Boosting Haplotype Inference with Local Search

Inês Lynce[1], João Marques-Silva[2], and Steve Prestwich[3]

[1] IST/INESC-ID, Technical University of Lisbon, Portugal
ines@sat.inesc-id.pt
[2] School of Electronics and Computer Science, University of Southampton, UK
jpms@ecs.soton.ac.uk
[3] Cork Constraint Computation Centre, University College Cork, Ireland
s.prestwich@cs.ucc.ie

Abstract. A very challenging problem in the genetics domain is to infer haplotypes from genotypes. This process is expected to identify genes affecting health, disease and response to drugs. One of the approaches to haplotype inference aims to minimise the number of different haplotypes used, and is known as haplotype inference by pure parsimony (HIPP). The HIPP problem is computationally difficult, being NP-hard. Recently, a SAT-based method (SHIPs) has been proposed to solve the HIPP problem. This method iteratively considers an increasing number of haplotypes, starting from an initial lower bound. Hence, one important aspect of SHIPs is the lower bounding procedure, which reduces the number of iterations of the basic algorithm, and also indirectly simplifies the resulting SAT model. This paper describes the use of local search to improve existing lower bounding procedures. The new lower bounding procedure is guaranteed to be as tight as the existing procedures. In practice the new procedure is in most cases considerably tighter, allowing significant improvement of performance on challenging problem instances.

1 Introduction

A recent and challenging problem in Bioinformatics is the process of inferring haplotype data from genotype data. Among a number of potential applications, haplotype inference can serve to identify genes affecting health, disease and response to drugs. Several approaches have been proposed for the haplotype inference problem in recent years. One of these approaches aims to minimise the number of different haplotypes used, and is known as Haplotype Inference by Pure Parsimony (HIPP). The HIPP problem is computationally difficult, being NP-hard, and several methods have been proposed to solve it.

The first exact solutions to the HIPP problem were based on Integer Linear Programming. More recently, Boolean Satisfiability (SAT) has been proposed for solving the HIPP problem. The SAT-based approach, SHIPs, has been shown to be very effective at solving hard HIPP problem instances. The organisation of SHIPs consists in iteratively considering an increasing number of haplotypes, starting from an initial lower bound. Hence, one important aspect of SHIPs is the lower bounding procedure, which reduces the number of iterations of the basic algorithm, and also indirectly simplifies the resulting SAT model.

SHIPs has been shown to be the most effective approach for solving the HIPP problem, but there are challenging instances that it cannot yet solve in a reasonable time. An important technique for improving performance is the computation of lower bounds for the problem, and several powerful and efficient techniques have already been proposed. This paper describes the use of local search to improve existing lower bounding procedures. The new lower bounding procedure is guaranteed to be as tight as existing procedures, but in practice the new procedure is in most cases considerably tighter, allowing significant performance improvements on challenging problem instances.

The paper is organised as follows. Section 2 introduces haplotype inference. Section 3 summarises previous work on the SHIPs approach. Section 4 surveys existing lower bounding techniques. Section 5 describes the new SAT-based local search method for improving lower bounds. Section 6 evaluates the new method empirically. Section 7 concludes the paper.

2 Haplotype Inference

A haplotype is the genetic constitution of an individual chromosome. The genetic constitution of a chromosome is described by a DNA sequence. A DNA sequence is specified by four nucleotides, which can be distinguished by the bases they contain: A (adenine), C (cytosine), T (thymine), and G (guanine). The DNA double helix consists of two molecules held together by weak bonds between base pairs of nucleotides. Given that base pairs are formed only between A and T and between G and C, the base sequence of each single strand can be deduced from that of the other strand in the DNA.

The underlying data that forms a haplotype can be the full DNA sequence in the region, or more commonly the Single Nucleotide Polymorphism (SNPs) in that region. An SNP is a genetic mutation that can occur within a DNA sequence. SNPs are the most common type of genetic mutation. An SNP variation occurs when a single nucleotide replaces one of the other three nucleotides. An example of an SNP is the alteration of the DNA sequence TTACCGT to TCACCGT, where the second T is replaced with a C. Although many SNPs do not produce physical changes, it is believed that other SNPs may predispose people to disease and even influence their response to drugs. The HapMap project [33, 34] (`http://www.hapmap.org`) represents an effort to identify such SNPs.

Figure 1 illustrates DNA sequences from five different individuals. Given these DNA samples, three SNPs are identified. SNPs correspond to the sites for which there are two different nucleotides amongst the different individuals. *SNPa* may contain base A or base G whereas *SNPb* may contain bases T/C and *SNPc* may contain bases T/A. The haplotypes are obtained from the SNPs in the DNA sequences.

The genetic code of different people is extraordinarily similar. Only about 0.1% of an individual DNA differs from that of any other individual. According

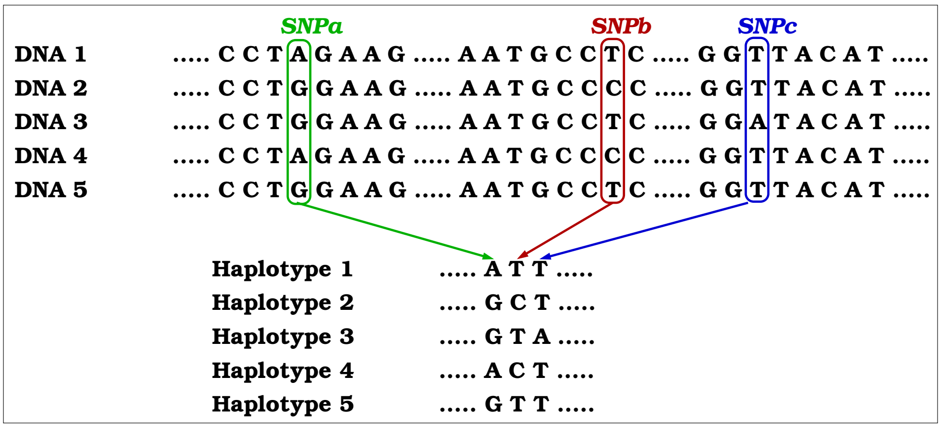
SNPa
SNPb
SNPc
DNA 1 ..... C C T A G A A G ..... A A T G C C T C ..... G G T T A C A T .....
DNA 2 ..... C C T G G A A G ..... A A T G C C C C ..... G G T T A C A T .....
DNA 3 ..... C C T G G A A G ..... A A T G C C T C ..... G G A T A C A T .....
DNA 4 ..... C C T A G A A G ..... A A T G C C C C ..... G G T T A C A T .....
DNA 5 ..... C C T G G A A G ..... A A T G C C T C ..... G G T T A C A T .....
Haplotype 1 ..... A T T .....
Haplotype 2 ..... G C T .....
Haplotype 3 ..... G T A .....
Haplotype 4 ..... A C T .....
Haplotype 5 ..... G T T .....

Fig. 1. Identifying SNPs in DNA sequences

to the information available from the HapMap project, the DNA sequences in the chromosomes of two individuals can be identical for hundreds of bases. But approximately once in every 1,200 bases, the sequences will exhibit an SNP. Overall, there are ten million estimated SNPs to occur in the human genome.

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 it is said to be *heterozygous*. The haplotype inference problem consists in inferring haplotypes from genotypes.

The identification of haplotypes may be useful for identifying specific diseases, as well as to predict patients response to drugs. Although some heritable disorders may depend on the variation of one single location in one single gene, common diseases usually depend on more than one variation. The study of the effects of particular variations is simplified by the fact that, in many cases, there exists a strong correlation between the allele present in a particular SNP and other nearby sites. Hence, identifying common haplotypes in the population represents a key first step towards the understanding of the pathogenesis of disease. However, current genotyping methods do not provide haplotype information, which is essential for detailed analysis of the mechanisms of disease. The same genotype may be related with different pairs of haplotypes, where some of the pairs may cause a specific disease, whereas the other pairs may not.

Definition 1 (Haplotype Inference). *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, m denotes the number of SNP sites, and g_i denotes a specific genotype where $1 \leq i \leq n$. (Furthermore g_{ij} denotes a specific site j in genotype g_i where $1 \leq j \leq m$.)*

Without loss of generality, we may assume that the values of the two possible values 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 to represent 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.

Example 1. The genotype 1201 *must* be explained by the pair of haplotypes 1001/1101. The genotype 2120 *may* be explained either by the pair 1110/0100 or by the pair 1100/0110.

2.1 Haplotype Inference by Pure Parsimony

The problem of haplotype inference is an active area of research. The most widespread approaches are based on statistical methods [32, 26]. An alternative is haplotype inference by pure parsimony [12].

Definition 2 (Haplotype Inference by Pure Parsimony (HIPP)). *Given a set of genotypes, the HIPP approach finds a solution to the haplotype inference problem such that the total number of distinct haplotypes used is minimum.*

The HIPP problem is NP-hard (see [11, 19] for proofs and historical perspective). Hence, this problem is also interesting from a computational point of view. Moreover, experimental results provide support for this method [35]: the number of haplotypes in a large population is typically very small, though genotypes exhibit a great diversity [27].

Example 2. 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.

A number of solutions exist for the HIPP problem, the majority of which are based on Integer Linear Programming (ILP) [11, 13, 1, 2] and which are reviewed below. A more recent approach proposes a SAT-based model [20, 21] and is described in the next section. Other approaches include branch-and-bound [35], heuristic approximation algorithms [17] and relaxations of a semidefinite programming model [18]. Moreover, there has been work on solving restricted cases of haplotype inference [14, 6] some of which are based on 2SAT [14].

The original ILP model, *TIP*, has linear space complexity on the number of possible haplotypes [11, 12], and so it is exponential on the number of given

Algorithm 1 Top-level SHIPs algorithm

```
SHIPs($\mathcal{G}$, $lb$)
$\mathcal{G} \leftarrow$ ApplySimplifications($\mathcal{G}$)
$r \leftarrow lb$
while (true)
    do Generate $\varphi^r$ given $\mathcal{G}$ and $r$
       if SAT($\varphi^r$) $= true$
          then return $r$
          else $r \leftarrow r + 1$
```

genotypes. 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. Each candidate haplotype is associated with a dedicated variable x_v, such that $x_v = 1$ if the haplotype is used. The cost function is the number of haplotypes used: minimise $\sum_v x_v$.

The *RTIP* model [11, 12] introduces one essential simplification. If the pair of haplotypes for a variable $y_{i,u}$ are such that they are not part of any other pair of haplotypes, then the $y_{i,u}$ variable and related x_v variables can be removed from the formulation.

A more recent ILP model, *PolyIP*, is polynomial in the number of sites m and population size n [13, 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 minimised. More recently, Brown and Harrower introduced a new polynomial-size formulation, *HybridIP*, representing a hybrid of the RTIP and PolyIP formulations [2].

3 SAT-based Haplotype Inference by Pure Parsimony

SHIPs is a SAT-based algorithm to solve the haplotype inference by pure parsimony problem [20, 21]. This section presents an overview of SHIPs; a more complete description of SHIPs can be found in the original publications.

3.1 SHIPs basic algorithm

Algorithm 1 summarises 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 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 satisfy $lb \leq r \leq 2\,n$. 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)) [4].

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, such that $1 \leq k \leq r$. As a result, $h_{kj} \in \{0, 1\}$ denotes the j^{th} site of haplotype k.

For a given value of r, the model considers r haplotypes and seeks to associate two haplotypes (possibly corresponding to the same haplotype) with each genotype g_i, where $1 \leq i \leq n$. 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 $k = 1, \dots, r$. 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 $s^b_{k_1 i} = 1$.

We can now derive the conditions for the SHIPs model:

- If a site g_{ij} is 0 then the model requires $(\neg s^a_{ki} \vee \neg h_{kj}) \wedge (\neg s^b_{ki} \vee \neg h_{kj})$ for $k = 1, \dots, r$. 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 $(\neg s^a_{ki} \vee h_{kj}) \wedge (\neg s^b_{ki} \vee h_{kj})$ for $k = 1, \dots, r$. 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 a 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 assumes the complementary value of t_{ij}. As a result the model requires $(h_{kj} \vee \neg t_{ij} \vee \neg s^a_{ki}) \wedge (\neg h_{kj} \vee t_{ij} \vee \neg s^a_{ki}) \wedge (h_{kj} \vee t_{ij} \vee \neg s^b_{ki}) \wedge (\neg h_{kj} \vee \neg t_{ij} \vee \neg s^b_{ki})$ for $k = 1, \dots, r$. 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:

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

[4] SHIPs implementation uses the MiniSat SAT solver (version 1.14) [7]. MiniSat is a complete solver implementing backtrack search enhanced with clause learning, lazy data structures and dynamic heuristics. It has been chosen among other complete solvers for being one of the most competitive solvers.

Since the proposed model is purely SAT-based, a simple alternative solution is used, which consists of the CNF representation of a simplified adder circuit and requiring the output of the adder to be equal to 1. The CNF representation of an adder circuit requires the utilization of additional variables v^a_{ki} and v^b_{ki}. 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:

$$\begin{array}{l} v^a_{1i} \leftrightarrow s^a_{1i} \\ (\neg v^a_{ki} \vee \neg s^a_{ki}) \\ v^a_{k+1\,i} = (s^a_{ki} \vee v^a_{ki}) \\ v^a_{ri} = 1 \end{array}$$

3.2 SHIPs improved algorithm

The core SHIPs model described above 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 range from the computation of lower bounds (described in the next section) to symmetry breaking techniques.

Breaking symmetries allows to further prune the search space. We may observe the existence of symmetries in the problem formulation on the h and on the s variables. For example, given a genotype 2101, it may either be explained by haplotypes 1101 and 0101 or by haplotypes 0101 and 1101. Also, each genotype may be explained by different arrangements of the s variables.

These symmetries are broken by ensuring ordering relations among the h and the s variables, similarly to what is done for matrix models [9]. Observe that 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 [22]. If matrix H is interpreted as a vector of strings of size m, $H = [h_1 h_2 \ldots h_r]^T$, then we can impose the condition $h_1 < h_2 < \ldots < h_r$, i.e. the haplotypes are lexicographically sorted. In addition, 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 Computing Lower Bounds

This section presents three approaches for computing lower bounds. The first one was proposed by Kalpakis and Namjoshi [18] and further studied by Brown and Harrower [3]. The two other approaches have been developed within SHIPs. One of the SHIPs lower bound approaches was used in earlier results on SHIPs [20, 21], and the other one improves on the original SHIPs lower bound approach [23].

4.1 Rank-based lower bound

The rank-based lower bound procedure considers a modified matrix representation of the set of genotypes. The rank of the modified matrix of genotypes is a lower bound on the number of haplotypes necessary for explaining the set of genotypes [18]. Instead of the problem formulation described earlier, the rank-based lower bound computation requires a different formulation for the values at each site. A homozygous site with value 0 remains unchanged. A homozygous site with value 1 in our formulation is now represented with value 2. Finally, heterozygous sites are represented with 1 instead of 2. This modification is required for the matrix rank to represent a lower bound on the number of haplotypes [18]. In the modified formulation, $\mathit{\Delta}$ denotes the matrix of genotypes, and we have the following result [3, 18]:

Proposition 1. *The rank of the genotype matrix is a lower bound on the number of haplotypes required to explain the set of genotypes $\mathit{\Delta}$.*

The rank-based lower bound provides a numerical value, which can be used for reducing the number of iterations of the top-level SHIPs algorithms. As we show next, the lower bound procedures associated with SHIPs also allow the inference of additional information that can be used to simplify the resulting SAT instances.

4.2 SHIPs lower bound

The basic SHIPs lower bound is based on the notion of incompatible genotypes.

Definition 3 (Incompatible Genotypes). *Two genotypes, g_i and g_l, are* incompatible *if and only if 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 by common haplotypes.

The SHIPs lower bound identifies a maximal clique of incompatible genotypes. 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 vertices 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.

Example 3. For example consider the following set of genotypes:

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

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

In order to maximise the computed lower bound, the objective is to identify the maximum clique in I. Since this problem is NP-hard [10], we use a simple greedy heuristic for estimating a maximal clique in the incompatibility graph. 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, it is possible to increase the lower bound that was 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.

4.3 Improved SHIPs lower bound

This section describes an alternative approach for computing lower bounds for SHIPs [23]. 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 increases the lower bound by searching for 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 summarises the alternative lower bound procedure. The procedure MergeGenotypes (shown in Algorithm 3) creates a new genotype from a set of genotypes such that the new genotype is heterozygous for any site where one of the genotypes is heterozygous or there exist different genotypes with different values. If all genotypes have the same homozygous site then the merged site retains 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. Several procedures can be devised for selecting a genotype at each step. The best results were obtained by selecting genotypes by increasing number of heterozygous sites, as shown in Algorithm 2.

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

Algorithm 2 Improved the clique-based LB

```
ImproveLB($G, G_C, lb$)
▷ $G$ is the set of genotypes
▷ $G_C$ is a clique of mutually incompatible genotypes
▷ $lb$ is the lower bound obtained from $G_C$
Sort $G_{NC}$ by increasing number of heterozygous sites          ▷ Optional step
$G_{NC} \leftarrow G - G_C$                               ▷ $G_{NC}$: set of non-clique genotypes
$G_S \leftarrow G_C$                              ▷ Working set of genotypes starts with $G_C$
for each $g \in G_{NC}$                          ▷ Analyze genotypes in (sorted) order
    do $S \leftarrow \{cg \in G_S : \text{Compatible}(g, cg)\}$
       if $(\exists_{1 \leq j \leq m} \text{Heterozygous}(g[j]) \wedge \exists_{v \in \{0,1\}} \forall_{s \in S}\, s[j] = v)$
          then
               $lb \leftarrow lb + 1$
               ▷ Set sites with differing values to 2 and update $G_S$
               $ng \leftarrow \text{MergeGenotypes}(S, g)$
               $G_S \leftarrow (G_S - S) \cup \{ng\}$
return $lb$
```

Algorithm 3 Merging genotypes

```
MergeGenotypes($S, g$)
$ng \leftarrow g$
for each $s \in S$
    do for $j = 1$ to $m$
           do if $(s[j] \neq ng[j])$
                 then $ng[j] \leftarrow 2$
return $ng$
```

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.

4.4 Integrating lower bounds

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.

Moreover, observe that the rank-based lower bound [18, 3] 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. Also, 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 for most matrices have a rank below R_m. In contrast, the lower bounds described in this section are larger than R_m for a large number of examples considered in Section 6.

Example 4. The different lower bounds are illustrated with the following example. Consider the set of genotypes:

$$G = \{g_1, g_2, g_3, g_4, g_5, g_6, g_7\}$$

where,

$$\begin{aligned} g_1 &= 0102 \\ g_2 &= 1021 \\ g_3 &= 2210 \\ g_4 &= 1101 \\ g_5 &= 1202 \\ g_6 &= 2021 \\ g_7 &= 2102 \end{aligned}$$

For this example, the rank lower bound is 4. As described in Example 3, the original SHIPs lower bound [20, 21] is 7, due to the clique composed of $\{g_1, g_2, g_3, g_4\}$. The contribution of each of the genotypes g_1, g_2 and g_3 is 2, because these genotypes have heterozygous sites, and the contribution of g_4 is 1, since this genotype has no heterozygous sites. Starting from the clique lower bound, Algorithm 2 increases the lower bound to 9, by considering g_5 and g_6, and increasing the lower bound by 1 due to each genotype. Let $\{h_1, h_2, h_3, h_4, h_5, h_6, h_7, h_8, h_9\}$ be the haplotypes identified with the improved lower bound. Due to the way the SHIPs lower bound is constructed, it is possible to associate haplotypes with specific genotypes. As a result, let $\{h_1, h_2\}$ be associated with g_1, $\{h_3, h_4\}$ with g_2, $\{h_5, h_6\}$ with g_3, $\{h_7\}$ with g_4, $\{h_8\}$ with g_5, and $\{h_9\}$ with g_6. As a result, the haplotypes associated with g_1, g_2 and g_3 are known, and so it is unnecessary to use clauses to select from the set of candidate haplotypes. The same holds true for g_4 with respect to h_7, because g_4 has no heterozygous sites. Finally, for g_5 and g_6, one of the haplotypes is known, respectively h_8 and h_9, and it is only necessary to select the remaining haplotype from the set of candidate haplotypes. The only haplotype not used for the lower bound, g_7, requires considering as candidate haplotypes all haplotypes not associated with genotypes

Table 1. Relations between genotypes and haplotypes in example

Genotype	Compatible Genotypes	Sets of Candidate Haplotypes
g_1	$\{g_7\}$	$\{h_1\}, \{h_2\}$
g_2	$\{g_5, g_6\}$	$\{h_3\}, \{h_4\}$
g_3	$\emptyset$	$\{h_5\}, \{h_6\}$
g_4	$\{g_5, g_7\}$	$\{h_7\}$
g_5	$\{g_2, g_4, g_6, g_7\}$	$\{h_8\}, \{h_3, h_4, h_7, h_9\}$
g_6	$\{g_2, g_5\}$	$\{h_9\}, \{h_3, h_4, h_8\}$
g_7	$\{g_1, g_4, g_5\}$	$\{h_1, h_2, h_7, h_8\}, \{h_1, h_2, h_7, h_8\}$

incompatible with g_7. Since the lower bound is 9, the first iteration of the SHIPs algorithm considers 9 haplotypes. As explained above, for g_1, g_2, g_3 and g_4 it is unnecessary to select haplotypes, since the associated haplotypes are known. For g_5 and g_6 only one of the explaining haplotypes needs to be selected from the set of candidate (and potentially compatible) haplotypes, since the other explaining haplotype is known to be h_8 and h_9, respectively. Finally, for g_7, both haplotypes explaining g_7 must be selected from the set of candidate haplotypes that are potentially compatible. These conditions are summarized in Table 1. Without simplifications from the lower bond information, each of the 7 genotypes would have to select two haplotypes from a set of 9 possible haplotypes. Besides the significant reduction in the number of clauses, the number of auxiliary variables is also reduced.

5 Improving Lower Bounds with Local Search

The clique-based method for computing lower bounds is not guaranteed to find the best lower bound. Firstly, the maximal clique is computed using a greedy algorithm, which may of course find a clique of less than maximum size. Secondly, even a maximum clique would not be guaranteed to lead to the best result, as a smaller clique might lead to a greater lower bound when adding genotypes in the extended method described above. Thirdly, the lower bound may be sensitive to the order in which genotypes are added during the extended method.

To find the best possible lower bound using the above forms of reasoning, we should in principle test all cliques in the first method and all orderings in the second method. This is clearly impractical and probably more expensive than using an inferior lower bound. But we can perform a limited search for a clique and genotype ordering that gives the best lower bound. This is an NP-hard problem: any maximal clique problem (which is itself NP-hard) can be expressed as the search for a clique plus an empty ordering, by constructing a set of genotypes that are heterozygous in all sites, and mutually incompatible if their corresponding graph vertices are adjacent.

5.1 The Problem of Finding an Optimal Lower Bound

The specific problem we aim to solve is as follows. Construct a sequence of genotypes such that each genotype has one of two possible justifications for being in the sequence: if it has an *I-justification* then it is incompatible with all earlier genotypes; if it has an *H-justification* it has a heterozygous site at which all earlier compatible genotypes have the same homozygous value. Define the *score* of a sequence as the sum of the scores of its members; the score of a genotype in the sequence is 2 if it is I-justified and has at least one heterozygous site, otherwise 1. The score s of a sequence establishes a lower bound of s for the population. This optimisation problem can be reduced to a series of feasibility problems: find a sequence with score 1, then 2, then 3, and so on until the score cannot be increased. Any known lower bound lb can be exploited by starting from score $lb + 1$.

We shall solve each feasibility problem by modelling it and applying a local search algorithm. The scores can be handled by using a linear pseudo-Boolean model, but as each score can only be either 1 or 2 it is easy to SAT-encode the problem instead. Local search is more well developed for SAT than for pseudo-Boolean models, and a wider range of local search algorithms are available. Instead of assigning a score to each genotype, we allow genotypes that would have a score of 2 to appear twice in the sequence. We must then modify the problem so that an I-justified genotype is incompatible with all earlier genotypes except for any copy of itself.

5.2 A SAT Model

Define SAT variables $S_{ik}, I_i, H_i, H_{ijb}, X_{ik}, Y_{ii'}$ where $1 \leq i < i' \leq n$, $1 \leq j \leq m$, $1 \leq k \leq l$, $b \in \{0,1\}$, n is the number of genotypes, m the number of sites in each genotype, l the sequence length, and $H_i, X_{ik}, Y_{ii'}$ are auxiliary variables introduced to reduce the space complexity of the model. The meanings of these variables are as follows: $S_{ik} = T$ means that sequence element k is genotype i, $I_i = T$ that genotype i is I-justified, H_{ijb} that genotype i is H-justified (the genotype must be heterozygous in site j and all earlier compatible genotypes have b in site j, and each H_{ijb} is defined only if genotype i is heterozygous in site j), $H_i = T$ that genotype i is H-justified for some j, b, $X_{ik} = T$ that genotype i does not occur in sequence position k' for any $k' < k$, and $Y_{ii'} = T$ means that genotype i' cannot precede genotype i in the sequence (i, i' are compatible). The clauses are as follows. Define the X_{ik} in terms of the S_{ik}:

$$\overline{X}_{ik} \vee \overline{S}_{ik'}$$

where $k' < k$. Define the $Y_{ii'}$ in terms of the S_{ik} and X_{ik}:

$$\overline{Y}_{ii'} \vee \overline{S}_{ik} \vee X_{i'k}$$

Define the H_i in terms of the H_{ijb}:

$$\overline{H}_i \vee \bigvee_j \bigvee_b H_{ijb}$$

Each sequence position takes exactly one genotype:

$$\bigvee_i S_{ik}$$

$$\overline{S}_{ik} \vee \overline{S}_{i'k}$$

Any non-totally homogeneous genotype in the sequence must be justified:

$$\overline{S}_{ik} \vee I_i \vee H_i$$

Any totally homogeneous genotype in the sequence must be I-justified:

$$\overline{S}_{ik} \vee I_i$$

Any H-justified genotype appears at most once in the sequence:

$$\overline{H}_i \vee \overline{S}_{ik} \vee X_{ik}$$

Any totally homozygous I-justified genotype appears at most once in the sequence:

$$\overline{I}_i \vee \overline{S}_{ik} \vee X_{ik}$$

Any I-justified genotype with at least one heterozygous site appears at most twice in the sequence:

$$\overline{I}_i \vee \overline{S}_{ik} \vee \overline{S}_{ik'} \vee X_{ik'}$$

where $k' < k$. Any I-justified genotype is incompatible with all earlier genotypes in the sequence:

$$\overline{I}_i \vee Y_{ii'}$$

where $i \neq i'$ are compatible. Any H-justified genotype constrains earlier genotypes in the sequence:

$$\overline{H}_{ijb} \vee Y_{ii'}$$

where $i' \neq i$ are compatible, genotype i is heterozygous in site j, and genotype i' has anything but b in site j. This SAT model has $\mathcal{O}(nm + nl + n^2)$ variables and $\mathcal{O}(nl^2 + n^2 l + n^2 m)$ literals. Without the use of auxiliary variables $H_i, X_{ik}, Y_{ii'}$ the complexity would be higher, which was the motivation for defining them.

5.3 Alternative models

There are several additional clauses that may be added to this model. The definition of the H_i can be made if-and-only-if:

$$\overline{H}_{ijb} \vee H_i$$

We could insist that no genotype in the sequence can be both I- and H-justified:

$$\overline{H}_i \vee \overline{I}_i$$

Any sequence can be transformed into one with all the I-justified genotypes at the start. We can shift any I-justified genotype G one place earlier in the sequence because it is incompatible with the genotype G' that preceded it. If G' was I-justified then it still is. On the other hand, if G' was H-justified then the swap has no effect: G' in its new position need not be H-justified with respect to G because they are incompatible. We can do this by adding:

$$\overline{H}_i \vee \overline{I}_{i'} \vee Y_{i'i}$$

The sequence then corresponds to the method of the previous section in which a clique is first constructed, then further genotypes are added. In experiments all these additional clauses either made little difference or slowed down local search. This is consistent with the known result that adding symmetry breaking and other clauses can harm local search performance [28]. It is often (though not always) best to omit clauses that are unnecessary for correctness.

Lower bounds might be further improved by devising additional justifications for adding a genotype to the sequence, using higher-order reasoning on multiple sites. Here is an example that works on triples of sites, which we call a *T-justification.*

Example 5. Suppose that genotype i is heterozygous in sites $j < j' < j''$, and all earlier genotypes in the sequence are homozygous and take one of the four patterns $bbb, \overline{b}bb, b\overline{b}b, bb\overline{b}$ in those sites, where $b \in \{0, 1\}$ and $\overline{b} = 1 - b$. Then we may add genotype i to the sequence because no pair of haplotypes explains the three heterozygous sites.

To use T-justifications we modify the SAT model by defining variables $T_{ijj'j''b}$ for each genotype i that is heterozygous in sites j, j', j'', if at least one other genotype follows one of the four allowed patterns in those sites. We also define auxiliary variables T_i by:

$$\overline{T}_i \vee \bigvee_j \bigvee_{j'} \bigvee_{j''} \bigvee_b T_{ijj'j''b}$$

Now any genotype in the sequence has a larger choice of justifications:

$$\overline{S}_{ik} \vee I_i \vee H_i \vee T_i$$

Any genotype with a T-justification appears at most once in the sequence:

$$\overline{T}_i \vee \overline{S}_{ik} \vee X_{ik}$$

Any genotype with a T-justification constrains earlier genotypes in the sequence:

$$\overline{T}_{ijj'j''b} \vee Y_{ii'}$$

where genotypes $i' \neq i$ are compatible, genotype i is heterozygous in sites j, j', j'', and genotype i' is homozygous and does not follow one of the four allowed

Algorithm 4 The new local search variant VW3

```
LOCAL-SEARCH($p, q, s$)
Initialise all variables to randomly selected truth values
Initialise all variable weights to 0
repeat
        Randomly select a violated clause C
        if C contains freebie variables
            then Randomly flip one of them
            else
                rp = random()
                if rp < $p$
                    then
                        rq = random()
                        if rq < $q$
                            then Flip a variable in C chosen randomly
                            else Flip a variable in C with least weight
                    else Flip a variable in C with fewest breaks,
                         break ties by least weight
        Update flipped variable weight using $s$
    until no clause is violated
```

patterns in those sites. A drawback with T-justifications is that they increase the complexity of the model to $\mathcal{O}(nm^3 + nl + n^2)$ variables and $\mathcal{O}(nl^2 + n^2l + n^2m^3)$ literals. We were therefore only able to perform experiments on small instances, on which no lower bound improvements were found. We do not use T-justifications in the experiments below but hope to develop them in future work, possibly by lifting the SAT model via quantification.

5.4 The Local Search Algorithm

As mentioned above, it would be impractical to explore all solutions to this SAT problem, and in preliminary experiments complete solvers took much longer than local search algorithms to find solutions. We experimented with several well-known algorithms and obtained our best results with a new variant of the VW algorithm [29], which we now describe.

VW is based on the SKC variant of the well-known WalkSAT algorithm [25, 31]. It dynamically adjusts weights associated with variables, instead of the clause weights used in several other algorithms (see [15] for a survey on these *dynamic local search* algorithms). Most SAT local search algorithms randomly select a violated clause, then use a heuristic to select a variable in that clause to flip (invert its truth current value). The heuristic used by WalkSAT/SKC is to choose the variable v with smallest *break count* b_v, defined as the number of satisfied clauses that would be violated by the flip. VW adjusts this heuristic by adding a term $c(w_v - M)$ to b_v where c is a constant, w_v is the current weight of

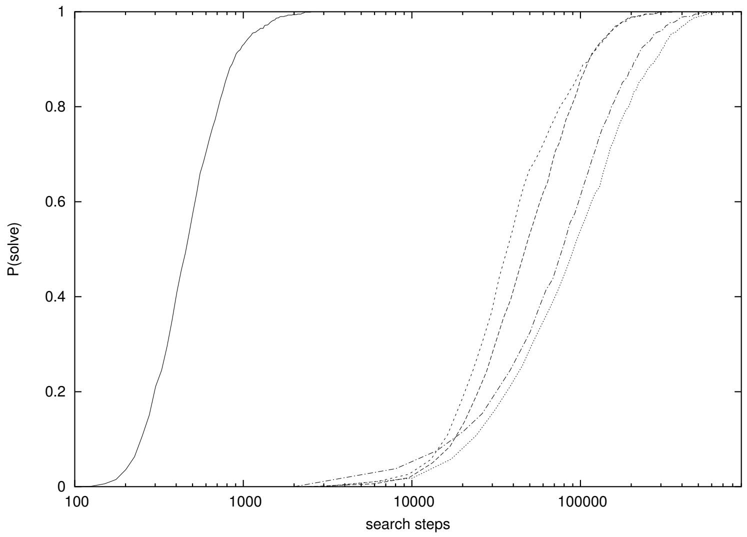

Fig. 2. Run length distributions for five HIPP instances

v, and M is the mean current weight over all variables. In common with SKC, VW also uses *freebie moves*: while checking the break counts of variables in a violated clause, if it detects one with zero break count then it is immediately selected. Also in common with SKC, if no freebie occurred then with probability p select a random variable from the clause, where p is a *noise parameter*. After selecting and flipping v its weight is adjusted according to the formula

$$w_v \leftarrow (1-s)(w_v+1) + s \times t$$

where s is a *smoothing actor*.

VW has three parameters that must be chosen by the user at runtime: c, s, p. This makes it quite difficult to tune. In particular, parameter c is a positive number with no upper bound, and best results may be obtained with a very small number such as 10^{-6} or a value greater than 1. To make tuning easier we now define a new VW variant in which parameter c is replaced by a probability q. The effect of nonzero q is to allow variables to be occasionally chosen purely on the basis of their weights, ignoring their break counts. This has a similar effect to the c parameter, which allows sufficiently large weights to override the effects of break counts. In experiments we found that setting $p = q$ usually gives reasonable results, so we can often eliminate parameter q. The new variant is much easier to tune, and in all the experiments below we use parameter settings $p = q = s = 0.1$. [29] defined two earlier variants VW1 and VW2 so we call the new variant VW3. It is summarised in Algorithm 4.

VW3 with these parameter settings was tested for search stagnation, which might occur if (for example) the noise parameter was set to a very low value.

We generated run-length distributions (as suggested in [16]) for five problems, using SAT instances corresponding to the highest lower bounds that we could establish, performing 10,000 runs in each case. The plot in Figure 2 shows that stagnation does not occur. Each of the five problem instance was solved 1,000 times and there is no flattening of the curves that would indicate stagnation.

6 Experimental Results

In this section we provide empirical evidence that the use of local search is able to tighten lower bounds for solving the HIPP problem. In addition, the identification of a tighter lower bound by local search significantly decreases the CPU time required by SHIPs to solve a given problem instance.

6.1 Experimental Setup

We have evaluated a set of 419 problem instances that can be divided in two categories:

- 90 problem instances were created based on the `ibd` problem described in [4] for which the corresponding haplotypes are available from `http://htsnp.stanford.edu/PCA/IBD.html`. Our 90 problem instances correspond to sets of genotypes that were obtained by pairing randomly chosen haplotypes. We should note that these instances have been selected for being the hardest for SHIPs to solve from those considered in [21]. (The remaining instances are trivially solved and are therefore irrelevant for our purposes.) The `ibd` problem described in [4] corresponds to a 500-kb region on human chromosome 5q31 that is implicated as containing a genetic risk factor for Crohn disease. After high-density SNP discovery, were selected 103 common SNPs (i.e. with >5% minor allele frequency) genotyped in 129 trios from a European-derived population. Half of the instances correspond to chromosomes transmitted to 258 individuals with Crohn disease and the remaining ones to 258 untransmitted chromosomes. An EM-type algorithm [5, 8] was used to include the minority of chromosomes that had one or more markers with ambiguous phase (that is, where both parents and offspring were heterozygous) or where one marker was missing genotype data.
- 329 problem instances correspond to the `SU1, SU2, SU3` and `SU-100kb` instances available from `http://www.stats.ox.ac.uk/∼marchini/phaseoff.html` and described in [30]. These simulated datasets have been used to evaluate phasing methods in the paper [24]. `SU1` instances correspond to 100 data sets of 90 unrelated individuals simulated with constant recombination rate across the region, constant population size, and random mating. Each of the 100 data sets consisted of 1 Mb of sequence. `SU2` instances are the same as `SU1`, but with the addition of a variable recombination rate across the region. `SU3` instances are the same as `SU2`, except that a model of demography consistent with white Americans was used. `SU-100kb` instances are identical

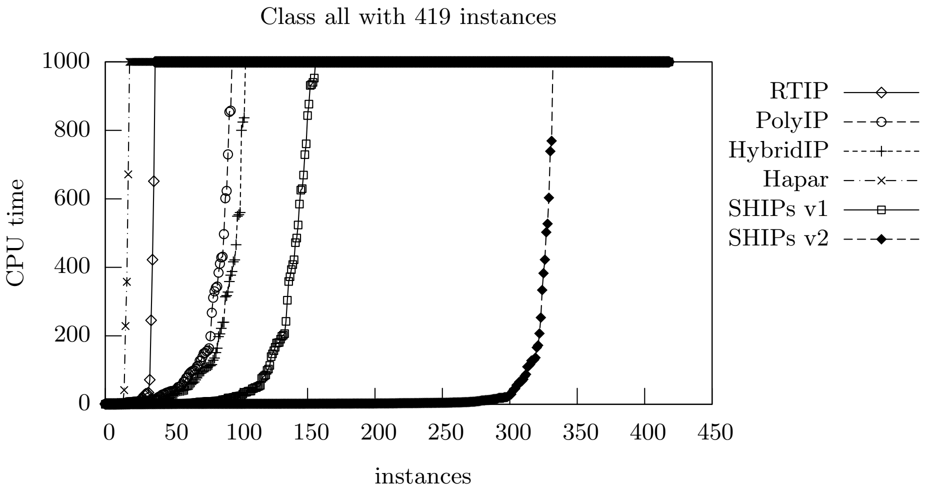

Fig. 3. Relative performance of HIPP solvers

to the SU3 set, except that the sequences are only 100 kb in length. Each of these 100-kb data sets was created by subsampling a set of 1,180 simulated haplotypes. The remaining 1,000 haplotypes were used to estimate the true population haplotype frequencies. This allowed a comparison of phasing algorithms ability to predict the haplotype frequencies in a small region of interest. These instances are in general harder for SHIPs than the ones used in [21].

6.2 SHIPs performance

A comparison of the performance of alternative approaches to the HIPP problem is summarized in Figure 3. A universe of 419 problem instances described above was used. All problem instances were simplified in a preprocessing step, according to what has been suggested in [2]: duplicated genotypes and sites were removed, as well as complemented sites.

The HIPP solvers RTIP [11], PolyIP [1], HybridIP [2], Hapar [35] and SHIPs [21] were considered[5]. In addition, we give results for two versions of SHIPs: the one described in [20] (SHIPs.v1) and the one introduced in this paper (SHIPs.v2). The run times for each solver were sorted and plotted, the cutoff point being 1000 seconds. The results shown were obtained on a 1.9 GHz AMD Athlon XP with 1GB of RAM. For the ILP-based HIPP solvers, the ILP package used was CPLEX version 7.5.

[5] All results were obtained with the tools provided by the authors.

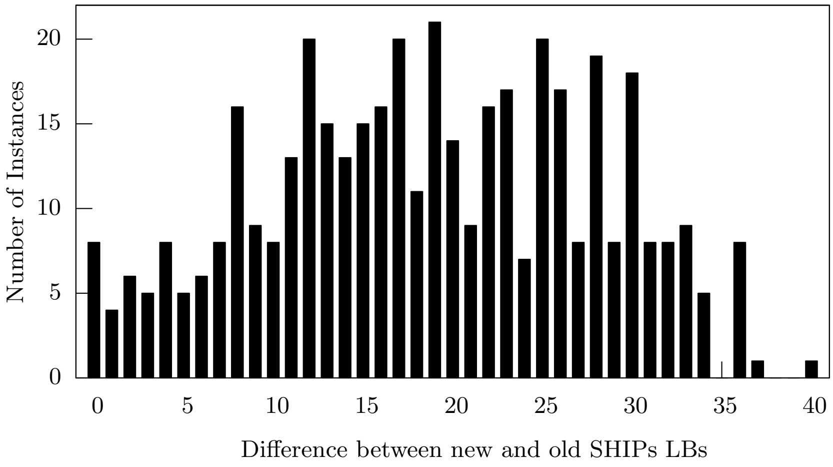

Fig. 4. Comparison of improved SHIPs lower bound with previous SHIPs lower bound

As can be concluded, SHIPs.v2 is the HIPP tool capable of solving the largest number of problem instances. SHIPs.v2 aborts 87 problem instances out of 419 instances, whereas SHIPs.v1 aborts 263 instances, HybridIP aborts 315 instances, PolyIP aborts 325 instances, RTIP aborts 382 instances and Hapar aborts 401 instances. We should also note that most of the problem instances aborted by RTIP were aborted due to memory exhaustion. These results provide additional support that better lower bounds can contribute decisively for improving SHIPs performance.

6.3 SHIPs performance with local search

We first compare the SHIPs lower bound described in [20, 21] with the SHIPs improved lower bound described earlier in this paper. Next we will evaluate the impact of the use of local search on SHIPs performance. These experiments were run on an Intel Xeon 5160 (3.0GHz with 4GB of RAM) with a timeout of 1000 seconds.

Our evaluation of the new lower bound considers two different aspects: the required CPU time and the quality of the new bound. In terms of CPU time the difference is negligible: for the 419 instances the new lower bound requires 7.93 seconds more than the old one (94.78 *versus* 86.85). The quality of the new lower bound can be analyzed from Figure 4. Overall, the new lower bound can represent as much as an increment of 40 haplotypes with respect to the previous bound. On average we observe an increment of 19 haplotypes. (From now on, whenever we refer to SHIPs we mean the SHIPs procedure that uses this improved lower bound.)

Next we evaluate the impact of the use of local search on SHIPs performance. We start by observing that although the selected classes of instances are among the hardest for SHIPs, some specific problem instances from different classes are easily solved in a few seconds. Moreover, the use of local search for improving the lower bound requires additional time, and therefore we decided to run SHIPs for a few seconds before using local search. Only for the unsolved instances is the use of local search justified.

After running SHIPs for 100 seconds we found that only 100 instances remained to be solved. SHIPs was able to solve 207 instances in less than 1 second, 299 instances in less than 10 seconds, 315 instances in less than 50 seconds and 319 instances in less than 100 seconds. Given the reduced number of instances solved between 50 and 100 seconds, it seems clear that running SHIPs for longer is not cost-effective. (Actually, we have run SHIPs for longer and observed that the improvements were negligible.)

The next step is to use local search to improve the lower bound. The local search procedure requires a lower bound value as input. Starting at this value, it tries to find a solution with the given size. In the next iteration this size is increased. We can now consider three different possibilities with respect to the lower bound value to be given as input:

1. The rank lower bound;
2. The lower bound computed by SHIPs;
3. The last problem proved unsatisfiable by SHIPs.

Also, we must decide for how long the local search procedure should run. We have already *lost* 100 seconds running SHIPs without finding a solution, but we can retry it after improving the lower bound. Regardless of the value to be passed as input to the local search procedure, we chose a run-time of 200 seconds as a good trade-off: if we run it for a shorter period of time then we are not able to find a new bound for many problem instances, or the improvement is not significant, and therefore the use of local search is not beneficial; if we run it for longer then we are not able to further increase the lower bound for most problem instances, and again most of the time is wasted for many problem instances.

With respect to the different possibilities to be given as input, we observed that the lower bound computed by SHIPs is the most effective one, despite the facts that the rank lower bound has in some cases a higher value and that the value given by the last problem proved to be unsatisfiable by SHIPs is almost always higher than the SHIPs lower bound. Both approaches for finding a lower bound - SHIPs and local search - are based on identifying a clique and therefore, not surprisingly, they seem to be complementary.

The lower bounds obtained from the matrix rank or from SHIPs last iteration are based on different concepts. Consequently, local search can hardly find a new lower bound or even prove that the input lower bound is indeed a lower bound. Given the SHIPs lower bound, local search is able to find a new bound for 99 out of 100 problem instances within 200 seconds. On the other hand, local search is able to find a new lower bound for 26/100 instances when given the

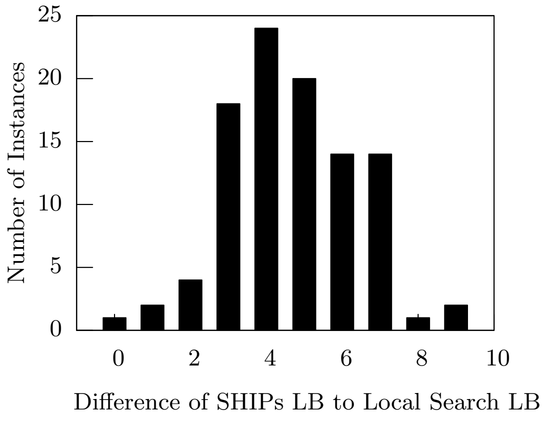

Fig. 5. Comparison of SHIPs lower bound with Local Search lower bound

rank based lower bound and for 5/100 instances when given the value corresponding to SHIPs last unsatisfiable iteration. From the 26 instances for which the local search procedure succeeds when given the rank lower bound, only for 6 instances the lower bound is better than the one computed when given the SHIPs lower bound. Clearly, for the lower bounds different from the one computed by SHIPs the use of local search is hardly an advantage and may even be seen as a disadvantage if we have to take into account for the time spent on local search.

At this point we have already defined the experimental procedure, that can be summarised as follows:

1. Run SHIPs for *100 seconds* (and terminate if solution is found);
2. Use local search for *200 seconds* to improve the lower bound computed by SHIPs;
3. Run SHIPs again for *700 seconds*.

The main goal is to get better results following this procedure when compared with running SHIPs for *1000 seconds*. We will first evaluate how does local search improve the lower bound and afterwards we will evaluate the impact of the new lower bound on solving the HIPP problem instances.

Figure 5 illustrates the difference between the new lower bound provided by the local search procedure and the (improved) SHIPs lower bound. The lower bound improvements range from 0 to 9 haplotypes, and local search does not improve the lower bound for only one instance. On average, the lower bound is incremented in 7 haplotypes.

Observe however that even a small increment in the lower bound may significantly reduce the size of the generated CNF formula, which is also expected to significantly reduce the required CPU time. Figure 6 compares the formulas

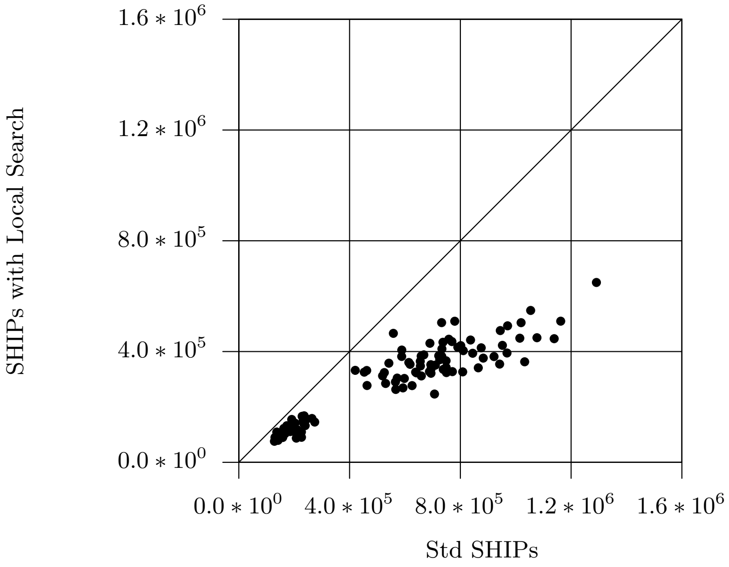

Fig. 6. Comparison of SHIPs and SHIPs with local search lower bound on the number of literals of the generated formulas

generated by SHIPs when not using the local search procedure with the formulas generated by SHIPs enhanced with the local search procedure. The comparison is made in terms of the number of literals in the formula. Given a lower bound *lb* generated by local search, we have compared the formula produced by the standard SHIPs procedure, given the value *lb*, with the formula produced by SHIPs when given not only the value *lb* but also the output produced by the local search procedure. This output contains information relating the lower bound value with specific genotypes. Overall, the reduction is significant, specially for larger formulas.

Figure 7 evaluates the impact of the local search procedure in SHIPs. Each point corresponds to a problem instance, where the x-axis corresponds to the CPU time required by SHIPs when enhanced with the local search procedure, and the y-axis corresponds to the CPU time required by the standard SHIPs procedure for solving the instance. We should note that the standard SHIPs procedure utilizes the improved lower bound procedure. Hence, these results evaluate exclusively the impact of the use of local search for improving SHIPs performance.

The most relevant result that can be inferred from Figure 7 is that SHIPs aborts 80 problem instances, whereas SHIPs enhanced with local search only aborts 43 problem instances. Considering that each unsolved problem instance counts for 1000 seconds (the CPU time limit), then SHIPs has used 87,030 seconds, whereas SHIPs enhanced with local search only uses 65,155 seconds. The

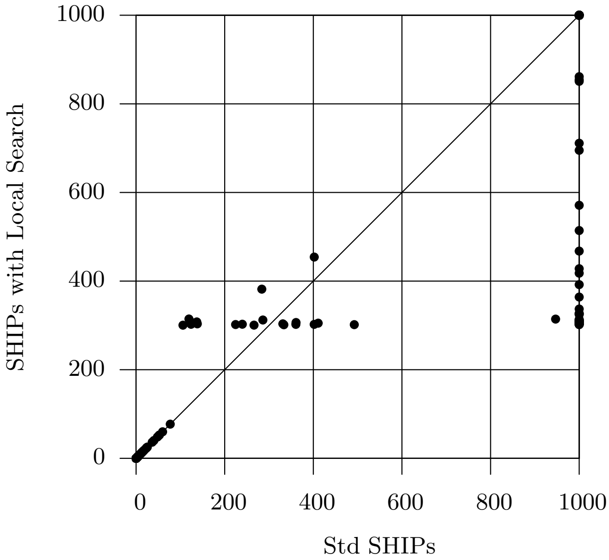

Fig. 7. Comparison of SHIPs and SHIPs with local search lower bound on the COU time (seconds)

time difference is not more significant because when using local search the first step consists in running SHIPs for 100 seconds. Consequently, for all instances solved in less than 100 seconds the time required for both approaches is the same. In addition, for the instances not solved in less than 100 seconds we have to account for at least 300 seconds (100 seconds for SHIPs + 200 seconds for local search). This explains why a significant subset of instances is solved in slightly over 300 seconds.

The speedup resulting from the use of local search is significant on instances previously aborted by SHIPs. 25 of the 37 problem instances that are only solved by SHIPs with local search are solved in less than 400 seconds. In practice, this means that SHIPs, when given the lower bound computed by local search, is now able to solve in less than 100 seconds a problem instance that could not be solved in less than 1000 seconds. (We should however take into account that we have spent at least 100 seconds and at most 300 seconds before running the new version of SHIPs.)

7 Conclusions

The SAT-based SHIPs approach was previously shown to be the most effective approach to the HIPP problem on a range of problem instances. This paper significantly improves its performance on challenging instances by using a SAT local search procedure to improve lower bounds, before using the complete SAT solver to find and prove an optimal result. The new procedure allows SHIPs to solve challenging instances that were previously unsolvable. It does so in two distinct ways. Firstly, it reduces the number of iterations that SHIPs needs to make to find an optimal solution (SHIPs increases the lower bound until the first solution is found). Secondly, it allows the SAT model to be considerably reduced in size, thus increasing efficiency.

It is often noted that the backtrack search and local search paradigms have complementary strengths and weaknesses, and an active research area is the design of hybrid approaches that exploit the strengths of both. In particular, local search can be used to improve upper bounds in a minimisation problem (or equivalently lower bounds in a maximisation problem) by quickly finding near-optimal solutions; backtrack search is then used to find an optimal solution and to prove it optimal. A novel aspect of our work is that local search is used to improve *lower* bounds for a minimisation problem, which is then used to improve backtrack search.

In future work we hope to improve lower bounds further, by investigating the extended reasoning outlined in Section 5. To handle the resulting greater space complexity of the SAT models, we may use quantification to generate clauses as needed during search. Alternatively, a Constraint Programming model might be more compact, if a large family of clauses can be represented by a global constraint.

Acknowledgments

This material is based in part upon works supported by the Science Foundation Ireland under Grant No. 05/IN/I886, 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 an 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. A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.
6. 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.
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. L. Excoffier and M. Slatkin. Maximum-likelihood estimation of molecular haplotype frequencies in a diploid population. *Molecular Biology and Evolution*, 12:921–927, 1995.
9. 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.
10. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.
11. D. Gusfield. Haplotype inference by pure parsimony. In *14th Annual Symposium on Combinatorial Pattern Matching (CPM'03)*, pages 144–155, 2003.
12. 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.
13. 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.
14. E. Halperin and R. Karp. Perfect phylogeny and haplotype assignment. In *Annual International Conference on Computational Molecular Biology*, pages 10–19, March 2003.
15. H. Hoos and T. Stützle. *Stochastic Local Search: Foundations and Applications.* Morgan Kaufmann, 2004.
16. H. H. Hoos and T. Stützle. *Stochastic Local Search: Foundations and Applications.* Morgan Kaufmann, 2004.
17. 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.
18. K. Kalpakis and P. Namjoshi. Haplotype phasing using semidefinite programming. In *International Symposium on Bioinformatic and Bioengineering*, pages 145–152, October 2005.
19. 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.
20. I. Lynce and J. Marques-Silva. Efficient haplotype inference with Boolean satisfiability. In *National Conference on Artificial Intelligence (AAAI)*, July 2006.
21. 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.
22. I. Lynce and J. Marques-Silva. Breaking symmetries in SAT matrix models. In *International Conference on Theory and Applications of Satisfiability Testing (SAT)*, May 2007.
23. I. Lynce, J. Marques-Silva, and A. L. Oliveira. Improved lower bounds for SAT-based haplotype inference. Technical Report 17, INESC-ID, June 2006.

24. 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.
25. D. McAllester, B. Selman, and H. Kautz. Evidence for invariants in local search. In *National Conference on Artificial Intelligence (AAAI)*, pages 321–326, 1997.
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. N. Patil, A. J. Berno, D. A. Hinds, W. A. Barrett, J. M. Doshi, C. R. Hacker, C. R. Kautzer, D. H. Lee, C. Marjoribanks, D. P. McDonough, B. T. N. Nguyen, M. C. Norris, J. B. Sheehan, N. Shen, D. Stern, R. P. Stokowski, D. J. Thomas, M. O. Trulson, K. R. Vyas, K. A. Frazer, S. P. A. Fodor, and D. R. Cox. Blocks of limited haplotype diversity revealed by high-resolution scanning of human chromosome 21. *Science*, 294:1719–1723, 2001.
28. S. Prestwich. Negative effects of modeling techniques on search performance. *Annals of Operations Research*, 118:137–150, 2003.
29. S. Prestwich. Random walk with continuously smoothed variable weights. In *International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 203–215, 2005.
30. 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.
31. B. Selman, H. Kautz, and B. Cohen. Noise strategies for improving local search. In *National Conference on Artificial Intelligence (AAAI)*, pages 337–343, 1994.
32. M. Stephens, N. Smith, and P. Donelly. A new statistical method for haplotype reconstruction. *American Journal of Human Genetics*, 68:978–989, 2001.
33. The International HapMap Consortium. The international hapmap project. *Nature*, 426:789–796, 2003.
34. The International HapMap Consortium. A haplotype map of the human genome. *Nature*, 437:1299–1320, 27 October 2005.
35. L. Wang and Y. Xu. Haplotype inference by maximum parsimony. *Bioinformatics*, 19(14):1773–1780, 2003.