Technical Report
EHU-KZAA-TR-2-2010

UNIVERSITY OF THE BASQUE COUNTRY
Department of Computer Science and Artificial Intelligence

A Preprocessing Procedure for Haplotype Inference by Pure Parsimony

Ekhine Irurozki, Borja Calvo, José A. Lozano

March 2010

San Sebastian, Spain
http://www.ccia-kzaa.ehu.es/

A Preprocessing Procedure for Haplotype Inference by Pure Parsimony

Ekhine Irurozki, Borja Calvo, Jose A. Lozano
ekhine.irurozqui@ehu.es, borja.calvo@ehu.es, ja.lozano@ehu.es
Intelligent Systems Group, University of the Basque Country, Spain
http://www.sc.ehu.es/isg

Abstract

Haplotype data is especially important in the study of complex diseases since it contains more information than genotype data. However, obtaining haplotype data is technically difficult and expensive. Computational methods have proved to be an effective way of inferring haplotype data from genotype data. One of these methods, the haplotype inference by pure parsimony approach (HIPP), casts the problem as an optimization problem and as such has been proved to be NP-hard. We have designed and developed a new preprocessing procedure for this problem. Our proposed algorithm works with groups of haplotypes rather than individual haplotypes. It iterates searching and deleting haplotypes that are not helpful in order to find the optimal solution. This preprocess can be coupled with any of the current solvers for the HIPP that need to preprocess the genotype data. In order to test it, we have used two state-of-the-art solvers, RTIP and GAHAP, and simulated and real HapMap data. Due to the computational time and memory reduction caused by our preprocess, problem instances that were previously unaffordable can be now efficiently solved.

1 Introduction

The genetic material of humans is organized in 23 pairs of chromosomes. In each pair, one of the chromosomes is inherited from the mother and the other from the father. The sequence of DNA in contiguous positions along a chromosomal region is called a haplotype, while mixed data of both chromosomes is called a genotype. In order to trace the structure of the human population and improve our ability to map disease genes, haplotype information is more relevant than genotype information. However, due to technology limitations, haplotype information is harder to obtain than genotype data: instruments can identify whether an individual is heterozygous at a site (alleles in both haplotypes are equal) or homozygous (alleles are different in the two haplotypes). Therefore, if the two alleles are different at a given site, we do not know which allele belongs to which haplotype. Hence, computational methods have been designed to obtain haplotype data from genotype data. This problem has been called *haplotype inference* (HI). There are several approaches to this problem: combinatorial, statistical, etc. Reviews about the haplotype inference problem

can be found in [1], [2], [3].

This paper focuses on one of the most popular approaches to the haplotype inference problem which is known as *haplotype inference by pure parsimony* (HIPP) [4]. This approach is based on the fact that, due to the process of genetic inheritance, the number of haplotypes in a given population is vastly smaller than the number of possible haplotypes. The goal is, therefore, to explain a given set of genotypes with the minimum possible number of haplotypes. This problem can be posed as a combinatorial optimization problem and it has been proved to be NP-hard [3] [5]. Although some authors have noticed the need to introduce additional properties for the model [6] the HIPP still is a lively approach to HI as shown in the number of recently published papers [7], [8], [9]. In the literature we can find several solvers for the problem, particularly those based on integer linear programming (TIP [4], RTIP [4], PolyIP, [10], [11], HybridIP [10], SM, RM and SMM [7]), branch and bound (HAPAR [12]), genetic algorithms (GAHAP [13]), local search [14], boolean satisfiability, SAT (SHIPs [15]) and pseudo boolean optimization (PolyPB, RPoly and NRPoly [16], [17]).

Many of the previously cited solving methods carry out a search over the space of possible resolving haplotype pairs of the genotypes. Therefore, before running the optimization algorithm, these solvers have to preprocess the genotype data by expanding all possible pairs of haplotypes for each genotype. Unfortunately, the number of resolving pairs for a genotype grows exponentially with respect to the number of heterozygous positions it has. Therefore, due to memory and computational time limitations, these optimization methods can not be applied to medium-large instances. Particularly, this problem affects some of the most widely applied algorithms such as TIP [4] and its optimized version RTIP [4], which cast the problem as an integer linear program (ILP). These formulations generate an exponential number of constraints on the number of ambiguous positions of the genotypes. It is also the case of GAHAP [13], the genetic algorithm, which requires an exponential amount of memory.

In this paper we propose a new preprocessing step for the HIPP problem. This preprocess drastically reduces the subset of resolving pairs generated for each genotype by detecting, before running the optimization algorithm, that some groups of haplotypes are not relevant. In other words, for every solution that includes elements of these groups, another solution can be built that does not include these elements and has, at most, the same number of different haplotypes. From now on we will refer to these haplotypes as *irrelevant haplotypes*.

We have tested our preprocessing algorithm with simulated and real data obtained from the HapMap project [18]. For small instances our preprocess shows quite a similar time performance to those in the literature. However, for medium-big instances when starting with the reduced subset of resolving pairs generated with our preprocess, the ILP formulation used in TIP and RTIP can deal with HI problems that were previously unaffordable. Moreover, for those problems of large size that can be solved with the original RTIP, we show a dramatic decrease in the computational time when departing from our reduced set of resolving pairs. Similar performance improvements are achieved when the applied solver is GAHAP.

The rest of the paper is organized as follows: in the next section we give a more detailed description of the HIPP, providing the background of the problem and presenting the RTIP and GAHAP algorithms. Our proposed algorithm is explained in Section 3. The results are presented and discussed in Section 4.

Genotype	Resolving pairs
221120	(110110, 111010)
	(110010, 111110)
201002	(100001, 101001)
211120	(100010, 111110)
	(100110, 111010)
	(101010, 111010)
	(101110, 110010)

Table 1: Three genotypes and every possible resolving pair for it

The paper ends with the conclusions and future work.

2 Haplotype inference by pure parsimony, TIP formulation and GAHAP algorithm

Humans share about 99% of the DNA, so, in order to study human genome, researches usually focus on the mutations. The most common form of variation is the Single Nucleotide Polymorphism (SNP) which affects one single base in a DNA region. Therefore, HI studies just consider the positions of the SNPs. When we look at a collection of members of a population and focus on a particular SNP position, it usually occurs that only two out of four bases appear in a significant percentage of the population. Formally, we represent haplotypes as length m binary vectors $(h[1], ..., h[m])$ such that $h[i] \in \{0, 1\},\ i = 1, ..., m$ where 0 represents the wild allele (the most frequent one) and 1 the mutated allele. Since the genotype information refers to the mixed information of its two haplotypes, we have three possible values at a given position of a genotype: 0, representing that it is homozygous with 0 (which means that both haplotypes have value 0 at the given position), 2 (denoting that it is homozygous with 1) or value 1 when it is heterozygous (one haplotype has value 0 and the other has value 1)[1]. Therefore, a genotype is represented as a length m ternary vector $(g[1], ..., g[m])$ such that $g[i] \in \{0, 1, 2\}$ for $i = 1, ..., m$. We say that genotype g is solved by haplotypes h_1 and h_2, and represent it by $g = h_1 \oplus h_2$, if $g[i] = h_1[i] + h_2[i]$ for $i = 1, ..., m$. An example of three genotypes and its possible resolving haplotype pairs is shown in Table 1.

HI studies do not deal with the SNPs over the entire genetic sequence at a time. Instead, the DNA sequence must be partitioned in such a way that the behavioral patterns of the genetic inheritance can be applied within them and, thus, accurate solutions from a biological point of view can be found. For the HIPP problem these regions are the ones between the hotspots (small regions with elevated recombination). It is known that the recombination process usually happens among the blocks of DNA between the hotspots, but not inside them. Thus, if we consider these blocks (and since a new mutation is very unlikely to happen at a given position) we can assume that the DNA blocks that an individual has are exactly the same as the ones its parents had. There-

[1]The notation used in this paper is not the standard one, but under it a genotype is explained by the sum, site by site, of its two resolving haplotypes. This representation was introduced by [19].

fore, if we look at a collection of individuals that are somehow related (for instance, members of a population) the number of haplotypes in such blocks will be much smaller than the number of possible ones. This observation and the fact that previous methods for HI found that their results were more likely correct when they returned small sets of haplotypes, were used by Gusfield to justify the HIPP [4] approach. The HIPP problem can be written as follows: Given a set of n genotypes $\mathcal{G} = \{g_1, ..., g_n\}$ find the smallest set of haplotypes $\mathcal{O}^* = \{h_1, ..., h_k\}$ such that for each genotype g in $\mathcal{G}$ there exist two haplotypes h_i, h_j in $\mathcal{O}^*$ such that $g = h_i \oplus h_j$.

The most widely used and referenced algorithm for the HIPP problem and one of the two that we have used for our experiments is the RTIP [4]. RTIP (*Reduced TIP*) is a more practical formulation of its first version TIP [4], which is actually considered a conceptual formulation. They both rewrite the HIPP problem as an integer linear program, including constraints for every different haplotype and haplotype pair which can participate in the resolution of the genotypes. Although the formulation is the same in both cases, the set of haplotypes, $\mathcal{O}$, and haplotype pairs considered by one and the other is different. Their formulation is shown in Fig. 1. The variables defined are the following: for each genotype g_i, its set of possible resolving haplotype pairs is denoted as $R_i = \{(j,k) : h_j \text{ and } h_k \text{ form an explaining pair for } g_i\}$. For each pair (j,k) in R_i they create a binary variable $w_{i,(j,k)}$ which is set to 1 iff the pair is selected to explain g_i. There is also a variable x_l for every distinct haplotype in the set $\mathcal{O}$ which is set to 1 iff it is used to explain a genotype. Constraint (1) in the model ensures that every genotype is explained by only one haplotype pair. Constraints (2) and (3) ensure that if $w_{i,(j,k)}$ is 1, and hence pair (h_j, h_k) is selected, then haplotypes h_j and h_k are in the set of chosen haplotypes. Finally, (4) ensures that the variables are binary. The objective function minimizes the total number of chosen haplotypes, which is the sum of the x_l variables. Therefore, constraint (1) generates n equations (where n is the number of genotypes) and constraint (2) generates two equations for each haplotype pair.

The RTIP algorithm searches among the set of resolving haplotype pairs, so before running the optimization algorithm it has to preprocess the genotype data by generating the set $\mathcal{O}$ of resolving pairs for each genotype. RTIP only considers pairs in which both haplotypes can participate in the resolution of any another genotype. Although this cut drastically decreases the number of possible haplotypes in the model, it still generates an exponential number of constraints in the general case. Gusfield [4] also introduced a procedure to generate this set of haplotypes in a time proportional to the length of the genotypes, m. This procedure consists of taking every pair of genotypes and looking for the haplotypes in the intersection, that is, those that can participate in the resolution of both genotypes at the same time.

The intersection of two genotypes is carried out as follows: Let g_i, g_j be two genotype vectors. To identify the haplotypes in $g_i \bigcap g_j$, both vectors must be scanned from left to right; if a site occurs with a 0 in one genotype and a 2 in the other, then the intersection is $\emptyset$; if a site occurs with a 1 in one vector and 0 or 2 in the other, then set that site to 0 if the site in the second vector is a 0 or to 1 if it is a 2. Then, if there are k remaining sites, where both g_i and g_j contain 1's, there are exactly 2^k distinct haplotypes in the intersection, and RTIP generates them by setting those k sites to 0 or 1 in every possible way. If, for example, we consider genotypes $g_a = 21101$ and $g_b = 00200$, there

$$\text{minimize} \sum_{l=1}^{|\mathcal{O}|} x_l$$

subject to

$$\sum_{(j,k)\in R_i} w_{i,(j,k)} = 1 \qquad \forall i \qquad (1)$$

$$x_j \geq w_{i,(j,k)} \qquad \forall i,j,k \qquad (2)$$

$$x_k \geq w_{i,(j,k)} \qquad \forall i,j,k \qquad (3)$$

$$x_l, w_{i,(j,k)} \in \{0,1\} \qquad \forall i,j,k,l \qquad (4)$$

Figure 1: TIP and RTIP formulation

is no haplotype that can solve both genotypes at the same time, because g_a requires its two resolving haplotypes to have a 1 in their first position and g_b requires its two resolving haplotypes to have a 0. Thus, their intersection is empty. Let us now consider genotypes $g_c = 22010$ and $g_d = 12100$. We can see that there exists one haplotype, which is 11000, that can solve both. For genotypes $g_e = 22110$ and $g_f = 11211$ the haplotypes that can explain both are 11100 and 11110.

The second algorithm used in our experiments is GAHAP [13], a heuristic method for the HIPP problem based on genetic algorithms. GAHAP represents an individual solution s as a length m binary vector (a bit array) where $s[i] = l$ represents that the haplotype pair chosen as an explanation for genotype g_i is the l^{th} pair in its resolution list. GAHAP also preprocesses the input genotype matrix before running the genetic algorithm. Its preprocess consists of that of RTIP with an additional cut which is as follows: Consider two genotypes g_i and g_j. Suppose g_i has two resolutions (h_1, h_2) and (h_4, h_5) and g_j has two resolutions (h_2, h_3) and (h_5, h_6). If h_1, h_3, h_4 and h_6 are only involved in the resolution of one genotype and h_2 and h_4 are involved in the resolution of two genotypes, then GAHAP only keeps pairs (h_1, h_2) and (h_2, h_3) and deletes pairs (h_4, h_5) and (h_5, h_6) (or equivalently keeps (h_4, h_5) and (h_5, h_6) and deletes (h_1, h_2) and (h_2, h_3)).

3 Set-based preprocess

In this section we present the main contribution of this paper, which is the set-based preprocess. As we have already stated, the preprocess consists of generating a list of possible resolving haplotype pairs for each genotype. Unfortunately, the number of resolving pairs for a genotype grows exponentially with respect to the number of ambiguous positions it has. Therefore, the larger the number of haplotypes is, the longer it will take the later applied solver to give a solution. Nevertheless, as we asserted in Section 1, not every haplotype is relevant to find the optimal solution, so in order to reduce the number of resolving pairs, these irrelevant haplotypes do not need to be considered. Gen-

erating every haplotype and then detecting and removing the irrelevant ones is, for medium-big size instances, an unaffordable task. Therefore, we have designed a procedure which deals with sets of haplotypes rather than individual haplotypes to generate just those that are relevant.

Once the initialization has been performed, the algorithm iterates a step where it looks for groups of irrelevant haplotypes. The haplotypes in such groups will not be part of the eventually generated haplotype pair lists. Therefore, once a group of irrelevant haplotypes is identified, it is not longer considered by the algorithm. The algorithm can run until no more haplotypes are found for elimination or another halting condition can be defined such as the time elapsed from the beginning.

3.1 Preliminaries

As already mentioned, haplotypes are represented as binary vectors and genotypes by ternary vectors. In order to deal with a large number of haplotypes, it is interesting to work with groups of them instead of individual haplotypes, thereby reducing the memory needed to store them. To represent sets of haplotypes we use *bases*, which are ternary vectors of 0's, 1's and question marks, '?'. The haplotypes included in a base are those having 0 at every position where the base has a 0, a 1 at every site where the base has a 1 and every combination of 1s and 0s at every site where the base has a '?'. Thus, the number of haplotypes included in a base A with p question marks is $|A| = 2^p$. If we consider for instance the base $A = 100??$, the haplotypes included in it are {10000, 10001, 10010, 10011}. Note also, that a base with no ambiguous position is a haplotype. By using this representation we avoid generating every haplotype and, therefore, we save computational space and time.

Since bases are sets of haplotypes, we can define the inclusion relation and the intersection for them.

Inclusion relation Given two bases A and B, we say that A is a subset of B if all the haplotypes in base A are also in base B and $A \neq B$. That happens, as summarized in (5), if every position of B with no ambiguous characters has the same value in A, every ambiguous position in A is also ambiguous in B and there is at least one ambiguous position in B that has not an ambiguous value in A. For instance, base $B = 10???$ is a superset of base $A = 1001?$ [2].

$$A \subset B \Leftrightarrow \begin{cases} \forall i \text{ s.t. } B[i] \neq ? \Rightarrow A[i] = B[i] \\ \forall i \text{ s.t. } A[i] = ? \Rightarrow B[i] = ? \\ \exists i \text{ s.t. } B[i] = ? \ \wedge \ A[i] \neq ? \end{cases} \tag{5}$$

Intersection Given two bases A and B, the haplotypes in the intersection are those that belong to both bases. Therefore, two bases have a non-empty intersection if a position in which one base happens to have a 1 and the other a 0 does not occur, i.e. $A \cap B \neq \emptyset \Leftrightarrow \forall i$ s.t. $A[i] \neq ? \wedge B[i] \neq ? \Rightarrow A[i] = B[i]$. In this case, the haplotypes in the intersection have a 1 at every position where any of the bases have a 1, a 0 at every position where any of the bases have a 0 and every combination of 0s and 1s where both

[2] Note that we define the inclusion relation as a proper inclusion.

bases have the ambiguous character. For example, for bases A=10??? and B=1? 00? the intersection is $A \cap B$=1000?.

In the same way as haplotypes, bases can also be paired in order to solve genotypes. Two bases A and B are complementary for a genotype g, denoted by $\gamma_g(A) = B$ or equivalently $\gamma_g(B) = A$, iff there exists a bijection between A and B such that for each haplotype $h_a \in A$ there exists a haplotype $h_b \in B$ such that $g = h_a \oplus h_b$. Note that if $\gamma_g(B) = A$, both A and B have the same number of ambiguities (and thus the same number of haplotypes) and in the same positions. Furthermore, given $h_a \in A$, the $h_b \in B$ such that $g = h_a \oplus h_b$ is built in the following way: if $i \in \{1, ...m\}$ is an index such that $A[i] = B[i] =?$ then $h_b[i] = 1 - h_a[i]$. For example, bases $A = 1?00?$ and $B = 1?10?$ are complementary for genotype $g = 21101$.

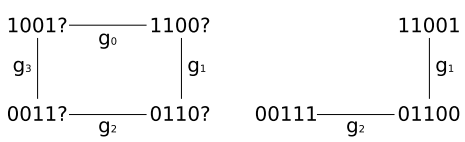
A function that has been widely used in the literature of the HI is the one called *cover*. This function associates to every haplotype the set of genotypes in whose resolution it can participate. This function can also be defined over the bases. The cover of a base A is the intersection of the cover of every haplotype included in A:

$$cover(A) = \bigcap_{h \in A} cover(h)$$

Our preprocessing algorithm considers several data structures in its execution. It considers two sets of bases, $\mathcal{H}$ and $\mathcal{D}$. The first structure, $\mathcal{H}$, contains the set of bases that the algorithm works with. This set is updated at each iteration by removing bases that contain irrelevant haplotypes. Set $\mathcal{D}$ contains the bases with non-irrelevant haplotypes. Every haplotype included in this set will be inserted into the haplotype pair lists generated at the end of the preprocess. Another important component of the algorithm is the *pair graph*, a graph $G = (V, E)$ which represents the complementary relation among the elements in $\mathcal{H}$. In this graph, the nodes V are the bases in $\mathcal{H}$ and there exists an edge between two nodes A and B iff there exists a genotype g such that $\gamma_g(A) = B$. Let us illustrate the above described graph with an example. Consider the genotypes and bases in Fig. 2a. The pair graph built using them has two connected components as shown in Fig. 2b and, as pointed out before, within each of them every base has the same number of ambiguous positions. We can also see that base 11001 (which has no ambiguous position and is actually a haplotype) is included in base 1100?. The cover of base 1001?, defined as the set of genotypes in whose it resolution can participate, is $cover(1001?) = \{g_0, g_3\}$.

Genotype	$\mathcal{H}$
g_0 : 21011	1001?
g_1 : 12101	1100?
g_2 : 01211	0110?
g_3 : 10121	0011?
	11001
	00111

(a)



(b)

Figure 2: Input list of genotypes, bases in $\mathcal{H}$ and the pair graph built with them

3.2 Initialization

The algorithm evolves by detecting and deleting groups of irrelevant haplotypes. In order to do that, it updates the set of bases $\mathcal{H}$ at each step by deleting bases which contain irrelevant haplotypes. The first step involves filling $\mathcal{H}$ with an initial set of bases. For this step we make use of an adaptation of the procedure introduced by Gusfield in his RTIP [4] (see Section 2). It consists of taking every pair of genotypes g_i, g_j and calculating the set of haplotypes that can solve both g_i and g_j, i.e. the intersection. If there exists at least one haplotype that can participate in the resolution of both g_i and g_j, their intersection can be written as a base, A_{ij}. In this case, the two complementary bases of A_{ij} are also calculated (one for each of the genotypes, $\gamma_{g_i}(A_{ij})$ and $\gamma_{g_j}(A_{ij})$) and the three bases are inserted into the set $\mathcal{H}$. The main difference between the original preprocess of RTIP and our adaptation is that RTIP explicitly generates every haplotype in the intersection and we generate the base which contains those haplotypes. Therefore the haplotypes are not explicitly generated reducing the computational space and time required by the algorithm. The set $\mathcal{D}$ is initially empty. To finish this initialization step, the pair graph is built using the initial set of bases in $\mathcal{H}$.

3.3 Elimination procedure

In this section we introduce the way in which bases are eliminated. We first introduce the steps of the algorithm, then illustrate it with an example and finally show that the deleted haplotypes are irrelevant. The pseudocode of this procedure is shown in Algorithm 1. The input for the elimination step consists of the set $\mathcal{H}$, the pair graph built with the bases in $\mathcal{H}$ and the set $\mathcal{D}$. The procedure outputs an updated set $\mathcal{H}$, the pair graph built with its bases and the updated set $\mathcal{D}$.

In the first three lines, the algorithm selects a set of bases to work with. The selected bases make a connected component in the pair graph and have the largest number of ambiguous positions of the bases in $\mathcal{H}$. If there is more than one connected component with the maximum number of ambiguous positions, the algorithm selects one randomly. These are maximal bases, which means that they are not included in any other base of $\mathcal{H}$.

Lines 4 to 11 calculate the intersections between every base in $\mathcal{C}$ and every maximal base which is not in $\mathcal{C}$, i.e. the bases in $\mathcal{F} \setminus \mathcal{C}$. For every non-empty intersection, the base containing the haplotypes in it is inserted into $\mathcal{H}$. The idea is to find bases whose haplotypes are involved in the resolution of as many genotypes as possible. Note that if haplotype h is included in the intersection of bases A and B, then h covers at least the same genotypes as both A and B.

Lines 12 to 22 insert into $\mathcal{H}$ the complementary bases of those that are maximal subsets of the bases in $\mathcal{C}$. In order to do that, for each base A in $\mathcal{C}$ we get its maximal subsets in $\mathcal{H}$, i.e. those bases $A' \subset A$ such that there is not $A'' \in \mathcal{H}$ and $A' \subset A'' \subset A$. Then, for each genotype g in the cover of A we calculate the complementary base of A' for g and insert it into $\mathcal{H}$. Note that once a new base is generated and introduced in $\mathcal{H}$, it can also be a new maximal subset for some base B in $\mathcal{C}$, so we will need to calculate its complementary bases. Therefore, this is a recursive process that finishes when the set is closed under the complementary operation. Given that our procedure works with maximal

Algorithm 1 Elimination procedure, set-based preprocess

1: Calculate $\mathcal{F} = \{A \,|\, A \in \mathcal{H} \wedge \nexists B \in \mathcal{H} \ s.t. \ A \subset B\}$ the set of maximal bases of $\mathcal{H}$
2: Let A be a base randomly drawn from $\{A \,|\, A \in \mathcal{F} \wedge |A| = max\{|B| \ |B \in \mathcal{F}\}\}$
3: Let $\mathcal{C}$ be the connected component of A in the pair-graph
4: **for all** $A_i \in \mathcal{C}$ **do**
5: **for all** $A_j \in \mathcal{F} \setminus \mathcal{C}$ **do**
6: $A_{ij} \leftarrow A_i \cap A_j$
7: **if** $A_{ij} \neq \emptyset$ **then**
8: Insert into set $\mathcal{H}$
9: **end if**
10: **end for**
11: **end for**
12: **while** $\{A' |\ A' \subset A \in \mathcal{C} \wedge \nexists A'' \in \mathcal{H} \ \ A' \subset A'' \subset A\}$ (the set of maximal subsets of the bases in $\mathcal{C}$) is not closed under the complementary operation **do**
13: **for all** $A \in \mathcal{C}$ **do**
14: **for all** $A' \subset A \ s.t. \ \nexists A'' \in \mathcal{H} \ \ A' \subset A'' \subset A$ **do**
15: **for all** $g \in cover(A)$ **do**
16: **if** $\gamma_g(A') \notin \mathcal{H}$ **then**
17: Generate and insert $\gamma_g(A')$ into set $\mathcal{H}$
18: **end if**
19: **end for**
20: **end for**
21: **end for**
22: **end while**
23: **if** $\nexists B \in \mathcal{H}$, $A \in \mathcal{C} \ \ s.t. \ B \subset A$ **then**
24: Add every base in $\mathcal{C}$ to $\mathcal{D}$
25: **end if**
26: Remove bases in $\mathcal{C}$ from $\mathcal{H}$
27: Calculate the pair graph $G = (V, E)$ with $V = \mathcal{H}$

bases and that the bases in $\mathcal{C}$ are deleted, this step is performed in order to keep the consistency in $\mathcal{H}$.

The last step, lines 23 to 26, removes the bases in $\mathcal{C}$ from $\mathcal{H}$. In the case that there is no base in $\mathcal{H}$ that is a subset of a base in $\mathcal{C}$ then the bases in $\mathcal{C}$ are introduced in $\mathcal{D}$. This last case means that the bases in $\mathcal{C}$ do not share any haplotype with any other base in $\mathcal{H}$ and can not be simplified. The bases included in the set $\mathcal{D}$ are not longer considered by the elimination procedure but they are not discarded by the algorithm. As we said in section 3.1, every haplotype in this set will be inserted into the eventually generated haplotype pair lists.

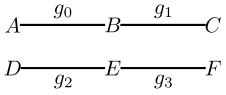
Now that we have described how the elimination procedure works, let us illustrate it with an example. Consider for this example the genotype matrix in Fig. 3a, and suppose that before an iteration of this procedure the current set of bases $\mathcal{H}$ and the pair graph built with them are those in Fig. 3b and 3c respectively. Assume also that the set of bases $\mathcal{D}$ is, at the beginning of this iteration, empty.

$g_0 : 122201$
$g_1 : 012201$
$g_2 : 001210$
$g_3 : 000110$

(a)

$A : 11110?$
$B : 01110?$
$C : 00110?$
$D : 0011?0$
$E : 0001?0$
$F : 0000?0$

(b)



(c)

Figure 3: Genotype matrix, bases in $\mathcal{H}$ and the pair graph built with them

In the first three lines the algorithm selects a set of bases to work with. In our example there are two connected components in the pair graph with the maximum number of ambiguous positions, which is 1, so suppose that we randomly choose the set $\{A, B, C\}$ as the connected component $\mathcal{C}$.

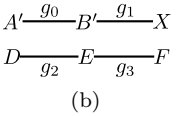
Lines 4 to 11 calculate the intersections between every base in $\mathcal{C}$ and every maximal base which is not in $\mathcal{C}$ and inserts them into $\mathcal{H}$. The algorithm finds just one non-empty intersection in these steps, $X = C \cap D = 001100$. Note that the cover of this base is $cover(X) = cover(C) \cup cover(D) = \{g_1, g_2\}$.

Lines 12 to 22 insert into $\mathcal{H}$ the complementary bases of those that are maximal subsets of the bases in $\mathcal{C}$. Here, the only maximal subset of the bases in $\mathcal{C}$ is X and therefore, these lines insert into $\mathcal{H}$ two new bases, B' (the complementary base of X for genotype g_1) and A' (the complementary base of B' for genotype g_0).

The last step, lines 23 to 26, removes the bases in $\mathcal{C}$ from $\mathcal{H}$. Since there exist bases in $\mathcal{H}$ that are included in bases of $\mathcal{C}$ ($A' \subset A$, $B' \subset B$ and $X \subset C$), the bases in $\mathcal{C}$ are not inserted into set $\mathcal{D}$. The output of this procedure consists of the updated set $\mathcal{H}$, the pair graph built with the bases in it and the set $\mathcal{D}$ (which has remain unchanged at this iteration). The set of bases $\mathcal{H}$ and the pair graph the procedure outputs are shown in Fig. 4.

$D : 0011?0$
$E : 0001?0$
$F : 0000?0$
$X : 001100$
$A' : 111100$
$B' : 011101$

(a)



(b)

Figure 4: Bases in $\mathcal{H}$ and the pair graph built with them after one iteration

Finally, we are going to prove that for a deleted haplotype h if there exists a solution S such that $h \in S$, then it is possible to build a new solution S' such that $h \notin S$ and S' has at most the same number of different haplotypes than S.

Before showing how to build this new solution, we should point out three facts about the deleted haplotypes. Let A be a base in the connected component $\mathcal{C}$. Suppose that at the end of an execution of the elimination procedure the bases in $\mathcal{C}$ are deleted from $\mathcal{H}$ without being inserted into $\mathcal{D}$.The first one is that for each base A in $\mathcal{C}$ not every haplotype is deleted, i.e. for each base $A \in \mathcal{C}$ there exists at the end of the elimination step a haplotype h such that $h \in A$ and $h \in B$ and $B \in \mathcal{H} \setminus \mathcal{C}$. This is clear as the set of maximal subsets of the

bases in $\mathcal{C}$ is not empty (since they are not inserted into $\mathcal{D}$, see line 23 in the algorithm) and we have closed this under the complementary operation.

The second fact is that if h is a removed haplotype at a given iteration, then all its complementary haplotypes for every genotype in its cover are also removed in the same iteration. In order to prove this statement, we must first see that if haplotype h is not removed at a given iteration then every haplotype for every genotype in its cover is not removed at the same iteration. For haplotype $h \in A$ not to be removed h must also be included in base B. If B is not a subset of A, lines 4 to 11 insert into $\mathcal{H}$ a base containing h. Let I be the maximal subset of A that contains h. The next step of the algorithm, lines 12 to 22, insert into $\mathcal{H}$ the complementary bases of every maximal subset of the bases in $\mathcal{C}$ until this set of bases is closed under the complementary operation. Since $\gamma_g(I)$ is now in $\mathcal{H} \setminus \mathcal{C}$, we know that haplotype $\gamma_g(h) \in \gamma_g(I)$ is not deleted. Moreover, the complementary relation is symmetric, so a haplotype is not removed if and only if its complementary haplotypes are not removed. Therefore, if a haplotype is removed, the complementary haplotypes for every genotype in its cover are removed.

The third fact to be considered about the deleted haplotypes is that their cover is equal to the cover of the base to which they belong. In an iteration of the elimination procedure a connected component of the pair graph is deleted. Thus, the complementary haplotype of a deleted haplotype must be in the same connected component, so it can not solve more genotypes than the base to which it belongs.

Considering these properties, it is possible to create a new solution S' as follows: we can delete haplotype $h \in A$ that solves genotype g in solution S and introduce a new haplotype $h_r \in A$ such that there exists $B \in \mathcal{F} \setminus \mathcal{C} \wedge h_r \in B$, i.e. h_r is not deleted at the current iteration. Clearly, if h appears in solution S solving genotype g, then $h' = \gamma_g(h) \in S$. Also, as we have already stated, if h is a deleted haplotype then $h' = \gamma_g(h)$ for all $h \in cover(A)$ is also deleted. Hence, we delete this haplotype h' from S and introduce $h'_r = \gamma_g(h_r)$ the complementary haplotype of the new introduced h_r for genotype g in S'. Of course, like as h', h'_r belongs to base $A' = \gamma_g(A)$. This argument could be recursively applied to h'_r and the process repeated until none of the deleted haplotypes from $\mathcal{H}$ at this iteration is in the new solution. Therefore, the new solution S' does not contain deleted haplotypes and has the same number of haplotypes as S.

3.4 The algorithm

In this section we describe the whole algorithm by combining the steps described in the previous sections. [3]

As shown in Algorithm 2, after the initial step has been performed, the elimination procedure is applied until a halting condition is reached. This halting condition can be defined in several terms such as the elapsed time from the beginning.

As previously mentioned, the output of the preprocess is a haplotype pair list associated to each genotype in the input. Therefore, the last step involves using the haplotypes in $\mathcal{H}$ and $\mathcal{D}$ to fill those lists. In order to do this, every

[3] An executable version of this algorithm is available at `http://www.sc.ehu.es/ccwbayes/members/ekhine/home/index.html`.

Algorithm 2 Set-based preprocess

1: Initialization
2: **while** ¬ halting condition reached **do**
3: Apply elimination procedure
4: **end while**
5: Generate and insert every haplotype in $\mathcal{H}$ into the pair lists

haplotype in each base is generated and inserted into the pair lists of every genotype in its cover, paired with its complementary in each case.

Let us consider the example in the previous section. Assuming that the step performed in that example was the last one before the halting condition was reached, the generated haplotypes are those shown in Fig. 5a and the resulting pair list is that in Fig. 5b.

$D : 0011?0 = \{X, D'\} = \{001100, 001110\}$
$E : 0001?0 = \{E', E''\} = \{000100, 000110\}$
$F : 0000?0 = \{F', F''\} = \{000000, 000010\}$
$X : 001100$
$A' : 111100$
$B' : 011101$

(a)

$g_0 : (A', B')$
$g_1 : (B', X)$
$g_2 : (X, E''), (D', E')$
$g_3 : (E', F''), (E'', F')$

(b)

Figure 5: Final set of haplotypes and resolving pairs for the genotypes

4 Experiments

4.1 Data sets

In order to evaluate the performance of our preprocessing algorithm, we have used synthetic and real haplotype inference problem instances. Synthetic instances have been obtained using the ms program [20] and the real data have been obtained from the International HapMap project [18]. The instances are divided into two different datasets that we have called *classical* and *new HapMap dataset*.

For every instance the duplicate rows and columns are removed before running the preprocess. In order to clearly present the results and due to the large number of instances, we have grouped them and report their average results. In order to group them, we have taken into account the average number of ambiguous positions per row and column the input genotype matrices have, parameters that we have called k_avg and l_avg respectively. These definitions are inspired by the characterization of the instances given in [5]. Its authors refer to an input genotype matrix as a *(k, l)-bounded instance* when it has at most k ambiguous positions per row and l ambiguous positions per column and they prove that the HIPP problem is NP-hard even for (4, 3)-bounded instances. In addition, we will see that this grouping has a nice property that helps to analyze the results. We have set the bounding values of k_avg and l_avg to make the groups in such a way that in a dataset all the groups have a similar number

of instances. The results are graphically presented. The bounding values are shown at the bottom of each figure, under each group. In order to know which group an instance belongs to, one must look for the first group, starting form the left, which matches its k_avg and l_avg values.

4.1.1 Classical dataset

This first dataset is a kind of standard in the HI literature, having been used by, among others, [10], [14] and [16]. It contains both simulated and real instances. The instances are divided into three groups[4]:

Uniform instances. The genotypes in these instances are synthetic and were generated using the ms software as follows: Generate a set of haplotypes, usually less than $2n$, remove the repeated ones and randomly select two haplotypes to generate each of the n genotypes [5]. There are 200 uniform instances, 110 of them generated under the assumption of no recombination, 30 assuming recombination level 4, another 30 with recombination level 16 and the last 30 assuming recombination level 40. These instances contain between 8 and 49 genotypes of a length which varies between 4 and 29 SNPs.

Non-uniform instances. The genotypes in these instances are also simulated and were generated as follows: Generate a set of $2n$ haplotypes with the ms software and randomly select two haplotypes (without removing the repeated ones) to generate each of the n genotypes. There are 90 non-uniform instances in the dataset. The instances in this dataset contain between 6 and 36 genotypes and between 9 and 46 SNPs.

HapMap instances. There are 24 real instances obtained from the HapMap project. These HapMap instances contain between 6 and 30 genotypes and between 5 and 29 SNPs.

4.1.2 New HapMap dataset

This second dataset only contains real data obtained from the HapMap project [18]. The whole sequence of chromosome 1 of the 95 individuals whose data is available was split by the hotspots because, as already stated, the recombination process is likely to happen among the blocks of DNA between the hotspots [18], but not inside them. The genotypes containing missing data were removed. For these instances we have also removed the duplicate and complementary rows and columns before running the preprocess, obtaining instances with a different number of SNPs and genotypes. Once this cut has been performed we obtained 2119 instances, with at most 63 genotypes and 56 SNPs.

4.2 Compared algorithms and parameters

In order to test our preprocessing procedure, we have combined it with two HIPP solvers, RTIP [4] and GAHAP [13]. They are both described in section 2. In the current section we compare both original algorithms, RTIP and GAHAP,

[4] These instances were kindly provided by Dan G. Brown and Ian M. Harrower

[5] Personal communication, Dan G. Brown and Ian M. Harrower

with their modified versions, SB-TIP and SB-GAHAP, which make use of our set-based preprocess instead of their original preprocess.

The preprocesses (that of RTIP and the set-based) were implemented in Java. The linear programming solver was GLPK, the GNU linear programming kit. RTIP and SB-TIP algorithms were executed on a different platform to GAHAP and SB-GAHAP. Therefore, the computational time of GAHAP and RTIP cannot be compared, as well as SB-TIP and SB-GAHAP.

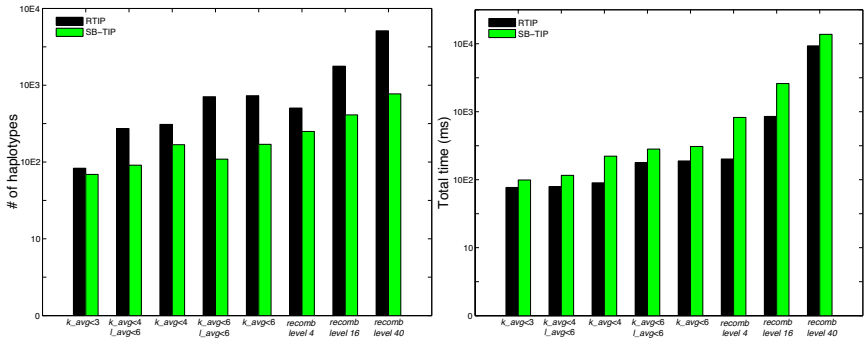
The halting condition of the set-based preprocess can be defined in several ways. For the experiments with the new HapMap dataset, we have halted each instance at five different moments using five different criteria. These criteria take into account the elapsed time and also the size of the stored bases in $\mathcal{H}$: The algorithm stops when the biggest base in the set $\mathcal{H}$ has 4, 3, 2, 1 and 0 ambiguous positions (this last condition means that the set of bases $\mathcal{H}$ is empty while $\mathcal{D}$ is not) or when the time limit has been exceeded, being this limit 1000 seconds. From now on, we will refer to these criteria as *EL4*, *EL3*, *EL2*, *EL1* and *EL0*. For the classical dataset the halting condition of the preprocess was defined in terms of computational time, being run with a time limit of 1000 seconds. We have used only this halting condition because the small size of these instances causes the differences among these criteria to be almost imperceptible. The time limit for the IP solver and the genetic algorithm is also 1000 seconds. Algorithms GAHAP and SB-GAHAP halt when one of these three conditions is met: (1) the 1000 seconds time limit is exceeded, (2) the number of generations is equal to 150 or (3) the optimal value is reached (the optimal value is obtained from the execution of SB-TIP). Following the recommendations of the authors, the population size was set to 150 for the instances in the classical dataset and to 600 for every instance in the new HapMap dataset.

4.3 Results

In this section we compare, on the one hand, the results of RTIP and SB-TIP and, on the other hand, the ones of GAHAP and SB-GAHAP. The comparisons are made in terms of haplotypes generated by the preprocessing procedures and execution time. As a general comment we will show that, as the size of the instances grow (i.e. for bigger values of k_avg and l_avg) the instances preprocessed with the set-based are solved much faster. However, for small size instances it is not always worth preprocessing the instances with our proposed procedure.

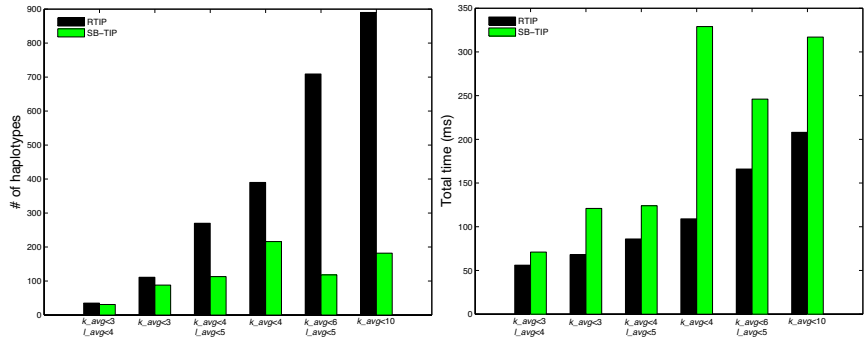
4.3.1 Classical dataset

Before analyzing the results in detail it is important to take into account the following facts. As we have already stated, GAHAP preprocesses the input genotype matrix before running the genetic algorithm using the RTIP preprocess and an additional cut. Therefore, the number of haplotypes generated by GAHAP is always less than or equal to the number of haplotypes generated by RTIP for every instance. This cut is implicitly performed by the set-based preprocess, so the number of haplotypes generated by the set-based preprocess when run until the final stage ($EL0$) is always less than or equal to the number of haplotypes generated by GAHAP. GAHAP needs to generate the haplotypes and then test if they match the requirement of the cut, while the set-based only



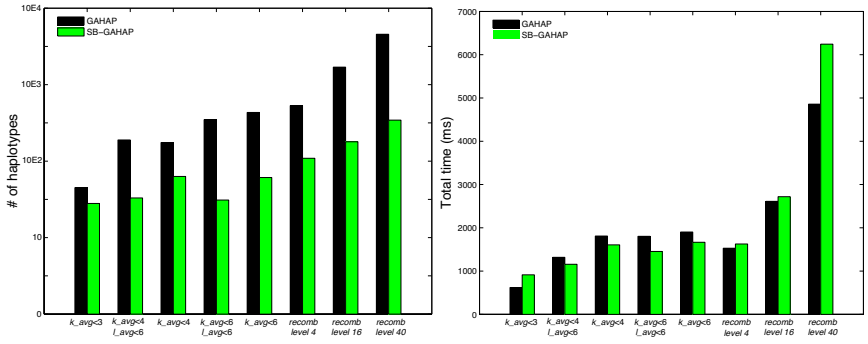
(a) (b)

Figure 6: Haplotypes generated (a) and execution time (b) for the uniform instances in the classical dataset with RTIP and SB-TIP



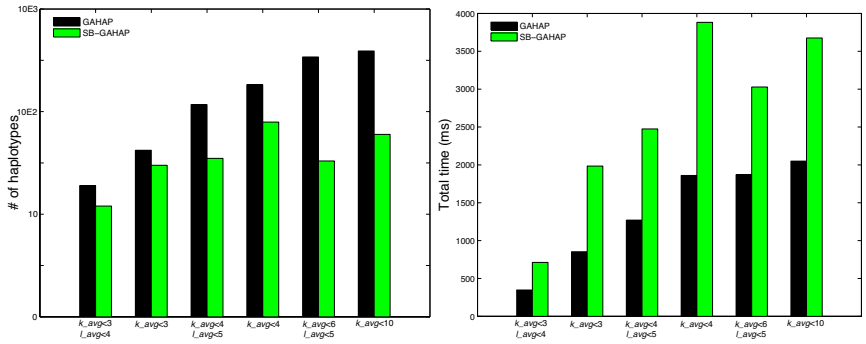
(a) (b)

Figure 7: Haplotypes generated (a) and execution time (b) for the non-uniform instances in the classical dataset with RTIP and SB-TIP



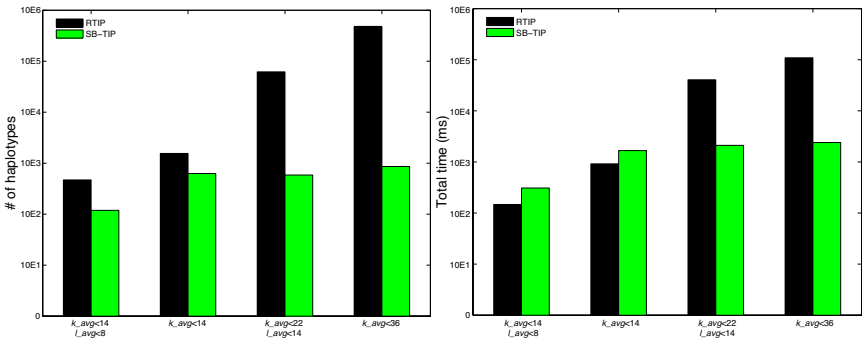
(a) (b)

Figure 8: Haplotypes generated (a) and execution time (b) for the uniform instances in the classical dataset with GAHAP and SB-GAHAP



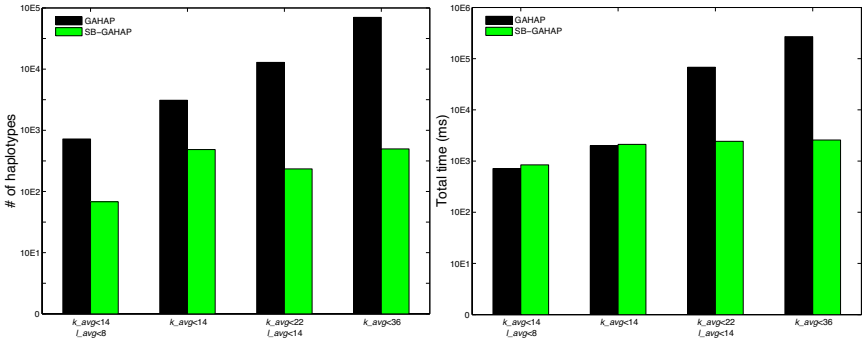
(a) (b)

Figure 9: Haplotypes generated (a) and execution time (b) for the non-uniform instances in the classical dataset with GAHAP and SB-GAHAP



(a) (b)

Figure 10: Haplotypes generated (a) and execution time (b) for the HapMap instances in the classical dataset with RTIP and SB-TIP



(a) (b)

Figure 11: Haplotypes generated (a) and execution time (b) for the HapMap instances in the classical dataset with GAHAP and SB-GAHAP

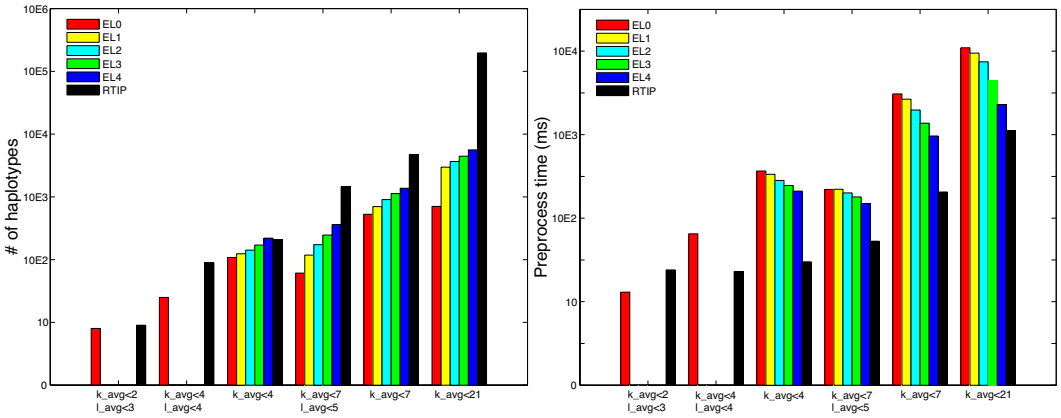
generates haplotypes that do not match the cut prerequisite. Note also that the number of haplotypes generated by SB-TIP and SB-GAHAP are not the same although the applied preprocess is the same. This is because GAHAP does not consider haplotypes that cover only one genotype, if they are ever needed they can be easily calculated (remember that under this notation a genotype is the sum of its two resolving haplotypes). Therefore, when preprocessing the data to which the GAHAP algorithm is going to be applied, our preprocessing procedure does not consider those haplotypes either.

In this section we show the haplotypes generated and the required time to preprocess the instances in the classical dataset. These are the smallest groups of instances and are quickly and correctly solved. Figs. 6a and 7a show the number of haplotypes generated by the RTIP preprocess and the set-based preprocess for the uniform and non-uniform instances respectively. Figs. 8a and 9a also show the number of haplotypes generated by GAHAP and SB-GAHAP for the same instances. When the set-based preprocess is used the number of haplotypes generated is smaller and therefore the time required by the IP and GAHAP solvers decreases. Despite this fact, the whole execution, preprocess and process, when using the set-based preprocess takes a little longer for these instances, as shown in Figs. 6b, 8b, 7b and 9b. The instances generated under recombination are the worst case. Although for these instances the number of haplotypes is vastly reduced when applying our proposed preprocessing procedure, the large number of non-empty intersections found during the execution due to the recombination, causes the preprocess time to increase.

The HapMap instances are the hardest ones in this classical dataset. The instances that have been preprocessed with the our proposed procedure are clearly solved faster as k_avg and l_avg grow, as shown in Figs. 10b and 11b. Fig. 10 compares RTIP and SB-TIP, the haplotypes generated in Fig. 10a, and the total time in Fig. 10b. We should point out that there is one instance that not one of them could preprocess in the given time limit. There also exists another instance which is correctly preprocessed using SB-TIP but could not be preprocessed when using RTIP due to memory overflow. Also, due to the large number of constraints needed to write the instances as an IP there are other three instances that could not be solved when using the RTIP preprocess that are correctly solved with SB-TIP. The situation is quite similar when comparing GAHAP and SB-GAHAP, as shown in Fig. 11. There also are three instances that GAHAP does not solve in the given time limit that SB-GAHAP correctly solves and one that none of them solve.

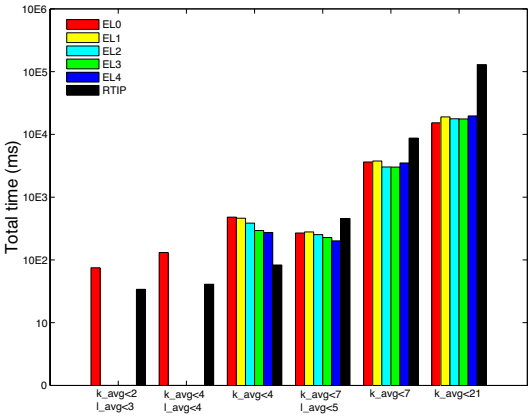
4.3.2 New HapMap dataset

In this section we give the experimental results of the described algorithms over the new HapMap dataset. It is worth noticing that for this dataset not every bar in each group of each figure has the same number of instances. This can happen for two different reasons. The first one is that there are instances that are correctly solved within the time limit when using our preprocess that did not finish when using the original algorithms, RTIP and GAHAP. Those instances will likely generate more haplotypes than the average, and therefore, this average will grow. This is due to the larger size of the instances in this dataset. Another cause is that small instances cannot be halted with some of our proposed criteria such as $EL4$ or $EL3$ because just after the initialization



(a) Haplotypes generated

(b) Preprocess time

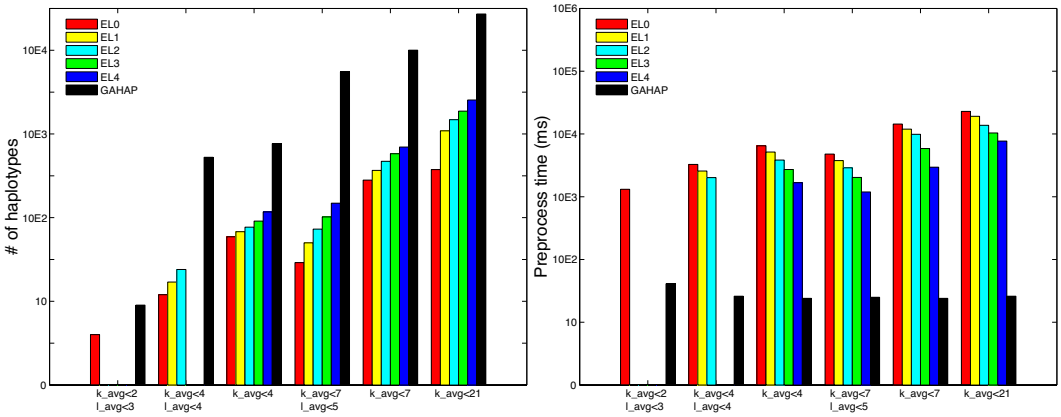


(c) Total time, preprocess + process

Figure 12: Results of RTIP and SB-TIP for the instances in the new HapMap dataset

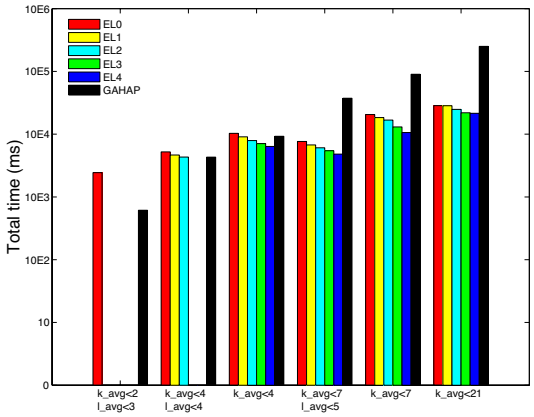
procedure has been performed the biggest base in $\mathcal{H}$ has less than 4 (respectively 3) ambiguous positions.

In Fig. 12a and 13a we can see how, for every group of instances, the number of haplotypes generated decreases when the set-based preprocess runs until the final stages. Fig. 12b shows the time required by the preprocess for RTIP and SB-TIP and Fig. 13b the required time by GAHAP and SB-GAHAP. In both cases one can see how the required time grows when the set-based preprocess is run until the final stages. The number of haplotypes generated is vastly reduced when using our preprocess and, when the applied process is TIP, so does the number of constraints in the IP formulation. Consequently, for instances with high values of k_avg and l_avg there is a dramatical decrease in the process time. This decrease causes a reduction of the total time (preprocess + process) required for solving an instance when it is preprocessed using the set-based procedure, as shown in Fig. 12c and 13c. Moreover, comparing RTIP and SB-TIP, 154 instances could not be solved when the applied preprocess was RTIP



(a) Haplotypes generated

(b) Preprocess time



(c) Total time, preprocess + process

Figure 13: Results of GAHAP and SB-GAHAP for the instances in the new HapMap dataset

while only five could not be solved with any of the five halting conditions defined for the set-based preprocess in the given time limit. On the other hand, 430 instances that were not solved by the original GAHAP were solved in the time limit when solved with our modification SB-GAHAP.

Fig. 14 and 15 show a comparison in terms of total execution time of the four algorithms. Each instance is plotted with a plus or a circular symbol depending whether it is faster solved (including preprocess and process) with the original algorithm (RTIP or GAHAP) or with the modified algorithm including the set-based preprocess (SB-TIP or SB-GAHAP).

Given that many instances have the same values of k_avg and l_avg and in order to better represent the results, each instance has been plotted in ($k\_avg+\epsilon$, $l\_avg+\gamma$) where the ϵ and γ parameters are random numbers sampled from a uniform distribution $(0, \frac{2}{5})$.

Fig. 14 compares the SB-TIP with RTIP. For almost every instance with $k\_avg > 5$, the best performance is obtained when the applied preprocess is our

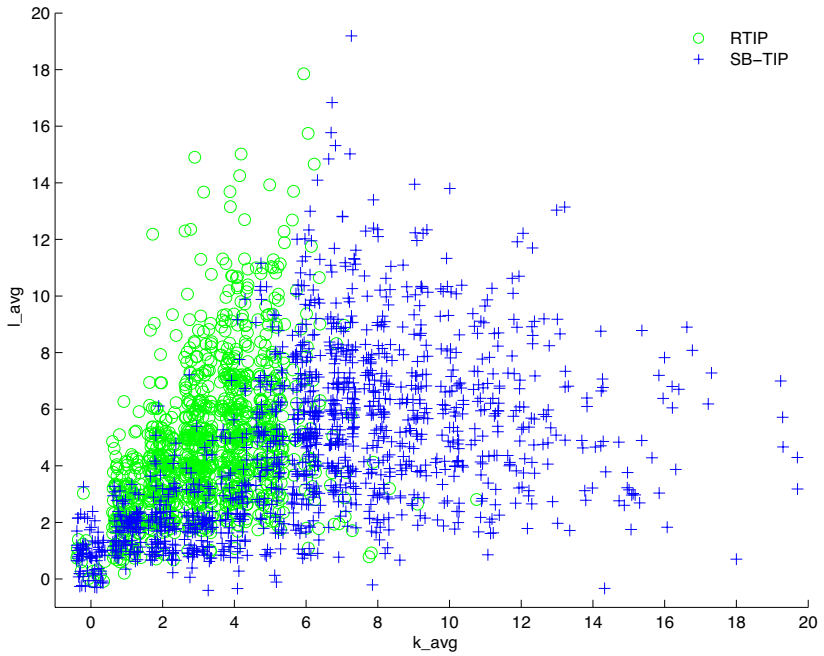


Figure 14: The plot shows, for each instance, the fastest algorithm, RTIP or SB-TIP

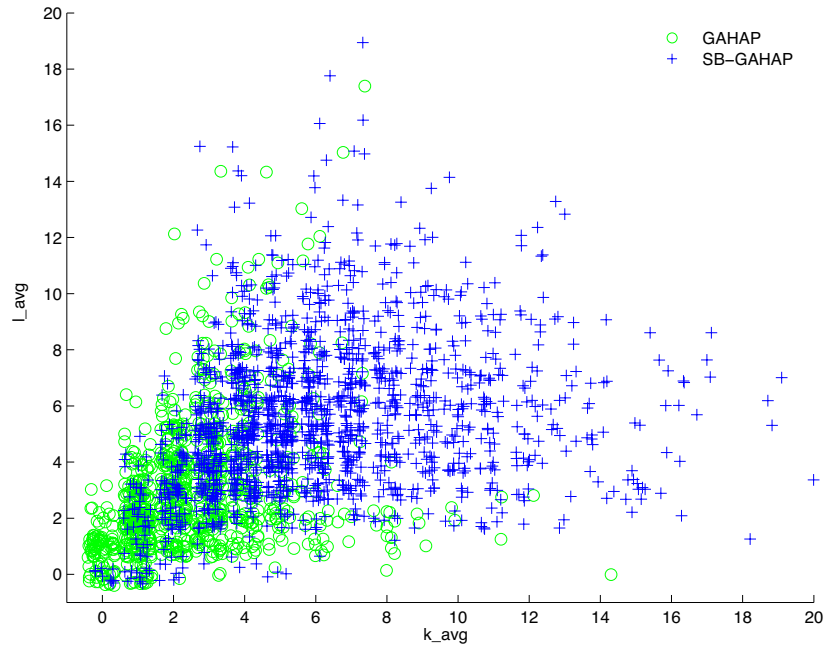


Figure 15: The plot shows, for each instance, the fastest algorithm, GAHAP or SB-GAHAP

proposed procedure. For these big instances it often happens that by performing few iterations of the elimination procedure a large number of haplotypes is discarded. On the other hand, RTIP generates them all, one by one, usually taking longer than the set-based preprocess. In extreme cases, RTIP does not even finish the preprocess in the given time limit. Moreover, even if it does finish, RTIP will generate a larger number of constraints than the set-based preprocess. Therefore, it will likely take the IP solver longer to solve an instance that has been preprocessed with RTIP. For instances with $k\_avg \leq 5$ and $l\_avg < 12$ both RTIP and SB-TIP show quite a similar performance. On the other hand, we can see that for the instances with $k\_avg \leq 5$ and $l\_avg \geq 12$ RTIP is faster than SB-TIP for most instances. These instances that are faster solved with RTIP contain a large number of genotypes (as a consequence of their large l_avg) of relatively few SNPs. Therefore, the number of intersections found during the initialization step is likely to be high. This implies that the algorithm deals with a large number of bases, but each containing relatively few haplotypes. The reduction in the number of haplotypes is not as significant as in the rest of the instances. Thus, the extra time consumed by the set-based preprocess compared to that of RTIP is much higher than the time saved when processing the instances.

Fig. 15 compares the SB-GAHAP with GAHAP. The plot is quite similar to the one in the previous figure. The instances with smaller values of k_avg or l_avg are faster solved with GAHAP while that with larger values of k_avg and l_avg are faster solved when the applied preprocess is the set-based.

The results of the tests of chromosome 1 are available at `http://www.sc.ehu.es/ccwbayes/members/ekhine/home/index.html`. In addition, the results of the 22 autosomal chromosomes will also be published.

5 Conclusions

In this paper we have introduced a new preprocessing procedure which can be coupled with several existing solvers for the haplotype inference by pure parsimony problem. This procedure, which we have called set-based preprocess, deals with sets of haplotypes instead of dealing with single haplotypes. It iterates searching and deleting groups of haplotypes such that, for every solution that includes haplotypes of these groups, another solution can be built that does not include these haplotypes and has, at most, the same number of different haplotypes. The use of groups of haplotypes instead of individual haplotypes reduces the memory needed to represent the initial set of haplotypes. The halting condition can be defined in several ways so one can adapt its behavior for different kinds of problems. The number of haplotypes generated by the set-based preprocess when run until the final stages is always, less than or equal to the number of haplotypes generated by any of the current preprocessing algorithms, usually being much smaller.

We have tested our procedure combining it with two existing and well known solvers for the HIPP problem using simulated and real data. These experiments showed the efficiency of this procedure in terms of the number of haplotypes generated and time performance, especially for instances with a large number of SNPs, it being possible to preprocess and process instances that were previously unaffordable in a reasonable time. Moreover, n set a lower limit for the number

of ambiguous positions per row and column for which it is worth using our proposed preprocessing procedure since the extra time consumed by it is going to be saved when processing the data. For these big instances it is usually not worth running the set-based preprocess until the final stages: the extra computational time required for the latest steps is higher than the time saved by the later applied solver with a reduced number of haplotypes. Finally, we have shown that the reduction in computational resources required when using our preprocess coupled with a state of the art solver allows us to solve HapMap problem instances that were previously unaffordable for the same solvers.

Acknowledgments

The authors are very grateful to Dan G. Brown and Ian M. Harrower for sending us their instances and to Rui-Sheng Wang for providing us with the source code of the GAHAP solver. This work has been partially supported by the Saiotek and Research Groups 2007-2012 (IT-242-07) programs (Basque Government), TIN2008-06815-C02-01 and Consolider Ingenio 2010 - CSD2007-00018 projects (Spanish Ministry of Science and Innovation) and COMBIOMED network in computational biomedicine (Carlos III Health Institute). Ekhine Irurozki holds the grant BES-2009-029143 from the Spanish Ministerio de Ciencia e Innovacion.

References

[1] P. Bonizzoni, G. Vedova, R. Dondi, and J. Li, "The haplotyping problem: An overview of computational models and solutions," *Journal of Computer Science and Technology*, vol. 18, pp. 675–688, noviembre 2003.

[2] D. Gusfield, "An overview of combinatiorial methods for haplotype inference," *Lecture Notes in Computer Science, Computational Methods for SNPs and Haplotype Inference, DIMACS/RECOMB Satellite Workshop*, vol. 2983/2004, pp. 599–600, 2004.

[3] G. Lancia, M. C. Pinotti, and R. Rizzi, "Haplotyping populations by pure parsimony. Complexity, exact and approximation algorithms," *INFORMS Journal on Computing*, vol. 16, pp. 348–359, 2004.

[4] D. Gusfield, "Haplotype inference by pure parsimony," *Combinatorial Pattern Matching: 14th Annual Symposium*, vol. 2676/2003, pp. 144–155, 2003.

[5] R. Sharan, B. V. Halldorsson, and S. Istrail, "Islands of tractability for parsimony haplotyping," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 3, no. 3, pp. 303–311, 2006.

[6] S. Climer, A. R. Templeton, and W. Zhang, "How frugal is mother nature with haplotypes?" *Bioinformatics*, vol. 25, pp. 68–74, 2009.

[7] D. Catanzaro, A. Godi, and M. Labbe, "A Class Representative Model for Pure Parsimony Haplotyping," *INFORMS Journal on Computing*, 2010. (Accepted)

[8] E. Erdem and F. Türe, "Efficient haplotype inference with answer set programming," in *AAAI'08: Proceedings of the 23rd national conference on Artificial intelligence*. AAAI Press, 2008, pp. 436–441.

[9] J. Neigenfind, G. Gyetvai, R. Basekow, S. Diehl, U. Achenbach, C. Gebhardt, J. Selbig, and B. Kersten, "Haplotype inference from unphased SNP data in heterozygous polyploids based on SAT," *BMC Genomics*, vol. 9, no. 1, p. 356, 2008.

[10] D. G. Brown and I. M. Harrower, "Integer programming approaches to haplotype inference by pure parsimony," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 3, no. 2, pp. 141–154, 2006.

[11] B. Halldorsson, V. Vafna, N. Edwards, R. Lippert, S. Yooseph, and S. Istrail, "A survey of computational methods for determining haplotypes," *Lecture Notes in Computer Science, Computational Methods for SNPs and Haplotype Inference, DIMACS/RECOMB Satellite Workshop*, vol. 2983/2004, pp. 26–47, 2004.

[12] L. Wang and Y. Xu, "Haplotype inference by maximum parsimony," *Bioinformatics*, vol. 19, pp. 1773–1780, 2003.

[13] R.-S. Wang, X.-S. Zhang, and L. Sheng, "Haplotype inference by pure parsimony via genetic algorithm," in *International Symposium on OR and Its Applications*, 2005, pp. 171–184.

[14] L. Di Gaspero and A. Roli, "Stochastic local search for large-scale instances of the haplotype inference problem by pure parsimony," *Journal of Algorithms*, vol. 63, no. 1-3, pp. 55–69, 2008.

[15] I. Lynce and a. Marques-Silva, Jo "Efficient haplotype inference with boolean satisfiability," in *AAAI'06: Proceedings of the 21st national conference on Artificial intelligence*. AAAI Press, 2006, pp. 104–109.

[16] A. Graça, J. Marques-Silva, I. Lynce, and A. L. Oliveira, "Efficient haplotype inference with pseudo-boolean optimization," *Algebraic Biology*, vol. 4545/2007, pp. 125–139, 2007.

[17] A. Graça, J. Marques-Silva, I. Lynce, and A. L. Oliveira, "Efficient haplotype inference with combined cp and or techniques," in *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 2008, pp. 308–312.

[18] The International HapMap Consortium, "A second generation human haplotype map of over 3.1 million SNPs," *Nature*, vol. 449, no. 7164, pp. 851–861, 2007.

[19] J. He and A. Zelikovsky, "Linear reduction for haplotype inference," *Lecture Notes in Bioinformatics, Algorithms in Bioinformatics*, vol. 3240/2004, pp. 242–253, 2004.

[20] R. R. Hudson, "Generating samples under a wright-fisher neutral model of genetic variation." *Bioinformatics*, vol. 18, no. 2, pp. 337–338, 2002.