

This paper describes SHIPs, a SAT-based approach for the problem of haplotype inference by pure parsimony (HIPP). The paper also provides experimental results demonstrating that SHIPs drastically outperforms all other existing exact approaches for the HIPP problem. Besides being much more efficient than alternative HIPP solvers, SHIPs is able to solve a large number of problem instances that no other HIPP solver is capable of. Moreover, SHIPs shares the accuracy of all HIPP solutions, and HIPP is competitive with the most widely used solutions.[12,16,34]

Despite the promising results, several challenges remain. As the results indicate,

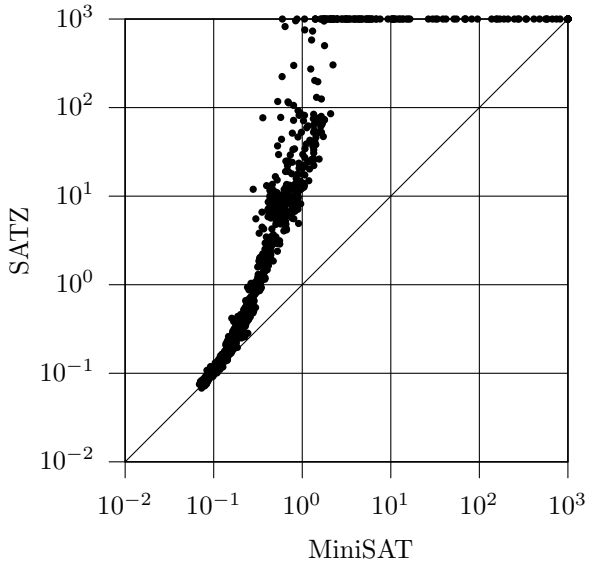

Fig. 13. SATZ vs MiniSAT on all instances

SHIPs is still unable to solve some instances from the phasing and the biological classes. This motivates additional optimizations to the SAT-based approach. Finally, the SAT-based approach allows considering criteria other than pure parsimony. For example, one may give preference to selecting haplotypes that explain the largest number of genotypes, even though the number of required haplotypes may not be minimum. The development of these alternative criteria and their practical evaluation is subject of future research work.

It should also be observed that the proposed SAT-based models also provide a new, essentially endless, source of challenging SAT instances, which can be used for driving the development of more optimized SAT solvers.

Acknowledgments

The authors thank A. Oliveira for having pointed out the haplotype inference by pure parsimony problem, and the authors of other tools [2,16] for providing assistance on the results reported in their papers.

This work is registered under Portuguese patent PT 103434, and is partially supported by Fundação para a Ciência e Tecnologia under research project POSC/EIA/61852/2004, by INESC-ID under research project SHIPs, and by Microsoft under contract 2007-017 of the Microsoft Research PhD Scholarship Programme.

References

1. 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)*, 2004.
2. 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.
3. D. Brown and I. Harrower. Toward and algebraic understanding of haplotype inference by pure parsimony. In *Computational Systems Bioinformatics Conference*, August 2006.
4. 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.
5. Z. Ding, V. Filkov, and D. Gusfield. A linear-time algorithm for the perfect phylogeny haplotyping (PPH) problem. In *International Conference on Research in Computational Molecular Biology (RECOMB)*, pages 585–600, May 2005.
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. P. Flener, A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Breaking row and column symmetries in matrix models. In *International Conference on Principles and Practice of Constraint Programming (CP)*, 2002.
9. A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, and T. Walsh. Global constraints for lexicographic orderings. In *International Conference on Principles and Practice of Constraint Programming (CP)*, 2002.
10. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
11. I. Gent. Arc consistency in SAT. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pages 121–125, July 2002.
12. D. Gusfield. Haplotype inference by pure parsimony. In *14th Annual Symposium on Combinatorial Pattern Matching (CPM'03)*, pages 144–155, 2003.
13. D. Gusfield and S. Orzach. *Handbook on Computational Molecular Biology*, volume 9 of *Chapman and Hall/CRC Computer and Information Science Series*, chapter Haplotype Inference. CRC Press, December 2005.
14. B. Halldórsson, V. Bafna, N. Edwards, R. Lippert, S. Yooseph, and S. Istrail. A survey of computational methods for determining haplotypes. In *Proceedings of the First RECOMB Satellite on Computational Methods for SNPs and Haplotype Inference*, volume 2983 of *LNBI*, pages 26–47, 2004.
15. E. Halperin and R. Karp. Perfect phylogeny and haplotype assignment. In *Annual International Conference on Computational Molecular Biology*, pages 10–19, March 2003.
16. Y.-T. Huang, K.-M. Chao, and T. Chen. An approximation algorithm for haplotype inference by maximum parsimony. *Journal of Computational Biology*, 12(10):1261–1274, December 2005.
17. R. R. Hudson. Generating samples under a wright-fisher neutral model of genetic variation. *Bioinformatics*, 18(2):337–338, February 2002.

18. K. Kalpakis and P. Namjoshi. Haplotype phasing using semidefinite programming. In *International Symposium on Bioinformatic and Bioengineering*, pages 145–152, October 2005.
19. 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.
20. 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.
21. 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.
22. 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.
23. I. Lynce and J. Marques-Silva. Efficient haplotype inference with Boolean satisfiability. In *National Conference on Artificial Intelligence (AAAI)*, July 2006.
24. 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)*, August 2006.
25. J. Marchini, D. Cutler, N. Patterson, M. Stephens, E. Eskin, E. Halperin, S. Lin, Z. Qin, H. Munro, G. Abecassis, P. Donnelly, and International HapMap Consortium. A comparison of phasing algorithms for trios and unrelated individuals. *American Journal of Human Genetics*, 78:437–450, 2006.
26. 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.
27. 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.
28. S. Schaffner, C. Foo, S. Gabriel, D. Reich, M. Daly, and D. Altshuler. Calibrating a coalescent simulation of human genome sequence variation. *Genome Reasearch*, 15:1576–1583, 2005.
29. C. Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In *Proc. of the 11th Intl. Conf. on Principles and Practice of Constraint Programming (CP 2005)*, pages 827–831, Sitges, Spain, October 2005.
30. M. Stephens, N. Smith, and P. Donelly. A new statistical method for haplotype reconstruction. *American Journal of Human Genetics*, 68:978–989, 2001.
31. The International HapMap Consortium. The international hapmap project. *Nature*, 426:789–796, 2003.
32. The International HapMap Consortium. A haplotype map of the human genome. *Nature*, 437:1299–1320, 27 October 2005.
33. T. Walsh. SAT *v* CSP. In *International Conference on Principles and Practice of Constraint Programming (CP)*, pages 441–456, September 2000.
34. L. Wang and Y. Xu. Haplotype inference by maximum parsimony. *Bioinformatics*, 19(14):1773–1780, 2003.