Alma Mater Studiorum
Università degli Studi di Bologna
DEIS

Genetic Master-Slave Algorithm for Haplotype Inference by Parsimony

Stefano Benedettini Luca Di Gaspero
Andrea Roli

January 10, 2009

DEIS Technical Report no. DEIS-LIA-003-09 *LIA Series no. 93*

Genetic Master-Slave Algorithm for Haplotype Inference by Parsimony

Stefano Benedettini [1] Luca Di Gaspero [2] Andrea Roli [1]

[1] *DEIS, Campus of Cesena*
Alma Mater Studiorum *Università di Bologna*
via Venezia 52, I-47023 Cesena, Italy
{stefano.benedettini|andrea.roli}@unibo.it
[2] *DIEGM, Università di Udine*
via delle Scienze 208, I-33100, Udine, Italy
l.digaspero@uniud.it

January 10, 2009

Abstract. Haplotype Inference is a challenging problem in bioinformatics that consists in inferring the basic genetic constitution of diploid organisms on the basis of their genotypes. This piece of information makes it possible to perform association studies for the genetic variants involved in multifactorial diseases and the individual responses to therapeutic agents. A notable approach to the problem is to encode it as a combinatorial problem (under certain hypotheses, such as the *pure parsimony* criterion) and to solve it using combinatorial optimization techniques. Recently, several new approaches to the problem were presented. Among them, solvers based on hybrid metaheuristics have been proven to be effective in solving large size instances. In this paper, we present an master-slave hybrid approach, in which a master solver optimize the parameters used by a slave solver for constructing a solution. By testing the algorithm on common Haplotype Inference benchmarks, we show that this approach can produce good quality solutions in a very short execution time.

Keywords: *Metaheuristics, haplotype inference, bioinformatics*

Contents

1 Introduction

The role of genetic variation and inheritance in human diseases is extremely important, though still largely unknown [21]. To this aim, the assessment of a full Haplotype Map of the human genome has become one of the current high priority tasks of human genomics [20]. Diploid organisms, such as mammals, have their genetic material arranged in pairs of chromosomes, one inherited from the father and the other from the mother. A haplotype is one of the two non identical copies of a chromosome of a diploid organism. The information conveyed by haplotypes allows to perform association studies for the genetic variants involved in diseases and the individual responses to therapeutic agents. The most important DNA variations considered by biologists are the *Single Nucleotide Polymorphisms* (SNPs), which occur when a nucleotide in the DNA sequence is replaced by another one. Technological limitations make it currently impractical to directly collect haplotypes by experimental procedures, but it is possible to collect *genotypes*, i.e., the conflation of pairs of haplotypes, in which the origins of SNPs can not be distinguished. Therefore, haplotypes have to be inferred from genotypes in order to reconstruct the detailed information and trace the precise structure of DNA variations in a population. This process is called *Haplotype Inference* (also known as *haplotype phasing*) and the goal is to find a set of haplotype pairs (i.e., a *phasing*) so that all the given genotypes are *resolved*, that is, they can be obtained by combining a pair of haplotypes from the set.

The main methods to tackle the Haplotype Inference are either combinatorial or statistical. Both, however, being of non-experimental nature, need some genetic model that could provide criteria for evaluating the solution returned with respect to actual genetic plausibility. In the case of the combinatorial methods, which are the subject of the present work, one of the most often used criterion is *pure parsimony*. The *pure parsimony* criterion [11] suggests to search for the smallest collection of distinct haplotypes that solves the Haplotype Inference problem. This criterion is consistent with current observations in natural populations for which the actual number of haplotypes is vastly smaller than the total number of possible haplotypes.

In this paper, we present a hybrid metaheuristic [4] technique to tackle the Haplotype Inference by pure parsimony. The method is a master-slave solver in which the master tries to optimize the parameters used by the slave to construct the solution.

The remainder of this paper is structured as follows. In Section 2 we briefly summarize the state of the art of Haplotype Inference solvers in the case of pure parsimony. In Section 3 we formally define the problem and its related main concepts. The master-slave solver is described in Section 4 and in Section 5 an experimental analysis on common benchmarks is illustrated. Finally, we conclude with Section 6 and we summarize ongoing and future work.

2 Related work

Current approaches for solving the Haplotype Inference problem by parsimony include simple greedy heuristics [6] and exact methods, such as Integer Linear Programming [5, 11, 12, 17], Semidefinite Programming [15, 16], SAT models [18, 19] and Pseudo-Boolean Optimization algorithms [10]. These approaches, however, at present seem not to be particularly adequate for very-large size instances.

Metaheuristic and hybrid approaches have been also proposed, trying to provide better scalability than exact approaches. To the best of our knowledge, the first attempt to employ metaheuristic techniques for the problem is a Genetic Algorithm [22]. However, in [22] results on real size instances are not reported. Recently, new metaheuristic approaches have been proposed. Stochastic local search [7], Ant colony optimization [2] and hybrid metaheuristic techniques [1] have been proven to efficiently solve large size instances. Among them, the hybrid approach dominates the other methods as it effectively combines the strengths of its components.

3 The Haplotype Inference problem

In the Haplotype Inference problem we deal with *genotypes*, that is, strings of length m that correspond to a chromosome with m sites. Each value in the string belongs to the alphabet $\{0, 1, 2\}$. A position in the genotype is associated with a site of interest on the chromosome and it has value 0 (wild type) or 1 (mutant) if the corresponding chromosome site is a homozygous site (i.e., it has that state on both copies) or the value 2 if the chromosome site is heterozygous. A *haplotype* is a string of length m that corresponds to only one copy of the chromosome (in diploid organisms) and whose positions can assume the symbols 0 or 1.

3.1 Genotype resolution

Given a chromosome, we are interested in finding an unordered[1] pair of haplotypes that can explain the chromosome according to the following definition:

Definition 1 (Genotype resolution)**.** Given a chromosome g, we say that the unordered pair $\langle h, k\rangle$ *resolves* g, and we write $\langle h, k\rangle \rhd g$ (or $g = h \oplus k$), if the following conditions hold (for $j = 1, \dots, m$):

$$\begin{aligned} g[j] = 0 \Rightarrow &\quad h[j] = 0 \wedge k[j] = 0 && \text{(1a)}\\ g[j] = 1 \Rightarrow &\quad h[j] = 1 \wedge k[j] = 1 && \text{(1b)}\\ g[j] = 2 \Rightarrow &\quad (h[j] = 0 \wedge k[j] = 1) \vee \\ &\quad (h[j] = 1 \wedge k[j] = 0) && \text{(1c)} \end{aligned}$$

[1]In the problem there is no distinction between the maternal and paternal haplotypes.

If $\langle h, k \rangle \rhd g$ we indicate the fact that the haplotype h (respectively, k) contributes in the resolution of the genotype g writing $h \trianglelefteq g$ (resp., $k \trianglelefteq g$). We also say that h is a *resolvent* of g. This notation can be extended to set of haplotypes, writing $H = \{h_1, \ldots, h_l\} \trianglelefteq g$, with the meaning that $h_i \trianglelefteq g$ for all $i = 1, \ldots, l$. The operator $\oplus$ is defined accordingly.

Conditions (1a) and (1b) require that both haplotypes must have the same value in all homozygous sites, while condition (1c) states that in heterozygous sites the haplotypes must have different values.

Observe that, according to the definition, for a single genotype string the haplotype values at a given site are predetermined in the case of homozygous sites, whereas there is a freedom to choose between two possibilities at heterozygous places. This means that for a genotype string with l heterozygous sites there are 2^{l-1} possible pairs of haplotypes that resolve it.

As an example, consider the genotype $g = (0212)$, then possible pairs of haplotypes that resolve it are $\langle (0110), (0011) \rangle$ and $\langle (0010), (0111) \rangle$.

After these preliminaries we can state the *Haplotype Inference* problem as follows:

Definition 2 (Haplotype Inference problem)**.** Given a population of n individuals, each of them represented by a genotype string g_i of length m we are interested in finding a set ϕ of n pairs of (not necessarily distinct) haplotypes $\phi = \{\langle h_1, k_1 \rangle, \ldots, \langle h_n, k_n \rangle\}$, so that $\langle h_i, k_i \rangle \rhd g_i, i = 1, \ldots, n$. We call H the set of haplotypes used in the construction of ϕ, i.e., $H = \{h_1, \ldots, h_n, k_1, \ldots, k_n\}$.

From the mathematical point of view, there are many possibilities for building the set H, since there is an exponential number of possible haplotypes for each genotype. Therefore, a criterion has to be added to the model for evaluating the solution quality.

One natural model of the Haplotype Inference problem is the already mentioned *pure parsimony* approach that consists in searching for a solution that minimizes the total number of distinct haplotypes used or, in other words, $|H|$, the cardinality of the set H. A trivial upper bound for $|H|$ is $2n$ in the case of all genotypes resolved by a pair of distinct haplotypes. It has been shown that the Haplotype Inference problem under the pure parsimony criterion is APX-hard [17] and therefore NP-hard.

It is important to stress, at this point, that finding a proven optimal solution is not particularly relevant, because the criteria defining the objective functions are an approximation of an (unknown) actual quality function. Therefore, approximate approaches that are able to return solutions of a good quality, even if not optimal, are of notable practical importance.

3.2 Compatibility and complementarity

It is possible to define a graph that expresses the compatibility between genotypes, so as to avoid unnecessary checks in the determination of the resolvents. In the graph $\mathcal{G} = (G, E)$, the set of vertices coincides with the set of the genotypes. Two genotypes g_1, g_2 are connected by an edge if they are *compatible*, i.e., one or more common haplotypes can resolve both of them. The formal definition of this property is as follows.

Definition 3 (Genotypes compatibility)**.** Let g_1 and g_2 be two genotypes, g_1 and g_2 are *compatible* if, for all $j = 1, \dots, m$, the following conditions hold:

$$g_1[j] = 0 \quad \Rightarrow \quad g_2[j] \in \{0, 2\} \tag{2a}$$

$$g_1[j] = 1 \quad \Rightarrow \quad g_2[j] \in \{1, 2\} \tag{2b}$$

$$g_2[j] = 2 \quad \Rightarrow \quad g_2[j] \in \{0, 1, 2\} \tag{2c}$$

The same concept can be expressed also between a genotype and a haplotype as in the following definition.

Definition 4 (Compatibility between genotypes and haplotypes)**.** Let g be a genotype and h a haplotype, g and h are *compatible* if, for all $j = 1, \dots, m$, the following conditions hold:

$$g[j] = 0 \quad \Rightarrow \quad h[j] = 0 \tag{3a}$$

$$g[j] = 1 \quad \Rightarrow \quad h[j] = 1 \tag{3b}$$

$$g[j] = 2 \quad \Rightarrow \quad h[j] \in \{0, 1\} \tag{3c}$$

We denote this relation with $h \mapsto g$, and we write $h[j] \mapsto g[j]$ when the conditions hold for the single site j. Moreover with an abuse of notation we indicate with $h \mapsto \{g_1, g_2, \dots\}$ the set of all genotypes that are compatible with haplotype h.

Notice that the set of genotypes that are compatible with a haplotype can contain only mutually compatible genotypes (i.e., they form a clique in the compatibility graph).

We also point out that disconnected components of the compatibility graph are necessarily resolved by distinct haplotypes, therefore the optimal set of haplotypes is the union of the optimal sets of each disconnected subgraph. This property is exploited in a specific preprocessing phase of our algorithm.

In the remainder of this paper, we will call *ambiguous* genotypes those which contains at least a heterozigous site. Conversely, *non-ambiguous* genotypes contains only homozigous sites and are therefore resolved by a pair of identical haplotypes, in turn identical to the genotype.

Due to the resolution definition, when one of the two haplotypes composing the pair, say h, has been selected, then the other haplotype can be directly inferred from h and the genotype g thanks to the resolution conditions:

Proposition 1 (Haplotype complement)**.** *Given a genotype g and a haplotype $h \mapsto g$, there exists a unique haplotype k such that $h \oplus k = g$. The haplotype k is called the* complement *of h with respect to g and is denoted with $k = g \ominus h$.*

4 Master-slave algorithm

The Haplotype Inference problem definition makes constructive procedures very appealing. Indeed, a constructive procedure can incrementally build a set H of haplotypes which, taken in pairs, resolve the genotypes. Such a procedure can start from an empty set and add one or two haplotypes at a time, while it scans the set of genotypes G. The objective is to build H as small as possible, i.e., to find a minimal cardinality set of haplotypes that composes the phasing. To this aim, new haplotypes should be added to H only when necessary, i.e., when no pair of haplotypes already in H resolves the current genotype g. In principle, an optimal solution could be found if an oracle were to indicate the right order of visiting the genotypes and the right starting pair of haplotypes, along with properly defined criteria for choosing the values to assign. This is in general not possible, but an iterative and adaptive search strategy could be very effective in exploring these possibilities.

Algorithm 1 Master-slave high-levelframework

```
1: $\mathcal{P} \leftarrow$ buildInitialPopulation($n$)
2: evaluate($\mathcal{P}$)
3: while terminating conditions not met do
4:     $\mathcal{P}' \leftarrow$ applyGeneticOperators($\mathcal{P}$)
5:     evaluate($\mathcal{P}'$) {use slave algorithm}
6:     $\mathcal{P} \leftarrow$ bestOf($n$, $\mathcal{P}$, $\mathcal{P}'$)
7: end while
8: return min($\mathcal{P}$)
```

The general idea of the master-slave algorithm we propose is that it is possible to split the solution construction in two, nested, phases: in the first phase the parameters of a solution construction procedure are set by a *master* solver and in the second phase the solution is actually built by a *slave* solver. For example, in the first phase genotypes are ordered and this sequence is used to construct a solution in the second phase. In a sense, this algorithmic structure is such that the problem is decomposed into two, interdependent, sub-problems. The solution to the first problem is an input to the procedure that resolves the second, that actually computes a solution. In our case, the algorithm that provides the input to the second is a population-based metaheuristic, namely a genetic algorithm, while the solver for the second sub-problem is a greedy procedure. The quality of the genotype order provided by the master is then evaluated on the basis of the objective value of the solution built by the slave. In fact, the master explores a search space of 'parameters': the objective value of the points of this space is the value of the solution returned by the slave. The master-slave scheme we designed is detailed in Algorithm 1. The division of the problem into master and slave facilitates a separation of concerns that helps to design a more extensible solver and to have a simple and neat implementation.

The slave algorithm can be, in general, any constructive procedure that needs to be fed with an initial set of resolving haplotypes and some criteria to complete the solution. The procedure we will describe is variations of a constructive procedure known as *Clark's rule* [6]. Clark's inference rule exploits the property of complementarity between a genotype and a haplotype and it works as follows:

1. Let $G' \subseteq G$ be a set containing only genotypes with zero or one ambiguous site only. From that, an initial set of haplotypes H explaining G' can be inferred by complementarity. Be $G \leftarrow G'$.
2. Choose a pair (g, h) $g \in G, h \in H \,|\, h \trianglelefteq g$. If such pair exists, set $H \leftarrow H \cup \{g \ominus h\}$ and $G \leftarrow G \setminus \{g\}$.
3. Iterate step 2 until:
   - $G = \emptyset$: in that case H is a solution to the problem;
   - Cannot find a pair (g, h): in that case the algorithm fails.

Although the procedure is very fast and simple, it has some drawbacks. First of all, it needs an initial haplotype set to 'bootstrap', which might be impossible to obtain. In fact, in non-trivial instances the presence of non-ambiguous genotypes or containing one heterozigous site is quite unlikely.

Secondly, the algorithm must make an arbitrary choice in step 2 because here there are two sources of non-determinism:

1. The set of genotypes that can be solved by a haplotype in the current set H can have cardinality grater than one;
2. The genotype chosen could be solvable by more than one haplotype in H.

As a consequence of that, the quality of the solution returned is strongly dependent on the order in which genotypes are explained and haplotypes selected each step.

Nevertheless, the application of Clark's rule to Haplotype Inference is appealing because it naturally tries to re-use haplotypes in the current partial solution to explain remaining genotypes. It is reasonable to think that some haplotypes in an optimal solution to the Haplotype Inference cover more than one genotype.

The main idea is thus to employ a learning procedure, in the form of a population-based metaheuristic, to guide the non-deterministic choices made in step 2; therefore, such procedure have to learn a good optimal genotype resolution order and some criteria to choose the most promising haplotype when asked to resolve a genotype.

4.1 Master solver

In order to effectively use heuristic constrcutive procedures *à la* Clark, we have to learn genotype resolution order and haplotype selection. Our first version of the master-slave uses a master genetic algorithm to compute a permutation of the genotypes composing the instance.

To fully define a genetic algorithm one has to specify:

- an solution coding for defining the individual structure
- a selection procedure
- a population update procedure
- genetic operators and how they are applied to the population
- an evaluation criterion for assigning fitness to the individuals

An individual represents a single permutation of genotypes, that is, a genotype ordering. That ordering is crucial for the slave procedure to compute a good solution. Moreover, the evaluation criterion, in our master-slave architecture, is actually provided by the slave algorithm, that builds a solution and computes its value according to the objective function. The selection procedure is a simple roulette-wheel, in which individuals are taken for mating with a probability proportional to their respective fitness values. As for population update, we chose to implement a steady-state genetic algorithm. Concerning genetic operators, we adopted the usual definition of crossover and mutation for permutations of length n, which are detailed in the following.

Mutation: the mutation of an individual encoded as a permutation simply involves a random swap of two elements.
Crossover: first a random point-cut is chosen and the permutations p, q are split into two sub-sequences $(p_1, p_2), (q_1, q_2)$. Then the new permutations p' and q' are constructed in this way[2]: at first $p' = p_1$, then all the elements of q_2 which are not in p_1 are orderly appended to p'; if the length of p' is still less than n, the procedure keeps appending elements to p' taken from sub-sequence q_1 if the given elements are not already present in p_1.

[2]We take into account only the construction of p', since the other is symmetrical.

4.2 Slave solver

Our implementation of the slave is similar to the constructive procedure first presented in [2]. Basically, it implements Clark's Algorithm, but uses:

- the provided permutation to establish a genotype resolution order;
- a deterministic heuristic procedure in order to select among multiple candidate haplotypes a resolvent for the currently visited genotype.

The algorithm starts from the first genotype of the permutation π supplied by the master. At each step, each visited genotype has its corresponding index in the permutation removed. The procedure then deterministically chooses a number of haplotypes to be included in the partial solution. Excluding the case in which H already contains a pair of haplotypes resolving g, the are three different cases to be considered for the resolution of a genotype g in this step of the algorithm: (i) no resolving candidates in H, (ii) one candidate, (iii) more than one candidate. In the following, we detail the procedure defined for these cases:

Case (i): The procedure takes the first non-visited neighbor, according to the superimposed visiting order, and deterministically produces a haplotype h that solves both. If needed, ambiguous sites are arbitrarily set to 0. h and $g \ominus h$ are added to the solution.

Case (ii): When only one resolvent candidate is available, $g \ominus h$ is added to the solution. This is a straightforward application of complementarity.

Case (iii): When more than one candidate is present in H the procedure chooses deterministically from these using a heuristic similar to case *(i)*. Let H' be this candidate set: the procedure takes the first non-visited neighbor, according to the supplied visiting order, and chooses deterministically a haplotype from H' that explains both. In the case that such a neighbor is not available, any haplotype of H' would be fine and one is deterministically chosen.

In Algorithm 2, we give the pseudocode of the slave procedure in which:

- π is the permutation of the genotypes. At each iteration, the visited genotype is removed from π (line 9);
- H_p is the partial solution, that is the set of haplotypes that explain at least all of the visited genotypes.
- *resolveGenotype*(g, H_p) is the procedure that implements the heuristic for haplotype selection described above and returns an updated partial solution.

- *pickGenotypeAccordingToOrder*(π, G') uses the current permutation π to establish an ordering of the genotypes in G'.

The algorithm stops when a feasible solution has been built, that is, when π is empty.

Algorithm 2 Slave: Haplotype selection

```
 1: while $\pi$ is not empty do
 2:   $G' \leftarrow \{g \in G \mid \exists h \in H_p, h \trianglelefteq g\}$ {first choose among genotypes with at least
      one candidate (cases ((ii)) or ((iii)))}
 3:   if $G' = \emptyset$ then
 4:     $g \leftarrow \pi[0]$ {else try with a genotype with no candidates (case ((i)))}
 5:   else
 6:     $g \leftarrow pickGenotypeAccordingToOrder(\pi, G')$
 7:   end if
 8:   $H_p \leftarrow resolveGenotype(g, H_p)$
 9:   $\pi \leftarrow removeVisitedGenotype(\pi, g)$
10: end while
```

The choice of a deterministic slave procedure has one major advantage over a stochastic one because the evaluation of the solution returned by the master, i.e., the permutation, has a unique evaluation. In case of a stochastic algorithm for the slave, solution quality would be a stochastic variable and one would need to estimate it, for example by taking the average of a sample. This issue can be tackled by using the techniques adopted for algorithms tackling stochastic problems [3].

4.3 Preprocessing phase

The instances of the Haplotype Inference Problem can be reduced by analyzing their structure, while preserving the property that a solution to the reduced instance is a solution to the original one.

The first preprocessing step consists in eliminating duplicated genotypes; in fact, some instances contain identical genotypes that can be substituted by a unique representant without loosing information.

Furthermore, as introduced in Section 3, the analysis of the structure of the compatibility graph enables us to identify independent sub-instances. Indeed, the genotypes belonging to an isolated sub-graph, i.e., a disconnected component, identify a sub-instance that can be solved independently. Therefore, a solution to the original instance can be found by separately solving the sub-instances composing it. A special case of independent instance is represented by isolated nodes. The contribution of such a genotype to the solution of the Haplotype Inference instance is composed by a pair of haplotypes that, by definition of compatibility, cannot be used to resolve any other genotype.

5 Experimental Analysis

To investigate the performance variance with respect to parameter selection and to measure the contribution of the learning component, we have run four different versions of the algorithm:

ga : is the plain master-slave genetic algorithm;

ga $^+$: is the master-slave genetic with increased computational resources;

ga $_{no}$: is the *ga* version of the algorithm, but with learning component disabled;

ga $\triangleright$ ***TS*** : is the *ga* hybridized with the local search procedure (a tabu search) introduced in [1].

As explained in [1], for *ga*$\triangleright$*TS* we chose to have a simple integration to better asses the contribution to solution quality of each algorithm. For this reason, the two algorithms have been serialized: first the master-slave is run and the best solution returned is passed to the local search as initial state.

In Table 1 the main parameters of the algorithms are summarized. Parameters have been set after a *brute-force* analysis on a subset of instances.

alg.	**pop. size**	**max iter.**	**max idle iter.**
ga	100	500	100
ga$^+$	200	500	200
ga$_{no}$	100	500	100
ga$\triangleright$*TS*	100	500	100

Table 1: Parameters of the solvers.

The algorithms have been tested on well known benchmarks such as Harrower Hapmap, Harrower Uniform, Marchini SU1, Marchini SU2, Marchini SU3 and Marchini SU-100kb whose main characteristics are summarized in Table 2.

In Figures 1–6, boxplots relative to solution quality (left) and running times (right) are drawn. The plots represent statistics of ten independent runs of the algorithms on all the instances of each set; values shown are sums of the aforementioned measures on all the instances.

The algorithms have been developed in C++; the software was compiled with the GNU g++ compiler v. 4.1.3 and tested on a Intel Pentium 4 3.0GHz machine

Benchmark set	**N. of inst.**	**N. of geno.**	**N. of sites**
Harrower Uniform	200	10÷100	30÷50
Harrower Hapmap	24	5÷68	30÷75
Marchini SU1	100	90	179
Marchini SU2	100	90	171
Marchini SU3	100	90	187
Marchini SU-100kb	29	90	18

Table 2: Main features of the benchmarks.

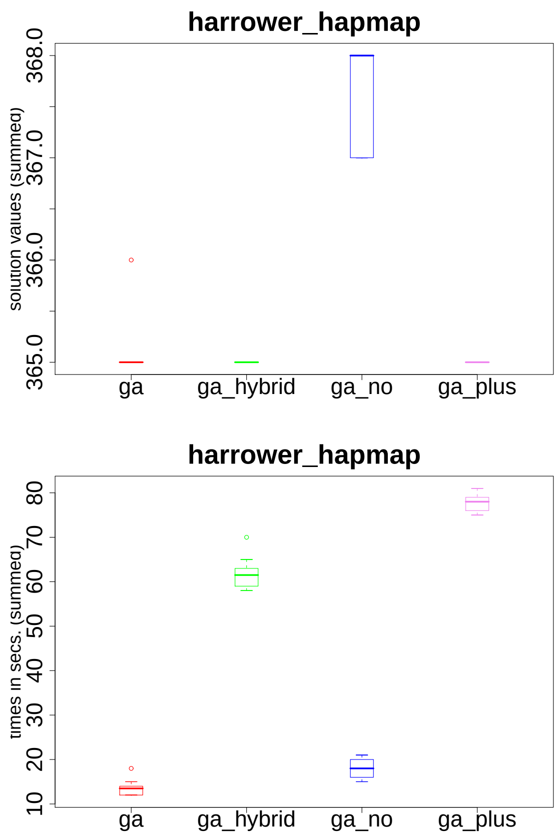


Figure 1: Harrower Hapmap

running Ubuntu 7.10 (kernel 2.6.22). The local search of ga▷TS was implemented with EasyLocal++ [8].

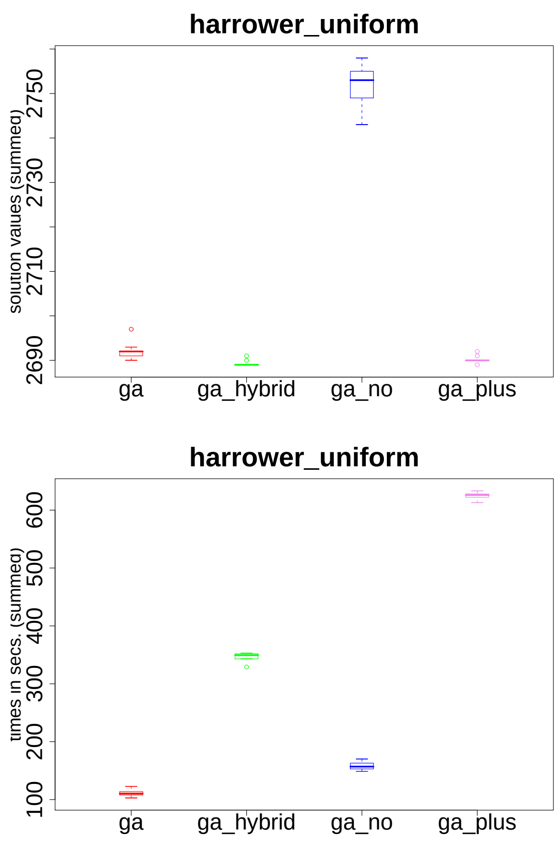


Figure 2: Harrower Uniform

If we consider ga and ga_{no}, graphics show that the learning component has a remarkable positive impact on solution quality while the cost in terms of computational resources is negligible. This proves that a simple constructive procedure can benefit from a careful selection of genotype resolution order.

In general, all solvers (except for ga_{no}) have the desirable property of being stable: although they are stochastic algorithms, the distribution of the solution values is rather packed around the mean and even zero for Harrower Hapmap.

As for solvers ga^{+} and $ga \triangleright TS$, their results in term of solution quality are comparable to ga, the basic version of the algorithm, while their execution times are considerably higher than that of ga. If we examine the graphics in more detail, we notice that the hybrid solver is better than ga^{+} in Harrower Uniform and Marchini SU-100kb since solution distribution has lower mean and standard deviation values. Analogously, ga^{+} is better than the hybrid in Marchini SU2 and Marchini SU3

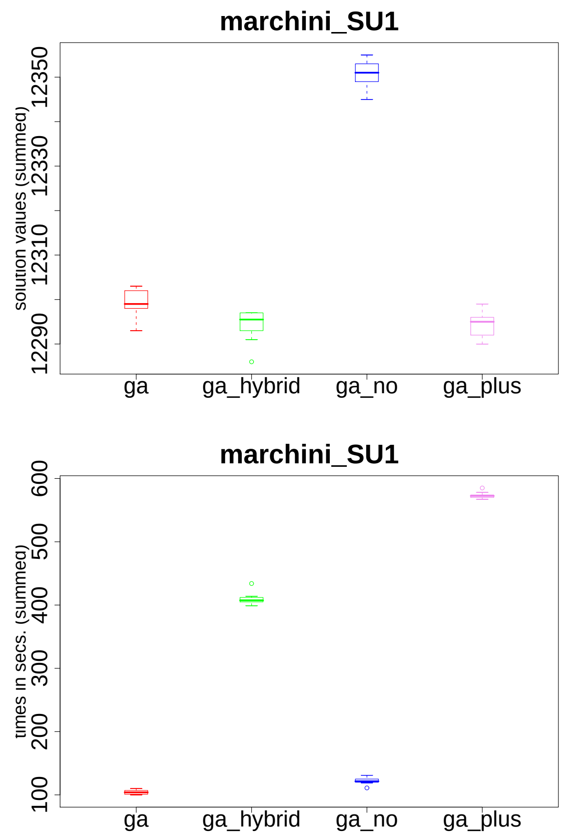


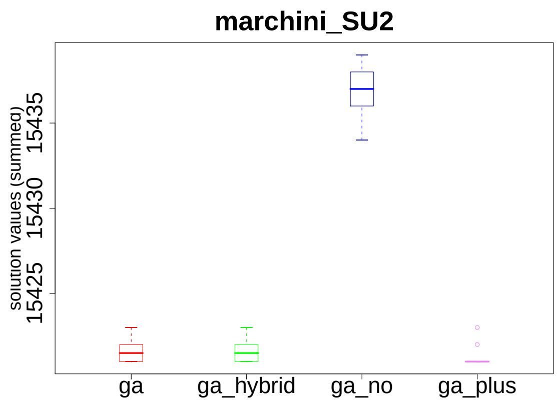
Figure 3: Marchini SU1

instances. On Marchini SU1 both algorithms exhibit the same behavior.

To assess the overall performace of the master-slave algorithm, we confronted *ga* to the hybrid solver presented in [1] that is, to the best of our knowledge, the state-of-the-art metaheuristic for Haplotype Inference. In Figures 7–12 the boxplots representing the statistics of both the algorithms w.r.t. solution quality and running time are plotted. We observe that the overall solution quality returned by *ga* is lower than that of the hybrid solver, but the execution time is remarkably shorter.

6 Conclusion and future work

We have presented a master-slave approach to Haplotype Inference by pure parsimony. The master searches in the space of parmeters used by the slave to build a solution. We have implemented the master as a genetic algorithm and the slave



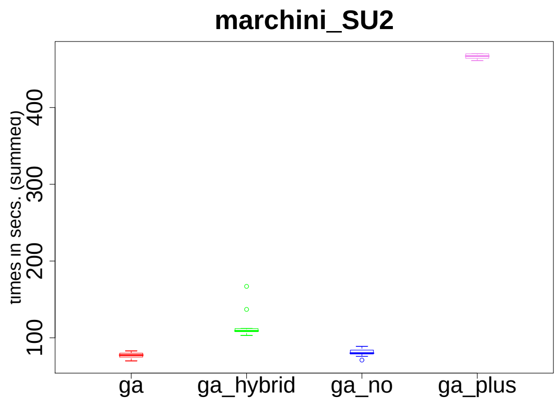
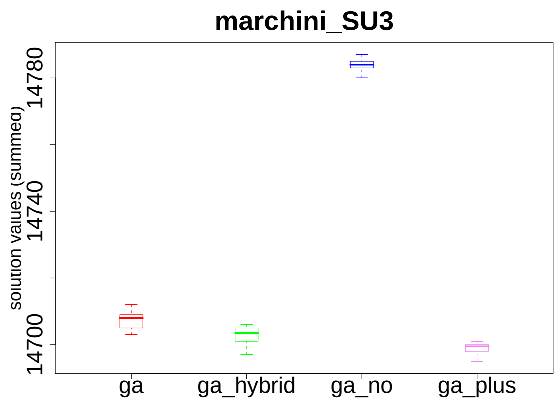


Figure 4: Marchini SU2

as a deterministic constructive procedure. Results show that the approach is very efficient and reaches a good balance between solution quality and execution time.

The approach is general and the algorithm can be extended in several directions. First of all, different procedures for implementing the slave can be adopted, especially stochastic ones, even though they require a more complex solution quality estimation. Besides extending the slave, also the algorithm of the master can be changed. For example, other population-based metaheuristics can be chosen, such as Ant colony optimization [9] or also stochastic local search techniques [14] can be tested. Furthermore, the approach could also be improved by adding a mechanism such as the one used in Benders decomposition techniques [13], in which the slave return the master a set of constraints to reduce the search space.



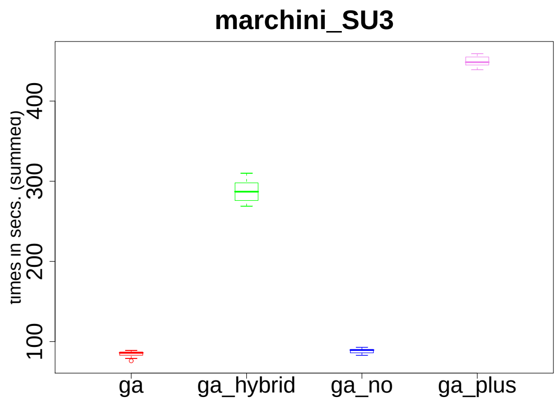


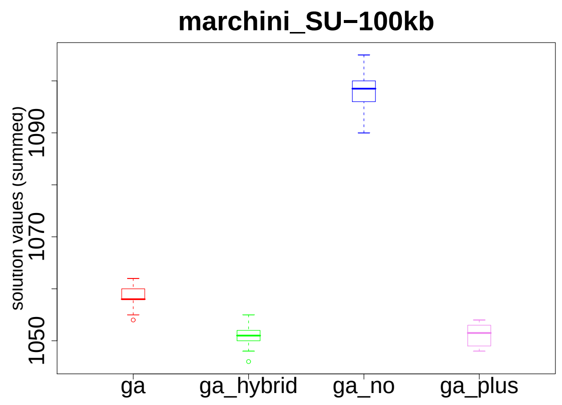
Figure 5: Marchini SU3

References

[1] S. Benedettini, L. Di Gaspero, and A. Roli. Towards a highly scalable hybrid metaheuristic for haplotype inference under parsimony. In *Proceedings of the 8th International Conference on Hybrid Intelligent Systems, HIS 2008*. IEEE Computer Society Press, 2008.

[2] S. Benedettini, A. Roli, and L. Di Gaspero. Two-level ACO for haplotype inference under pure parsimony. In *Ant Colony Optimization and Swarm Intelligence, 6th International Workshop, ANTS 2008*, volume 5217 of *Lecture Notes in Computer Science*. Springer–Verlag, 2008.

[3] M. Birattari, P. Balaprakash, T. Stützle, and M. Dorigo. Estimation-based local search for stochastic combinatorial optimization using delta evaluations: A case



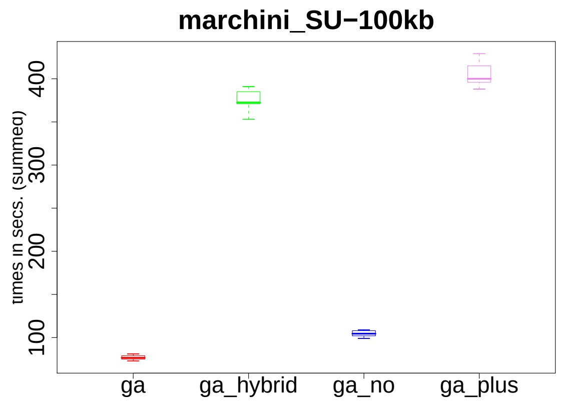


Figure 6: Marchini SU-100kb

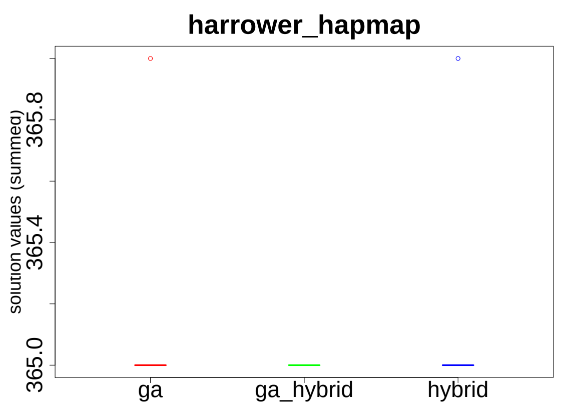
study on the probabilistic traveling salesman problem. *INFORMS Journal on Computing*, 20(4), 2008.

[4] C. Blum and A. Roli. *Hybrid Metaheuristics: An Introduction*, chapter 1. Hybrid Metaheuristics: An Emerging Approach to Optimization. Springer-Verlag, 2008.

[5] D. G. Brown and I. M. Harrower. Integer programming approaches to haplotype inference by pure parsimony. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(2):141–154, 2006.

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

[7] L. Di Gaspero and A. Roli. Stochastic local search for large-scale instances of



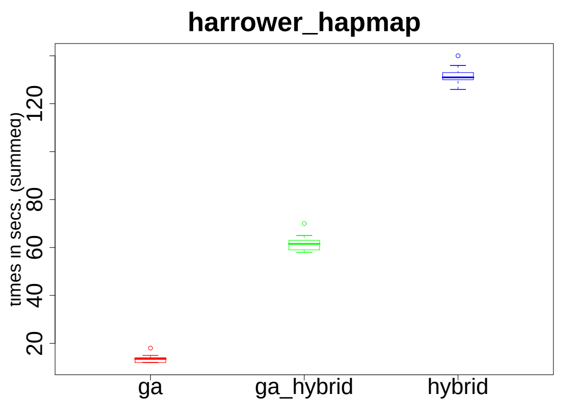
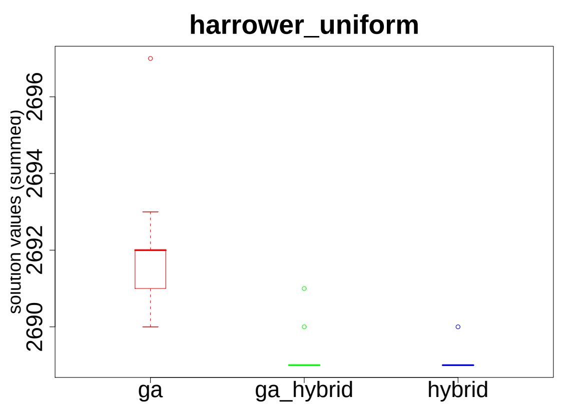


Figure 7: Harrower Hapmap



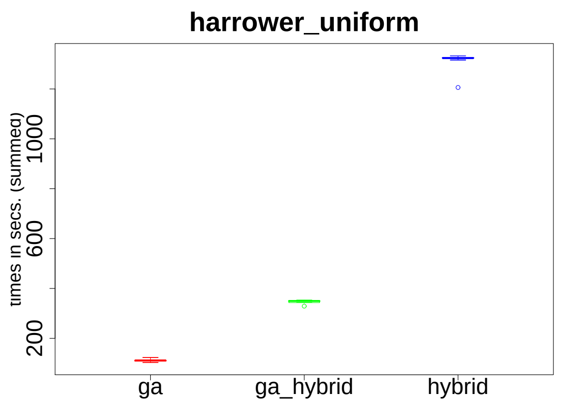
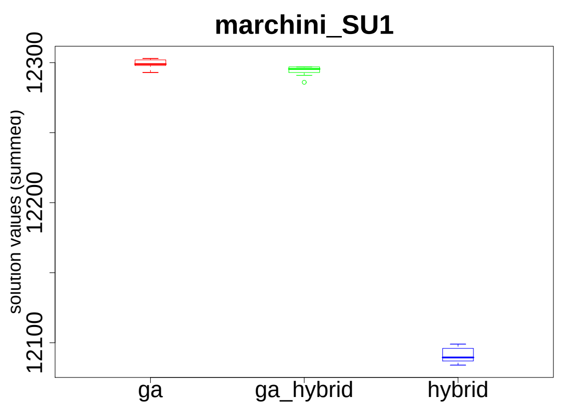


Figure 8: Harrower Uniform



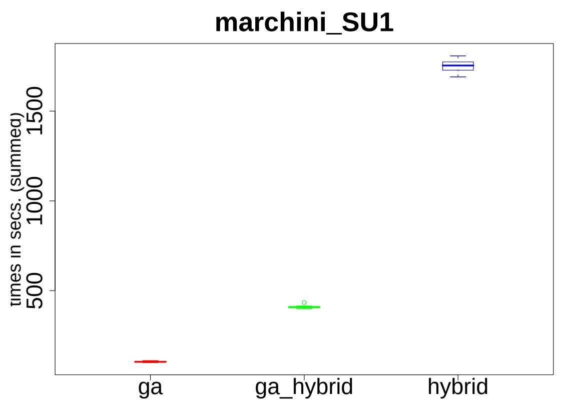
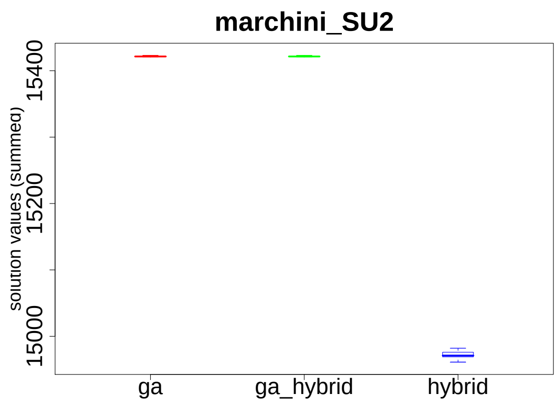


Figure 9: Marchini SU1



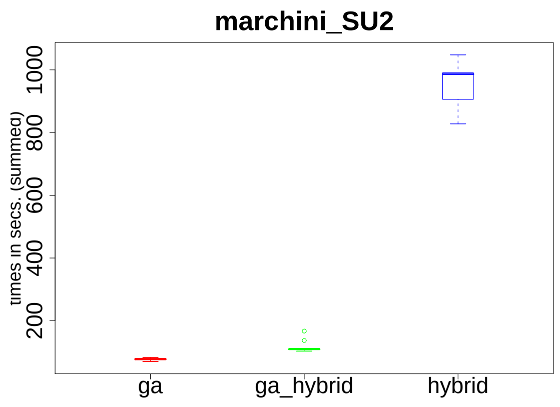


Figure 10: Marchini SU2

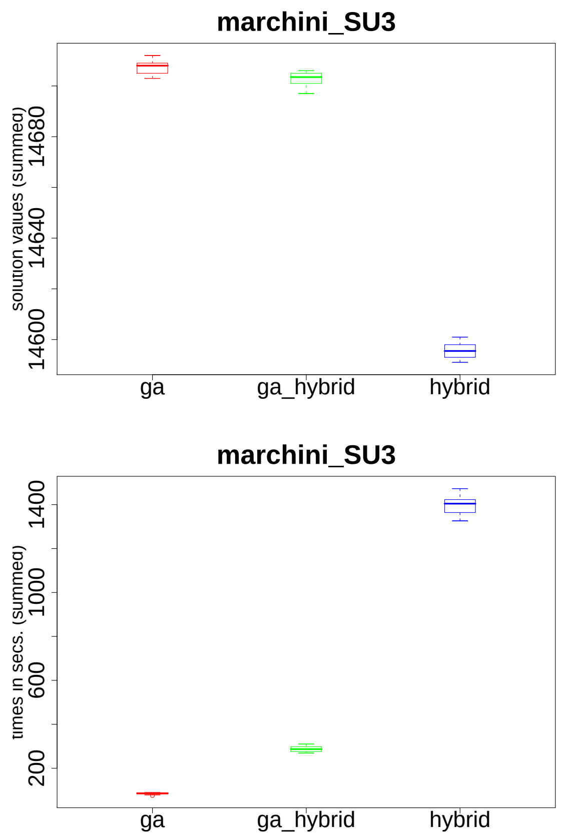


Figure 11: Marchini SU3

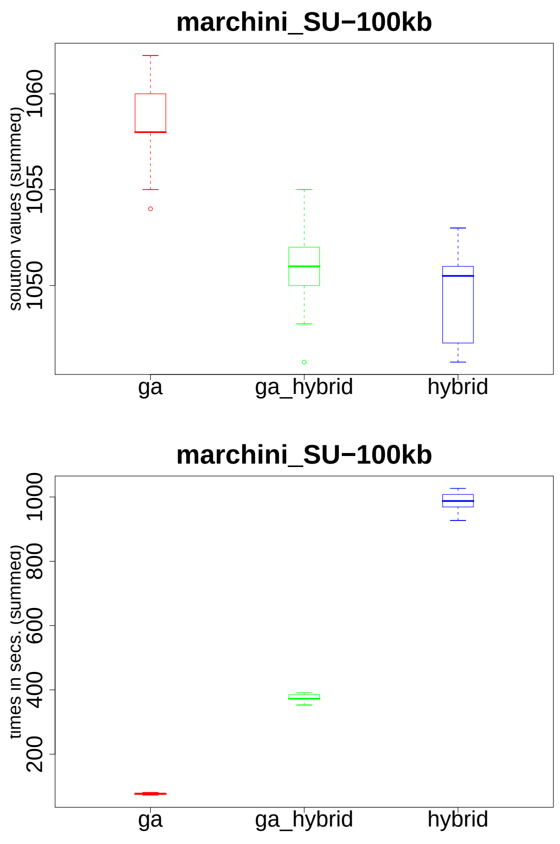


Figure 12: Marchini SU-100kb

the haplotype inference problem by pure parsimony. *Journal of Algorithms in Logic, Informatics and Cognition*, 2008. To appear.

[8] L. Di Gaspero and A. Schaerf. EasyLocal++: An object-oriented framework for flexible design of local search algorithms. *Software—Practice and Experience*, 33(8):733–765, 2003.

[9] M. Dorigo and T. Stützle. *Ant Colony Optimization.* MIT Press, Cambridge, MA, USA, 2004.

[10] A. Graça, J. Marques-Silva, I. Lynce, and A. L. Oliveira. Efficient haplotype inference with pseudo-boolean optimization. In *Algebraic Biology, Second International Conference, AB 2007, Castle of Hagenberg, Austria, July 2-4, 2007, Proceedings*, volume 4545 of *Lecture Notes in Computer Science*, pages 125–139. Springer-Verlag, Berlin-Heidelberg, Germany, 2007.

[11] D. Gusfield. Haplotype inference by pure parsimony. In *Combinatorial Pattern Matching (CPM 2003), Proceedings of the 14th Annual Symposium*, volume 2676 of *Lecture Notes in Computer Science*, pages 144–155, Berlin-Heidelberg, Germany, 2003. Springer-Verlag.

[12] B. V. Halldórsson, V. Bafna, N. Edwards, R. Lippert, S. Yooseph, and S. Istrail. A survey of computational methods for determining haplotypes. In *Computational Methods for SNPs and Haplotype Inference*, volume 2983 of *Lecture Notes in Computer Science*, pages 26–47. Springer, 2002.

[13] J. Hooker. *Integrated Methods for Optimization.* Operations Research & Management Science. Springer, 2007.

[14] H. Hoos and T. Stützle. *Stochastic Local Search Foundations and Applications.* Morgan Kaufmann Publishers, San Francisco, CA (USA), 2005.

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

[16] K. Kalpakis and P. Namjoshi. Haplotype phasing using semidefinite programming. In *BIBE*, pages 145–152. IEEE Computer Society, 2005.

[17] G. Lancia, M. C. Pinotti, and R. Rizzi. Haplotyping populations by pure parsimony: Complexity of exact and approximation algorithms. *INFORMS Journal on Computing*, 16(4):348–359, 2004.

[18] I. Lynce and J. Marques-Silva. Efficient haplotype inference with boolean satisfiability. In *Proceedings of the 21st National Conference on Artificial Intelligence*

and the Eighteenth Innovative Applications of Artificial Intelligence Conference, Menlo Park, CA, USA, 2006. AAAI Press.

[19] I. Lynce and J. Marques-Silva. SAT in bioinformatics: Making the case with haplotype inference. In *SAT*, volume 4121 of *Lecture Notes in Computer Science*, pages 136–141, Berlin-Heidelberg, Germany, 2006. Springer-Verlag.

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

[21] The International HapMap Consortium. A haplotype map of the human genome. *Nature*, 437, 2005.

[22] R.-S. Wang, X.-S. Zhang, and L. Sheng. Haplotype inference by pure parsimony via genetic algorithm. In *Operations Research and Its Applications: the Fifth International Symposium (ISORA'05), Tibet, China, August 8–13*, volume 5 of *Lecture Notes in Operations Research*, pages 308–318. Beijing World Publishing Corporation, Beijing, People Republic of China, 2005.